Node, React ve Mongo uygulamasının Dockerize edilip Amazon EKS servisi ile canlıya alınması

Şükrü ÇİRİŞ

Arçelik A.Ş.

1. Başta node backend'ine, react frontend'ine ve mongo database'ine sahip basit todo web uygulamamı yaptım. Projenin tüm kodu ve ilerdeki aşamalarda kullanılan dosyaların bulunduğu github reposu burada: https://github.com/SUKRUCIRIS/node-react-mongodb-kubernetes Basit node uygulamasının EKS ile canlıya alınması ise burada: https://github.com/SUKRUCIRIS/Amazon-EKS-turkish-tutorial Önceki dokümanda anlattığım gerekli rollerin yaratılması gibi bazı şeyleri bu dokümanda tekrar anlatmayacağım.

2. Client tarafında react uygulaması default olarak 3000 portunda çalışır. Server tarafında node uygulaması 8080 portunda çalışacak. React uygulamasını node uygulamasına proxy yapmamız gerekicek client'in server'a erişebilmesi için. Bunun için "package.json" dosyasına "proxy" parametresi ile server'ın adresi eklenmeli. Eğer her şey lokalde çalışacaksa oraya "http://127.0.0.1:8080" yazılmalı. Eğer docker'da container olarak çalışacaksa oraya container ismi ile yazılmalı. Mesela container ismi "server" ise "http://server:8080" yazılmalı. Eğer uygulamayı kubernetes üstünde çalıştırıyorsak oraya kubernetes servisinin ismi ile yazılmalı. Ben kubernetes üzerinde çalıştırıcam. Backend'in çalıştığı kubernetes servisinin ismini "service-np-server" yapıcam o yüzden o parametreyi "http://service-np-server:8080" yaptım.

```
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "proxy": "http://service-np-server:8080",
  "dependencies": {
    "@babel/plugin-proposal-private-property-in-object": "^7.21.11",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0"
```

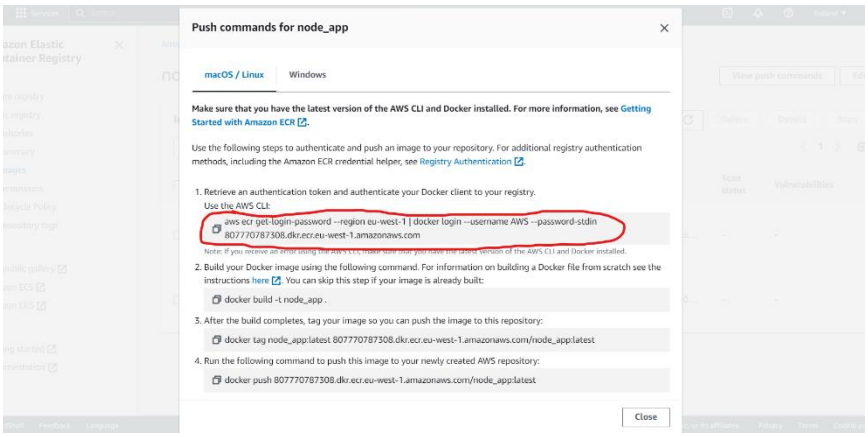
3. Aynı şekilde backend'i mongo database'e bağlarken mongoose aracılığıyla adres vermek gerekecek. İkinci adımda olduğu gibi mongo kubernetes servisinin ismini ve portunu veriyorum. Mongo database'i default olarak 27017 portunu kullanıyor. Uygulamamın kullandığı database ismi "tododb".

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

mongoose.connect("mongodb://service-np-mongo:27017/tododb");

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

4. Amazon ECR servisinde oluşturacağımız frontend, backend ve database imageları için üç tane repository oluşturdum. Docker compose ile image'ları build ettikten sonra Amazon EKS'de kullanmak için oralara yükleyeceğim. Docker'ın çalıştığından ve AWS CLI'nin kurulu ve konfigüre olduğundan emin olduktan sonra bu repolardan birine girip orada gösterilen push command'lerden ilkini çalıştır. Bu komut çalışınca docker ile AWS hesabının repolarına push etme yetkin olacak.



5. Şimdi backend, client ve database için docker dosyalarını ve docker compose dosyasını hazırlamamız lazım. Ayrıca backend ve client için ".dockerignore" dosyaları da koydum node_modules klasörü gibi gereksiz şeyler image'a aktarılmasın diye. Gerekli modüller image'ın içinde komut ile baştan inecek. Backend için docker 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" ]
```

EXPOSE ile kullanacağım portu expose ettim. RUN komutları image yaratılırken çalışıyor. CMD komutları image çalıştırılınca çalışıyor. İkinci ve altıncı satırlardaki komutlar ile uygulamayı production için kurmuş oluyorum.

Frontend için docker 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  CMD ["npm","start"]
```

Database için docker dosyası:

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

Database için sadece resmi base image'ı yüklemek yeterli.

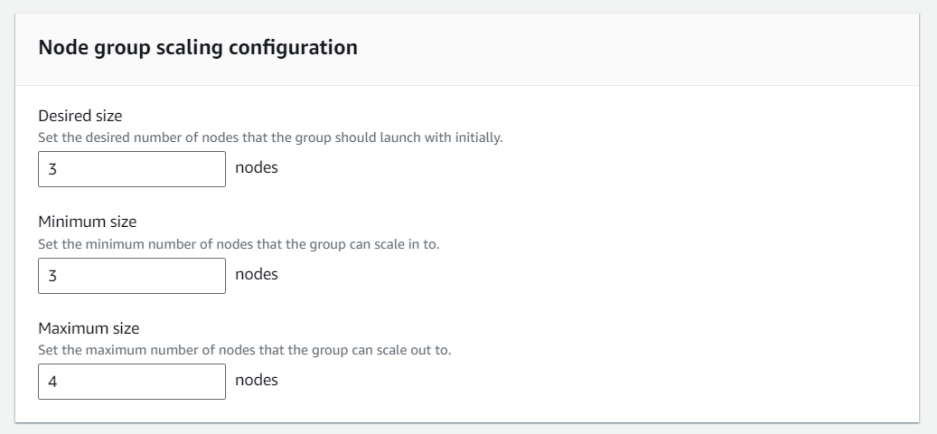
Docker compose dosyası:

```yaml
version: "3.8"
services:
  client:
    build: client
    stdin_open: true
    volumes:
      - ./client:/usr/src/app
      - /usr/src/app/node_modules
    container_name: client
    image: 807770787308.dkr.ecr.eu-west-1.amazonaws.com/todo_client:latest
    restart: always
    depends_on:
      - server
    ports:
      - 3000:3000
    networks:
      - react-express
  server:
    container_name: server
    restart: always
    build: server
    volumes:
      - ./server:/usr/src/app
      - /usr/src/app/node_modules
    depends_on:
      - mongo
    ports:
      - 8080:8080
    networks:
      - express-mongo
      - react-express
    image: 807770787308.dkr.ecr.eu-west-1.amazonaws.com/todo_server:latest
  mongo:
    container_name: mongo
    restart: always
    build: mongo
    volumes:
      - ./data:/data/db
    ports:
      - 27017:27017
    networks:
      - express-mongo
    image: 807770787308.dkr.ecr.eu-west-1.amazonaws.com/todo_mongo:latest
networks:
  react-express:
  express-mongo:

```

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 ECR servisinde oluşturduğum repoların URI'sını latest tagi ile koydum. "container_name" parametresi bu compose dosyası ile container oluşturulursa oluşacak container'ın ismini belirler. Docker compose dosyasının klasöründe "docker compose build" komutunu çalıştır, bittiğinde image'lar oluşmuş olacak. "docker compose push" komutunu çalıştır, bittiğinde image'lar ECR'ye yüklenmiş olacak.

6. Amazon EKS servisinde “todo_cluster” isminde tamamen default ayarlarla bir cluster oluşturdum. Daha sonra bir t3.micro tipinde instance’lar ile görseldeki scaling ayarlarıyla bir node group oluşturdum.



7. Üç image için de kubernetes service ve deployment dosyalarını hazırladım.

   client_deploy.yml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-client
  labels:
    name: testing-kube-deploy
    app: testing-kube-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: testing-kube-pod
      app: testing-kube-app
  template:
    metadata:
      name: testing-kube-pod
      labels:
        name: testing-kube-pod
        app: testing-kube-app
    spec:
      containers:
        - name: testing-kube-container
          image: "807770787308.dkr.ecr.eu-west-1.amazonaws.com/todo_client:latest"
          imagePullPolicy: Always
          ports:
            - containerPort: 3000

```

   Image parametresi için client’in kullandığı port olmalı, image parametresi ECR’deki 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 çalışıcak demek. Dördüncü satırdaki deployment ismi unique olmalı.

client_service.yml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: service-np-client
  labels:
    name: testing-kube-deploy
    app: testing-kube-app
spec:
  type: NodePort
  selector:
    app: testing-kube-app
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
      nodePort: 30001

```

Dokuzuncu satırda belirtildiği üzere bu servisin tipi “NodePort” yani dış ağa açık. “nodePort” parametresi ile servisin dışa açılan portu ayarlanır, ben 30001 yaptım. “port” ve “targetPort” parametresi için client’in kullandığı port olan 3000 değerini verdim.

mongo_deploy.yml:

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: deploy-mongo
  labels:
    name: testing-kube-deploy
    app: testing-kube-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: testing-kube-pod
      app: testing-kube-app
  template:
    metadata:
      name: testing-kube-pod
      labels:
        name: testing-kube-pod
        app: testing-kube-app
    spec:
      containers:
        - name: testing-kube-container
          image: "807770787308.dkr.ecr.eu-west-1.amazonaws.com/todo_mongo:latest"
          imagePullPolicy: Always
          ports:
            - containerPort: 27017

```

Burada yine image, containerPort ve name parametresini özelleştirdim.

mongo_service.yml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: service-np-mongo
  labels:
    name: testing-kube-deploy
    app: testing-kube-app
spec:
  type: ClusterIP
  selector:
    app: testing-kube-app
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017

```

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

server_deploy.yml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-server
  labels:
    name: testing-kube-deploy
    app: testing-kube-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: testing-kube-pod
      app: testing-kube-app
  template:
    metadata:
      name: testing-kube-pod
      labels:
        name: testing-kube-pod
        app: testing-kube-app
    spec:
      containers:
        - name: testing-kube-container
          image: "807770787308.dkr.ecr.eu-west-1.amazonaws.com/todo_server:latest"
          imagePullPolicy: Always
          ports:
            - containerPort: 8080

```

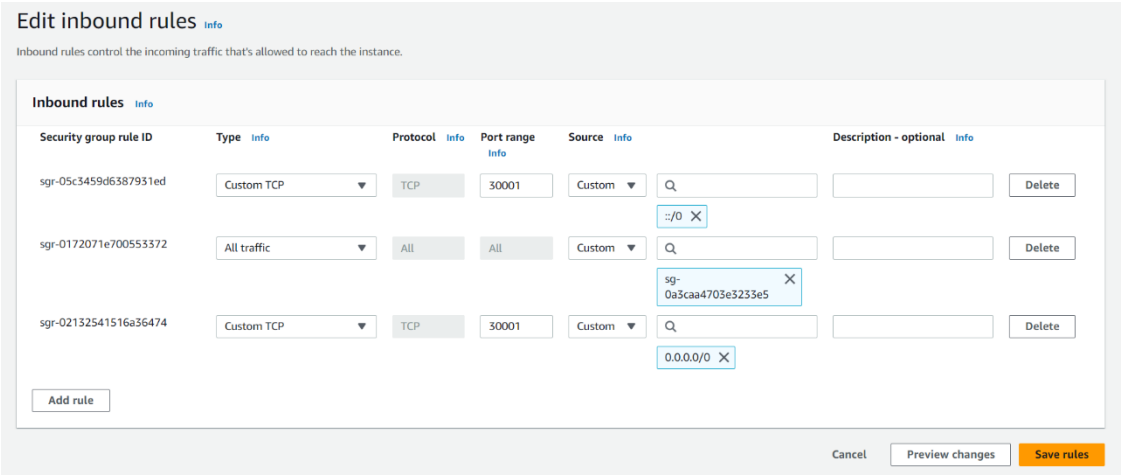
server_service.yml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: service-np-server
  labels:
    name: testing-kube-deploy
    app: testing-kube-app
spec:
  type: ClusterIP
  selector:
    app: testing-kube-app
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080

```

“aws eks update-kubeconfig --name <cluster_ismi>” komutunu çalıştırarak otomatik olarak kubectl için config oluştur. Oluşturduğumuz kubernetes dosyalarının olduğu klasörde aşağıdaki komutları çalıştır:
“kubectl create -f mongo_deploy.yml”
“kubectl create -f mongo_service.yml”
“kubectl create -f server_deploy.yml”
“kubectl create -f server_service.yml”
“kubectl create -f client_deploy.yml”
“kubectl create -f client_service.yml”
Bunları yaptıktan sonra uygulamanın sorunsuz çalışmaya başlamış olması lazım. "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.

8. EC2 servisine gidip cluster’ına ait instance’ların security group’una client servisi’nin node portuna (30001) sahip custom TCP ipv4 ve ipv6 için iki tane inbound rule ekle.



9. Node grubuna ait EC2 servisindeki instance’lardan biri için http://<Public-IPv4-DNS>:30001 url’sine gittiğimde uygulamanın çalıştığını gördüm.