SIT323/SIT737- Cloud Native Application Development
9.1P: Adding a database to your application

Overview :

When developing an application, it's common to work with data, files, or both. Microservices are no exception to this. In order to store dynamic data that's generated and updated by the microservice, we need to have a database. Additionally, we need to have a storage location for assets that are either served by the application or uploaded to it. For this task, your objective is to integrate a database into your existing containerized microservice application.

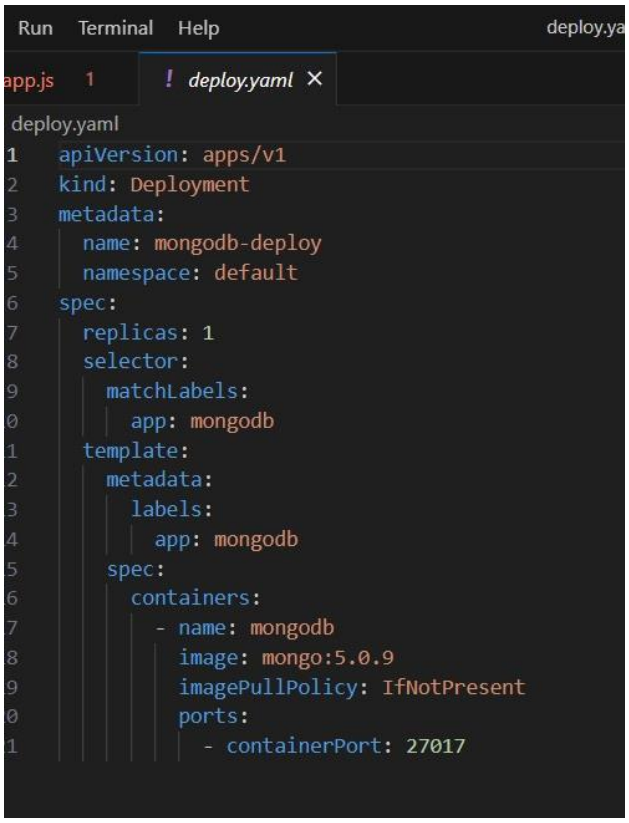
Documetation on Deploying MongoDB application on Kubernetes

Instructions

1. **Deploy MongoDB using a Kubernetes configuration**

Create a file named mongodb.yaml and add the following content to it:

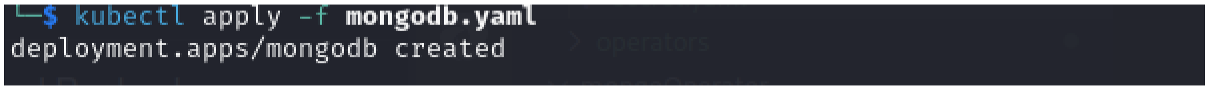
```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
```



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deploy
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
        - name: mongodb
          image: mongo:5.0.9
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 27017
```

Then apply the configuration file –

$ kubectl apply -f mongodb.yaml



```
└─$ kubectl apply -f mongodb.yaml
deployment.apps/mongodb created
```

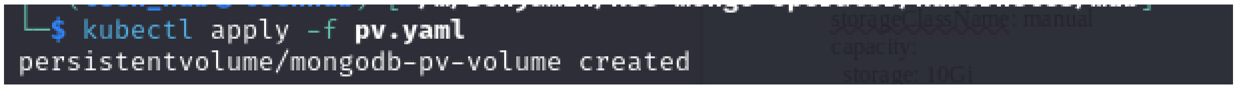
2. Create a Persistent Volume and Persistent Volume Claim

Create a file named pv.yaml and add the following content to it:

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: mongodb-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Then apply the configuration file –

$ kubectl apply -f pv.yaml



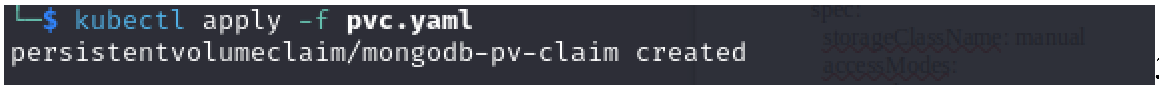
```
└─$ kubectl apply -f pv.yaml
persistentvolume/mongodb-pv-volume created
```

Create a file named pvc.yaml and add the following content to it:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Then apply the configuration file –

$ kubectl apply -f pvc.yaml



```
└─$ kubectl apply -f pvc.yaml
persistentvolumeclaim/mongodb-pv-claim created
```

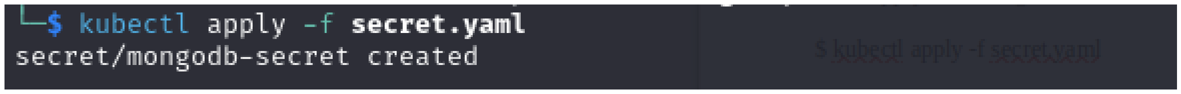
3. Create a Secret for the MongoDB User Credentials

Create a file named secret.yaml and add the following content to it:

```
apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
type: Opaque
stringData:
  MONGO_USERNAME: <username>       # set the MongoDB username
  MONGO_PASSWORD: <password>       # set the MongoDB password
```

Then apply the configuration file –

$ kubectl apply -f secret.yaml



```
└─$ kubectl apply -f secret.yaml
secret/mongodb-secret created
```

4. Modify the Deployment Manifest

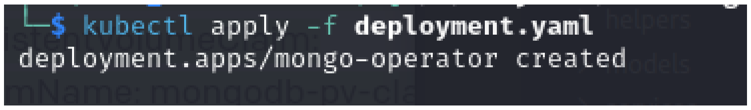
Update the deployment manifest to include the secret and the MongoDB database parameters.
For example,

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: <deployment-name>
spec:
  selector:
    matchLabels:
      app: <application-name>
  template:
    metadata:
      labels:
        app: <application-name>
    spec:
      containers:
      - name: <docker-container-name>
        image: <docker-image-name>
        ports:
        - containerPort: 80
        env:
        - name: MONGO_USERNAME
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: MONGO_USERNAME
        - name: MONGO_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: MONGO_PASSWORD
        - name: MONGO_HOST
          value: mongodb
        - name: MONGO_PORT
          value: "27017"   # set the MongoDB port
      volumes:
      - name: mongodb-persistent-storage
```

```
        persistentVolumeClaim:
          claimName: mongodb-pv-claim
```

Then apply the configuration file –

$ kubectl apply -f deployment.yaml



```
└─$ kubectl apply -f deployment.yaml
deployment.apps/mongo-operator created
```

5. Configure Application to Connect to MongoDB

Update the application code to use the MongoDB credentials and connection string from the
deployment manifest.

For example:

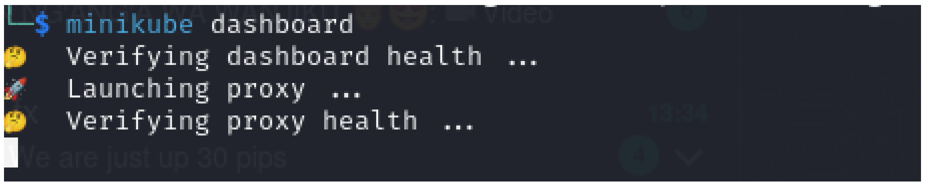
```
const MongoDB = require('mongodb');
// get the username and password from the environment variables
let username= process.env.MONGO_USERNAME;
let password = process.env.MONGO_PASSWORD;

// Get the MongoDB host and port from the environment variables
let host = process.env.MONGO_HOST;
let port = process.env.MONGO_PORT;

// create the MongoDB connection string
let url = `mongodb://${username}:${password}@${host}:${port}`;

// establish a connection to the MongoDB
MongoDB.connect(url, function(err, client) {
  if (err) throw err;
  // perform application tasks
});
```

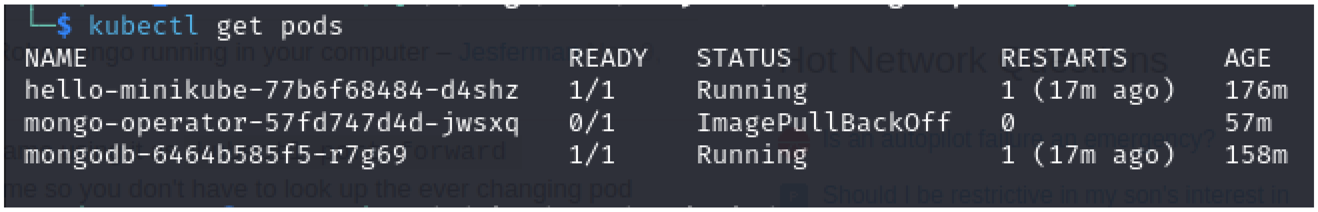
Launch the Kubernetes dashboard.



```
└─$ minikube dashboard
🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
```

6. Test the Deployment

Test the deployment to ensure that the application can connect to the MongoDB database and perform basic CRUD (Create, Read, Update, Delete) operations.



```
└─$ kubectl get pods
NAME                               READY   STATUS             RESTARTS      AGE
hello-minikube-77b6f68484-d4shz    1/1     Running            1 (17m ago)   176m
mongo-operator-57fd747d4d-jwsxq    0/1     ImagePullBackOff   0             57m
mongodb-6464b585f5-r7g69           1/1     Running            1 (17m ago)   158m
```

7. Set Up Database Backups and Disaster Recovery

Backup your MongoDB database using tools such as mongodump or mongo-backup-utils and configure automated backups for periods of time of your choice. Additionally, you can set up disaster recovery mechanisms such as replication or replica sets to ensure that your data is secure in case of an emergency.

8. Monitor Performance

Monitor the performance of the MongoDB database and the application to ensure that the database is running smoothly and efficiently. You can use monitoring tools such as MongoDB's Atlas monitoring tool or MongoDB Ops Manager to monitor the performance of your MongoDB instance. Additionally, you can use application performance monitoring (APM) tools such as New Relic or AppDynamic to monitor the performance of your application.