AIN3003

Database and Cloud Computing Project

Yaser Z. K. Shoshaa

Student ID 2104449

Table of Contents

Task 1: Construct MongoDB

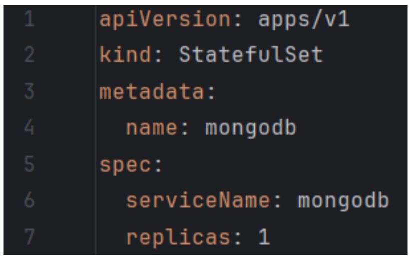
Kubernetes StatefulSet

We use the following configuration for it.

- replicas:

  For this use case, a single MongoDB instance in the setup file for simplicity, for scaling we can run this command.

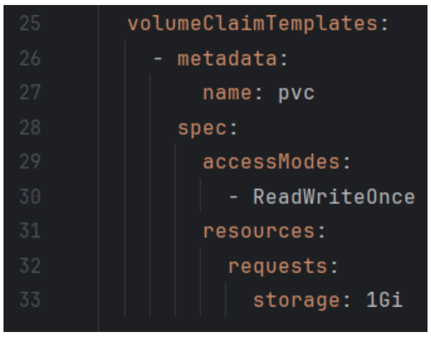
  kubectl scale statefulset mongodb --replicas=4



```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongodb
spec:
  serviceName: mongodb
  replicas: 1
```

- volumeClaimTemplates:

  Utilizing a Persistent Volume Claim (PVC) ensures that MongoDB data persists beyond the lifecycle of the Pod.

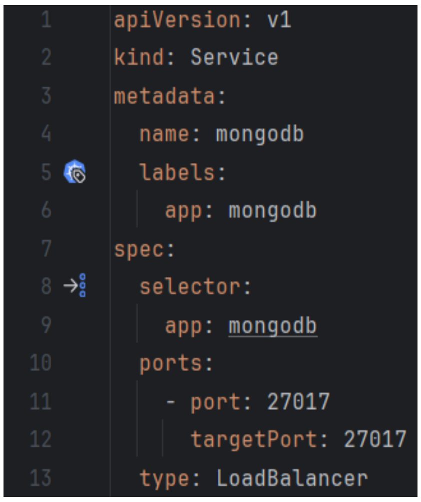


```yaml
  volumeClaimTemplates:
    - metadata:
        name: pvc
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 1Gi
```

- ReadWriteOnce Access Mode:

  ReadWriteOnce access mode is appropriate for the PVC, ensuring that the volume is mounted as read-write by a single pod at a time.

Kubernetes Service



```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongodb
  labels:
    app: mongodb
spec:
  selector:
    app: mongodb
  ports:
    - port: 27017
      targetPort: 27017
  type: LoadBalancer
```

we create a Kubernetes service with the following configuration using port 27017 to match the MongoDB instance
and Type LoadBalancer to have external access as well as distribute the traffic into the pods.

after configuring both we apply both files into azure Kubernetes service using these commands

kubectl apply -f mongodb-statefulset.yaml

kubectl apply -f mongodb-svc.yaml

Task 2: Develop Python Application

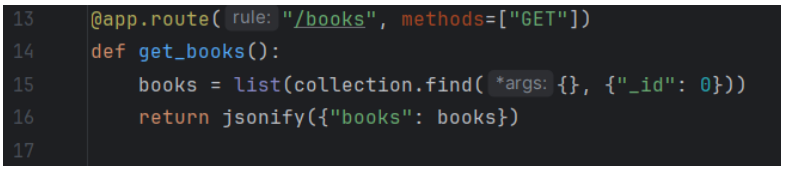
We have used Flask to ensure RESTful.

i. **Get All Books**

Endpoint: /books

Method: GET

Description: Retrieve a list of all books in the bookstore.



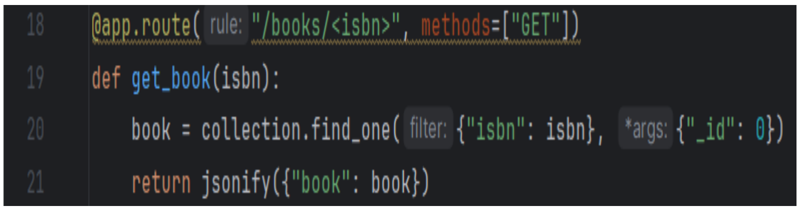
```python
@app.route( rule: "/books", methods=["GET"])
def get_books():
    books = list(collection.find( *args: {}, {"_id": 0}))
    return jsonify({"books": books})
```

ii. **Get a Specific Book**

Endpoint: /books/<isbn>

Method: GET

Description: Retrieve details of a specific book using its ISBN.



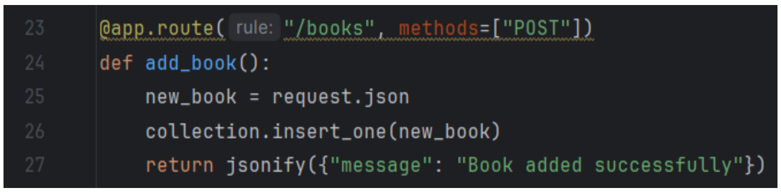
```python
@app.route( rule: "/books/<isbn>", methods=["GET"])
def get_book(isbn):
    book = collection.find_one( filter: {"isbn": isbn}, *args: {"_id": 0})
    return jsonify({"book": book})
```

iii. **Add a New Book**

Endpoint: /books

Method: POST

Description: Add a new book to the bookstore.



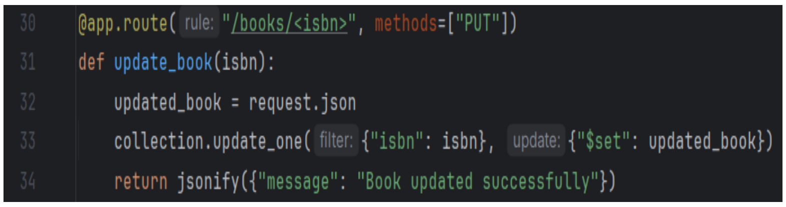
```python
@app.route( rule: "/books", methods=["POST"])
def add_book():
    new_book = request.json
    collection.insert_one(new_book)
    return jsonify({"message": "Book added successfully"})
```

iv. **Update a Book**

Endpoint: /books/<isbn>

Method: PUT

Description: Update the details of a specific book using its ISBN.



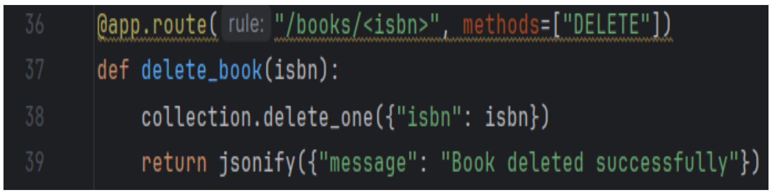
```python
@app.route( rule: "/books/<isbn>", methods=["PUT"])
def update_book(isbn):
    updated_book = request.json
    collection.update_one( filter: {"isbn": isbn}, update: {"$set": updated_book})
    return jsonify({"message": "Book updated successfully"})
```

v. **Delete a Book**

Endpoint: /books/<isbn>

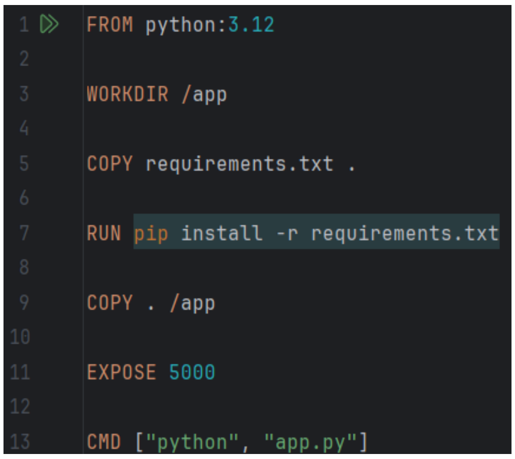
Method: DELETE

Description: Delete a specific book from the bookstore.



```python
@app.route( rule: "/books/<isbn>", methods=["DELETE"])
def delete_book(isbn):
    collection.delete_one({"isbn": isbn})
    return jsonify({"message": "Book deleted successfully"})
```

Containerizing the Python App



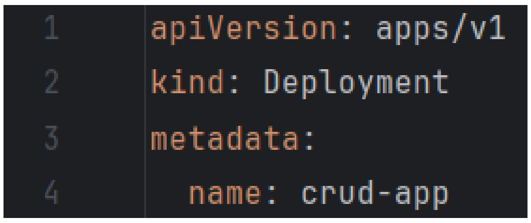
```
FROM python:3.12

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . /app

EXPOSE 5000

CMD ["python", "app.py"]
```

docker login

docker build -t crud-app:aks .

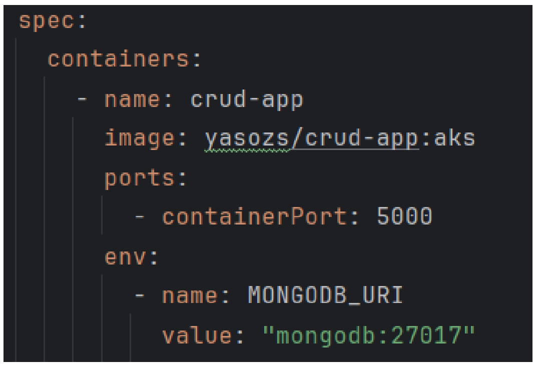
docker push yasozs/crud-app:aks

Task 3: Kubernetes Deployment



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: crud-app
```

Using the docker image we pushed to Docker hub and setting other specs like kind, name, etc.



```
spec:
  containers:
    - name: crud-app
      image: yasozs/crud-app:aks
      ports:
        - containerPort: 5000
      env:
        - name: MONGODB_URI
          value: "mongodb:27017"
```

We used environmental variables for configuration.

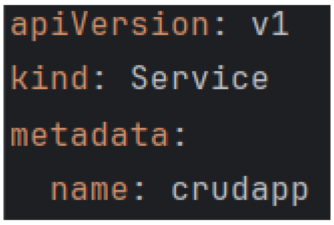
We can create Kubernetes Deployment for our python application.

kubectl apply -f app.yaml

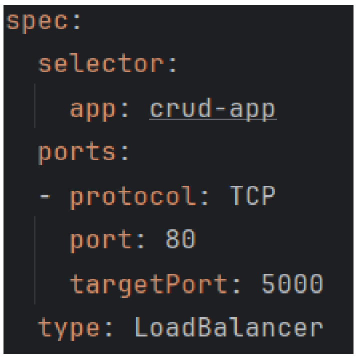
We pair it with a Kubernetes Service

We set the target port to match our deployment port.

kubectl apply -f app-svc.yaml



```
apiVersion: v1
kind: Service
metadata:
  name: crudapp
```

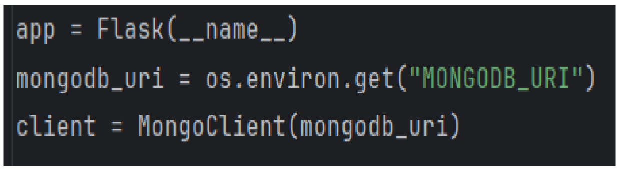


```
spec:
  selector:
    app: crud-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
  type: LoadBalancer
```

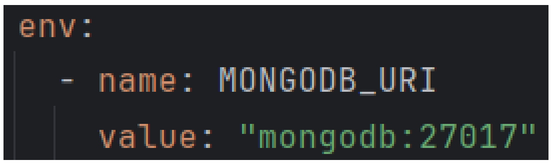
Task 4:

Service Discovery

After deploying the Python app into Azure and having both of Python apps running on the cloud, as well as the Mongo instance we can implement service discovery between them.



```python
app = Flask(__name__)
mongodb_uri = os.environ.get("MONGODB_URI")
client = MongoClient(mongodb_uri)
```



```yaml
env:
  - name: MONGODB_URI
    value: "mongodb:27017"
```

using our service name for connection with the specified port