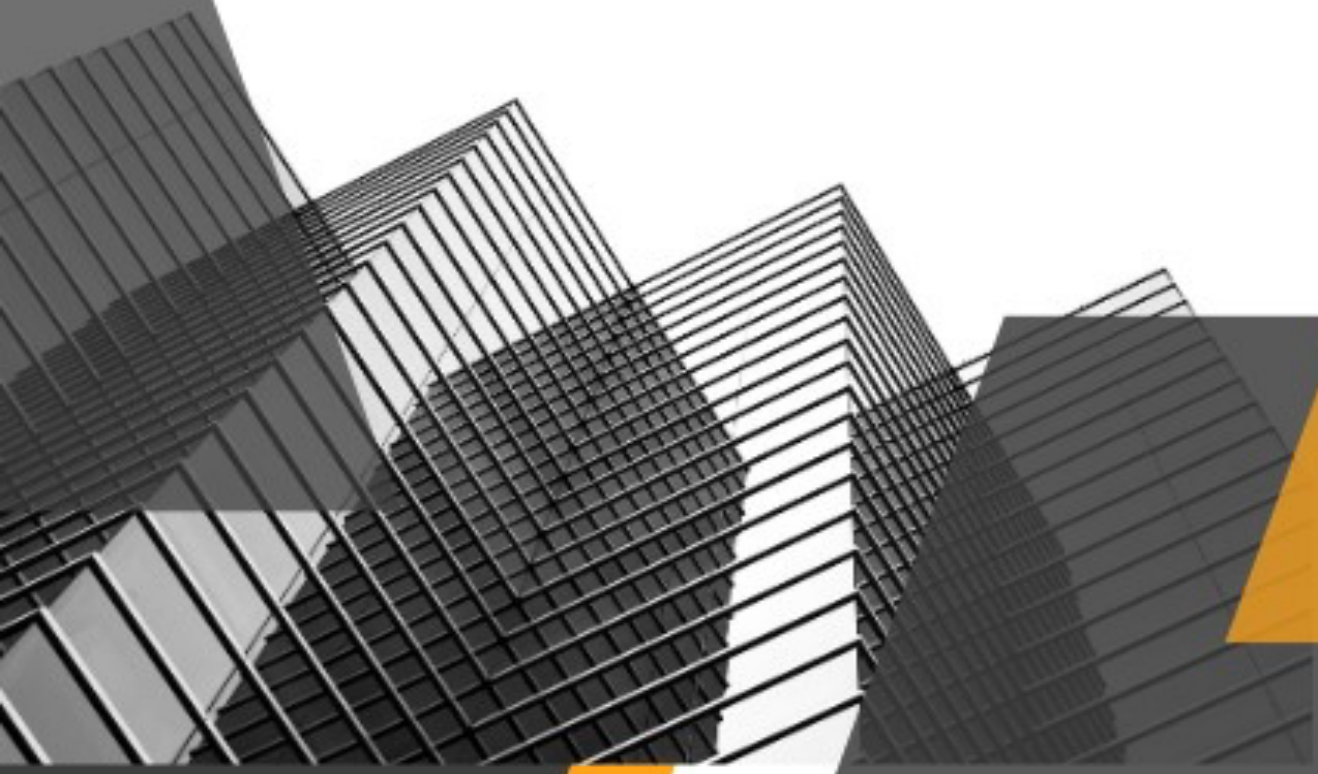
S1 INFORMATIKA

Pembangunan Perangkat Lunak

Orientasi Service



2025

FAKULTAS ILMU KOMPUTER
PEMBANGUNAN NASIONAL VETERAN

LEMBAR PENGESAHAN

Modul
Pembangunan Perangkat Lunak Orientasi Service

Oleh:
Muhammad Panji Muslim, S.Pd., M.Kom
NIP. 199308312022031007

Modul ini disusun sebagai pedoman dan acuan dalam pelaksanaan
Perangkat Lunak Orientasi Service Semester VI
Program Studi S1 Informatika, Fakultas Ilmu Komputer
Universitas Pembangunan Nasional "Veteran" Jakarta

Disahkan pada tanggal 10 Maret 2025

Koordinator Program Studi S1 Informatika

Dr. Ridwan Raafiudin, S.Kom., M.Kom

KATA PENGANTAR

Alhamdulillahirabbil'alamin. Puji dan syukur senatiasa penyusun panjatkan kehadirat Allah SWT yang telah melimpahkan rahmat dan karunia-Nya sehingga penyusun dapat menyelesaikan Modul Praktikum Perangkat Lunak Orientasi Service sebagai pusat kegiatan belajar mahasiswa. Modul ini juga dilengkapi dengan Langkah-langkah dari Implementasi Service menggunakan beberapa tahapan API.

Penyusun menyadari masih banyak kekurangan dalam penyusunan modul ini. Oleh karena itu, kami sangat mengharap kritik dan saran demi perbaikan dan optimalisasi modul ini.

Penyusun mengucapkan terima kasih kepada berbagai pihak yang telah membantu proses penyelesaian modul ini. Semoga modul ini dapat bermanfaat bagi kita semua, khususnya para mahasiswa Service Oriented Informatika.

Jakarta, 10 Oktober 2025

Penyusun

MATERI XV
Orkestrasi Kubernetes

Topik:

1. Men-deploy layanan menggunakan Docker untuk isolasi dan deployment yang efisien.
2. Membuat **Dockerfile** untuk layanan berbasis Node.js
3. Dan Melakukan Orkestrasi menggunakan kubernetes

Mengapa Orkestrasi Penting?

1. Pengelolaan Skala Besar: Dalam sistem modern, aplikasi sering kali dibagi menjadi banyak *microservices*. Orkestrasi memastikan semua layanan berjalan harmonis.
2. Penskalaan Otomatis: Kubernetes dapat menambah atau mengurangi jumlah *instances* layanan berdasarkan kebutuhan.
3. Pemulihan Kesalahan: Jika ada kontainer yang gagal, Kubernetes secara otomatis menggantinya dengan yang baru.
4. Distribusi Beban: Kubernetes mengatur distribusi lalu lintas ke berbagai layanan agar beban kerja merata.
5. Efisiensi Sumber Daya: Dengan orkestrasi, penggunaan CPU, memori, dan jaringan dapat dioptimalkan.

Komponen Orkestrasi Kubernetes

1. Node:
   - *Master Node:* Bertugas mengelola kluster Kubernetes.
   - *Worker Node:* Tempat kontainer dijalankan.
2. Pod: Unit terkecil di Kubernetes, berisi satu atau beberapa kontainer.
3. Deployment: Mengelola penyebaran aplikasi, seperti menentukan jumlah replika (*replica sets*).
4. Service: Memberikan akses ke pod melalui alamat IP tetap, meskipun pod tersebut berubah.
5. Scheduler: Memastikan setiap pod dijalankan pada node yang sesuai berdasarkan kebutuhan sumber daya.
6. Controller: Memastikan bahwa kluster berjalan sesuai konfigurasi (contoh: mempertahankan jumlah replika).
7. API Server: Mengelola komunikasi antara pengguna/CLI dengan kluster Kubernetes.

Proses Orkestrasi di Kubernetes

1. Penyebaran Aplikasi: Pengguna membuat konfigurasi (file YAML/JSON) untuk mendefinisikan bagaimana aplikasi harus dijalankan.
2. Penjadwalan Pod: Kubernetes Scheduler menentukan di mana pod harus dijalankan berdasarkan ketersediaan sumber daya.
3. Penskalaan Aplikasi: Deployment menentukan jumlah replika yang dibutuhkan dan memastikan layanan selalu tersedia.
4. Load Balancing: Service mendistribusikan lalu lintas ke semua pod secara merata.
5. Pemulihan Kesalahan: Jika pod mengalami kegagalan, Kubernetes otomatis memulai ulang pod baru.

Contoh Orkestrasi

Misalnya, aplikasi Node.js berjalan dengan 3 replika (*instances*) untuk menangani lalu lintas

pengguna. Dengan Kubernetes:

1. Jika beban lalu lintas meningkat, Kubernetes dapat meningkatkan replika menjadi 5.
2. Jika salah satu replika gagal, Kubernetes akan memulai ulang pod baru secara otomatis.
3. Melalui Service, Kubernetes mengatur distribusi lalu lintas agar semua replika mendapat beban yang merata.

Implementasi Orkestrasi dengan Kubernetes dengan membuat fullstack (menggabungkan Frontend dan Backend)

Langkah Awal sebagai berikut:
Langkah Awal Donwload Kubernet disini dan Download Minikubes untuk lokal kubernetes disini.

Lalu apabila telah terinstall di komputer cek instalasi dengan menjalankan perintah diterminal. Untuk taruh File Instalasi di PATH rekan rekan, dan lalu cek dengan command **"kubectl version –client"**

```
C:\Users\Muhammad Panji>kubectl version --client
Client Version: v1.31.0
Kustomize Version: v5.4.2
```

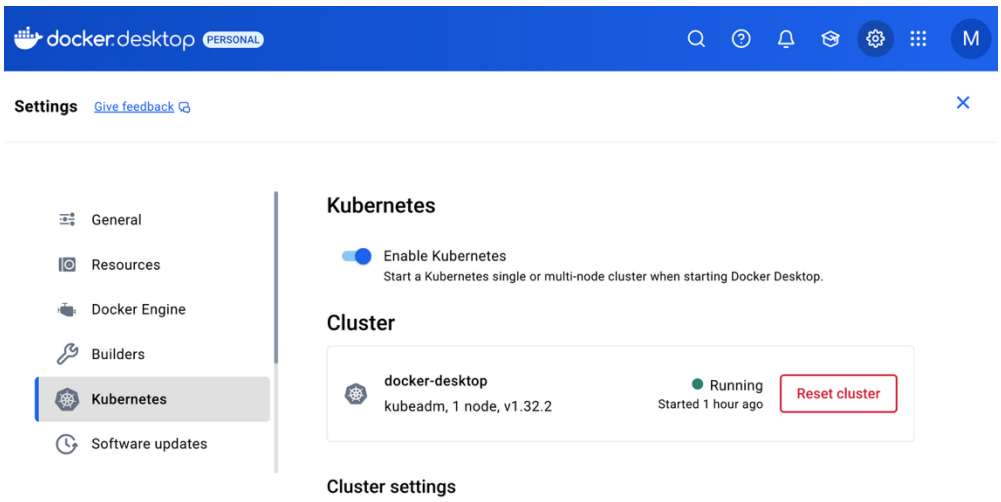
Gambar 1. Cek Kubernetes

Begitu pula apabila telah melakukan instalasi minikube, maka cek instalasi untuk mengecek Kubernetes local (minikube) sudah berjalan dengan baik di device rekan rekan dengan command berikut:

```
C:\Users\Muhammad Panji>minikube version
minikube version: v1.34.0
commit: 210b148df93a80eb872ecbeb7e35281b3c582c61
```

Gambar 2. Cek Minikube

Atau bisa rekan rekan ke Docker Hub menyalakan Kubernetes yang sudah ada di Docker Hub, dengan cara ke Setting lalu ke Tab Kubernetes, Jika Kubernetes belum terinstall Docker Hub akan merekomendasikan Apply dan Restart untuk instalasi dan Running Kubernetes



Gambar 3. Docker Hub

Jika berhasil Kubernetes terinstall dan running di docker hub rekan rekan maka akan ada notif Kubernetes running, atau bisa di cek menggunakan command berikut:

```
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack % kubectl version --client
Client Version: v1.32.2
Kustomize Version: v5.5.0
```

Gambar 4. Cek Instalasi

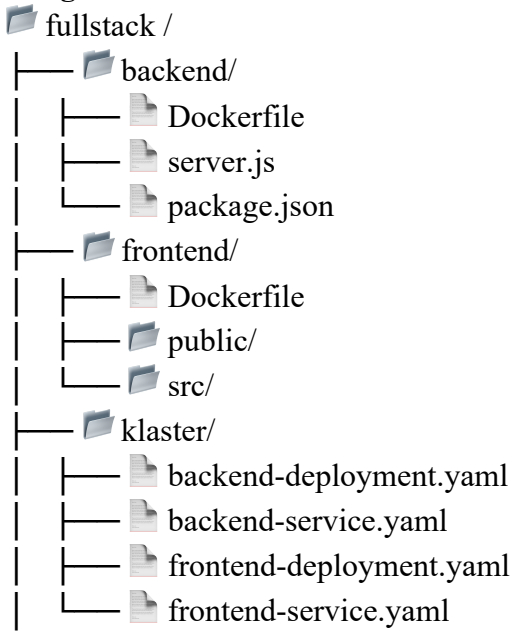
Langkah selanjutnya Inisialisasi Project Node

Buat folder baru untuk project dan inisialisasi Node project dengan nama **"fullstack"** maka akan secara otomatis melakukan instalasi project Node dengan nama **fullstack**

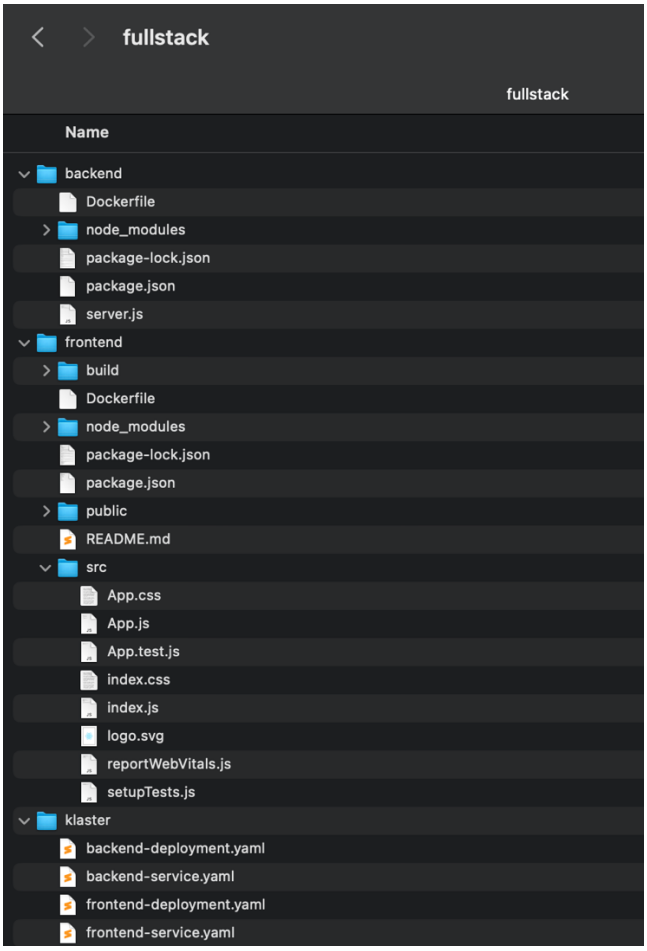
```
muhammadpanji@Muhammads-MacBook-Pro-4 ~ % mkdir fullstack
muhammadpanji@Muhammads-MacBook-Pro-4 ~ % cd fullstack
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack %
```

Gambar 5. Buat Project fullstack

Dengan Direktori Folder



Gambar 6. Direktori Folder



Gambar 7. Tampilan Direktori File

Langkah 3. Set Backend

Buat folder backend (mkdir biasa) lalu konfigurasi seperti biasa (npm init -y) dan (npm install express) dan Membuat Server menggunakan Express.js dengan menggunakan nama file server.js dengan script sebagai berikut:

```
const express = require('express');
const app = express();
const PORT = 3011;
const cors = require('cors');

app.use(cors());

app.get('/api', (req, res) => {
  res.json({ message: 'tes backend di kubernetes' });
});

app.listen(PORT, () => console.log(`Server jalan pada port
${PORT}`));
```

Selanjutnya update file backend/package.json dengan script:
package.json:

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^5.1.0"
  }
}
```

Buat File backend/Dockerfile

```
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3011
CMD ["node", "server.js"]
```

Langkah 4. Set Frontend (React Js)
Dalam membuat folder frontend tidak bisa menggunakan mkdir biasa, yang rekan rekan lakukan adalah menggunakan, command untuk membuat project react
"npm create-react-app frontend"

Lalu tunggu sebentar sampai endline menunjukkan "Happy Hacking". Maka file frontend react telah berhasil terbuat. Jika berhasil terbuat terdapat beberapa file yang rekan-rekan akan update dan buat, antara lain:

frontend/src/App.js

```
import { useEffect, useState } from 'react';

function App() {
  const [data, setData] = useState('');

  useEffect(() => {
    fetch('http://localhost:30011/api')
      .then(res => res.json())
      .then(json => setData(json.message))
      .catch(err => console.error("API error:", err));
  }, []);

  return (
    <div>
      <h1>Tes Frontend</h1>
      <p>coba tes dari backend: {data}</p>
    </div>
  );
}

export default App;
```

Lalu buatlah file frontend/Dockerfile

```
FROM node:18 as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
```

Update pada frontend/package.json (dengan menambahkan proxy: dengan backend-service:3011 pada line 5)

```
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "proxy": "http://backend-service:3011",
  "dependencies": {
    "@testing-library/dom": "^10.4.0",
    "@testing-library/jest-dom": "^6.6.3",
    "@testing-library/react": "^16.3.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^19.1.0",
    "react-dom": "^19.1.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Jika sudah, build dan push backend dan frontend rekan rekan pada docker dengan command berikut:

1. **Build and Push Backend to Docker Hub**

```
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack % docker build -t muhammadpanjiii/backend:v1 ./ba
ckend
[+] Building 1.2s (10/10) FINISHED                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                        0.0s
 => => transferring dockerfile: 149B                                        0.0s
 => [internal] load metadata for docker.io/library/node:18                  1.1s
 => [internal] load .dockerignore                                           0.0s
 => => transferring context: 2B                                             0.0s
 => [1/5] FROM docker.io/library/node:18@sha256:c6ae79e38498325db67193d     0.0s
 => [internal] load build context                                           0.0s
 => => transferring context: 40.09kB                                        0.0s
 => CACHED [2/5] WORKDIR /app                                               0.0s
 => CACHED [3/5] COPY package*.json ./                                      0.0s
 => CACHED [4/5] RUN npm install                                            0.0s
 => CACHED [5/5] COPY . .                                                   0.0s
 => exporting to image                                                      0.0s
 => => exporting layers                                                     0.0s
 => => writing image sha256:f64fe632665dee127ca6bf42233e63671e658de2ae0     0.0s
 => => naming to docker.io/muhammadpanjiii/backend:v1                       0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/2s0441a2k3hbk5ja
sn6pibvvw
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack % docker push muhammadpanjiii/backend:v1
The push refers to repository [docker.io/muhammadpanjiii/backend]
192859a67f87: Pushed
0d62f33e3d30: Pushed
5707c70a2b7b: Pushed
a71f28766dc4: Pushed
e78159dbd370: Pushed
a358a725b813: Pushed
cd8a6003174c: Pushed
abb63e49e652: Pushed
6cc65bdde70e: Pushed
41a4e3939504: Pushed
3520c50ae60e: Pushed
75ba6634710f: Pushed
v1: digest: sha256:2a4075a2cde32054e3a97ee883c426e449da3a604da7c54df217e4f11512b962 size: 2839
```

Gambar 8. Build dan Push Backend

2. Build and Push Frontend to Docker Hub

```
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack % docker build -t muhammadpanjiii/frontend:v1 ./frontend
[+] Building 39.2s (16/16) FINISHED                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                        0.0s
 => => transferring dockerfile: 220B                                        0.0s
 => WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (     0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine             3.8s
 => [internal] load metadata for docker.io/library/node:18                  2.4s
 => [auth] library/node:pull token for registry-1.docker.io                 0.0s
 => [auth] library/nginx:pull token for registry-1.docker.io                0.0s
 => [internal] load .dockerignore                                           0.0s
 => => transferring context: 2B                                             0.0s
 => [build 1/6] FROM docker.io/library/node:18@sha256:c6ae79e38498325db     0.0s
 => [stage-1 1/2] FROM docker.io/library/nginx:alpine@sha256:65645c7bb6     6.7s
 => => resolve docker.io/library/nginx:alpine@sha256:65645c7bb6a0661892     0.0s
 => => sha256:65645c7bb6a0661892a8b03b89d0743208a18dd 10.33kB / 10.33kB     0.0s
 => => sha256:63ffc0d1f14e4082b832c6a42e606e9a0384a526f 2.50kB / 2.50kB     0.0s
 => => sha256:c60e446e49a0b607fd79968afdc54cedce6764499 1.79MB / 1.79MB     1.2s
 => => sha256:96868d9fa38f469a86d2f25787e43ee9ad33033 10.80kB / 10.80kB     0.0s
 => => sha256:6e771e15690e2fabf2332d3a3b744495411d6e0b0 3.99MB / 3.99MB     0.9s
 => => sha256:e6557c42ebeaea010d8a8883fdacdc5a17dea1221416d 627B / 627B     1.1s
 => => extracting sha256:6e771e15690e2fabf2332d3a3b744495411d6e0b00b2ae     0.2s
 => => sha256:d3282d7e6b7633cf153dce0ca1c72b6ea574edb0b3435 957B / 957B     1.3s
 => => sha256:a4ce1202d74643d4e4ce15787afc2402a18802e3d7bfd 405B / 405B     1.4s
 => => extracting sha256:c60e446e49a0b607fd79968afdc54cedce67644990a393     0.1s
 => => sha256:37aca2470cdaf48d251d9308c09dc8735b45ebcbe 1.40kB / 1.40kB     1.6s
 => => sha256:1ab010a063387e697fc32bb43022532c0e275d2e5 1.21kB / 1.21kB     1.6s
 => => sha256:9994ea1088e3f1d0eb3dea855f32e7e63742b26 16.03MB / 16.03MB     5.2s
 => => extracting sha256:e6557c42ebeaea010d8a8883fdacdc5a17dea1221416d0     0.0s
 => => extracting sha256:d3282d7e6b7633cf153dce0ca1c72b6ea574edb0b34351     0.0s
 => => extracting sha256:a4ce1202d74643d4e4ce15787afc2402a18802e3d7bfd0     0.0s
 => => extracting sha256:1ab010a063387e697fc32bb43022532c0e275d2e51fb65     0.0s
 => => extracting sha256:37aca2470cdaf48d251d9308c09dc8735b45ebcbefada4     0.0s
 => => extracting sha256:9994ea1088e3f1d0eb3dea855f32e7e63742b2644c8611     1.4s
 => [internal] load build context                                           4.8s
 => => transferring context: 287.00MB                                       4.4s
 => CACHED [build 2/6] WORKDIR /app                                         0.0s
 => [build 3/6] COPY package*.json ./                                       1.6s
 => [build 4/6] RUN npm install                                            13.8s
 => [build 5/6] COPY . .                                                    9.5s
 => [build 6/6] RUN npm run build                                           4.8s
 => [stage-1 2/2] COPY --from=build /app/build /usr/share/nginx/html        0.0s
 => exporting to image                                                      0.0s
 => => exporting layers                                                     0.0s
 => => writing image sha256:24023dfc7713f3ced409c68b843a8e1d1e1de8c6096     0.0s
 => => naming to docker.io/muhammadpanjiii/frontend:v1                      0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/vqbuyrxq6rhcusxs0q36sr688

 1 warning found (use docker --debug to expand):
 - FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 1)
```

```
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack % docker push muhammadpanjiii/frontend:v1
The push refers to repository [docker.io/muhammadpanjiii/frontend]
18ffcb3b84eb: Pushed
96a0d9b1e80c: Mounted from library/nginx
97ca58bb468d: Mounted from library/nginx
4a651ac2c218: Mounted from library/nginx
0fe7bdfe639d: Mounted from library/nginx
c32437183174: Mounted from library/nginx
9aad78ecf380: Mounted from library/nginx
3459dc8f926a: Mounted from library/nginx
a16e98724c05: Mounted from library/nginx
v1: digest: sha256:27b2ebe1dba97ea4ba3071ecf62fe80b9fcdc553606a523f4e46da6ad085e8af size: 2199
```

Gambar 9. Build dan Push Frontend

Jika ingin melihat hasil push nya bisa ke docker hub

☐	○	muhammadpanji v1	f64fe632665d	2 hours ago	1.09 GB ▷ ⋮
☐	○	muhammadpanji v1	24023dfc7713	1 hour ago	50.81 MB ▷ ⋮

Gambar 10. Docker Hub

Dari Kontainer Backend dan Frontend ini mari kita buat satu agar bisa menjadi satu kesatuan (klaster) menggunakan Kubernetes

Langkah 5. Buat File yaml untuk kebutuhan Kubernetes

Klaster/backend-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: muhammadpanjiii/backend:v1
        ports:
        - containerPort: 3011
```

Klaster/backend-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  type: NodePort
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 3011
      targetPort: 3011
      nodePort: 30011
```

Klaster/frontend-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: muhammadpanjiii/frontend:v1
        ports:
        - containerPort: 80
```

Klaster/frontend-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
    nodePort: 30080
  type: NodePort
```

Langkah 6. Selanjutnya eksekusi atau buat Klaster menggunakan comman Kuberenetesnya (kubectl apply -f folder/file)

```
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack % kubectl apply -f klaster/backend-deployment.yaml
deployment.apps/backend created
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack % kubectl apply -f klaster/backend-service.yaml
service/backend-service created
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack % kubectl apply -f klaster/frontend-deployment.yaml
deployment.apps/frontend created
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack % kubectl apply -f klaster/frontend-service.yaml
service/frontend-service created
```

Gambar 11. Klaster Kubernetes

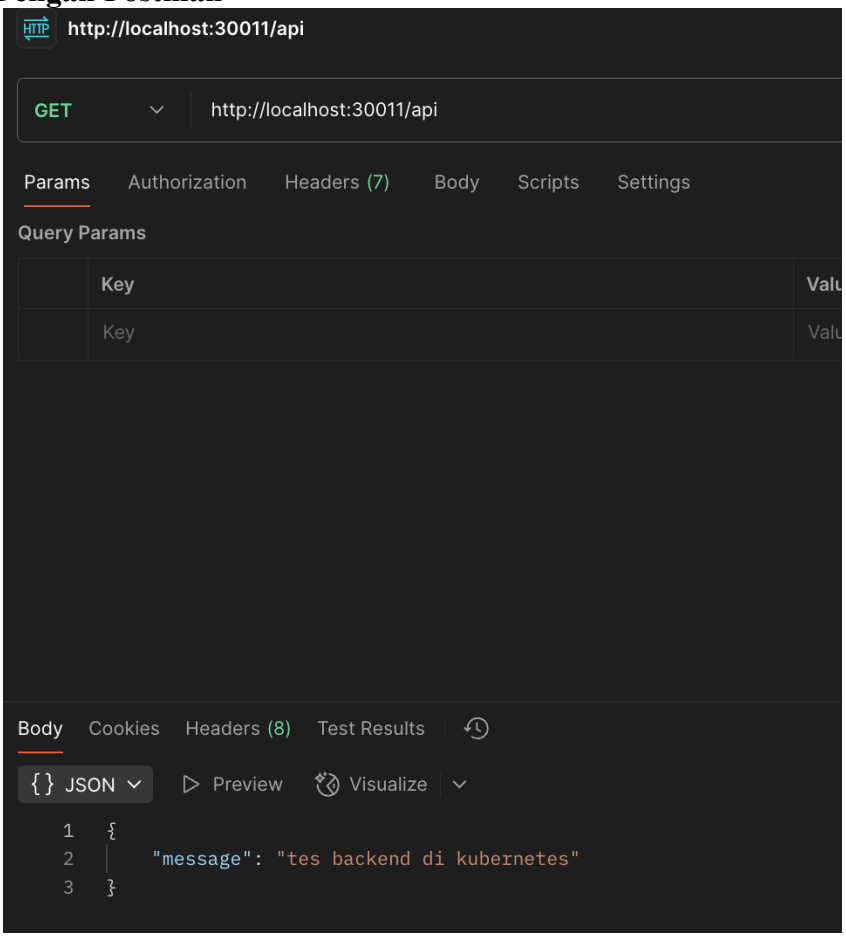
Lalu cek port dan service yang terbentuk dengan command kubectl get svc

```
muhammadpanji@Muhammads-MacBook-Pro-4 fullstack % kubectl get svc
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
backend-service    NodePort    10.108.96.53    <none>        3011:30011/TCP   13m
frontend-service   NodePort    10.97.69.133    <none>        80:30080/TCP     40m
kubernetes         ClusterIP   10.96.0.1       <none>        443/TCP          82m
```

Gambar 12. Cek Port dan Cluster IP (get service)

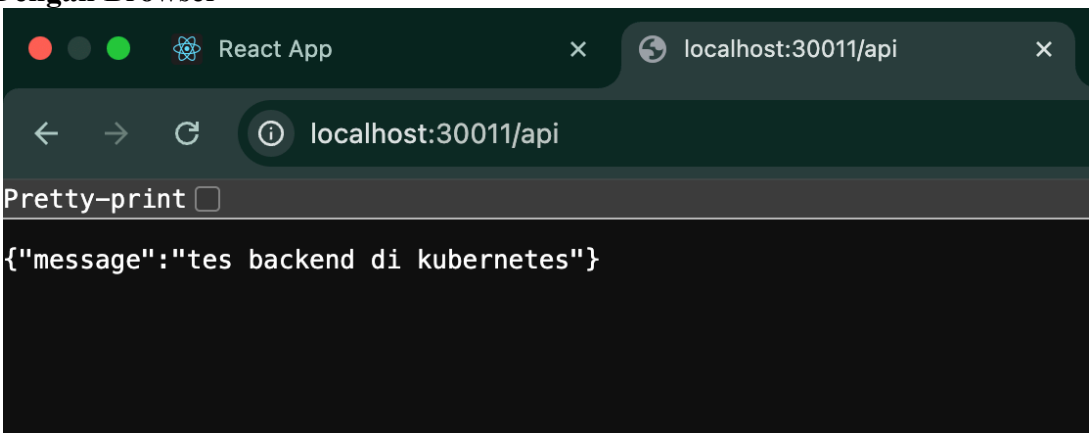
Langkah 9. Menguji Backend dan Frontend yang sudah terklaster dan terhubung

1. **Dengan Postman**



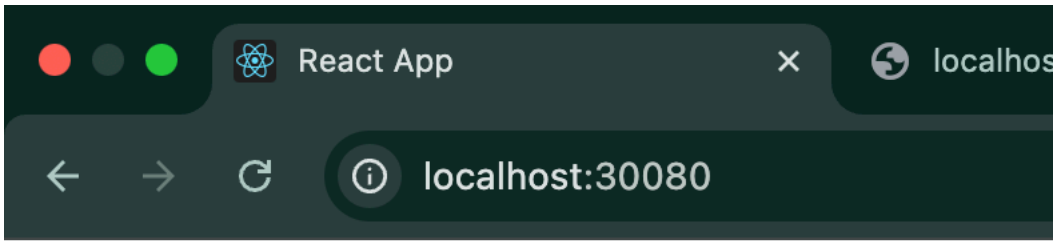
Gambar 13. Uji Postman (Backend)

2. **Dengan Browser**



Gambar 14. Uji Browser (Backend)

Lalu Uji Frontend yang sudah terhubung dengan Backend dengan klaster yang terbentuk dengan link : localhost:30080



Tes Frontend

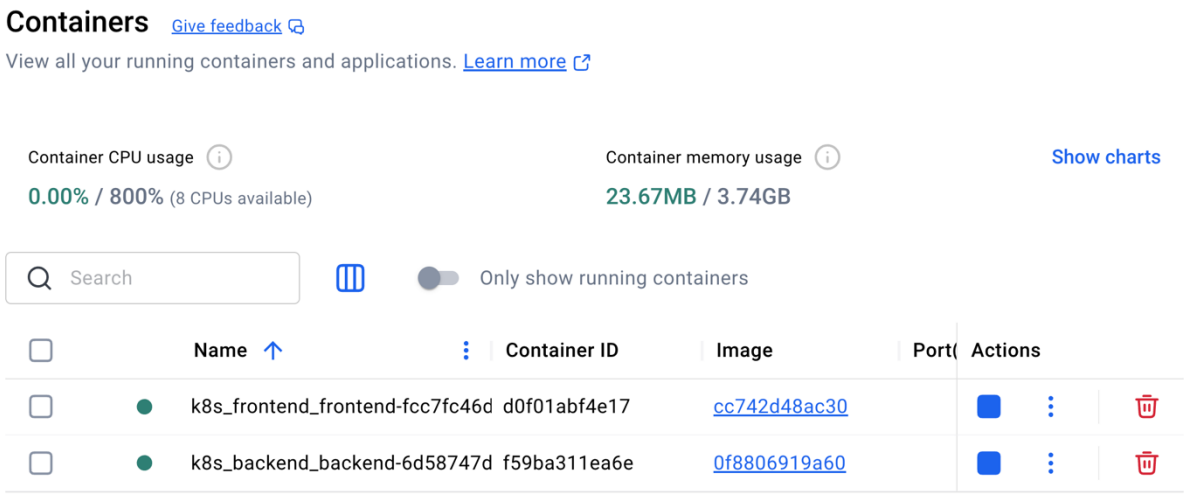
coba tes dari backend: tes backend di kubernetes

Gambar 15. Uji Fullstack (Frontend dan Backend)

Langkah Selanjutnya untuk mengecek service yang aktif menggunakan command berikut:

```
NAME                        READY   STATUS    RESTARTS   AGE
backend-6d58747dbd-lk6td    1/1     Running   0          62m
frontend-fcc7fc46d-scbpx    1/1     Running   0          71m
```

Gambar 16. Get pods



Gambar 17. Container Docker GUI

Perbandingan Docker Compose dan Kubernetes

Fitur	Docker Compose	Kubernetes
Skalabilitas	Cocok untuk pengembangan dan aplikasi kecil	Dapat menangani aplikasi besar dengan skala tinggi
Manajemen Layanan	Mengelola beberapa container dengan satu file	Mengelola ribuan container dan layanan secara otomatis
Keamanan	Menyediakan kontrol dasar untuk isolasi	Menyediakan kontrol keamanan dan otentikasi lebih canggih
Penggunaan	Mudah digunakan untuk aplikasi kecil hingga menengah	Lebih kompleks, cocok untuk aplikasi besar dan enterprise
Pengelolaan Lalu Lintas	Tidak memiliki load balancing otomatis	Built-in load balancing dan manajemen beban
Orkestrasi	Tidak ada orkestrasi otomatis	Orkestrasi otomatis dengan kemampuan scaling dan recovery

Dengan menggunakan Docker Compose atau Kubernetes, rekan-rekan dapat memilih solusi orkestrasi yang tepat berdasarkan kebutuhan aplikasi rekan-rekan apakah aplikasi tersebut masih dalam tahap pengembangan atau sudah membutuhkan skalabilitas dan manajemen yang lebih kompleks.

Selamat Bereksplorasi