MERN GKE

ŞÜKRÜ ÇİRİŞ

ARÇELİK A.Ş.

1- Giriş:
MERN uygulamanı oluştur. Uygulamamın kodunu, dosya yapısını ve ilerdeki adımlarda kullandığım dosyalar ve komutların hepsi buradaki repoda: https://github.com/SUKRUCIRIS/MERN-kubernetes

2- Docker dosyaları ve nginx:
Dockerhub hesabın yoksa oluştur. Orada üç tane repo oluştur backend, frontend ve database için. Docker compose ile oluşturduğumuz image'ları oraya push edicez. GKE oradan pull edicek.

a. Client için nginx dosyası:

```
server {
    listen 80;

    location /server/ {
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_pass http://server:8080/;
    }

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    error_page 405 =200 $uri;
    error_page 500 502 503 504 /50x.html;

    location = 50x.html {
        root /usr/share/nginx/html;
    }
}
```

80 portunu dinleyecek. Dördüncü satırda /server/ ile başlayan adreslerin backend'e gitmesi sağlanıyor. Backend'in kubenetes servisinin ismi server ve portu 8080 olduğu için onuncu satır o şekilde yazıldı. Ayrıca uygulama 405 hatası veriyordu backend'e attığı sorgularda, on dokuzuncu satır bunu engelliyor.

b. Client için Dockerfile dosyası:

```
FROM node:20.4-bookworm-slim as build
ENV NODE_ENV=production
RUN apt-get update
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm ci --omit=dev
COPY . .
RUN npm run build

FROM nginx:1.16.0
COPY --from=build /usr/src/app/build /usr/share/nginx/html
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx/nginx.conf /etc/nginx/conf.d
EXPOSE 80
CMD ["/bin/bash", "-c", "nginx -g \"daemon off;\""]
```

İlk adımda react frontend'i production için build ediyorum. İkinci adımda nginx ile canlıya alıyorum. On birinci satırda ilk adımda oluşturduğum dosyaları yeni image'a taşıyorum. 80 portunu expose ediyorum. Ayrıca node_modules gibi gereksiz dosyaları image' a aktarmamak için bir .dockerignore dosyası da oluşturdum. CMD komutları image çalışınca çalışır, RUN komutları image oluşturulurken çalışır.

c. Server için Dockerfile dosyası:

```
1   FROM node:20.4-bookworm-slim
2   ENV NODE_ENV=production
3   RUN apt-get update
4   WORKDIR /usr/src/app
5   COPY package*.json ./
6   RUN npm ci --omit=dev
7   COPY . .
8   EXPOSE 8080
9   CMD [ "node", "index.cjs" ]
```

Resmi node image'ını taban alarak production modunda gerekli modülleri kuruyorum. 8080 portunu expose ediyorum. Backend kodumun olduğu dosya ismi index.cjs. Mongodb'ye mongoose modülü ile bağlanıyor.

```
const mongoose = require("mongoose");
const bodyParser = require("body-parser");

mongoose.connect("mongodb://mongo:27017/tododb?directConnection=true");

const userSchema = new mongoose.Schema({
  username: { type: String, unique: true },
```

Database için kullandığım kubernetes servisinin ismi mongo, portu 27017 ve database ismi tododb dolayısıyla bu şekilde bağlandım.

d. Database için Dockerfile dosyası:

```
1   FROM mongo:7.0.0-rc8
```

Database için sadece resmi image'ı kurmak yeterli.

e. Docker compose dosyası:

Docker compose kolayca multi container uygulama oluşturmak için kullanılan bir araçtır. Burada 3 tane image oluşturulması için komutlar var. "build" parametresi docker dosyasının yolunu gösterir. "ports" parametresinde kullandığınız portları kullanmanız lazım. "image" parametresi ile oluşturulan image'ı isimlendirip tag'leyebiliriz. Bu parametreye dockerhub'da oluşturduğum repoların ismini latest tagi ile koydum. "container_name" parametresi bu compose dosyası ile container oluşturulursa oluşacak container'ın ismini belirler. Github actions'da "docker compose build" ve "docker compose push" komutlarını kullanarak image'ları bu dosya ile build ve push edicem.

```yaml
version: "3.8"
services:
  client:
    build: client
    stdin_open: true
    container_name: client
    image: sukruciris/mern_client:latest
    restart: always
    depends_on:
      - server
    ports:
      - 80:80
    networks:
      - react-express

  server:
    container_name: server
    restart: always
    build: server
    depends_on:
      - mongo
    ports:
      - 8080:8080
    networks:
      - express-mongo
      - react-express
    image: sukruciris/mern_server:latest

  mongo:
    container_name: mongo
    restart: always
    build: mongo
    ports:
      - 27017:27017
    networks:
      - express-mongo
    image: sukruciris/mern_mongo:latest

networks:
  react-express:
  express-mongo:

```

3- Kubernetes dosyaları:

a. Client için deployment dosyası:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-client
  labels:
    name: deploy-client
    app: todo-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: deploy-client
      app: todo-app
  template:
    metadata:
      name: deploy-client
      labels:
        name: deploy-client
        app: todo-app
    spec:
      containers:
        - name: deploy-client
          image: "sukruciris/mern_client:latest"
          imagePullPolicy: Always
          ports:
            - containerPort: 80

```

containerPort parametresi uygulamanın kullandığı port olmalı, image parametresi dockerhub'daki kullanmak istediğimiz image'ın adresi olmalı. imagePullPolicy'nin Always olması, image'ın çalıştığı podu silersek her seferinde ECR'deki repodan belirttiğimiz lastest tagine sahip image'ı tekrar çekip onunla çalışacak demek. Dördüncü satırdaki deployment ismi unique olmalı.

b. Client için servis dosyası:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: client
  labels:
    name: client
    app: todo-app
spec:
  type: LoadBalancer
  selector:
    app: todo-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80

```

Bu servisin tipi LoadBalancer, dış ağa açık olması için. "port" ve "targetPort" parametresi için client'in kullandığı port olan 80 değerini verdim.

c. Server için deployment dosyası:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-server
  labels:
    name: deploy-server
    app: todo-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: deploy-server
      app: todo-app
  template:
    metadata:
      name: deploy-server
      labels:
        name: deploy-server
        app: todo-app
    spec:
      containers:
        - name: deploy-server
          image: "sukruciris/mern_server:latest"
          imagePullPolicy: Always
          ports:
            - containerPort: 8080

```

d. Server için servis dosyası:

Dokuzuncu satırda belirtildiği üzere bu servisin tipi "ClusterIP" yani dış ağa kapalı. Açık olmasına gerek yok çünkü backend sadece database ile etkileşecek.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: server
  labels:
    name: server
    app: todo-app
spec:
  type: ClusterIP
  selector:
    app: todo-app
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080

```

e. Storage class dosyası:

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: mongo-storage
provisioner: kubernetes.io/gce-pd
volumeBindingMode: Immediate
allowVolumeExpansion: true
reclaimPolicy: Delete
parameters:
  type: pd-standard
  fstype: ext4
  replication-type: none

```

Database için oluşturacağım persistent volume'ün kullanacağı storage class'ı yaratılıyor burda. Altıncı satırdaki parametrenin immediate olması önemli yoksa database ve volume'ün ikisi de birbirinin yaratılmasını bekliyor.

f. Persistent volume claim dosyası:

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-persistent-storage
spec:
  storageClassName: mongo-storage
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi

```

Burada database için 10gb'lık bir persistent volume oluşturuyorum. accessModes parametresinin ReadWriteOnce olması bu volume'e aynı anda tek bir node'un yazılıp okunmasına izin verildiği anlamına geliyor.

g. Database için deployment dosyası:
Daha önce oluşturduğum volume'ü burada kullandım.

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: deploy-mongo
  labels:
    name: deploy-mongo
    app: todo-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: deploy-mongo
      app: todo-app
  template:
    metadata:
      name: deploy-mongo
      labels:
        name: deploy-mongo
        app: todo-app
    spec:
      containers:
        - name: deploy-mongo
          image: "sukruciris/mern_mongo:latest"
          imagePullPolicy: Always
          ports:
            - containerPort: 27017
          volumeMounts:
            - name: mongo-persistent-storage
              mountPath: /data/db
      volumes:
        - name: "mongo-persistent-storage"
          persistentVolumeClaim:
            claimName: "mongo-persistent-storage"

```

h. Database için servis dosyası:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongo
  labels:
    name: mongo
    app: todo-app
spec:
  type: ClusterIP
  selector:
    app: todo-app
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017

```

4- GKE(Google Kubernetes Engine) cluster oluşturma:

Google Cloud hesabın yoksa oluştur. Bilgisayarına gcloud cli indir. “gcloud auth login” komutu ile bilgisayarından login ol. “gcloud projects create <id>” komutu ile yeni proje oluştur. “gcloud config set project <id>” komutu ile bundan sonraki komutların o proje içinde çalışmasını sağla. “gcloud projects list” komutu ile tüm projelerini görebilirsin. Google Cloud servisleri projeler içinde çalışır. Şimdi bu proje içinde kullanacağımız servisleri aktif hale getirmemiz lazım. Bu üç komut ile kullanacağımız üç servisi aktif hale getiriyoruz:

“gcloud services enable container.googleapis.com”

“gcloud services enable compute.googleapis.com”

“gcloud services enable iam.googleapis.com”

IAM servisini otomatik CI/CD yaparken github actions'da login olmak için kullanıcaz.
"gcloud container clusters create todo-cluster --num-nodes 3 --machine-type e2-small --zone europe-west1-b" komutu ile todo-cluster isminde istediğim bölgeyi, makine tipini ve makine sayısını belirterek bir cluster oluşturdum.

5- GitHub Actions ile CI/CD:

```yaml
name: Docker Image CI-CD

on:
  push:
    branches: ["master"]
  pull_request:
    branches: ["master"]

jobs:
  runMultipleCommands:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}
      - run: docker compose build
      - run: docker compose push
      - run: echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg] https://packages.cloud.google.com/apt cloud-sdk main" | sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list
      - run: curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key --keyring /usr/share/keyrings/cloud.google.gpg add -
      - run: sudo apt-get update
      - run: sudo apt-get install google-cloud-sdk-gke-gcloud-auth-plugin
      - uses: "google-github-actions/auth@v1"
        with:
          credentials_json: "${{ secrets.GOOGLE_CREDENTIALS }}"
      - run: gcloud container clusters get-credentials todo-cluster --zone europe-west1-b
      - run: kubectl delete -R -f ./kubernetes-configs/ --ignore-not-found=true
      - run: kubectl create -R -f ./kubernetes-configs/
```

GitHub Actions ile github reposuna her push yaptığımda o repodan image'ları oluşturup dockerhub'a push ediyor. Daha sonra GKE'de onları canlıya alıyor. Github actions basit olarak bir bilgisayar açıp üstünde reponuzdaki dosyalar ile komut çalıştırmaktan ibaret. Public repolar için tamamen bedava. On birinci satırda yazdığım üzere Ubuntu kullanıyorum. Her step'te uses veya run kullanılmalı. run ile bildiğimiz komut çalışıyor, uses ile github'ın hazır bulundurduğu komut kümeleri. On üçüncü satırdaki github reposundaki dosyaları kullanabilmemizi sağlıyor. On dördüncü satırda dockerhub'a login oluyorum, kullanıcı adı ve token kullanarak. Bunları public repodaki bir dosyaya yazmak tehlikeli olacağından github secrets kullandım. Reponun ayarlarından secret ekleyerek burada kullanabilirsiniz. On sekizinci satırda image'ları build ediyorum. On dokuzuncu satırda image'ları push ediyorum. Yirminci satırdan yirmi üçe kadar gke kullanmak için olması gereken bir plugin'i indiriyorum. Yirmi dördüncü satırda Google Cloud'a login oluyorum credentials ile. Google Cloud'a login olmak için gereken credentials json dosyasını "gcloud iam service-accounts keys create <dosya_ismi> --iam-account=<DEVELOPER_USERNAME>" komutu ile oluşturabilirsiniz. Daha sonra bunu github secrets'a eklemeniz lazım. Yirmi yedinci satırda kubectl'in hedefini todo-cluster yapıyorum. Burada kendi bölge ve cluster isminizi kullanmanız gerek. Yirmi sekizinci satırda cluster'daki eski kaynakları siliyorum. Son satırda ise yenilerini oluşturuyorum.

6- Debug etme:
Bunları yaptıktan sonra uygulamanın çalışmaya başlamış olması lazım. "gcloud container clusters get-credentials todo-cluster --zone europe-west1-b" komutu ile cluster ismini ve bölgeni kullanarak kubectl için config oluştur bilgisayarında. "kubectl get all" komutu ile servislerin, pod'ların durumunu görebilirsin. Eğer birinde sorun varsa "kubectl logs <pod_ismi>" komutu ile pod'un loglarını görüp debug edebilirsin. "kubectl delete <isim>" komutu ile servisleri, deployment'ları veya pod gibi çalışan şeyleri silebilirsin.

7- Siteye giriş:
Client'in external ip'sini kullanarak siteyi görebilirsiniz.

```
C:\Program Files (x86)\Google\Cloud SDK>kubectl get all
NAME                                    READY   STATUS    RESTARTS      AGE
pod/deploy-client-77bfd887b8-w7xx8      1/1     Running   0             18h
pod/deploy-mongo-0                      1/1     Running   0             18h
pod/deploy-server-764c697984-cp967      1/1     Running   1 (18h ago)   18h

NAME                 TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
service/client       LoadBalancer   10.36.8.236    34.78.38.111    80:32683/TCP   18h
service/kubernetes   ClusterIP      10.36.0.1      <none>          443/TCP        25h
service/mongo        ClusterIP      10.36.10.77    <none>          27017/TCP      18h
service/server       ClusterIP      10.36.13.98    <none>          8080/TCP       18h

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/deploy-client     1/1     1            1           18h
deployment.apps/deploy-server     1/1     1            1           18h

NAME                                         DESIRED   CURRENT   READY   AGE
replicaset.apps/deploy-client-77bfd887b8     1         1         1       18h
replicaset.apps/deploy-server-764c697984     1         1         1       18h

NAME                                READY   AGE
statefulset.apps/deploy-mongo       1/1     18h
```