



kubernetes

Kubernetes Signature Project

Python Flask framework + MongoDB + REST API + Kubernetes + GKE
Node.js server + MongoDB + Kubernetes + GKE

By
Maryam Taherzadeh
Prepared Under the direction of Professor Henry Chang

School of Engineering
Northwestern Polytechnic University
117 Fourier Avenue, Fremont, CA 94539

April 2021

Table Of Content

Content	Page number
○ Abstraction	4
○ Introduction	5
○ Design	6-8
- System Requirement	
- Kubernetes Concept	
○ Implementation	9-33
○ Test Result	34-38
○ Future Enhancement	39
○ Conclusion	40
○ References	41

Table of Figures

Figure 1 : Key Kubernetes Concept 9

Abstract

This project applies the **following techniques** and runs on Google Kubernetes Engine.

One of the techniques in this project is **Pods** which are used to run two applications and the other techniques is **service** which is useful for outside access the application. **Persistence Volumes** are also techniques are applied to store the data with MongoDB. **Ingress** is expose both applications under same domain but different path. ConfigMaps play role in storing MongoDB service address restarts with a different service address, and with ConfigMaps, we don't need to build the docker image again with the new address.

Introduction

In this project ingress is used on Kubernetes for hosting applications. Infact, two application applied MongoDB which is hosted on Google Kubernetes Engine (GKE) for storing data. Two application under the same domain (but in different path) will be hosted and also used MongoDB hosted on GKE for storing data.

Design

System Requirement

In order to run the Project on Google cloud Platform, and to make use of it, you need to follow the below steps which is important to implement it correctly in your project.

- a. On your GCP control panel, type “GKE” and enable Google Kubernetes Engine .On top right corner of the webpage, click to open the gcp terminal.
- b. Create a cluster as usual on GKE gcloud container clusters
- c. Create two pods: pod one for creating student records & pod 2 for creating bookstore
- d. Create service is infact a resource to make single, constant point of entry to a group of pods to expose your application / service for access from outside the cluster
- e. Persistence Volumes for storing the data with MongoDB
- f. Ingress for exposing both applications under same domain but different path
- g. ConfigMaps for storing MongoDB service address, in case MongoDB is down and restarts with a different service address, and with ConfigMaps, we don't need to build the docker image again with the new address

Design

Kubernetes Concepts:

To better understand the project let's to understand some concepts used in this project

-**NODE:** A Node is a worker machine provisioned to run Kubernetes. Each Node is managed by the Kubernetes master.

-**POD:** is the basic building block of Kubernetes tightly-coupled group of application containers that run on a Node. Containers in a Pod are deployed together and share resources (like data volumes and network addresses).

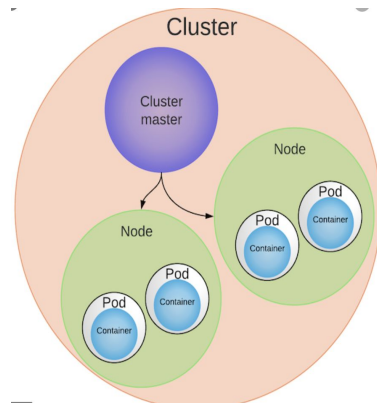
Multiple Pods can run on a single Node.

Concept

-**SERVICE:** is a logical set of Pods that perform a similar function. It can load balancing and service discovery.

-**INGRESS:** is a set of routing rules applied to control the external access to Services based on the request host or path.

-**CONFIGMAP:** is an API object applied for storing non-confidential data based on key-value pairs. Pods consume ConfigMaps as environment variables as configuration files in a volume or command-line arguments



(Fig 1) kubernetes concept

Design

Cs571.project.com is a domain name

cs571.project.com/studentserver (a Node.js server that returns a student's score with related student ID)

cs571.project.com/bookshelf (a bookshelf application with REST API implemented using Python Flask)

Implementation

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
```

NAME	LOCATION	MASTER VERSION	MASTER IP	MACHINE TYPE	NODE VERSION	NUM_NODES	STATUS
kubia	us-west1	1.18.16-gke.302	34.105.38.17	e2-micro	1.18.16-gke.302	3	RUNNING

2. create a mongodb deployment with this yaml filec

```
Kubecttl apply -f mongodb-deployment.yaml
```

```
raherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$ kubectl apply -f mongodb-deployment.yaml
deployment.apps/mongodb-deployment created
raherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$ cat 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:
        # 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
```

Implementation

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

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

`kubectl get pods`

```
taherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$ kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
mongodb-deployment-554cbb9965-qtip2 0/1     ContainerCreating   0           4m34s
```

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

```
taherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$ vim mongodb-service.yaml
taherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$ cat mongodb-service.yaml
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
```

Implementation

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

6- In this step you should wait and check whether the service is up or not

```
taherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$ kubectl apply -f mongodb-service.yaml
service/mongodb-service created
taherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mongodb-deployment-554cbb9965-swn9w 1/1     Running   0           102s
taherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.3.240.1	<none>	443/TCP	9m34s
mongodb-service	LoadBalancer	10.3.255.95	34.82.148.16	27017:31465/TCP	4m14s

Implementation

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

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

```
kubectil exec -it mongodb-deployment-replace-with-your-pod-name -- bash
```

```
taherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$ kubectl exec -it mongodb-deployment-554cbb9965-swn9w -- bash
root@mongodb-deployment-554cbb9965-swn9w:/# mongo 34.82.148.16
MongoDB shell version v4.4.4
connecting to: mongodb://34.82.148.16:27017/test?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("846727fd-b939-47a2-940e-753b5b3c24f4") }
MongoDB server version: 4.4.4
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
  https://community.mongodb.com
---
The server generated these startup warnings when booting:
2021-04-06T20:13:28.004+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2021-04-06T20:13:29.585+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()
---
> |
```

Implementation

Step 1 Create MongoDB using Persistent Volume on GKE, and insert records into it

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

```
---  
> exit  
bye  
root@mongodb-deployment-554cbb9965-swn9w:/# exit  
exit  
taherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$
```

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

```
taherzadeh19529@cloudshell:~/mongodb/yaml (phonic-axle-307118)$ cd  
taherzadeh19529@cloudshell:~ (phonic-axle-307118)$ cd mongodb/  
taherzadeh19529@cloudshell:~/mongodb (phonic-axle-307118)$ node  
Welcome to Node.js v12.14.1.  
Type ".help" for more information.  
>
```

Implementation

Step 1 Create MongoDB using Persistent Volume on GKE, and insert records into it

10-Type the following

Enter the following line by line(pay attention to indentation)

```
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

Implementation

Step 1 Create MongoDB using Persistent Volume on GKE, and insert records into it

11-If the in the previous slide everything is correct, you should see this output:

3 means we inserted three records . in here as you can see we search for student id:11111

```
> 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);
...     });
... });
undefined
> 3
{
  _id: 606cca839329c7042b79484e,
  student_id: 11111,
  student_name: 'Bruce Lee',
  grade: 84
}
> |
```

Implementation

Step 2 :Modify our studentServer to get records from MongoDB and deploy to GKE

1.Create a studentServer

```
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
The and
/api/score?student_id=1111
JSON response should contain only 'student_id', 'student_name' '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=1111
  var parsedUri = url.parse(req.url, true);

  var student_id = parseInt(parsedUri.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, { useNewUriParser: 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);
```


Implementation

Step 2 :Modify our studentServer to get records from MongoDB and deploy to GKE

2-Create a Dockerfile



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

3-Build the student server docker image

```
docker build -t yourdockerhubID/studentserver .
```

```
Successfully built d0f1bc1ca093
Successfully tagged tahezadeh19529/studentserver:latest
taherzadeh19529@cloudshell:~/mongodb (phonic-axle-307118) $
```

Implementation

Step 2 :Modify our studentServer to get records from MongoDB and deploy to GKE

4. Push the docker image

```
docker push yourdockerhubID/studentserver
```

```
taherzadeh19529@cloudshell:~/mongodb (phonic-axle-307118)$ docker push taherzadeh19529/studentserver
Using default tag: latest
The push refers to repository [docker.io/taherzadeh19529/studentserver]
a6f26b0f2fdc: Pushed
999b173318b7: 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: sha256:9e45ac50094d6a3700223fe90b7cfe090936250bb51ae3711a00046442d3dc4f size: 2424
taherzadeh19529@cloudshell:~/mongodb (phonic-axle-307118)$
```

Implementation

Step 3 Create a python Flask bookshelf REST API and deploy on GKE

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/<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)
```

Implementation

Step 3 Create a python Flask bookshelf REST API and deploy on GKE

2.Create a Dockerfile

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ vim bookshelf.py
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ vim Dockerfile
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ cat 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" ]
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$
```

Implementation

Step 3 Create a python Flask bookshelf REST API and deploy on GKE

3- Build the bookshelf app into a docker image

`docker build -t tahezadeh19529/bookshelf`

```
tahezadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ docker build -t tahezadeh19529/bookshelf .
Sending build context to Docker daemon 13.17MB
Step 1/8 : FROM python:alpine3.7
--> 00be2573e9f7
Step 2/8 : COPY . /app
--> c70329a5b548
Step 3/8 : WORKDIR /app
--> Running in 45901f1c815e
Removing intermediate container 45901f1c815e
--> b59d8be293e6
Step 4/8 : RUN pip install -r requirements.txt
--> Running in ce3eae5ce50a
Collecting Flask (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/f2/28/2a03252dfb9ebf377f40fba6a7841b47083260bf8bd8e737b0c6952df83f/Flask-1.1.2-py2.py3-none-any.whl (94kB)
Collecting Flask-PyMongo (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/67/b8/0322016b9ce09a64fba9018211e7c35fd51380527ffd9ea248744f389239/Flask_PyMongo-2.3.0-py2.py3-none-any.whl
Collecting click>=5.1 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/d2/3d/fa76db83bf75c4f8d338c2fd15c8d33fdd7ad23a9b5e57eb6c5de26b430e/click-7.1.2-py2.py3-none-any.whl (82kB)
Collecting itsdangerous>=0.24 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2.py3-none-any.whl
Collecting Jinja2>=2.10.1 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7e/c2/1eece8c95ddbc9b1aeb64f5783a9e07a286de42191b7204d67b7496ddf35/Jinja2-2.11.3-py2.py3-none-any.whl (125kB)
Collecting Werkzeug>=0.15 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/cc/94/5f7079a0e00bd6863ef8f1da638721e9da21e5bacee597595b318f71d62e/Werkzeug-1.0.1-py2.py3-none-any.whl (298kB)
Collecting PyMongo>=3.3 (from Flask-PyMongo->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/72/82/e7196f2f69318dd206db26db68fcfa0ff821d88fbca6d0f0c7b678ba0353/pymongo-3.11.3.tar.gz (777kB)
Collecting MarkupSafe>=0.23 (from Jinja2->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/b9/2e/64db92e53b86efccfaea71321f597fa2e1b2bd3853d8ce658568f7a13094/MarkupSafe-1.1.1.tar.gz
Building wheels for collected packages: PyMongo, MarkupSafe
  Building wheel for PyMongo (setup.py): started
  Building wheel for PyMongo (setup.py): finished with status 'done'
  Building wheel for MarkupSafe (setup.py): finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/97/64/bb/be01adf5254f3e63c246204e2df51543af23e24e5531f8cf2a
  Building wheel for MarkupSafe (setup.py): started
```

Implementation

Step 3 Create a python Flask bookshelf REST API and deploy on GKE

4-Push the docker image to your docker hub

Docker push tahezadeh19529/bookshelf

```
tahezadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ docker push tahezadeh19529/bookshelf
Using default tag: latest
The push refers to repository [docker.io/tahezadeh19529/bookshelf]
b7c879419fe5: Pushed
0aea60415351: Pushed
5fa31f02caa8: Mounted from library/python
88e61e328a3c: Mounted from library/python
9b77965e1d3f: Mounted from library/python
50f8b07e9421: Mounted from library/python
629164d914fc: Mounted from library/python
latest: digest: sha256:e4ef4e52545f0db9c90e9a87618afde2700c8aae584562823f044ef9c42bf855 size: 1790
tahezadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$
```

Implementation

Step 4 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
```

Implementation

Step 5 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

1- Create studentserver-deployment.yaml

```
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: tahezadeh19529/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
```


Implementation

Step 5 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

2. Create bookshelf-deployment.yaml

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: tahezadeh19529/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
```

Project setup & Implementation

Step 5 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

3. Create sutdentserver-service.yaml

```
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

```
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
```

Implementation

Step 5 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

5. Start minikube

minikube start

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axe-307118)$ minikube start
* minikube v1.18.1 on Debian 10.9 (amd64)
- MINIKUBE_FORCE_SYSTEMD=true
- MINIKUBE_HOME=/google/minikube
- MINIKUBE_WANTUPDATENOTIFICATION=false
* Automatically selected the docker driver. Other choices: none, ssh
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Downloading Kubernetes v1.20.2 preload ...
  > preloaded-images-k8s-v9-v1...: 491.22 MiB / 491.22 MiB 100.00% 145.93 M
* Creating docker container (CPUs=2, Memory=4000MB) ...
* Preparing Kubernetes v1.20.2 on Docker 20.10.3 ...
- Generating certificates and keys ...
- Booting up control plane ...
- Configuring RBAC rules ...
* Verifying Kubernetes components...
- Using image gcr.io/k8s-minikube/storage-provisioner:v4
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Implementation

Step 5 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

5. Start minikube

minikube start

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axe-307118)$ minikube start
* minikube v1.18.1 on Debian 10.9 (amd64)
  - MINIKUBE_FORCE_SYSTEMD=true
  - MINIKUBE_HOME=/google/minikube
  - MINIKUBE_WANTUPDATENOTIFICATION=false
* Automatically selected the docker driver. Other choices: none, ssh
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Downloading Kubernetes v1.20.2 preload ...
  > preloaded-images-k8s-v9-v1...: 491.22 MiB / 491.22 MiB 100.00% 145.93 M
* Creating docker container (CPUs=2, Memory=4000MB) ...
* Preparing Kubernetes v1.20.2 on Docker 20.10.3 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v4
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Implementation

Step 5 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

5. Start minikube

minikube start

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axe-307118)$ minikube start
* minikube v1.18.1 on Debian 10.9 (amd64)
- MINIKUBE_FORCE_SYSTEMD=true
- MINIKUBE_HOME=/google/minikube
- MINIKUBE_WANTUPDATENOTIFICATION=false
* Automatically selected the docker driver. Other choices: none, ssh
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Downloading Kubernetes v1.20.2 preload ...
  > preloaded-images-k8s-v9-v1...: 491.22 MiB / 491.22 MiB 100.00% 145.93 M
* Creating docker container (CPUs=2, Memory=4000MB) ...
* Preparing Kubernetes v1.20.2 on Docker 20.10.3 ...
- Generating certificates and keys ...
- Booting up control plane ...
- Configuring RBAC rules ...
* Verifying Kubernetes components...
- Using image gcr.io/k8s-minikube/storage-provisioner:v4
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Implementation

Step 5 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

6.Ingress

minikube addons enable ingress

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ minikube addons enable ingress
- Using image us.gcr.io/k8s-artifacts-prod/ingress-nginx/controller:v0.40.2
- Using image jettech/kube-webhook-certgen:v1.2.2
- Using image jettech/kube-webhook-certgen:v1.3.0
* Verifying ingress addon...
* The 'ingress' addon is enabled
```

7. Create student server and bookshelf related pods and start service using the above yaml file

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ kubectl apply -f studentserver-deployment.yaml
deployment.apps/web created
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ kubectl apply -f bookshelf-deployment.yaml
deployment.apps/bookshelf-deployment created
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ kubectl apply -f studentserver-service.yaml
service/web created
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ kubectl apply -f bookshelf-service.yaml
service/bookshelf-service created
```

bash: kubectl: command not found

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ kubectl apply -f bookshelf-configmap.yaml
configmap/bookshelf-config created
```

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ kubectl apply -f studentserver-configmap.yaml
configmap/studentserver-config created
```

Implementation

Step 5 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

8. Check if all the pods are running correctly

kubectl get pods

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
bookshelf-deployment-646c59bd88-v7sd5  1/1     Running   0           26s
web-6554866b97-88x8v                 1/1     Running   0           35s
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ kubectl get svc
```

9. 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
```

Implementation

Step 5 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

10. Create the ingress service using the above yaml file

```
kubectl apply -f studentservermongoIngress.yaml
```

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ kubectl create -f studentservermongoIngress.yaml
ingress.networking.k8s.io/server created
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$
```

11. Check if ingress is running

```
kubectl get ingress
```

Please wait until you see the Address, then move forward

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ kubectl get ingress
```

NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
server	<none>	cs571.project.com	192.168.49.2	80	140m

Implementation

Step 5 Create ConfigMap for both applications to store MongoDB URL and MongoDB name

12. Add Address 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

```
# 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-864692867660-default-boost-79btk
192.168.49.2 cs571.project.com
~
~
~
~
~
~
~
```

Test Result

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

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ curl http://cs571.project.com/studentserver/api/score?student_id=22222
{"_id":"606f926dfbaf28003fa09842","student_id":22222,"student_name":"Jackie Chen","grade":93}
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ curl http://cs571.project.com/studentserver/api/score?student_id=33333
{"_id":"606f926dfbaf28003fa09843","student_id":33333,"student_name":"Jet Li","grade":88}
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ curl http://cs571.project.com/studentserver/api/score?student_id=11111
{"_id":"606f926dfbaf28003fa09841","student_id":11111,"student_name":"Bruce Lee","grade":84}
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$
```

Test Result

list all books.

```
curl cs571.project.com/bookshelf/books
```

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ curl cs571.project.com/bookshelf/books
[
  {
    "Book Author": "unkown",
    "Book Name": "cloud computing",
    "ISBN": "123456",
    "id": "606f9c26640cd7763066f15a"
  }
]
```

Test Result

Add a book

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

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axe-307118)$ curl -X POST -d '{"book_name": "cloud computing","book_author":
> "unkown", "isbn": "123456"}' http://cs571.project.com/bookshelf/book
{
  "message": "Task saved successfully!"
}
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axe-307118)$ curl http://cs571.project.com/bookshelf/books
[
  {
    "Book Author": "unkown",
    "Book Name": "cloud computing",
    "ISBN": "123456",
    "id": "606f94b6640cd7763066f159"
  },
  {

```

Test Result

Update a book

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

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axe-307118)$ curl -X PUT -d '{"book_name": "123","book_author": "test", "isbn": "123updated"}' http://cs571.project.com/bookshelf/book/606f94b6640cd7763066f159
{"message": "Task updated successfully!"}
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axe-307118)$ curl http://cs571.project.com/bookshelf/books
[{"Book Author": "test", "Book Name": "123", "ISBN": "123updated", "id": "606f94b6640cd7763066f159"}, {"Book Author": "unkown", "Book Name": "cloud computing", "ISBN": "123456", "id": "606f9c26640cd7763066f15a"}]
```

Test Result

Delete a book

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

```
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ curl -X DELETE cs571.project.com/bookshelf/book/606f94b6640cd7763066f159
{
  "message": "Task deleted successfully!"
}
taherzadeh19529@cloudshell:~/mongodb/bookshelf (phonic-axle-307118)$ curl http://cs571.project.com/bookshelf/books
[
  {
    "Book Author": "unkown",
    "ISBN": "123456",
    "id": "606f9c26640cd7763066f15a"
  }
]
```

Future Enhancement

- Inserting TLS to Ingress for making both application under HTTPS
- Processing a real domain name and host both of the applications for public Test

Conclusion

This project applied techniques such as pod, persistence volume, Ingress, Service & ConfigMap. All test have done on Google Kubernetes Engine. Under the test Domain name (cs571.project.com) several applications can be hosted .

- Several applications can be hosted
 - /studentserver
 - a node.js server application
 - /bookshelf
 - a python Bookshelf REST API applications

Reference

https://npu85.npu.edu/~henry/npu/classes/capstone/rest/slide/index_slide.html

https://npu85.npu.edu/~henry/npu/classes/cloud_computing/mongodb/slide/index_slide.html

<https://v1-19.docs.kubernetes.io/docs/tasks/access-application-cluster/ingress-minikube/>

<https://github.com/Quan25>