CS571 Group 3 Project Report

Members: Sangmin Lee (20253), Ayoyimika Folashade Ajibade, Nang Thiri Wutyi (20113), Shahriar Fahim (20361)

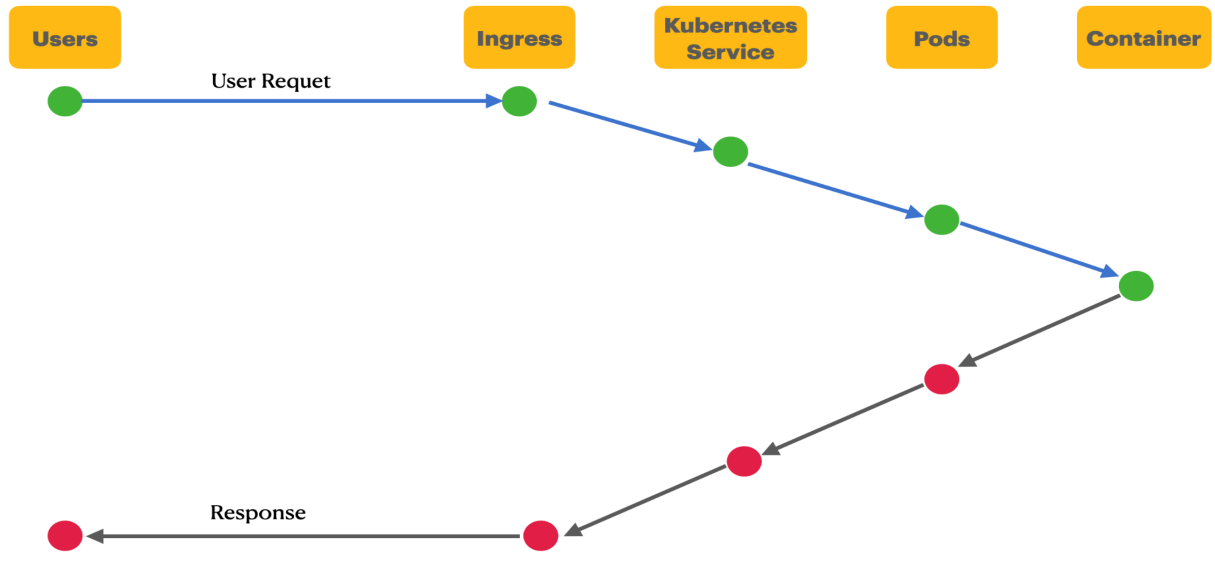
GitHub repository: https://github.com/AyoyimikaAjibade/Task-Mgt-App?tab=readme-ov-file
(Note: All dockerfiles, .dockerignore, yaml, yml files are not included in this git repository. However, you can find all detail at Dockerization and Orchestration Instruction part below)

System Design & Architecture

1. System Architecture Overview:

- Frontend: React.js, interacting with the backend via REST API.
- Backend: NestJS, handling task management operations.
- Database: PostgreSQL (hosted on Cloud SQL, GCP).
- Containerization: Docker containers for the backend and frontend.
- Orchestration: Kubernetes (GKE) manages deployment, scaling, and networking.
- Load Balancing: Kubernetes Ingress Controller distributes traffic.
- Monitoring: Google Cloud Logging, Grafana for performance tracking.

2. Sequence Diagram



3. Deployment Strategy

- Containerization:
    - The application is packaged in Docker containers.
    - Images are stored in DockerHub for deployment.
- Kubernetes Orchestration:
    - Deployments & Services manage API instances.
    - ConfigMaps & Secrets store environment variables securely.
    - Horizontal Pod Autoscaler (HPA) adjusts replicas based on CPU load.
- Load Balancing & Traffic Management:
    - Kubernetes Ingress Controller handles routing across backend instances.
    - Supports SSL termination for security.
- Monitoring & Performance Optimization:
    - Google Cloud Logging & Monitoring track logs and metrics.
    - Grafana visualizes system health and resource consumption.

Dockerization and Orchestration Instruction

1. Install git in VM

sudo apt update
sudo apt install git -y

2. Pull repo from GitHub

it clone https://<username>:<PAT>@github.com/AyoyimikaAjibade/Task-Mgt-App.git
(Note: you need PAT. Please search Google how to get it)

3. Create **Dockerfile** and .**dockerignore** inside of frontend and backend directory

NOTE: No Dockerfile and .dockerignore for db since it use its official image

Frontend at frontend directory
Dockerfile

```
# Stage 1: build React app
FROM node:20-alpine AS build

WORKDIR /app
COPY package*.json ./
RUN npm install --frozen-lockfile
COPY . .
RUN npm run build

# Stage 2: serve with Nginx
FROM nginx:alpine

COPY --from=build /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
```

.dockerignore

```
node_modules
dist
build
.env
```

```
Dockerfile
.dockerignore
*.log
```

Add .env.production file to connect with API and REACT at frontend directory
.env.production

```
REACT_APP_API_URL=/api
```

Backend at backend directory
Dockerfile

```
  GNU nano 7.2                                  Dockerfile
# Use official Node.js image as the base
FROM node:20-alpine

# Set the working directory
WORKDIR /app

# Copy package files and install dependencies
COPY package*.json ./
RUN npm ci||  upn install

# Copy the rest of the backend source code
COPY . .

# Build the NestJS app
RUN npm run build

# Expose the application port (adjust if different)
EXPOSE 3001

# Start the NestJS application
CMD ["npm", "run", "start:prod"]
```

.dockerignore

```
node_modules
dist
.dockerignore
Dockerfile
```

```
.env
*.log
```

4. GCP authentication and make cluster

gcloud auth login

gcloud services enable container.googleapis.com

sudo apt-get install google-cloud-cli-gke-gcloud-auth-plugin

sudo apt-get install kubectl

```
gcloud container clusters create task-mgt-cluster \
  --num-nodes=1 \
  --zone=us-central1-a
```

gcloud container clusters get-credentials task-mgt-cluster --zone us-central1-a

5. Build docker images and push them.

In my example, repository name: task-mgt-app / region - us-central1

Backend

docker build -t gcr.io/YOUR_PROJECT_ID/task-mgt-app/backend .
gcloud auth configure-docker us-central1-docker.pkg.dev
docker push gcr.io/YOUR_PROJECT_ID/task-mgt-backend

Frontend

docker build -t gcr.io/YOUR_PROJECT_ID/task-mgt-app/frontend .
gcloud auth configure-docker us-central1-docker.pkg.dev
docker push gcr.io/YOUR_PROJECT_ID/task-mgt-frontend

6. For deployment

Create Development.yml and Service.yml file for frontend, backend, and db

frontend-deployment.yml - in frontend directory

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 2
```

```
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: <YOUR IMAGE >
          ports:
            - containerPort: 80
```

backend-deployment.yml- in backend directory

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          image: us-central1-docker.pkg.dev/tranquil-lore-449501-j8/task-mgt-app/backend
          ports:
            - containerPort: 3001
          env:
            - name: DB_HOST
              value: postgres
            - name: DB_PORT
              value: "5432"
            - name: DB_USERNAME
              value: taskadmin
            - name: DB_PASSWORD
              value: strongpassword
            - name: DB_NAME
              value: task_mgt_db
            - name: JWT_SECRET
```

```
        value: jwtverysecretkey571
```

postgres-deployment.yml- in Task_Mgt_App directory

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:15
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_DB
              value: task_mgt_db
            - name: POSTGRES_USER
              value: taskadmin
            - name: POSTGRES_PASSWORD
              value: strongpassword
          volumeMounts:
            - name: postgres-data
              mountPath: /var/lib/postgresql/data
      volumes:
        - name: postgres-data
          emptyDir: {}
```

frontend-service.yml - in frontend directory

```
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
```

```
  selector:
    app: frontend
  ports:
    - port: 80
      targetPort: 80
  type: LoadBalancer
```

backend-service.yml- in backend directory

```
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
    - port: 3001
      targetPort: 3001
  type: ClusterIP
```

postgres-service.yml- in TMA directory

```
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  selector:
    app: postgres
  ports:
    - port: 5432
  type: ClusterIP
```

Make nginx.conf at frontend directory

```
#nginx.conf
server {
  listen 80;

  location / {
    root /usr/share/nginx/html;
```

```
    index index.html index.htm;
    try_files $uri $uri/ /index.html;
  }

  location /api/ {
    proxy_pass http://backend-service:3001/;
    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;
  }
}
```

Make nodePort.yml at frontend directory

```
#nodePort.yml
apiVersion: v1
kind: Service
metadata:
  name: frontend-nodeport
spec:
  type: NodePort
  selector:
    app: frontend
  ports:
    - port: 80             # Port inside the cluster (container port)
      targetPort: 80       # Port your frontend app listens on
      nodePort: 30080      # External port exposed on nodes
```

Run -

kubectl apply -f frontend-deployment.yml
kubectl apply -f frontend-service.yml

kubectl apply -f backend-deployment.yml
kubectl apply -f backend-service.yml

kubectl apply -f postgres-deployment.yml
kubectl apply -f postgres-service.yml

kubectl apply -f nodePort.yml

To check service status: (Make sure your pods and services are running)

kubectl get pods
kubectl get services

7. Set up firewall:

```
gcloud compute firewall-rules create allow-nodeport-30080 \
  --allow tcp:30080 \
  --direction=INGRESS \
  --priority=1000 \
  --network=default \
  --source-ranges=0.0.0.0/0
```

8. Test from local machine

Note 1 : Set up the firewall first
Run gcloud compute instances list

NOTE 2 : Make sure you are using 30080 port
Ex: 34.16.53.104:**30080**

9. Set up HPA

Make sure you installed metrics server:
kubectl get deployment metrics-server -n kube-system

If not:
kubectl apply -f
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

Create backend-hpa.yaml in the backend directory

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: backend-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: backend
  minReplicas: 2
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
```

```
    type: Utilization
    averageUtilization: 70
```

Run: kubectl apply -f backend-hpa.yaml

Check: kubectl get hpa

10. Debugging

To check Pods status - kubectl get pods
To check service status - kubectl get services
To check logs - kubectl logs -f deploy/backend / kubectl logs -f deploy/frontend

If you don't have enough space in VM, this command will delete unused images
docker system prune -af
docker volume prune -f
df -h

11. (Optional) Rebuild docker image and Redepoly

It's already in GitHub repository, so you can use with commands below
This is a sh file to help you to rebuild docker image and redeploy with one command.
NOTE: Create sh file at Task-mgt-app directory, not backend or frontend directory.
—----------------------------—----------------------------
nano build-frontend.sh

```
#!/bin/bash

PROJECT_ID="tranquil-lore-449501-j8"
REGION="us-central1"
REPO="task-mgt-app"
IMAGE_NAME="frontend"
BACKEND_URL="http://backend-service:3001"
IMAGE_URI="$REGION-docker.pkg.dev/$PROJECT_ID/$REPO/$IMAGE_NAME"

echo "🧼 Cleaning..."
rm -rf ./frontend/build ./frontend/dist

echo "🐳 Building frontend..."
docker build --no-cache \
  --build-arg REACT_APP_API_URL=$BACKEND_URL \
  -t $IMAGE_URI ./frontend

echo "📦 Pushing to registry..."
docker push $IMAGE_URI
```

```
echo "🚀 Restarting frontend deployment..."
kubectl rollout restart deployment frontend

echo "✅ Frontend deployed!"
```

chmod +x build-frontend.sh

Run sh file: ./build-frontend.sh

Log: kubectl logs -f deploy/frontend

———————————————————————————————————

nano build-backend.sh

```
#!/bin/bash

PROJECT_ID="tranquil-lore-449501-j8"
REGION="us-central1"
REPO="task-mgt-app"
IMAGE_NAME="backend"
IMAGE_URI="$REGION-docker.pkg.dev/$PROJECT_ID/$REPO/$IMAGE_NAME"

echo "🧼 Cleaning..."
rm -rf ./backend/dist

echo "🐳 Building backend..."
docker build --no-cache -t $IMAGE_URI ./backend

echo "📦 Pushing to registry..."
docker push $IMAGE_URI

echo "🚀 Restarting backend deployment..."
kubectl rollout restart deployment backend

echo "✅ Backend deployed!"
```

chmod +x build-backend.sh

Run sh file: ./build-backend.sh
Log : kubectl logs -f deploy/backend

Load Balancing with Ingress

(GKE Ingress Controller & Let's Encrypt -HTTPS)

1. Set up ingress controller:

To enable the NGINX Ingress Controller
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.9.4/deploy/static/provider/cloud/deploy.yaml

Install cert-manager for Let's Encrypt
kubectl apply -f
https://github.com/cert-manager/cert-manager/releases/download/v1.13.3/cert-manager.yaml

Get ingress external IP and point a domain:
kubectl get svc ingress-nginx-controller -n ingress-nginx

2. Create cluster-issuer.yaml and ingress.yaml files

```
# cluster-issuer.yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    email: your-email@example.com
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: letsencrypt-prod-private-key
    solvers:
    - http01:
        ingress:
          class: nginx
```

```
# ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
```

```
metadata:
  name: task-mgt-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  tls:
  - hosts:
    - taskmgtapp.duckdns.org
    secretName: taskapp-tls
  rules:
  - host: taskmgtapp.duckdns.org
    http:
      paths:
      - path: /()(.*)
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
      - path: /api(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: backend-service
            port:
              number: 3001
```

Run:
kubectl apply -f cluster-issuer.yaml
kubectl apply -f ingress.yaml

3. Debugging

1. After you set up the domain (or if you didn't set up domain), the domain address will send you to different site:

   nslookup taskapp.yourdomain.com
   kubectl get svc ingress-nginx-controller -n ingress-nginx

   Run the above command and make sure that the ingress external IP is matched. If there are not same, you need to update DNS A record to point to the ingress external IP

If you need to set up DNS, use DuckDNS - Free DNS

2. If your certificate is not working, it means HTTPS is not working and you will see a “Not Secure” sign.

   Run to check your certificates and check if there are any failed event
   kubectl get certificate
   kubectl describe certificate taskmgtapp-tls