CS 571 Advanced cloud computing

19615 Gayatri Kolekar

Signature Project

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 kubia --num-nodes=1 --machine-type=e2-micro --region=us-west1

Wait for the creation to finish,

2. Let's create a Persistent Volume first, if you have created a persistent volume for the week10's homework, you can skip this one

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

3. Now create a mongodb deployment with this yaml filec

```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:
        # by default, the image is pulled from docker hub
        - image: mongo
          name: 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

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl apply -f mongodb-deployment.yaml
deployment.apps/mongodb-deployment 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

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl get pods
NAME                                 READY   STATUS              RESTARTS   AGE
mongodb-deployment-57dc68b4bd-28cd6   0/1     ContainerCreating   0          103s
```

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

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  type: LoadBalancer
  ports:
      # service port in cluster
    - port: 27017
      # port to contact inside container
      targetPort: 27017
  selector:
    app: mongodb
```

kubectl apply -f mongodb-service.yaml

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ 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

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

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl get svc
NAME              TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)           AGE
kubernetes        ClusterIP      10.36.0.1      <none>          443/TCP           3d22h
mongodb-service   LoadBalancer   10.36.13.86    35.230.79.93    27017:32717/TCP   3d17h
```

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

kubectl exec -it mongodb-deployment-replace-with-your-pod-name -- bash

Now you are inside the mongodb deployment pod

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl exec -it mongodb-deployment-57dc68b4bd-xfwsd
 -- bash
error: unable to upgrade connection: container not found ("mongo")
```

Try

mongo External-IP

You should see something like this, which means your mongoDB is up and can be accessd using the External-IP

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

Node

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ node
Welcome to Node.js v12.14.1.
Type ".help" for more information.
>
```

Enter the following line by line

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://EXTERNAL-IP/mydb"
// Connect to the db
MongoClient.connect(url,{ useNewUrlParser: true, useUnifiedTopology: true }, function(err, client){
if (err)
 throw err;
 // create a document to be inserted
var 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}
]
db.collection("students").insertMany(docs, function(err, res){
if(err) throw err;
console.log(res.insertedCount);
client.close();
});
db.collection("students").findOne({"student_id": 11111},
```

function(err, result){

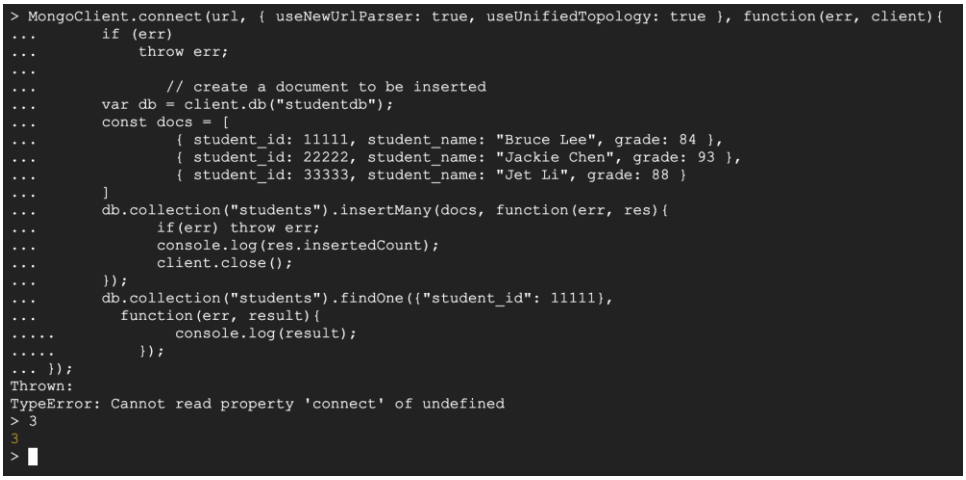
console.log(result);

});

});

If Everything is correct, you should see this,

3 means three records was inserted, and we tried search for student_id=11111



Step2 Modify our studentServer to get records from MongoDB and deploy to GKE

1. Create a studentServer

var http = require('http');

var url = require('url');

var mongodb = require('mongodb');

const { MONGO_URL, MONGO_DATABASE } = process.env;

// - Expect the request to contain a query

// string with a key 'student_id' and a student ID as

// the value. For example

```
// /api/score?student_id=1111
// - The JSON response should contain only 'student_id', 'student_name'
// and 'student_score' properties. For example:
//
// {
 // "student_id": 1111,
// "student_name": Bruce Lee,
 // "student_score": 84
 // }
//
var MongoClient = mongodb.MongoClient;
var uri = `mongodb://${MONGO_URL}/${MONGO_DATABASE}`;
// Connect to the db
console.log(uri);
var server = http.createServer(function (req, res) {
var result;
// req.url = /api/score?student_id=11111
var parsedUrl = url.parse(req.url, true);
var student_id = parseInt(parsedUrl.query.student_id);
// match req.url with the string /api/score
if (/^\/api\/score/.test(req.url)) {
// e.g., of student_id 1111
MongoClient.connect(uri,{ useNewUrlParser: true, useUnifiedTopology: true }, function(err, client){
if (err)
throw err;
var db = client.db("studentdb");
db.collection("students").findOne({"student_id":student_id},
(err, student) => {
if(err)
```

```
throw new Error(err.message, null);
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");
}
 });
});
} else {
res.writeHead(404);
res.end("Wrong url, please try again\n");
}
});
server.listen(8080);
```

2. Create Dockerfile

```
FROM node:7
ADD studentServer.js /studentServer.js
ENTRYPOINT ["node", "studentServer.js"]
RUN npm install mongodb
```

3. Build the studentserver docker image

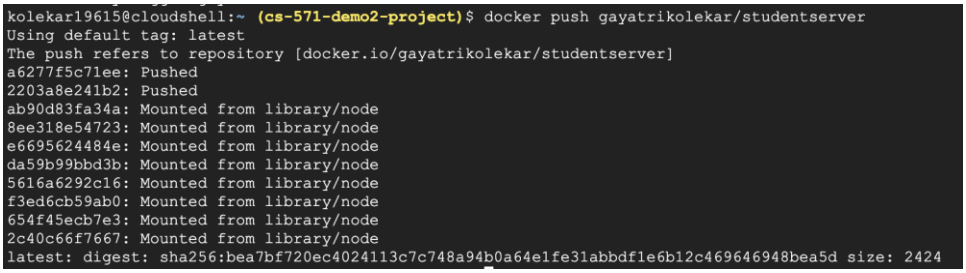
docker build -t yourdockerhubID/studentserver .

Make sure there is no error

```
Successfully built 96417c5af9c6
Successfully tagged gayatrikolekar/studentserver:latest
kolekar19615@cloudshell:~ (cs-571-demo2-project)$
```

4. Push the docker image

docker push yourdockerhubID/studentserver



Step3 Create a python Flask bookshelf REST API and deploy on GKE

1. Create bookshelf.py

```
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/", 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/", 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 Dockerfile

```
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

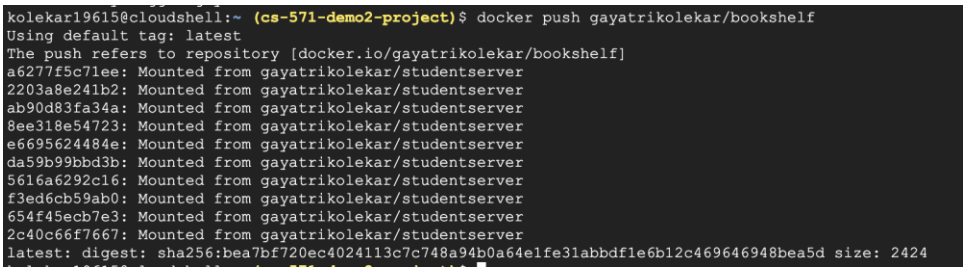
docker build -t zhou19539/bookshelf .

Make sure this step build successfully

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ docker build -t gayatrikolekar/bookshelf .
Sending build context to Docker daemon  217.6MB
Step 1/4 : FROM node:7
 ---> d9aed20b68a4
Step 2/4 : ADD studentServer.js /studentServer.js
 ---> Using cache
 ---> 2efb89e0806a
Step 3/4 : ENTRYPOINT ["node","studentServer.js"]
 ---> Using cache
 ---> 139d581f2979
Step 4/4 : RUN npm install mongodb
 ---> Using cache
 ---> 96417c5af9c6
Successfully built 96417c5af9c6
Successfully tagged gayatrikolekar/bookshelf:latest
```

4. Push the docker image to your dockerhub

docker push yourdockerhubID/bookshelf



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: Change-this-to-your-mongoDB-EXTERNAL-IP

MONGO_DATABASE: mydb

2. Create a file named bookshelf-configmap.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: bookshelf-config

data:

SERVICE_NAME.NAMESPACE.svc.cluster.local:SERVICE_PORT

MONGO_URL: Change-this-to-your-mongoDB-EXTERNAL-IP

MONGO_DATABASE: mydb

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

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
labels:
app: studentserver-deploy
spec:
replicas: 1
selector: matchLabels:
app: web
template:
metadata:
labels:
app: web
spec:
containers:
- image: zhou19539/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: zhou19539/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 sutdentserver-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

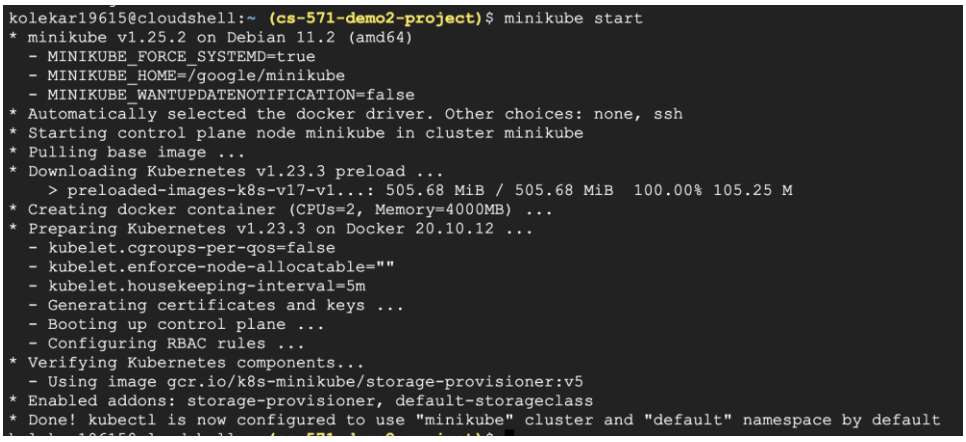
targetPort: 5000

selector:

app: bookshelf-deployment

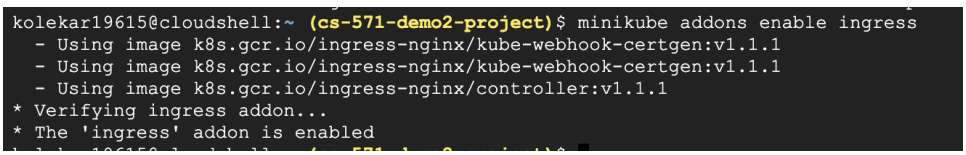
5. Start minikube

minikube start



6. Start Ingress

minikube addons enable ingress



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

kubectl apply -f studentserver-deployment.yaml

kubectl apply -f studentserver-configmap.yaml

kubectl apply -f studentserver-service.yaml

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl apply -f studentserver-deployment.yaml
deployment.apps/web created
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl apply -f studentserver-configmap.yaml
configmap/studentserver-config created
```

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl apply -f studentserver-service.yaml
service/web created
```

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

kubectl apply -f bookshelf-deployment.yaml

kubectl apply -f bookshelf-configmap.yaml

kubectl apply -f bookshelf-service.yaml

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl apply -f bookshelf-deployment.yaml
deployment.apps/bookshelf-deployment created
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl apply -f bookshelf-configmap.yaml
configmap/bookshelf-config created
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl apply -f bookshelf-service.yaml
service/bookshelf-service created
```

9. Check if all the pods are running correctly

kubectl get pods

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS      AGE
bookshelf-deployment-7bb7d899-xj6jj   0/1     Error     3 (32s ago)   75s
web-56686484b9-w5mkv                  0/1     Error     4 (63s ago)   2m5s
```

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

- path: /bookshelf(/|$)(.*)

pathType: Prefix

backend:

service:

name: bookshelf-service

port:

number: 5000

11. Create the ingress service using the above yaml file

kubectl apply -f studentservermongoIngress.yaml

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl apply -f studentservermongoIngress.yaml
ingress.networking.k8s.io/server created
```

12. Check if ingress is running

kubectl get ingress

Please wait until you see the Address, then move forward

```
kolekar19615@cloudshell:~ (cs-571-demo2-project)$ kubectl get ingress
NAME     CLASS   HOSTS              ADDRESS        PORTS   AGE
server   nginx   cs571.project.com  192.168.49.2   80      16s
```

13. Add Addreee to /etc/hosts

vi /etc/hosts

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

Your-address cs571.project.com

Your /etc/hosts file should look something like this after adding the line, but your address should be different from mine

```
# Kubernetes-managed hosts file.
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
172.17.0.4      cs-773365444975-default
192.168.49.2 cs571.project.com
```

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

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

On another path, you should be able to use the REST API with bookshelf application I.e list all books

curl cs571.project.com/bookshelf/books

Add a book

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

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/id