Step1 Create MongoDB using Persistent Volume on GKE, and insert records into it
1. Enable Kubernetes Engine API and Active Google Cloud Shell
create a kubernetes cluster with three nodes:
gcloud container clusters create kubia --num-nodes=3 --machine-type=e2-micro --zone=us-central1

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ gcloud container clusters create kubia --num-nodes=1 --machine-type=e2-micro --region=us-central1
NAME: kubia
LOCATION: us-central1
MASTER_VERSION: 1.30.5-gke.1443001
MASTER_IP: 34.172.211.35
MACHINE_TYPE: e2-micro
NODE_VERSION: 1.30.5-gke.1443001
NUM_NODES: 3
STATUS: RUNNING
```

2. Let's create a Persistent Volume first,
gcloud compute disks create --size=10GiB --zone=us-west1-b mongodb

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ gcloud compute disks create --size=10GiB --zone=us-central1-a mongodb
WARNING: You have selected a disk size of under [200GB]. This may result in poor I/O performance. For more information, se
Created [https://www.googleapis.com/compute/v1/projects/cs571-project-1-438018/zones/us-central1-a/disks/mongodb].
NAME: mongodb
ZONE: us-central1-a
SIZE_GB: 10
TYPE: pd-standard
STATUS: READY
```

3. Now create a mongodb deployment with this yaml file:
$ nano mongodb-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
spec:
  selector:
    matchLabels:
      app: mongodb
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongo
        image: mongo
```

```
      ports:
      - containerPort: 27017
      volumeMounts:
      - name: mongodb-data
        mountPath: /data/db
    volumes:
    - name: mongodb-data
      gcePersistentDisk:
        pdName: mongodb
        fsType: ext4
```

$ kubectl apply -f mongodb-deployment.yaml

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl apply -f mongodb-deployment.yaml
deployment.apps/mongodb-deployment created
rpuranda464@cloudshell:~ (cs571-project-1-438018)$
```

4. Check if the deployment pod has been successfully created and started running:
$ kubectl get pods

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
mongodb-deployment-5b7dc756c6-rnfrr   1/1     Running   0          32s
rpuranda464@cloudshell:~ (cs571-project-1-438018)$
```

5. Create a service for the mongodb, so it can be accessed from the outside
$ nano mongodb-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  type: LoadBalancer
  ports:
    - port: 27017
      targetPort: 27017
      nodePort: 30002
  selector:
    app: mongodb
```

$ kubectl apply -f mongodb-service.yaml

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ nano mongodb-service.yaml
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl apply -f mongodb-service.yaml
service/mongodb-service created
rpuranda464@cloudshell:~ (cs571-project-1-438018)$
```

6. Wait couple of minutes, and check if the service is up:
$ kubectl get svc

```
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)           AGE
kubernetes        ClusterIP      34.118.224.1    <none>         443/TCP           21m
mongodb-service   LoadBalancer   34.118.229.93   34.56.112.37   27017:31929/TCP   35s
```

7. Now try and see if MongoDB is functioning for connections using External-IP
$ kubectl exec -it mongodb-deployment-replace-with-your-pod-name -- bash
$ kubectl exec -it mongodb-deployment-594c77dcdf-rm5c2 -- bash

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl exec -it mongodb-deployment-5b7dc756c6-rnfrr -- bash
root@mongodb-deployment-5b7dc756c6-rnfrr:/#
```

8. 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 accessed using the External-IP

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl exec -it mongodb-deployment-5b7dc756c6-rnfrr -- bash
root@mongodb-deployment-5b7dc756c6-rnfrr:/# mongo 34.56.112.37
```

Type exit to exit mongodb and back to our google console

```
MongoDB shell version v4.4.4
connecting to: mongodb://35.197.111.141:27017/test?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("b6684190-20f0-40d9-8051-e8bc05d08b6c") }
MongoDB server version: 4.4.4
---
The server generated these startup warnings when booting:
        2021-03-25T00:01:30.075+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engi
ne. See http://dochub.mongodb.org/core/prodnotes-filesystem
        2021-03-25T00:01:31.629+00:00: Access control is not enabled for the database. Read and write access to data and
 configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

```
test> exit
root@mongodb-deployment-5b7dc756c6-rnfrr:/# exit
exit
rpuranda464@cloudshell:~ (cs571-project-1-438018)$
```

9. We need to insert some records into the MongoDB for later use:
$ node
Enter the following line by line:

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ node
Welcome to Node.js v22.11.0.
Type ".help" for more information.
> const { MongoClient } = require('mongodb');
undefined
> const url = "mongodb://34.56.112.37/mydb";
undefined
>
> async function main() {
...   const client = await MongoClient.connect(url);
...   const db = client.db("studentdb");
...
...   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 }
...   ];
...
...   try {
...     const res = await db.collection("students").insertMany(docs);
...     console.log(res.insertedCount);  // Prints the number of inserted documents
...
...     const result = await db.collection("students").findOne({ "student_id": 11111 });
...     console.log(result);  // Prints the result of the find query
...   } catch (err) {
...     console.error(err);
...   } finally {
...     client.close();  // Close the connection after operations are done
...   }
... }
```

If everything is correct, you should see this:
3 means three records was inserted, and we tried search for student_id=11111

```
...    const client = await MongoClient.connect(url);
...    const db = client.db("studentdb");
...
...    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 }
...    ];
...
...    try {
...      const res = await db.collection("students").insertMany(docs);
...      console.log(res.insertedCount);  // Prints the number of inserted documents
...
...      const result = await db.collection("students").findOne({ "student_id": 11111 });
...      console.log(result);  // Prints the result of the find query
...    } catch (err) {
...      console.error(err);
...    } finally {
...      client.close();  // Close the connection after operations are done
...    }
... }
undefined
>
> main();
Promise {
  <pending>,
  [Symbol(async_id_symbol)]: 50,
  [Symbol(trigger_async_id_symbol)]: 6
}
> 3
{
  _id: new ObjectId('6732a59a6edbeca9c11cebbb'),
  student_id: 11111,
  student_name: 'Bruce Lee',
  grade: 84
}
```

Step2: Modify our student Server to get records from MongoDB and deploy to GKE

1. Create a studentServer.js
$ nano studentServer.js

```
var http = require('http');
var url = require('url');
var mongodb = require('mongodb');
const { MONGO_URL, MONGO_DATABASE } = process.env;

// Check if MONGO_URL and MONGO_DATABASE are set
if (!MONGO_URL || !MONGO_DATABASE) {
    console.error("MONGO_URL or MONGO_DATABASE environment variable is missing");
    process.exit(1);  // Exit the program if necessary environment variables are missing
}

// MongoDB connection string
var MongoClient = mongodb.MongoClient;
var uri = `mongodb://${MONGO_URL}/${MONGO_DATABASE}`;
```

```
console.log("MongoDB URI:", uri);  // Log the connection string for debugging

var server = http.createServer(function (req, res) {
   // Parse the request URL, e.g., /api/score?student_id=1111
   var parsedUrl = url.parse(req.url, true);
   var student_id = parseInt(parsedUrl.query.student_id, 10); // Ensure the student_id is parsed
as an integer

   // Check if the URL matches /api/score
   if (/^\/api\/score/.test(req.url)) {
      if (isNaN(student_id)) {
         res.writeHead(400, { 'Content-Type': 'application/json' });
         res.end(JSON.stringify({ error: 'Invalid student_id parameter' }) + '\n');
         return;
      }

      // Connect to the MongoDB database
      MongoClient.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true }, function
(err, client) {
         if (err) {
            res.writeHead(500, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify({ error: 'Failed to connect to database' }) + '\n');
            return;
         }

         var db = client.db(MONGO_DATABASE); // Use the database name from the
environment variable
         db.collection("students").findOne({ "student_id": student_id }, (err, student) => {
            if (err) {
               res.writeHead(500, { 'Content-Type': 'application/json' });
               res.end(JSON.stringify({ error: err.message }) + '\n');
               return;
            }

            if (student) {
               // Only return the required fields
         const response = {
                  student_id: student.student_id,
                  student_name: student.student_name,
                  student_score: student.student_score
               };
               res.writeHead(200, { 'Content-Type': 'application/json' });
               res.end(JSON.stringify(response) + '\n');
            } else {
```

```
                res.writeHead(404, { 'Content-Type': 'application/json' });
                res.end(JSON.stringify({ error: 'Student not found' }) + '\n');
            }
        });
    });
  } else {
    res.writeHead(404, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({ error: 'Wrong URL, please try again' }) + '\n');
  }
});

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

2. Create a Dockerfile
$ nano Dockerfile

```
FROM node:18
WORKDIR /app
ADD studentServer.js /app/studentServer.js
# Install dependencies, including mongodb
RUN npm install -g npm@latest
RUN npm install mongodb
CMD ["node", "studentServer.js"]
```

3. Build the studentserver docker image:
Docker login first:
$ docker build -t rash0101/studentserver

Make sure the image create successfully:
$ docker images

4. Push the docker image to dockerhub:
$ docker push rash0101/studentserver

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ docker tag yourdockerhubID/studentserver rash0101/studentserver
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ docker push rash0101/studentserver
Using default tag: latest
The push refers to repository [docker.io/rash0101/studentserver]
c2215f6b11f8: Pushed
ab90d83fa34a: Mounted from library/node
8ee318e54723: Mounted from library/node
e6695624484e: Mounted from library/node
da59b99bbd3b: Mounted from library/node
5616a6292c16: Mounted from library/node
f3ed6cb59ab0: Mounted from library/node
654f45ecb7e3: Mounted from library/node
2c40c66f7667: Mounted from library/node
latest: digest: sha256:8e0d45873e33392992dd961b7afffcd95919ae679e2e452293602a2dffc7f72a size: 2213
rpuranda464@cloudshell:~ (cs571-project-1-438018)$
```

Step3 Create a python flask bookshelf REST API and deploy on GKE
1. Created bookshelf.py

```
from flask import Flask, request, jsonify
from flask_pymongo import PyMongo
from bson.objectid import ObjectId
import socket
import os

app = Flask(__name__)

# MongoDB URI configuration
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_books():
    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="Book 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 = "Book updated successfully!"
    else:
        message = "No book found!"
    return jsonify(message=message)

@app.route("/book/<id>", methods=["DELETE"])
def delete_book(id):
    response = db.bookshelf.delete_one({"_id": ObjectId(id)})
    if response.deleted_count:
        message = "Book deleted successfully!"
    else:
        message = "No book found!"
    return jsonify(message=message)

@app.route("/tasks/delete", methods=["POST"])
def delete_all_books():
    db.bookshelf.remove()
    return jsonify(message="All books deleted!")
```

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

2. Create a requirements.txt file and a Dockerfile:

```
Flask==1.1.2
Flask-PyMongo==2.3.0
gunicorn==20.0.4
requests==2.25.1
https://storage.googleapis.com/velostrata-release/gce-v2v/gce-v2v.tar.gz
```

```
FROM python:alpine3.7
COPY . /app
WORKDIR /app
RUN pip install --upgrade pip
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 rash0101/bookshelf

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ docker build -t rash0101/bookshelf .
[+] Building 0.6s (7/7) FINISHED
```

Make sure this step build successfully
4. Push the docker image to your dockerhub
$ docker push rash0101/bookshelf

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ docker push rash0101/bookshelf
Using default tag: latest
The push refers to repository [docker.io/rash0101/bookshelf]
c2215f6b11f8: Mounted from rash0101/studentserver
ab90d83fa34a: Mounted from rash0101/studentserver
8ee318e54723: Mounted from rash0101/studentserver
e6695624484e: Mounted from rash0101/studentserver
da59b99bbd3b: Mounted from rash0101/studentserver
5616a6292c16: Mounted from rash0101/studentserver
f3ed6cb59ab0: Mounted from rash0101/studentserver
654f45ecb7e3: Mounted from rash0101/studentserver
2c40c66f7667: Mounted from rash0101/studentserver
latest: digest: sha256:8e0d45873e33392992dd961b7afffcd95919ae679e2e452293602a2dffc7f72a size: 2213
```

Step4 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

1. Create a file named studentserver-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: studentserver-config
data:
  MONGO_URL: "mongodb://34.56.112.37:27017"
  MONGO_DATABASE: mydb
```

2. Create a file named bookshelf-configmap.yaml
In MONGO_URL section, you need to change it to your own External_IP address
Notice: the reason of creating those two ConfigMap is to avoid re-building docker image again if the mongoDB pod restarts with a different External-IP

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: bookshelf-config
data:
  MONGO_URL: "mongodb://34.56.112.37:27017”
  MONGO_DATABASE: "mydb"
```

Step5 Expose 2 application using ingress with Nginx, so we can put them on the same domain but different Path

1. Create studentserver-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web  # This should be 'studentserver-deployment'
  labels:
    app: studentserver-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web  # This can stay as 'web', as it defines the container label
    spec:
```

```
    containers:
     - image: rash0101/studentserver
       imagePullPolicy: Always
       name: web
       ports:
         - containerPort: 8080
       env:
         - name: MONGO_URL
           valueFrom:
             configMapKeyRef:
               name: studentserver-config
               key: MONGO_URL
         - name: MONGO_DATABASE
           valueFrom:
             configMapKeyRef:
               name: studentserver-config
               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: rash0101/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 a studentserver-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  type: LoadBalancer
  ports:
    # Service port within the cluster
    - port: 8080
      # Port to contact inside the container
      targetPort: 8080
      nodePort: 30001
  selector:
    app: web  # Ensure this matches the labels on the pod/deployment
```

4. Create a bookshelf-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: bookshelf-service
spec:
  type: LoadBalancer
  ports:
    # Service port within the cluster
    - port: 5000
      # Port to contact inside the container
```

```
    targetPort: 5000
    nodePort: 30000
  selector:
    app: bookshelf-deployment  # Ensure this matches the labels on the pod/deployment
```

5. Start minikube
$ minikube start

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ minikube start
* minikube v1.34.0 on Ubuntu 24.04 (amd64)
  - MINIKUBE_FORCE_SYSTEMD=true
  - MINIKUBE_HOME=/google/minikube
  - MINIKUBE_WANTUPDATENOTIFICATION=false
* Automatically selected the docker driver. Other choices: none, ssh
* Using Docker driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.45 ...
* Downloading Kubernetes v1.31.0 preload ...
    > preloaded-images-k8s-v18-v1...:  326.69 MiB / 326.69 MiB  100.00% 252.26
    > gcr.io/k8s-minikube/kicbase...:  487.90 MiB / 487.90 MiB  100.00% 100.05
* Creating docker container (CPUs=2, Memory=4000MB) ...
* Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
  - kubelet.cgroups-per-qos=false
  - kubelet.enforce-node-allocatable=""
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
rpuranda464@cloudshell:~ (cs571-project-1-438018)$
```

6. Start Ingress
$ minikube addons enable ingress

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ 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/controller:v1.9.4
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20231011-8b53cabe0
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20231011-8b53cabe0
* Verifying ingress addon...
* The 'ingress' addon is enabled
```

7. Create studentserver related pods and start service using the above yaml files:
$ kubectl apply -f studentserver-deployment.yaml
$ kubectl apply -f studentserver-configmap.yaml
$ kubectl apply -f studentserver-service.yaml

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl apply -f studentserver-deployment.yaml
deployment.apps/web created
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl apply -f studentserver-configmap.yaml
configmap/studentserver-config created
```

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl apply -f studentserver-service.yaml
service/web created
```

8. Create bookshelf related pods and start service using the above yaml files:
$ kubectl apply -f bookshelf-deployment.yaml
$ kubectl apply -f bookshelf-configmap.yaml
$ kubectl apply -f bookshelf-service.yaml

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl apply -f bookshelf-deployment.yaml
deployment.apps/bookshelf-deployment created
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl apply -f bookshelf-configmap.yaml
configmap/bookshelf-config created
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl apply -f bookshelf-service.yaml
service/bookshelf-service created
```

9. Check if all the pods are running correctly
$ kubectl get pods

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
bookshelf-deployment-5886b446bd-qxb52  1/1     Running   0          4m18s
web-6bdd9dbfc4-z95vm                   1/1     Running   0          20m
rpuranda464@cloudshell:~ (cs571-project-1-438018)$
```

10. Create an ingress service yaml file 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
```

11. Create the ingress service using the above yaml file
$ kubectl apply -f studentservermongoIngress.yaml

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl apply -f /home/rpuranda464/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

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ kubectl get ingress
NAME     CLASS   HOSTS               ADDRESS        PORTS   AGE
server   nginx   cs571.project.com   192.168.49.2   80      102s
rpuranda464@cloudshell:~ (cs571-project-1-438018)$
```

13. Add Address to /etc/hosts
$ sudo vi etc/hosts
Add the address you got in the last step in the file

192.168.49.2 cs571.project.com

```
192.168.49.2 cs571.project.com
```

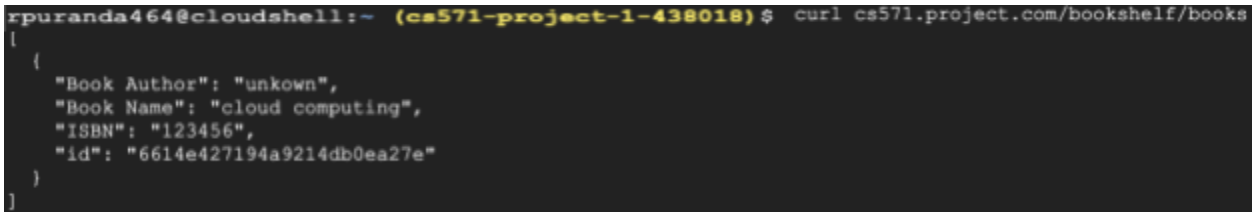
14. If everything goes smoothly, you should be able to access your application:
$ curl cs571.project.com/studentserver/api/score?student_id=11111

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ curl cs571.project.com/studentserver/api/score?student_id=11111
{"_id":"6614e38bc9cba3f570b486b5","student_id":11111,"student_name":"Bruce Lee","grade":84}
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ curl cs571.project.com/studentserver/api/score?student_id=22222
{"_id":"6614e39bc9cba3f570b486b6","student_id":22222,"student_name":"Jackie Chen","grade":93}
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ curl cs571.project.com/studentserver/api/score?student_id=33333
{"_id":"6614e3a8c9cba3f570b486b7","student_id":33333,"student_name":"Jet Li","grade":88}
```

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

```
rpuranda464@cloudshell:~ (cs571-project-1-438018)$ curl -X POST -d "{\"book_name\": \"cloud computing\",\"book_author\": \"unknown\", \"isbn\": \"123456\"
}" http://cs571.project.com/bookshelf/book
{
  "message": "Task saved successfully!"
}
```

$ curl cs571.project.com/bookshelf/books



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

Delete a book
$ curl -X DELETE cs571.project.com/bookshelf/book/6052fffbd09c0d7f8cf1f93