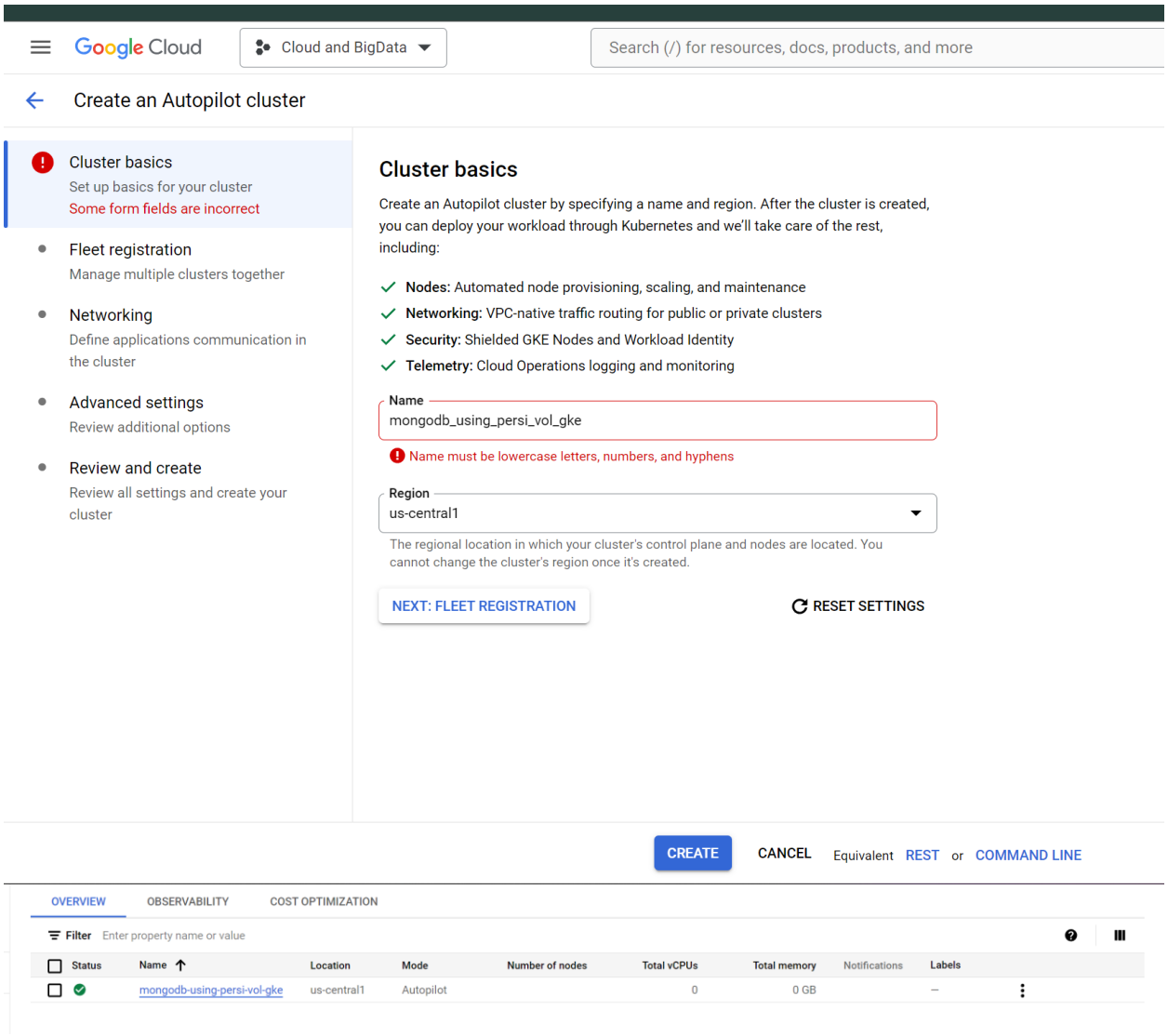
Step 1: Create MongoDB Using Persistent Volume on GKE and Insert Records

1. Create a Cluster on GKE:



- Create a Kubernetes cluster named `kubia` using Google Kubernetes Engine (GKE) with 0 node.
- This command sets up a Kubernetes cluster that will host the MongoDB deployment.

2. Create a Persistent Volume:

```
gcloud compute disks create mongodb --size=10GiB --zone=us-central1-a
```

- Create a persistent disk of size 10GiB named `mongodb` in the `us-central1` zone.
    - Persistent disks are necessary for MongoDB to store data reliably, ensuring data is retained even if the pod restarts.

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ gcloud compute disks create mongodb --size=10GB --zon
e=us-central1-f
WARNING: You have selected a disk size of under [200GB]. This may result in poor I/O performance. F
or more information, see: https://developers.google.com/compute/docs/disks#performance.
Created [https://www.googleapis.com/compute/v1/projects/cloud-and-bigdata/zones/us-central1-f/disks
/mongodb].
NAME: mongodb
ZONE: us-central1-f
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
```

3. **Create MongoDB Deployment:**

`kubectl apply -f mongodb-deployment.yaml`

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ vi mongodb-deployment.yaml
```

Error: In case you get an error like below follow the steps to fix that:

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl apply -f mongodb-deployment.yaml
Unable to connect to the server: dial tcp 34.29.198.63:443: i/o timeout
```

We have to configure the cluster to the one we just created.

`gcloud container clusters get-credentials mongodb-using-persi-vol-gke --region us-central1`
`Fetching cluster endpoint and auth data.`
`kubeconfig entry generated for mongodb-using-persi-vol-gke.`

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ gcloud config get-value project
Your active configuration is: [cloudshell-15631]
cloud-and-bigdata
adagniew407@cloudshell:~ (cloud-and-bigdata)$ gcloud container clusters list
NAME: mongodb-using-persi-vol-gke
LOCATION: us-central1
MASTER_VERSION: 1.29.6-gke.1038001
MASTER_IP: 34.42.195.227
MACHINE_TYPE: e2-small
NODE_VERSION: 1.29.6-gke.1038001
NUM_NODES:
STATUS: RUNNING
adagniew407@cloudshell:~ (cloud-and-bigdata)$ gcloud container clusters get-credentials mongodb-usi
ng-persi-vol-gke --region us-central1
Fetching cluster endpoint and auth data.
kubeconfig entry generated for mongodb-using-persi-vol-gke.
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl config current-context
gke_cloud-and-bigdata_us-central1_mongodb-using-persi-vol-gke
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl get nodes
No resources found
```

(cloud-and-bigdata) ×

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  gcePersistentDisk:
    pdName: mongodb
    fsType: ext4
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
spec:
  selector:
    matchLabels:
      app: mongodb
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
```

```yaml
  name: mongodb-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
spec:
  selector:
    matchLabels:
      app: mongodb
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - image: mongo
        name: mongo
        ports:
        - containerPort: 27017
        volumeMounts:
        - name: mongodb-data
          mountPath: /data/db
      volumes:
      - name: mongodb-data
        persistentVolumeClaim:
          claimName: mongodb-pvc
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  gcePersistentDisk:
    pdName: mongodb
    fsType: ext4
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
spec:
  selector:
    matchLabels:
      app: mongodb
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - image: mongo
        name: mongo
        ports:
        - containerPort: 27017
        volumeMounts:
        - name: mongodb-data
          mountPath: /data/db
      volumes:
      - name: mongodb-data
        persistentVolumeClaim:
          claimName: mongodb-pvc
```

- Deploy MongoDB to the Kubernetes cluster using the configuration specified in the `mongodb-deployment.yaml` file.
- This YAML file contains the configurations needed to deploy MongoDB, including volume mounts, container specifications, and deployment strategy.

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl apply -f mongodb-deployment.yaml
persistentvolume/mongodb-pv created
persistentvolumeclaim/mongodb-pvc unchanged
deployment.apps/mongodb-deployment unchanged
```

4. **Check Deployment Pod:**

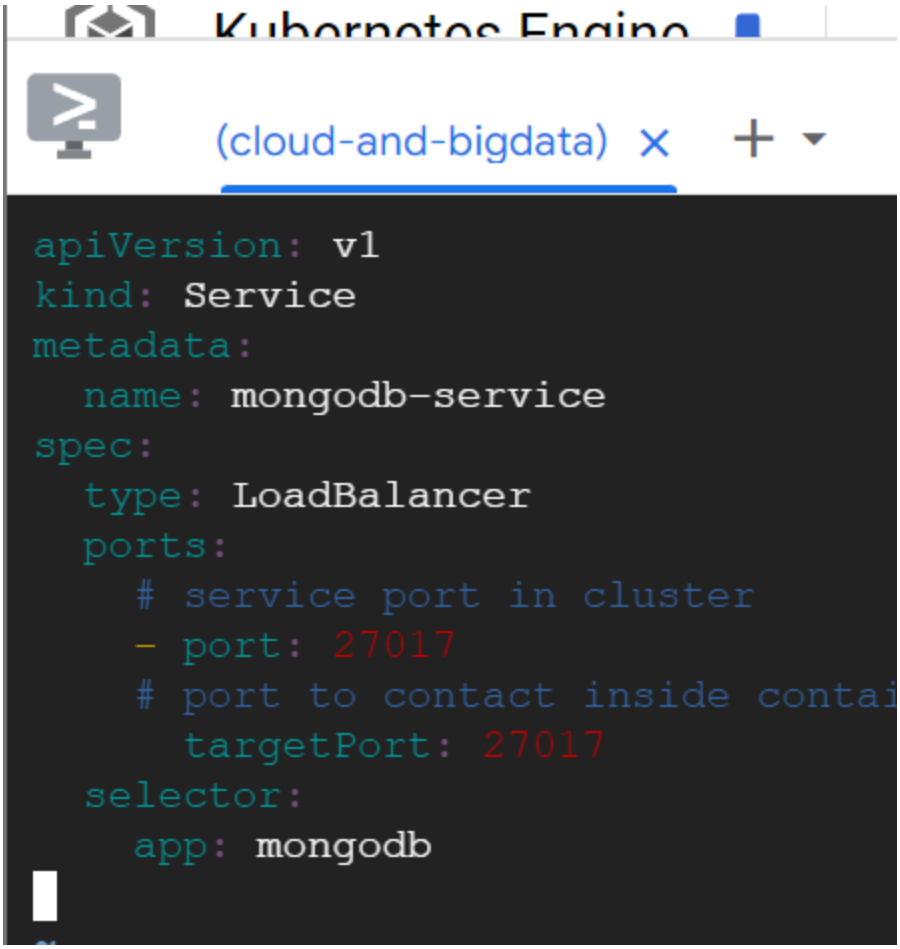
```
kubectl get pods
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
mongodb-deployment-694c495656-28wjl   1/1     Running   0          7m10s
```

- List all pods in the current namespace to check if the MongoDB pod is running.
- Verifying the pod's status ensures that MongoDB is correctly deployed and running.
- Please wait until you see the STATUS is running, then you can move forward

5. **Create MongoDB Service:**

```
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
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl apply -f mongodb-service.yaml
service/mongodb-service created
```

- Create a service for MongoDB to allow external access to the MongoDB instance using the configuration in `mongodb-service.yaml`.
- This service exposes the MongoDB deployment, making it accessible via an external IP address.

6. **Check Service Status:**

```
kubectl get svc
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl get svc
NAME              TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)           AGE
kubernetes        ClusterIP      34.118.224.1     <none>         443/TCP           10m
mongodb-service   LoadBalancer   34.118.228.218   34.71.187.5    27017:30319/TCP   72s
```

- List all services in the current namespace to check the status of the MongoDB service.
- Confirming the service status ensures that the external IP is assigned and the service is accessible.

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

7. **Test MongoDB Connection:**

```
kubectl exec -it mongodb-deployment-694c495656-28wjl - bash
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl exec -it mongodb-deployment-694c495656-28wjl -- bash
root@mongodb-deployment-694c495656-28wjl:/#
```

- Execute an interactive bash shell on the MongoDB pod to test the connection using the external IP.
- This step allows you to enter the MongoDB pod and manually verify that MongoDB is running and accessible.

Connecting to MongoDB:

- Use the `mongosh` command to connect to the MongoDB instance. The correct command should be:

```
mongosh --host 34.71.187.5 --port 27017
```

```
root@mongodb-deployment-694c495656-28wjl:/# mongosh --host 34.71.187.5 --port 27017
Current Mongosh Log ID: 66a7b71cc4ad305e34149f47
Connecting to:          mongodb://34.71.187.5:27017/?directConnection=true&appName=mongosh+2.2.10
Using MongoDB:          7.0.12
Using Mongosh:          2.2.10
mongosh 2.2.12 is available for download: https://www.mongodb.com/try/download/shell

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


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2024-07-29T15:28:42.548+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
   2024-07-29T15:28:44.126+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2024-07-29T15:28:44.127+00:00: vm.max_map_count is too low
------

test>
```

```
root@mongodb-deployment-694c495656-78x2m:/# mongod --version
db version v7.0.12
Build Info: {
    "version": "7.0.12",
    "gitVersion": "b6513ce0781db6818e24619e8a461eae90bc94fc",
    "openSSLVersion": "OpenSSL 3.0.2 15 Mar 2022",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "ubuntu2204",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}
```

8. **Exit MongoDB Pod:**

```
exit
```

```
}
root@mongodb-deployment-694c495656-78x2m:/# exit
exit

test> exit
root@mongodb-deployment-694c495656-78x2m:/# mongod --version
```

   - Exit the MongoDB pod and return to the main console.
   - After testing the connection, exiting the pod returns you to the original shell.

9. **Insert Records into MongoDB:**

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

async function run() {
  const url = "mongodb://34.71.187.5/studentdb";
  const client = new MongoClient(url);

  try {
    // Connect to the MongoDB cluster
    await client.connect();

    // Specify the database and collection
    const db = client.db("studentdb");
    const collection = db.collection("students");

    // Create documents to be inserted
    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 the documents
    const insertResult = await collection.insertMany(docs);
    console.log(`${insertResult.insertedCount} documents were
inserted`);

    // Find one document
    const result = await collection.findOne({ student_id: 11111 });
    console.log(result);
  } finally {
    // Close the connection
    await client.close();
  }
}

run().catch(console.dir);
```

   - Use Node.js to connect to MongoDB and insert sample student records into the database.

- This script connects to MongoDB using the provided external IP, inserts sample data into the `students` collection, and retrieves one of the inserted records to verify the insertion.

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ npm install mongodb

added 12 packages in 2s
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ node
Welcome to Node.js v20.15.1.
Type ".help" for more information.
> const { MongoClient } = require('mongodb');
undefined
>
> const url = "mongodb://34.71.187.5/studentdb"; // Your MongoDB server IP and database name
undefined
> const client = new MongoClient(url, { useNewUrlParser: true, useUnifiedTopology: true });
undefined
>
> async function run() {
...   try {
...     // Connect to the MongoDB cluster
...     await client.connect();
...
...     // Specify the database and collection
...     const db = client.db("studentdb");
...     const collection = db.collection("students");
...
...     // Create documents to be inserted
...     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 the documents
...     const insertResult = await collection.insertMany(docs);
...     console.log(`${insertResult.insertedCount} documents were inserted`);
...
...     // Find one document
...     const result = await collection.findOne({ student_id: 11111 });
...     console.log(result);
...
...   } finally {
...     // Close the connection
...     await client.close();
...   }
... }
```

```
>
> run().catch(console.dir);
Promise {
  <pending>,
  [Symbol(async_id_symbol)]: 105,
  [Symbol(trigger_async_id_symbol)]: 73
}
3 documents were inserted
{
  _id: new ObjectId('66a7b86ff6b133130c0aff9f'),
  student_id: 11111,
  student_name: 'Bruce Lee',
  grade: 84
}
>
```

Step 2: Modify Student Server to Get Records from MongoDB and Deploy to GKE

1. **Create `studentServer.js`:**

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

// Connection URI
const uri = `mongodb://${MONGO_URL}/${MONGO_DATABASE}`;
console.log(`MongoDB URI: ${uri}`);

// Create a server
const server = http.createServer(async (req, res) => {
  try {
    // Parse the URL and query string
    const parsedUrl = url.parse(req.url, true);
    const student_id = parseInt(parsedUrl.query.student_id);

    // Match req.url with the string /api/score
    if (/^\/api\/score/.test(req.url) && student_id) {
      // Connect to the database
      const client = new MongoClient(uri);
      await client.connect();
      const db = client.db(MONGO_DATABASE);

      try {
        // Find the student document
        const student = await db.collection("students").findOne({
"student_id": student_id });

        if (student) {
          // Prepare the response object
          const response = {
            student_id: student.student_id,
            student_name: student.student_name,
            student_score: student.grade
          };

          // Send the response
          res.writeHead(200, { 'Content-Type': 'application/json' });
          res.end(JSON.stringify(response));
        } else {
          res.writeHead(404, { 'Content-Type': 'text/plain' });
          res.end("Student Not Found\n");
        }
      } finally {
        // Ensure the client is closed
        await client.close();
      }
    } else {
      res.writeHead(404, { 'Content-Type': 'text/plain' });
      res.end("Wrong URL, please try again\n");
    }
  } catch (err) {
    console.error(err);
    res.writeHead(500, { 'Content-Type': 'text/plain' });
```

```
    res.end("Internal Server Error\n");
  }
});

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


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

// Connection URI
const uri = `mongodb://${MONGO_URL}/${MONGO_DATABASE}`;
console.log(uri);

// Create a server
const server = http.createServer(async (req, res) => {
  try {
    // Parse the URL and query string
    const parsedUrl = url.parse(req.url, true);
    const student_id = parseInt(parsedUrl.query.student_id);

    // Match req.url with the string /api/score
    if (/^\/api\/score/.test(req.url)) {
      // Connect to the database
      const client = new MongoClient(uri);
      await client.connect();
      const db = client.db("studentdb");

      // Find the student document
      const student = await db.collection("students").findOne({ "student_id": student_id });
      await client.close();
```

```
      // Connect to the database
      const client = new MongoClient(uri);
      await client.connect();
      const db = client.db("studentdb");

      // Find the student document
      const student = await db.collection("students").findOne({ "student_id": student_id });
      await client.close();

      if (student) {
        // Prepare the response object
        const response = {
          student_id: student.student_id,
          student_name: student.student_name,
          student_score: student.grade
        };

        // Send the response
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify(response) + '\n');
      } else {
        res.writeHead(404);
        res.end("Student Not Found\n");
      }
    } else {
      res.writeHead(404);
      res.end("Wrong URL, please try again\n");
    }
  } catch (err) {
    console.error(err);
    res.writeHead(500);
    res.end("Internal Server Error\n");
  }
});

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

- **Description:** Create a Node.js server to retrieve student records from MongoDB and respond with JSON.
- This server listens for HTTP requests on port 8080, connects to MongoDB, retrieves student records based on the student ID provided in the request, and returns the results as JSON.

2. **Create Dockerfile:**

```
# Use a smaller base image
FROM node:alpine

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install mongodb

# Copy the rest of the application code
COPY . .

# Expose the port your app runs on
```

```
EXPOSE 3000

# Use CMD to run your application
CMD ["node", "studentServer.js"]
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ cat Dockerfile
# Use a smaller base image
FROM node:alpine

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install mongodb

# Copy the rest of the application code
COPY . .

# Expose the port your app runs on
EXPOSE 3000

# Use CMD to run your application
CMD ["node", "studentServer.js"]
```

- **Description:** Define the Dockerfile to containerize the student server application.
- **Explanation:** This Dockerfile sets up a Node.js environment, adds the server script, installs the necessary dependencies, and specifies the entry point for the container.

3. **Build Docker Image:**

```
docker build -t student-server .
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ docker build -t student-server .
[+] Building 66.1s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 405B
 => [internal] load metadata for docker.io/library/node:alpine
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/node:alpine@sha256:39005f06b2fae765764d6fdf20ad1c4d0890f5ad3e1f39b56a18768334b8ecd6
 => => resolve docker.io/library/node:alpine@sha256:39005f06b2fae765764d6fdf20ad1c4d0890f5ad3e1f39b56a18768334b8ecd6
 => => sha256:c4f54159f74a5dccd97b9af978fab507483785736e822851638f20f275716fc3 1.39MB / 1.39MB
 => => sha256:39005f06b2fae765764d6fdf20ad1c4d0890f5ad3e1f39b56a18768334b8ecd6 6.62kB / 6.62kB
 => => sha256:c83e6e8aa2c458cf740b18b7b13e546751fe081d36223aac253b5ec0da2cd89d 1.72kB / 1.72kB
 => => sha256:5c4cc5767575c711b99b1b077ad8afb8cfbf407aca174b7cbc998ce6db1e4f93 6.36kB / 6.36kB
 => => sha256:c6a83fedfae6ed8a4f5f7cbb6a7b6f1c1ec3d86fea8cb9e5ba2e5e6673fde9f6 3.62MB / 3.62MB
 => => sha256:8d90f41c769e0bfd90a1e8456db9f590ae8dc42842ffa098693b6ed4bd44eba3 47.36MB / 47.36MB
 => => extracting sha256:c6a83fedfae6ed8a4f5f7cbb6a7b6f1c1ec3d86fea8cb9e5ba2e5e6673fde9f6
 => => sha256:6ecb2bd0d8e8f8628fe4a7cf14404c59d67a5547c6e5faa4a507f4b232bd316e 449B / 449B
 => => extracting sha256:8d90f41c769e0bfd90a1e8456db9f590ae8dc42842ffa098693b6ed4bd44eba3
 => => extracting sha256:c4f54159f74a5dccd97b9af978fab507483785736e822851638f20f275716fc3
 => => extracting sha256:6ecb2bd0d8e8f8628fe4a7cf14404c59d67a5547c6e5faa4a507f4b232bd316e
 => [internal] load build context
 => => transferring context: 1.41GB
 => [2/5] WORKDIR /app
 => [3/5] COPY package*.json ./
 => [4/5] RUN npm install mongodb
 => [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:1b6eabcd81cffa3a6d8b5b076990bf3d554cd96396b05986a3e4395dc2ae7993
 => => naming to docker.io/library/student-server
```

- **Description:** Build the Docker image for the student server.
- **Explanation:** This command creates a Docker image named `studentserver` tagged with your Docker Hub ID, packaging the Node.js server for deployment.

4. **Push Docker Image:**

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ docker tag student-server:latest asdg124/mydbd:latest
```

`docker push asdg124/mydbd:latest`

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ docker push asdg124/mydbd:latest
The push refers to repository [docker.io/asdg124/mydbd]
c74d2426dbf0: Pushed
2e4e4b289fa0: Pushed
3088b4cfb57b: Pushed
94ff3700f4c5: Pushed
a64fa369054d: Mounted from library/node
1a944437090a: Mounted from library/node
4b76468bfe06: Mounted from library/node
78561cef0761: Mounted from library/node
latest: digest: sha256:83fec71dc62438554d13cad2cc5133827da6b116d5f28653b0af04dddc22585f size: 1996
```

- Push the Docker image to Docker Hub.
- Uploading the Docker image to Docker Hub makes it available for deployment on GKE.

Step 3: Create 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 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)
    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)
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ vi bookshelf.py
adagniew407@cloudshell:~ (cloud-and-bigdata)$ vi Dockerfile
adagniew407@cloudshell:~ (cloud-and-bigdata)$ vi requirements.txt
```

(cloud-and-bigdata) × + ▾ Editor

```
from flask import Flask, request, jsonify
from flask_pymongo import PyMongo
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"]
    })
-- INSERT --                                                    1,1           Top
```

- Define a Flask application that provides a REST API for managing a bookshelf.
- This Flask app connects to MongoDB, provides routes for CRUD operations on books, and returns responses in JSON format. It runs on port 5000.

2. **Create 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"]
```

(cloud-and-bigdata) ×

```
FROM python:alpine3.7

COPY . /app
WORKDIR /app

RUN pip install -r requirements.txt

ENV PORT 5000
EXPOSE 5000

ENTRYPOINT ["python3"]
CMD ["bookshelf.py"]
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ cat requirements.txt
Flask==2.0.1
flask_pymongo==2.3.0
pymongo==3.11.4
```

- Make sure to add a requirement file
- Define the Dockerfile to containerize the Flask application.
- This Dockerfile sets up a Python environment, copies the application code, installs dependencies, and specifies the entry point for the container.

3. **Build Docker Image:**

```
docker build -t asdg124/bookshelf .
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ docker build -t asdg124/bookshelf .
[+] Building 37.9s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 197B
 => [internal] load metadata for docker.io/library/python:alpine3.7
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 1.30MB
 => [1/4] FROM docker.io/library/python:alpine3.7@sha256:35f6f83ab08f98c727dbefd53738e3b3174a48b4571ccb1910bae480dcdba847
 => => resolve docker.io/library/python:alpine3.7@sha256:35f6f83ab08f98c727dbefd53738e3b3174a48b4571ccb1910bae480dcdba847
 => => sha256:00be2573e9f79754b17954ba7a310a5f70c25b8f5bb78375e27e9e86d874877e 6.13kB / 6.13kB
 => => sha256:48ecbb6b270eb481cb6df2a5b0332de294ec729e1968e92d725f1329637ce01b 2.11MB / 2.11MB
 => => sha256:692f29ee68fa6bab04aa6a1c6d8db0ad44e287e5ff5c7e1d5794c3aabc55884d 308.48kB / 308.48kB
 => => sha256:6439819450d10d1aae92561f3ffff722137aada46d509644e8de4ca82bb26b07 25.90MB / 25.90MB
 => => sha256:35f6f83ab08f98c727dbefd53738e3b3174a48b4571ccb1910bae480dcdba847 2.04kB / 2.04kB
 => => sha256:014f52b0e7ae4fcd43201bfa4c4e0320c8517b611d7daa0e41ba33a0cb1fab80 1.37kB / 1.37kB
 => => extracting sha256:48ecbb6b270eb481cb6df2a5b0332de294ec729e1968e92d725f1329637ce01b
 => => sha256:3c7be240f7bfb19ec575d8547832a9f20b95eec9b4cc94fe717dd047ad661159 230B / 230B
 => => sha256:ca4b349df8ed83a59776df8f3868ece2783aa1ee2e9f052c9c9f3b54ae51a593 1.81MB / 1.81MB
 => => extracting sha256:692f29ee68fa6bab04aa6a1c6d8db0ad44e287e5ff5c7e1d5794c3aabc55884d
 => => extracting sha256:6439819450d10d1aae92561f3ffff722137aada46d509644e8de4ca82bb26b07
 => => extracting sha256:3c7be240f7bfb19ec575d8547832a9f20b95eec9b4cc94fe717dd047ad661159
 => => extracting sha256:ca4b349df8ed83a59776df8f3868ece2783aa1ee2e9f052c9c9f3b54ae51a593
 => [2/4] COPY . /app
 => [3/4] WORKDIR /app
 => [4/4] RUN pip install -r requirements.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:db07fb6c82dc89ad105670e43ad69f1cede068f3ad19e5836a135c8aa214cf00
 => => naming to docker.io/asdg124/bookshelf

 1 warning found (use --debug to expand):
 - LegacyKeyValueFormat: "ENV key=value" should be used instead of legacy "ENV key value" format (line 8)
```

- Build the Docker image for the Flask application.
- This command creates a Docker image named `bookshelf` tagged with your Docker Hub ID, packaging the Flask application for deployment.

4. **Push Docker Image:**

```
docker push asdg124/bookshelf
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ docker tag asdg124/bookshelf:latest asdg124/bookshelf
:latest
adagniew407@cloudshell:~ (cloud-and-bigdata)$ docker push asdg124/bookshelf:latest
The push refers to repository [docker.io/asdg124/bookshelf]
5194c359e180: Pushed
5f70bf18a086: Pushed
75760539bf52: Pushed
5fa31f02caa8: Layer already exists
88e61e328a3c: Layer already exists
9b77965e1d3f: Layer already exists
50f8b07e9421: Layer already exists
629164d914fc: Layer already exists
latest: digest: sha256:2a1496479808f5d048527061d4efb9a6eab21260320bd3ded9cce26f38500b7a size: 1998
```

- Push the Docker image to Docker Hub.
- Uploading the Docker image to Docker Hub makes it available for deployment on GKE.

Step 4: Create ConfigMap for Both Applications

1. **Create `studentserver-config.yaml`:**

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: studentserver-config
data:
```

```
  MONGO_URL: 34.71.187.5
  MONGO_DATABASE: studentdb
```

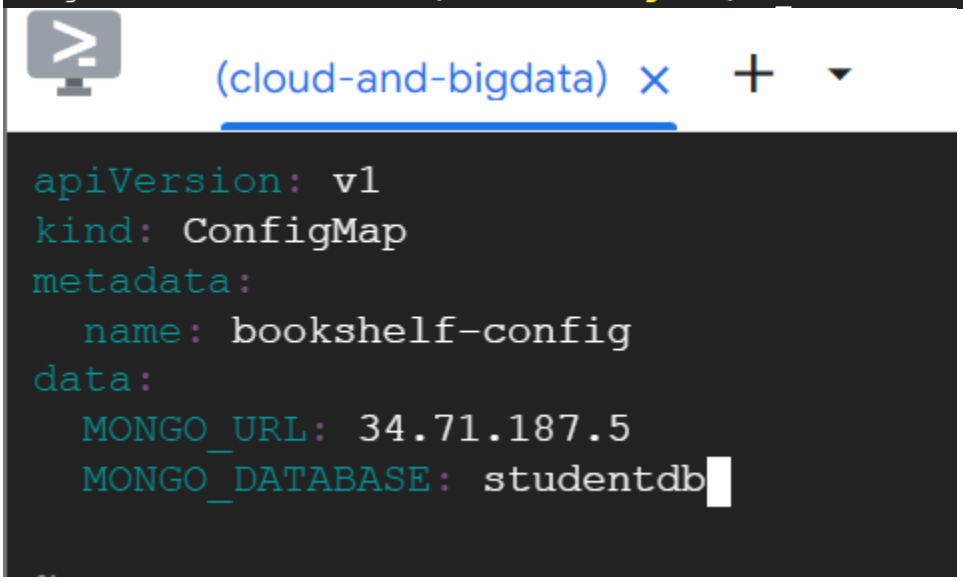
```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ vi studentserver-configmap.yaml
adagniew407@cloudshell:~ (cloud-and-bigdata)$ cat studentserver-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: studentserver-config
data:
  MONGO_URL: 34.71.187.5
  MONGO_DATABASE: studentdb
```

- Define a ConfigMap for the student server application.
- This ConfigMap provides configuration data (MongoDB URL and database name) to the student server, allowing it to connect to MongoDB.

2. **Create `bookshelf-configmap.yaml`:**

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: bookshelf-config
data:
  MONGO_URL: 34.71.187.5
  MONGO_DATABASE: studentdb
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ vi bookshelf-configmap.yaml
```



- Define a ConfigMap for the bookshelf application.
- This ConfigMap provides configuration data (MongoDB URL and database name) to the bookshelf application, allowing it to connect to MongoDB.
- 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

Create `studentserver-deployment.yaml`

This YAML file describes the deployment configuration for the Student Server application, including the number of replicas, container image, environment variables, and ports.

```
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: asdg124/mydbb
        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
```

(cloud-and-bigdata) ×

```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: asdg124/mydbd
        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
```

This configuration specifies the deployment of the `studentserver` application using the Docker image `zhou19539/studentserver`. It pulls environment variables for MongoDB connection from a ConfigMap.

Step 2: Create `bookshelf-deployment.yaml`

This YAML file describes the deployment configuration for the Bookshelf application, similar to the student server deployment.

```
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: asdg124/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
```

(cloud-and-bigdata) ×

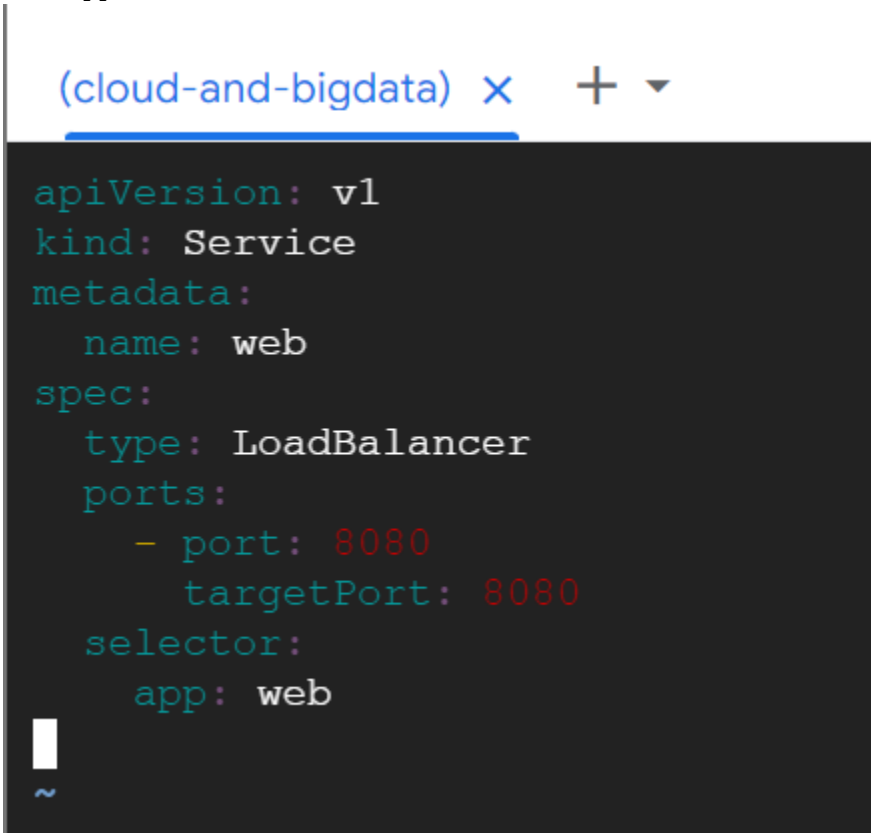
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bookshelf-deployment
  labels:
    app: bookshelf-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bookshelf-deployment
  template:
    metadata:
      labels:
        app: bookshelf-deployment
    spec:
      containers:
      - image: asdg124/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
```

This configuration specifies the deployment of the `bookshelf` application using the Docker image `zhou19539/bookshelf`. It also pulls environment variables for MongoDB connection from a ConfigMap.

Step 3: Create `studentserver-service.yaml`

This YAML file defines a service for the Student Server application to expose it outside the Kubernetes cluster.

```
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  type: LoadBalancer
  ports:
    - port: 8080
      targetPort: 8080
  selector:
    app: web
```

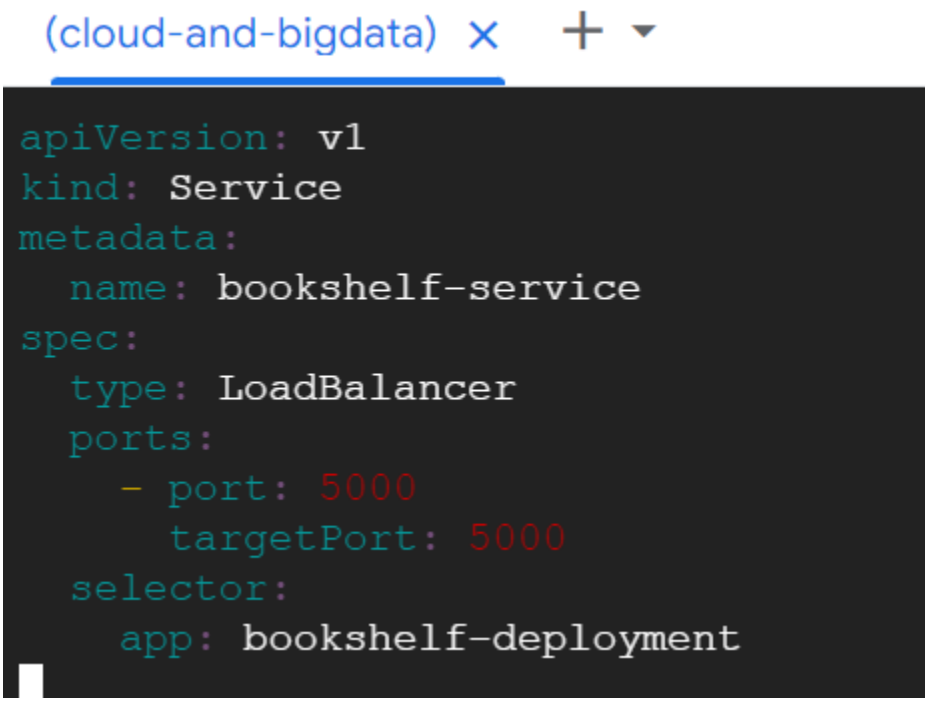


This service exposes the Student Server application on port 8080 using a LoadBalancer, which routes external traffic to the application pods.

Step 4: Create `bookshelf-service.yaml`

This YAML file defines a service for the Bookshelf application to expose it outside the Kubernetes cluster.

```
apiVersion: v1
kind: Service
metadata:
  name: bookshelf-service
spec:
  type: LoadBalancer
  ports:
    - port: 5000
      targetPort: 5000
  selector:
    app: bookshelf-deployment
```



This service exposes the Bookshelf application on port 5000 using a LoadBalancer, which routes external traffic to the application pods.

Step 5: Start Minikube

This command starts a Minikube cluster, which is a local Kubernetes environment.

```
minikube start
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ minikube start
* minikube v1.33.1 on Ubuntu 22.04 (amd64)
  - MINIKUBE_FORCE_SYSTEMD=true
  - MINIKUBE_HOME=/google/minikube
  - MINIKUBE_WANTUPDATENOTIFICATION=false
* Automatically selected the docker driver. Other choices: ssh, none
* Using Docker driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.44 ...
* Downloading Kubernetes v1.30.0 preload ...
    > preloaded-images-k8s-v18-v1...:  342.90 MiB / 342.90 MiB  100.00% 210.10
    > gcr.io/k8s-minikube/kicbase...:  481.58 MiB / 481.58 MiB  100.00% 88.07 M
* Creating docker container (CPUs=2, Memory=4000MB) ...
* Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
  - 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
```

Minikube provides a simple way to create a local Kubernetes cluster for development and testing purposes.

Step 6: Start Ingress

This command enables the Nginx ingress controller in Minikube.

```
minikube addons enable ingress
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ 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/maste
r/OWNERS
  - Using image registry.k8s.io/ingress-nginx/controller:v1.10.1
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.1
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.1
* Verifying ingress addon...
* The 'ingress' addon is enabled
```

Enabling the ingress addon allows you to manage inbound traffic to your Kubernetes services using the Nginx ingress controller.

Step 7: Apply Student Server Deployment and Service

These commands create the deployment and service for the Student Server application using the YAML files created earlier.

```
kubectl apply -f studentserver-deployment.yaml
kubectl apply -f studentserver-configmap.yaml
kubectl apply -f studentserver-service.yaml
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl apply -f studentserver-deployment.yaml
deployment.apps/web created
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl apply -f studentserver-configmap.yaml
configmap/studentserver-config created
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl apply -f studentserver-service.yaml
service/web created
```

These commands apply the deployment and service configurations to the Kubernetes cluster, creating the necessary pods and services for the Student Server application.

Step 8: Apply Bookshelf Deployment and Service

These commands create the deployment and service for the Bookshelf application using the YAML files created earlier.

```
kubectl apply -f bookshelf-deployment.yaml
kubectl apply -f bookshelf-configmap.yaml
kubectl apply -f bookshelf-service.yaml
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl apply -f bookshelf-deployment.yaml
deployment.apps/bookshelf-deployment created
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl apply -f bookshelf-configmap.yaml
configmap/bookshelf-config created
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl apply -f bookshelf-service.yaml
service/bookshelf-service created
```

These commands apply the deployment and service configurations to the Kubernetes cluster, creating the necessary pods and services for the Bookshelf application.

Step 9: Check Pods Status

This command checks the status of all the pods in the Kubernetes cluster.

```
kubectl get pods
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
bookshelf-deployment-5dbd8d848c-92pfz   1/1     Running   0          40s
web-7c75fd7b65-r2c79                   1/1     Running   0          4m31s
```

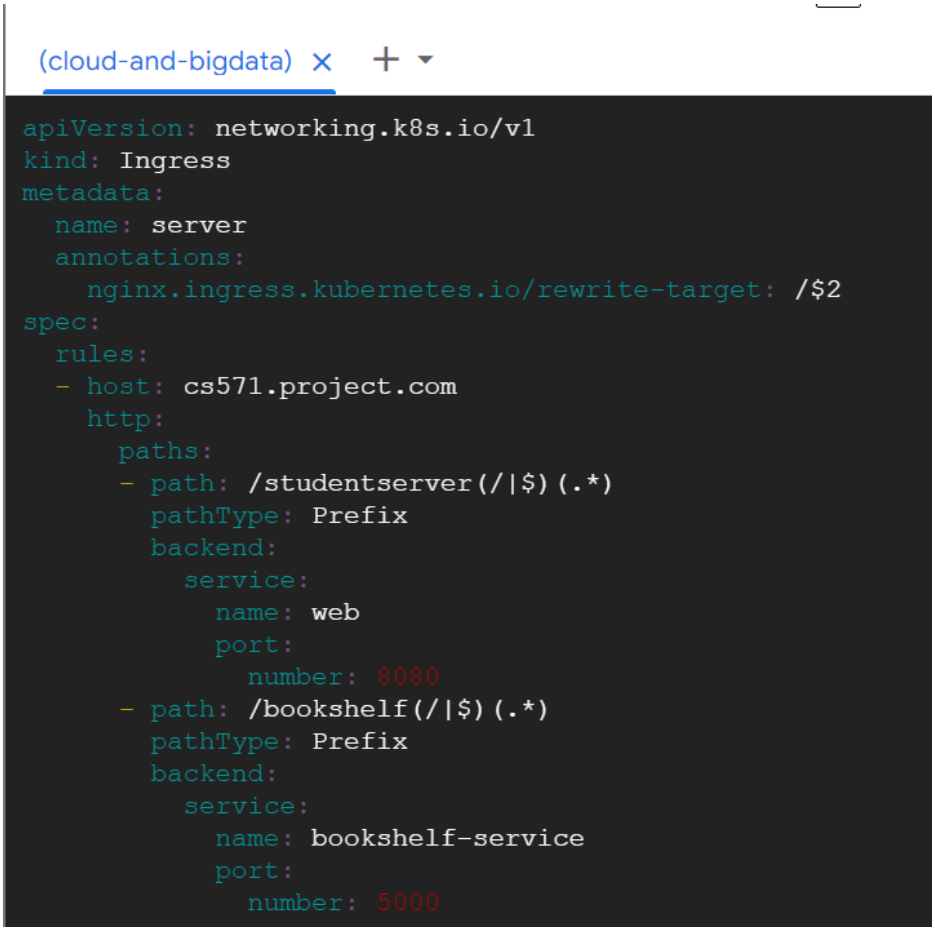
This command lists all the pods in the cluster, allowing you to verify that the deployments are running correctly.

Step 10: Create Ingress Configuration

This YAML file defines an ingress resource that routes traffic to the Student Server and Bookshelf services based on the URL path.

```
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: ImplementationSpecific
        backend:
          service:
            name: web
            port:
              number: 8080
      - path: /bookshelf(/|$)(.*)
        pathType: ImplementationSpecific
        backend:
          service:
            name: bookshelf-service
            port:
              number: 5000
```



This ingress resource directs requests to `cs571.project.com/studentserver` to the Student Server service and requests to `cs571.project.com/bookshelf` to the Bookshelf service.

Step 11: Apply Ingress Configuration

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

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ 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
```

Applying this ingress configuration allows you to route traffic to the Student Server and Bookshelf services based on the URL path.

Step 12: Check Ingress Status

This command checks the status of the ingress resource.

```
kubectl get ingress
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl get ingress
NAME     CLASS   HOSTS               ADDRESS   PORTS   AGE
server   nginx   cs571.project.com             80      8s
```

This command lists the ingress resources, allowing you to verify that the ingress is set up correctly and has an assigned address.

Step 13: Update `/etc/hosts`

This step updates the `/etc/hosts` file on your local machine to map the ingress address to the domain name `cs571.project.com`.

Before applying lets do some checkups:

Get the Minikube IP:

minikube ip

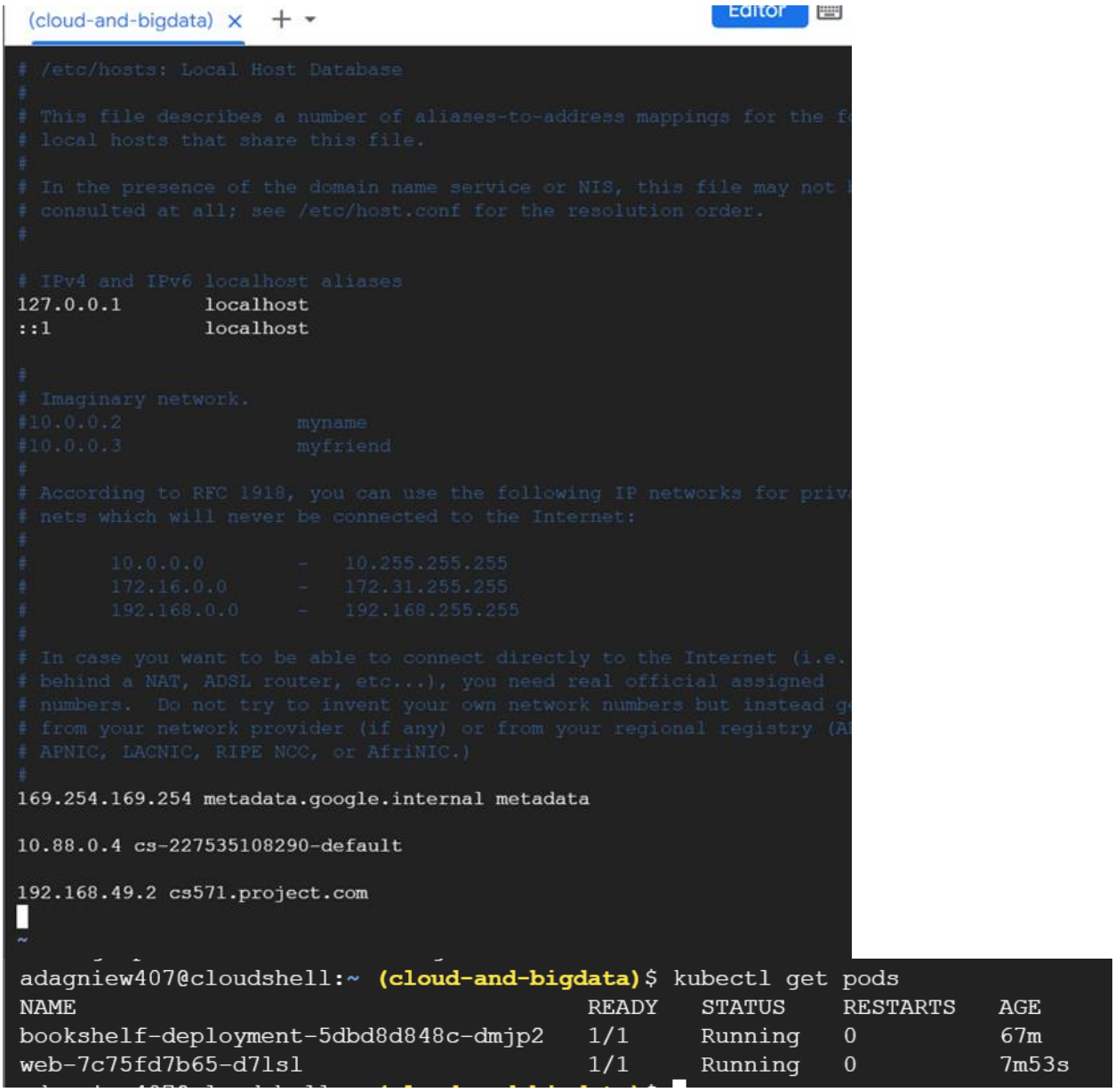
```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ minikube ip
192.168.49.2
```

```
vi /etc/hosts
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ sudo vi /etc/hosts
```

Add the line:

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



Adding this line to the `/etc/hosts` file allows you to access the applications using the custom domain name in your local environment.

Step 14: Access the Applications

These commands use `curl` to test the endpoints of the Student Server and Bookshelf applications.

```
# Access Student Server
curl cs571.project.com/studentserver/api/score?student_id=11111
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ curl cs571.project.com/studentserver/api/score?student_i
1
{"student_id":11111,"student_name":"Bruce Lee","student_score":84}adagniew407@cloudshell:~ (cloud-and-
```

```
# List all books in Bookshelf
curl cs571.project.com/bookshelf/books
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ curl cs571.project.com/bookshelf/books
[]
```

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

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ curl -X POST -d "{\"book_name\": \"cloud computing\",\"book_author\":
\"unkown\", \"isbn\": \"123456\" }" http://cs571.project.com/bookshelf/book
{
  "message": "Task saved successfully!"
}
adagniew407@cloudshell:~ (cloud-and-bigdata)$ curl cs571.project.com/bookshelf/books
[
  {
    "Book Author": "unkown",
    "Book Name": "cloud computing",
    "ISBN": "123456",
    "id": "66a87500f086ee9aa5b9bae1"
  }
]
```

```
# Update a book
curl -X PUT -H "Content-Type: application/json" -d '{"book_name": "Updated
Book Name", "book_author": "Updated Author", "isbn": "Updated ISBN"}'
http://cs571.project.com/bookshelf/book/66a87500f086ee9aa5b9bae1
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ curl -X PUT -H "Content-Type: application/json" -d '{"book_name": "Updated Book Name", "book_author": "Updated Author", "isbn": "Updated ISBN"}' http://cs57
1.project.com/bookshelf/book/66a87500f086ee9aa5b9bae1
{
  "message": "Task updated successfully!"
}
adagniew407@cloudshell:~ (cloud-and-bigdata)$
adagniew407@cloudshell:~ (cloud-and-bigdata)$ curl cs571.project.com/bookshelf/books
[
  {
    "Book Author": "Updated Author",
    "Book Name": "Updated Book Name",
    "ISBN": "Updated ISBN",
    "id": "66a87500f086ee9aa5b9bae1"
  }
]
```

```
# Delete a book
curl -X DELETE
http://cs571.project.com/bookshelf/book/66a87500f086ee9aa5b9bae1
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ curl -X DELETE http://cs571.project.com/bookshelf/book/66a87500f086ee9aa5b9bae1
{
  "message": "Task deleted successfully!"
}
adagniew407@cloudshell:~ (cloud-and-bigdata)$ curl cs571.project.com/bookshelf/books
[]
```

These commands test the functionality of the deployed applications by performing various CRUD operations via the REST API endpoints. By following these steps, you can deploy and expose your Student Server and Bookshelf applications on the same domain with different paths using Kubernetes and Nginx Ingress.