Final Project (Tugas 5) Komputasi Awan

Kubernetes in Docker (KIND and K3D)

Anggota Kelompok

- Gloriyano Cristho Daniel Pepuho (5025201121)
- Yehezkiel Wiradhika (5025201086)
- Muhammad Dzakwan (5027201065)

Alamat Repository
https://github.com/danielcristho/k3d-kind-exploration

Pendahuluan

Dalam era modernisasi teknologi, kontainerisasi menjadi salah satu solusi utama dalam pengelolaan aplikasi yang skalabel, portabel, dan efisien. Kubernetes, sebagai salah satu platform orkestrasi container terpopuler, menawarkan kemudahan dalam mengatur dan skalabilitas aplikasi berbasis kontainer. Namun, untuk memahami dan mengimplementasikan Kubernetes secara efektif, diperlukan lingkungan yang fleksibel dan mudah dikonfigurasi. Kubernetes in Docker (KIND) hadir sebagai alat yang memungkinkan pengguna untuk membuat cluster Kubernetes lokal menggunakan container Docker. Alat ini sangat bermanfaat untuk pengembangan, pengujian, dan simulasi lingkungan Kubernetes tanpa memerlukan infrastruktur fisik yang kompleks.

Dalam arsitektur Kubernetes, K3s bisa dimanfaatkan sebagai distribusi Kubernetes ringan untuk mengatur orkestrasi aplikasi Flask, PostgreSQL, dan Nginx. K3D menjalankan cluster Kubernetes sepenuhnya di dalam container Docker, sementara K3s berfungsi untuk provisioning pod dan service dengan overhead minimal. Flask berperan sebagai aplikasi backend yang berinteraksi dengan PostgreSQL untuk operasi database, dan Nginx sebagai reverse proxy yang menerima permintaan HTTP dari klien serta meneruskannya ke Flask. Dengan K3s pada KIND, deployment menjadi lebih sederhana dan efisien, karena cluster Kubernetes dikelola dalam container Docker dengan pengaturan yang mendukung skalabilitas, portabilitas, serta testing di lingkungan lokal sebelum diterapkan ke cluster yang lebih besar.

Tugas

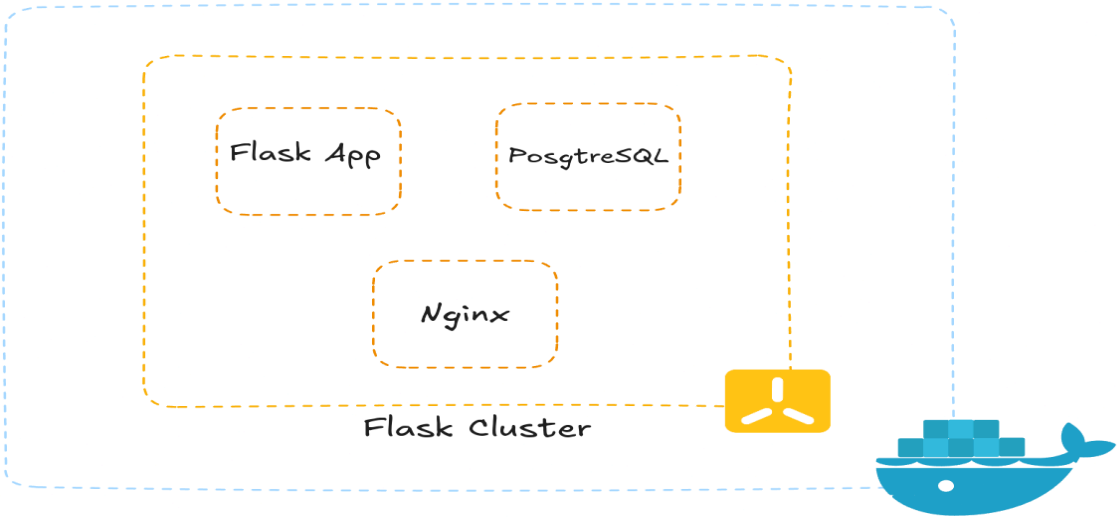
Berdasarkan pada https://github.com/rm77/cloud2023/tree/master/kubernetes

- Pada repo yang tersedia, cluster menggunakan KIND (kubernetes in docker) yang bisa dijalankan pada satu komputer saja, tapi jika bisa menggunakan jenis kubernetes yang lain
- kembangkan project studi kasus dengan kreasi anda sendiri, untuk men-deploy container di lingkungan kubernetes, container bisa menggunakan hasil dari tugas sebelumnya atau membuat baru tergantung dari situasi yang anda buat

- (optional) Kembangkan suatu platform Continuous deployment, yang membuat deployment ke cluster kubernetes bisa otomatis, dengan menggunakan scripting
- Berikan penjelasan tentang kapan skenario kreasi anda penting/cocok untuk dilakukan
- Buatlah gambar arsitektur
- Buatlah script yang mendukung
- Tunjukkan dashboard dari cluster yang dibuat
- Berikan capture screenshot dan penjelasan

Penjelasan

Gambar Arsitektur



Arsitektur ini menggunakan **K3s**, distribusi Kubernetes yang ringan, di dalam Docker untuk provisioning aplikasi berbasis Flask. Komponen utama terdiri dari **Flask App** sebagai backend, **PostgreSQL** sebagai database, dan **Nginx** sebagai reverse proxy untuk menangani permintaan HTTP dari klien. K3s mengelola orkestrasi komponen ini di dalam cluster Kubernetes, memungkinkan interaksi terorganisir antara Flask App, PostgreSQL, dan Nginx. Semua komponen dijalankan dalam container Docker, sehingga mempermudah deployment, portabilitas, dan isolasi aplikasi. Dengan pendekatan ini, aplikasi dapat diskalakan secara otomatis dan efisien, meskipun memberikan overhead tambahan pada resource di lingkungan lokal. Arsitektur ini ideal untuk pengujian awal aplikasi berbasis Flask sebelum deployment ke cluster Kubernetes yang lebih besar.

Penjelasan Kode

$ up-cluster.sh

```
#!/bin/env bash

k3d cluster create --config manifests/create-config.yaml

kubectl label node k3d-flask-todo-agent-0 role=lb && \
kubectl label node k3d-flask-todo-agent-1 role=app && \
kubectl label node k3d-flask-todo-agent-2 role=app && \
kubectl label node k3d-flask-todo-agent-3 role=db
```

manifests/create-config.yaml

```
apiVersion: k3d.io/v1alpha5
kind: Simple
metadata:
  name: flask-todo
servers: 1
agents: 4
image: rancher/k3s:v1.30.4-k3s1
ports:
- port: 30000-30100:30000-30100
  nodeFilters:
  - server:*
options:
  k3s:
    extraArgs:
    - arg: --disable=traefik
      nodeFilters:
      - server:*
```

Script `create-config.yaml` adalah konfigurasi untuk membuat cluster Kubernetes menggunakan **k3d**. Cluster diberi nama `flask-todo` dan terdiri dari 1 server node dan 4 agent nodes menggunakan image Kubernetes ringan `rancher/k3s:v1.30.4-k3s1`. Konfigurasi ini membuka rentang port 30000-30100 di node server untuk mendukung akses layanan melalui NodePort. Pada bagian `options`, **Traefik**, ingress controller bawaan K3s, dinonaktifkan melalui argumen `--disable=traefik`, sehingga memberikan fleksibilitas untuk menggunakan alat atau proxy lain, seperti Nginx, untuk manajemen trafik. Config ini memungkinkan pengaturan cluster yang optimal untuk aplikasi Flask-Todo.

```
→ ./up-cluster.sh
INFO[0000] Using config file manifests/create-config.yaml (k3d.io/v1alpha5#
INFO[0000] Prep: Network
INFO[0000] Created network 'k3d-flask-todo'
INFO[0000] Created image volume k3d-flask-todo-images
INFO[0000] Starting new tools node...
INFO[0000] Starting node 'k3d-flask-todo-tools'
INFO[0001] Creating node 'k3d-flask-todo-server-0'
INFO[0001] Creating node 'k3d-flask-todo-agent-0'
INFO[0001] Creating node 'k3d-flask-todo-agent-1'
INFO[0001] Creating node 'k3d-flask-todo-agent-2'
INFO[0001] Creating node 'k3d-flask-todo-agent-3'
INFO[0001] Creating LoadBalancer 'k3d-flask-todo-serverlb'
INFO[0001] Using the k3d-tools node to gather environment information
INFO[0001] HostIP: using network gateway 172.22.0.1 address
INFO[0001] Starting cluster 'flask-todo'
INFO[0001] Starting servers...
INFO[0001] Starting node 'k3d-flask-todo-server-0'
INFO[0006] Starting agents...
INFO[0006] Starting node 'k3d-flask-todo-agent-0'
INFO[0006] Starting node 'k3d-flask-todo-agent-3'
INFO[0006] Starting node 'k3d-flask-todo-agent-2'
INFO[0006] Starting node 'k3d-flask-todo-agent-1'
INFO[0015] Starting helpers...
INFO[0015] Starting node 'k3d-flask-todo-serverlb'
INFO[0026] Injecting records for hostAliases (incl. host.k3d.internal) and
INFO[0028] Cluster 'flask-todo' created successfully!
INFO[0028] You can now use it like this:
kubectl cluster-info
node/k3d-flask-todo-agent-0 labeled
node/k3d-flask-todo-agent-1 labeled
node/k3d-flask-todo-agent-2 labeled
node/k3d-flask-todo-agent-3 labeled
```

Script `up-cluster.sh` digunakan untuk membuat dan mengatur cluster Kubernetes berbasis **k3d**—alat untuk menjalankan cluster **K3s** di dalam Docker. Perintah pertama, `k3d cluster create --config manifests/create-config.yaml`, membuat cluster berdasarkan konfigurasi yang ditentukan dalam file `create-config.yaml`. Setelah cluster dibuat, script melabeli node dalam cluster untuk mendefinisikan peran masing-masing node: `k3d-flask-todo-agent-0` diberi label `role=lb` sebagai load balancer, `k3d-flask-todo-agent-1` dan `k3d-flask-todo-agent-2` masing-masing dilabeli `role=app` untuk menjalankan aplikasi Flask, dan `k3d-flask-todo-agent-3` diberi label `role=db` untuk menjalankan database PostgreSQL. Label ini mempermudah pengelolaan node berdasarkan fungsi mereka dalam arsitektur aplikasi.

deployment/flask-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-todo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-todo
  template:
    metadata:
      labels:
        app: flask-todo
    spec:
      nodeSelector:
        role: app
      containers:
      - name: flask-todo
        image: danielcristh0/flask-todo:1.1
        ports:
        - containerPort: 5000
        env:
        - name: FLASK_ENV
          value: "production"
        - name: DATABASE_URL
          value: "postgresql://admin:dbPassword@db:5432/todo"
---
```

(gambar dipotong untuk menghemat tempat, lebih lengkapnya ada di: https://github.com/danielcristho/k3d-kind-exploration/blob/main/k3d-flask-postgres/deployment/flask-deployment.yaml)

Script `flask-deployment.yaml` mendefinisikan **Deployment** dan **Service** untuk aplikasi Flask dalam cluster Kubernetes. **Deployment** ini bernama `flask-todo` dengan 2 replika pod yang menjalankan image Docker `danielcristh0/flask-todo:1.1`. Pod dijadwalkan pada node yang memiliki label `role: app` menggunakan `nodeSelector`. Container di setiap pod berjalan pada port 5000 dan memiliki variabel lingkungan `FLASK_ENV` untuk mode produksi serta `DATABASE_URL` untuk koneksi ke database PostgreSQL. Bagian **Service** membuat resource bernama `flask-todo` dengan tipe `ClusterIP`, yang memungkinkan komunikasi antar-pod melalui port 5000 menggunakan label `app: flask-todo` sebagai selektor untuk mengarahkan trafik ke pod yang sesuai.

deployment/nginx-configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  default.conf: |
    upstream flask {
        server flask-todo:5000;
    }

    server {
        listen 8082;

        location / {
            proxy_pass http://flask;
        }
    }
```

Script `nginx-configmap.yaml` mendefinisikan sebuah **ConfigMap** bernama `nginx-config` untuk menyimpan konfigurasi Nginx dalam cluster Kubernetes. File konfigurasi `default.conf` di dalamnya mengatur sebuah upstream bernama `flask`, yang mengarahkan trafik ke service Flask pada `flask-todo:5000`. Server Nginx diatur untuk mendengarkan pada port 8082 dan meneruskan semua permintaan yang datang ke endpoint root (`/`) menuju upstream `flask` menggunakan `proxy_pass`. ConfigMap ini biasanya digunakan untuk menyediakan konfigurasi Nginx yang dinamis ke container Nginx tanpa memodifikasi image Docker-nya.

deployment/nginx-deployment.yaml

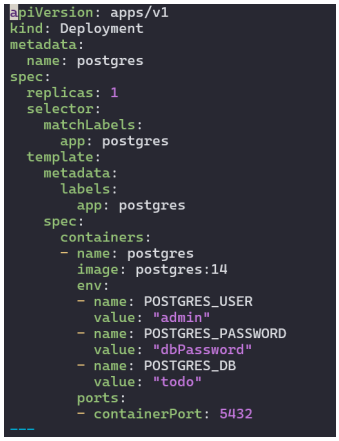
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      nodeSelector:
        role: lb
      containers:
      - name: nginx
        image: nginx:1.19.2-alpine
        ports:
        - containerPort: 8082
        resources:
          requests:
            memory: "64Mi"
            cpu: "100m"
          limits:
            memory: "128Mi"
            cpu: "200m"
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx/conf.d/default.conf
          subPath: default.conf
```

(gambar dipotong untuk menghemat tempat, lebih lengkapnya ada di: https://github.com/danielcristho/k3d-kind-exploration/blob/main/k3d-flask-postgres/deployment/nginx-deployment.yaml)

Script `nginx-deployment.yaml` mendefinisikan **Deployment** dan **Service** untuk Nginx dalam cluster Kubernetes. **Deployment** ini menjalankan 1 replika pod dengan image

`nginx:1.19.2-alpine`, dijadwalkan pada node berlabel `role: lb` menggunakan `nodeSelector`. Pod mendengarkan pada port 8082, dan container Nginx menggunakan konfigurasi yang disediakan oleh **ConfigMap** `nginx-config`, yang di-mount ke `/etc/nginx/conf.d/default.conf`. Sumber daya container dibatasi dengan request 64Mi memori dan 100m CPU, serta limit 128Mi memori dan 200m CPU. Bagian **Service** membuat resource bertipe `LoadBalancer`, memungkinkan Nginx menerima trafik eksternal pada port 8082 dan meneruskannya sesuai konfigurasi proxy-nya.

deployment/postgres-deployment.yaml



```yaml
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:14
        env:
        - name: POSTGRES_USER
          value: "admin"
        - name: POSTGRES_PASSWORD
          value: "dbPassword"
        - name: POSTGRES_DB
          value: "todo"
        ports:
        - containerPort: 5432
---
```

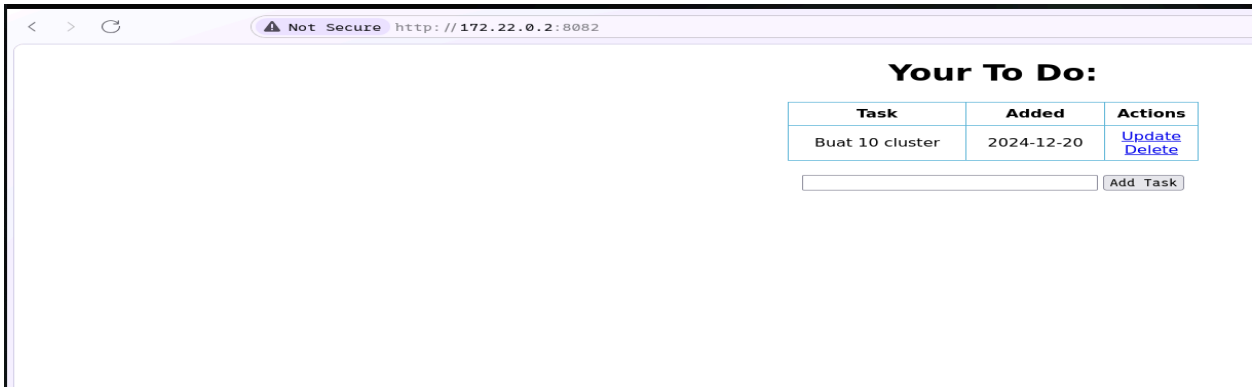
(gambar dipotong untuk menghemat tempat, lebih lengkapnya ada di: https://github.com/danielcristho/k3d-kind-exploration/blob/main/k3d-flask-postgres/deployment/postgres-deployment.yaml)

Script `postgres-deployment.yaml` mendefinisikan **Deployment** dan **Service** untuk PostgreSQL dalam cluster Kubernetes. **Deployment** ini menjalankan 1 replika pod menggunakan image `postgres:14`. Container PostgreSQL diatur dengan variabel lingkungan `POSTGRES_USER`, `POSTGRES_PASSWORD`, dan `POSTGRES_DB` untuk mengonfigurasi pengguna, kata sandi, dan nama database (`todo`). Container mendengarkan pada port 5432 untuk koneksi database. Bagian **Service** mendefinisikan resource bertipe `ClusterIP`, yang memungkinkan akses ke PostgreSQL secara internal di cluster Kubernetes melalui nama service `db` pada port 5432. Hal ini mempermudah aplikasi lain, seperti Flask, untuk terhubung ke database.

```
⎈ k3d-flask-todo in k3s-flask-postgres/k3d-flask-postgres on  main [!]
→ ./create-deployment.sh
deployment.apps/postgres created
service/db created
deployment.apps/flask-todo created
service/flask-todo created
configmap/nginx-config created
deployment.apps/nginx created
service/nginx created

⎈ k3d-flask-todo in k3s-flask-postgres/k3d-flask-postgres on  main [!]
→ kubectl get pods
NAME                          READY   STATUS              RESTARTS   AGE
flask-todo-67db5b7fbd-jvqk8   0/1     ContainerCreating   0          9s
flask-todo-67db5b7fbd-l77hg   0/1     ContainerCreating   0          9s
nginx-fd54c6fff-5gpkx         0/1     ContainerCreating   0          9s
postgres-5797cf68b9-p2km2     0/1     ContainerCreating   0          9s

⎈ k3d-flask-todo in k3s-flask-postgres/k3d-flask-postgres on  main [!]
→ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
flask-todo-67db5b7fbd-jvqk8   1/1     Running   0          5m1s
flask-todo-67db5b7fbd-l77hg   1/1     Running   0          5m1s
nginx-fd54c6fff-5gpkx         1/1     Running   0          5m1s
postgres-5797cf68b9-p2km2     1/1     Running   0          5m1s

⎈ k3d-flask-todo in k3s-flask-postgres/k3d-flask-postgres on  main [!]
→
```

Hasil deployment, akses webserver menggunakan EXTERNAL-IP dari LoadBalancer. Lalu jika sudah kita bisa menambahkan data, yang mana nantinya data tersebut akan tersimpan ke database.

```
→ kubectl get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP                                                  PORT(S)          AGE
db           ClusterIP      10.43.191.205   <none>                                                       5432/TCP         9m36s
flask-todo   ClusterIP      10.43.211.20    <none>                                                       5000/TCP         9m36s
kubernetes   ClusterIP      10.43.0.1       <none>                                                       443/TCP          10m
nginx        LoadBalancer   10.43.49.180    172.22.0.2,172.22.0.3,172.22.0.4,172.22.0.5,172.22.0.6       8082:31951/TCP   9m35s
```



```
⎈ k3d-flask-todo in k3s-flask-postgres/k3d-flask-postgres on  main [!]
❯ kubectl exec -it postgres-5797cf68b9-sc2sc -- psql -U admin  -d todo
psql (14.15 (Debian 14.15-1.pgdg120+1))
Type "help" for help.

todo=# \dt
       List of relations
 Schema | Name | Type  | Owner
--------+------+-------+-------
 public | todo | table | admin
(1 row)

todo=# select * from todo;
 id |      content      |         date_created
----+-------------------+-------------------------------
  1 | Buat 10 cluster | 2024-12-20 08:52:11.225342
(1 row)

todo=#
```

Kind-cases

Create cluster

```
ster/kind$ sudo sh 1-create-cluster.sh
Creating cluster "mylab99" ...
 ⠈⠁ Ensuring node image (kindest/node:v1.32.0) 🖼
 ✓ Ensuring node image (kindest/node:v1.32.0) 🖼
 ✓ Preparing nodes 📦 📦 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-mylab99"
You can now use your cluster with:

kubectl cluster-info --context kind-mylab99

Have a nice day! 👋
```

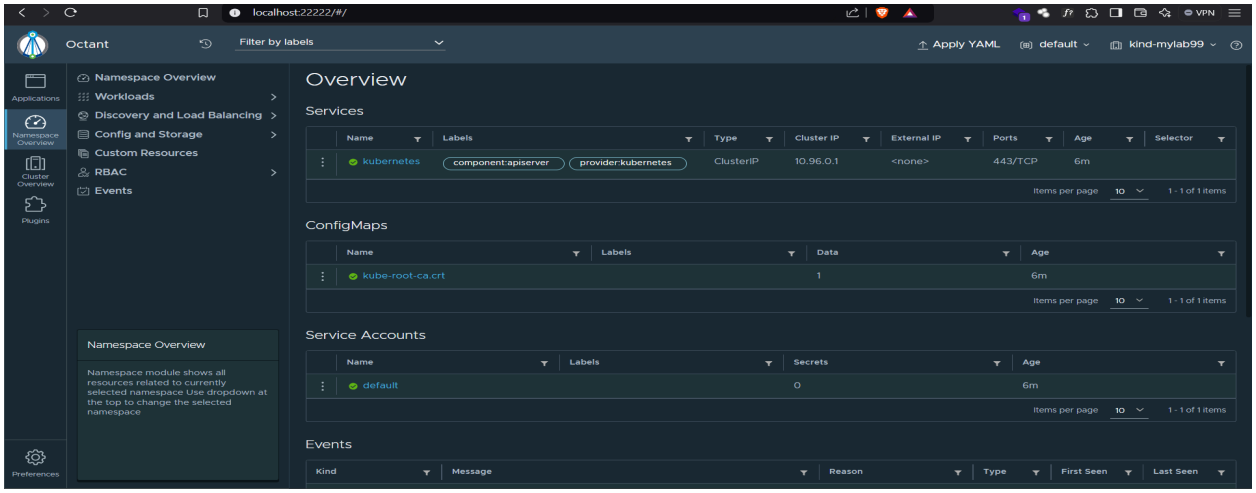
Set config & install ingress

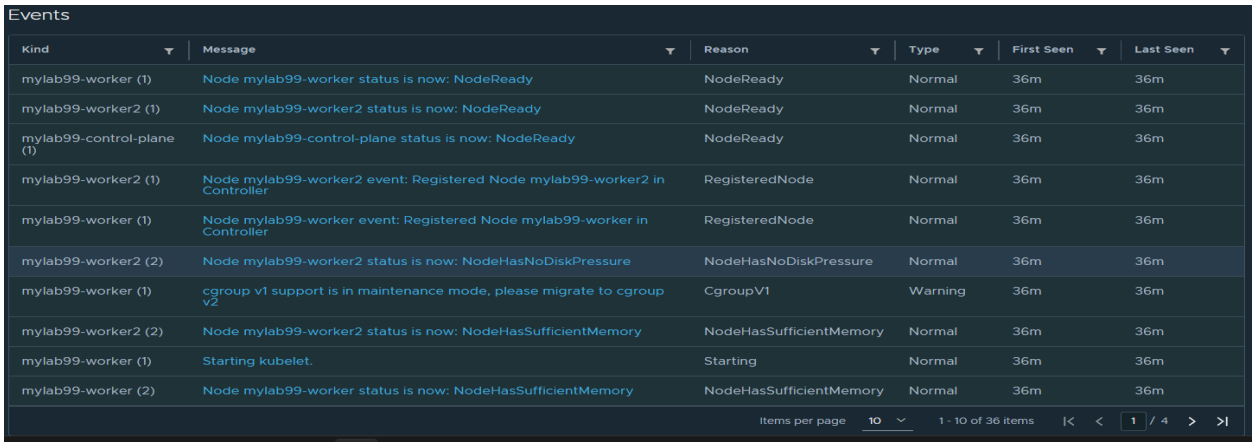
```
ster/kind$ sudo sh 2-set-config.sh
Set kubectl context to "kind-mylab99"
NAME                   STATUS   ROLES           AGE     VERSION
mylab99-control-plane   Ready    control-plane   3m6s    v1.32.0
mylab99-worker          Ready    <none>          2m54s   v1.32.0
mylab99-worker2         Ready    <none>          2m54s   v1.32.0
benji_1086@DESKTOP-68TBQ31:~/k3d-kind-exploration/kind-cases/setup-clu
ster/kind$ sudo sh 3-install-ingress.sh
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
```

Testing

```
ster/kind$ sudo sh 4-cek-ingress.sh
pod/ingress-nginx-controller-7f7d6896c9-n245w condition met
```

```
sualizer$ sudo docker ps -a
CONTAINER ID   IMAGE                  COMMAND                  CREATED
         STATUS          PORTS

      NAMES
335f064e9781   kindest/node:v1.32.0   "/usr/local/bin/entr…"   6 minut
es ago   Up 6 minutes

      mylab99-worker
81452eb14621   kindest/node:v1.32.0   "/usr/local/bin/entr…"   6 minut
es ago   Up 6 minutes

      mylab99-worker2
c9022953bd37   kindest/node:v1.32.0   "/usr/local/bin/entr…"   6 minut
es ago   Up 6 minutes   0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 0.0.
0.0:30080->30080/tcp, 0.0.0.0:30443->30443/tcp, 127.0.0.1:16443->6443/
tcp   mylab99-control-plane
```





Kesimpulan

Implementasi arsitektur KIND (Kubernetes in Docker) berhasil dilakukan, serta konfigurasi Flask, Nginx, dan PostgreSQL menggunakan arsitektur K3D dan juga pemanfaatan CI/CD sederhana, menunjukkan kemudahan dalam membangun dan mengelola aplikasi berbasis kontainer di lingkungan lokal dengan efisiensi tinggi. Arsitektur ini memungkinkan pengembangan yang lebih cepat dan skalabilitas yang lebih baik melalui penggunaan container Docker untuk menjalankan cluster Kubernetes yang ringan. Dengan CI/CD yang terintegrasi, proses otomatisasi deployment dan testing menjadi lebih lancar, sementara Flask berfungsi sebagai backend yang terhubung dengan datbase PostgreSQL dan dilindungi oleh Nginx sebagai reverse proxy, memberikan solusi yang mudah dikelola dan terstruktur untuk aplikasi yang skalabel dan portabel. Kami juga berhasil melakukan setup cluster arsitektur KIND yang telah disediakan sebelumnya dengan kubectl, rke, serta visualizer octant.