Moch. Avin
5025221061

Laporan Tugas 2 Komputasi Awan
Link repo: https://github.com/mochavin/komputasi-awan
Deskripsi tugas: Tugas 2

Pendahuluan

Teknologi kontainerisasi menggunakan Docker telah menjadi standar dalam proses *deployment* aplikasi modern karena kemampuannya menyediakan *environment* yang konsisten, portabel, dan terisolasi antar sistem. Dengan Docker, setiap aplikasi dapat dikemas bersama dependensinya ke dalam sebuah container, sehingga meminimalkan konflik konfigurasi antar *environment*. Pendekatan ini memungkinkan *developer* untuk membangun, menguji, dan mendistribusikan aplikasi dengan lebih efisien dan dapat direproduksi di berbagai platform.

Namun, dalam kasus aplikasi yang terdiri dari banyak *service* seperti web server, database, dan backend API, mengelola setiap *container* secara manual menjadi tidak efisien. Untuk itu, **Docker Compose** hadir sebagai alat orkestrasi sederhana yang memungkinkan pengguna mendefinisikan dan menjalankan kumpulan *container* melalui satu file konfigurasi berbasis YAML. Dengan Docker Compose, hubungan antar *service*, variabel *environment*, port, serta volume dapat diatur secara terpusat, sehingga mempermudah proses pengembangan, pengujian, dan *deployment*. Laporan ini disusun untuk mendemonstrasikan pemahaman dalam menggunakan Docker Compose.

Tugas yang diberikan

Tugas ini terdiri dari dua bagian utama. Pertama, menjalankan dan mendokumentasikan empat studi kasus yang ada pada repository, meliputi konfigurasi Nginx dengan dan tanpa SSL, implementasi WordPress, serta penyiapan *development environment* PHP. Dokumentasi mencakup penjelasan kode, langkah-langkah eksekusi, serta *screenshot* sebagai bukti. Bagian kedua adalah mengembangkan sebuah case baru (dinamakan Case 5). Pada case 5, saya mengembangkan aplikasi *microservice* sederhana. Untuk kasus ini, saya merancang arsitektur, menjelaskan skenario penggunaannya, membuat skrip yang diperlukan, dan menyajikan laporan implementasi dengan *screenshot* dan penjelasan.

Case 1: Docker compose nginx without SSL

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case1$ ls
docker-compose.yml  html  nginx-conf
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case1$ ls html/
assets  blog  grid  heroes  index.html  jumbotron
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case1$ ls nginx-conf/
nginx.conf
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case1$ cat ./nginx-conf/nginx.conf
server {
        listen 80;
        listen [::]:80;

        server_name _;

        index index.html index.htm;

        root /var/www/html;

        location = /favicon.ico {
                log_not_found off; access_log off;
        }
        location = /robots.txt {
                log_not_found off; access_log off; allow all;
        }
        location ~* \.(css|gif|ico|jpeg|jpg|js|png)$ {
                expires max;
                log_not_found off;
        }
}
```

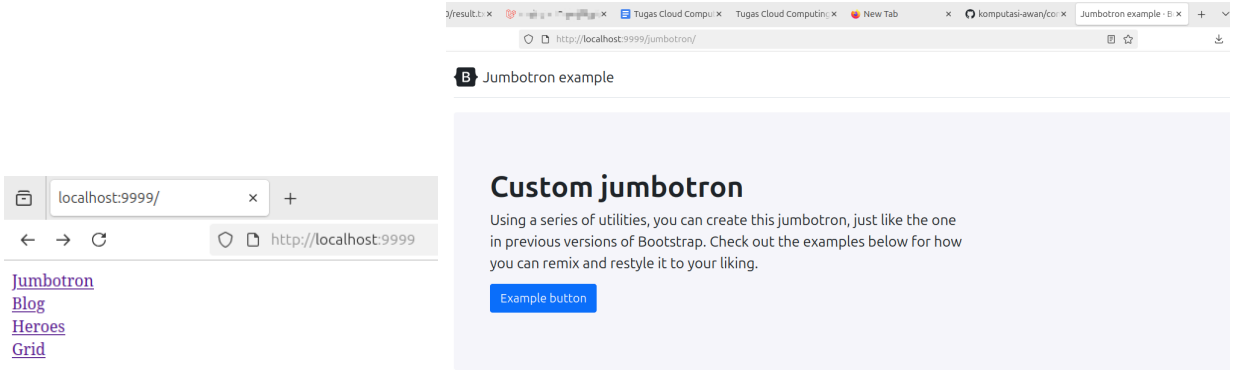
Gambar 1.1 Isi dari case 1

Pada Case 1 terdapat tiga komponen utama, yakni config Nginx, HTML dan kontennya, serta Docker Compose. Config Nginx diatur untuk listen pada port 80 dan serve HTML yang ada di direktori **/var/www/html** di dalam container. Konten HTML berada di **$(pwd)/html** pada host, dan Docker Compose berada di **$(pwd)/docker-compose.yml**.

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case1$ cat docker-compose.yml
version: '3'

services:
  webserver:
    image: nginx:1.15.12-alpine
    restart: unless-stopped
    ports:
      - "9999:80"
    volumes:
      - ./html:/var/www/html
      - ./nginx-conf:/etc/nginx/conf.d
    networks:
      - app-network
networks:
  app-network:
    driver: bridge
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case1$ docker compose up -d
WARN[0000] /home/cc25/cloudcomputing/cloud2023/containers/compose/compose/case1/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove
it to avoid potential confusion
[+] Running 2/2
 ✔ Network case1_app-network    Created                                                                                                                    0.0s
 ✔ Container case1-webserver-1  Started                                                                                                                    0.1s
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case1$ docker ps
CONTAINER ID   IMAGE                 COMMAND                  CREATED         STATUS         PORTS                                       NAMES
e41ad1fdb094   nginx:1.15.12-alpine  "nginx -g 'daemon of…"   5 seconds ago   Up 4 seconds   0.0.0.0:9999->80/tcp, [::]:9999->80/tcp     case1-webserver-1
```

Gambar 1.2 Docker Compose

Gambar 1.2 menunjukkan proses menjalankan container Nginx menggunakan **Docker Compose** yang didefinisikan di dalam **docker-compose.yml** dimana dibuat satu service dengan nama **webserver** dengan image *nginx:1.15.12-alpine.* Service ini listen pada port 9999 pada host dan meneruskannya ke port 80 di dalam container sesuai dengan config Nginx (nginx.conf). Terdapat 2 volume yang dipasang, yakni di dalam folder *./html* ke */var/www/html* untuk contennya dan *./nginx-conf* ke */etc/nginx/conf.d* untuk config *nginx*-nya. Untuk menjalankan konfigurasi docker composenya dijalankan perintah **docker compose up -d**. Untuk melihat apakah sudah jalan, bisa cek dengan perintah **docker ps**. Seperti yang terlihat pada Gambar 1.2, container sudah berhasil dijalankan dengan nama **case1-webserver-1.**



Gambar 1.3 Hasil port 9999 pada host

Gambar 1.3 menunjukkan hasil **localhost:9999**, pada "/" terlihat 4 *link* sesuai dengan isi dari **index.html.** Ketika diklik salah satu dari *link* tersebut, user akan diarahkan sesuai dengan *link* tersebut. Pada contoh di gambar 1.3, ketika diklik *jumbotron* maka akan diarahkan ke **/jumbotron** dan akan muncul tampilan index.html yang ada di folder jumbotron.

Moch. Avin
5025221061

Case 2: nginx with SSL

Case 2 sama halnya dengan case 1, dimana docker compose nginx tetapi dengan tambahan SSL

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case2$ cat docker-compose.yml
version: '3'

services:
  webserver:
    image: nginx:1.15.12-alpine
    restart: unless-stopped
    privileged: true
    ports:
      - "443:443"
      - "80:80"
    volumes:
      - ./certs:/certs
      - ./html:/var/www/html
      - ./nginx-conf/nginx.secure.conf:/etc/nginx/conf.d/nginx.conf
    networks:
      - app-network
networks:
  app-network:
    driver: bridge
```

Gambar 2.1 docker-compose.yml

Gambar 2.1 menunjukkan bahwa docker compose akan *spawn* satu service dengan nama **webserver.** Dimana *service* ini binding port 80 ke 80 di container dan 443 di host ke 443 di container. Pada case 2 ini juga mount volumes untuk certs, content HTML, dan config secure nginx.

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case2$ docker compose up -d
WARN[0000] /home/cc25/cloudcomputing/cloud2023/containers/compose/compose/case2/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potentia
on
[+] Running 2/2
 ✔ Network case2_app-network    Created                                                    0.0s
 ✔ Container case2-webserver-1  Started                                                    0.2s
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case2$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS                                                                      NAMES
c15993d70411   nginx:1.15.12-alpine   "nginx -g 'daemon of…"   4 seconds ago   Up 3 seconds   0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp   case2-webserver-1
cc25@cc25:~/cloudcomputing/cloud2023/containers/compose/compose/case2$ curl -v https://localhost --insecure
* Host localhost:443 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:443...
* Connected to localhost (::1) port 443
* ALPN: curl offers h2,http/1.1
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384 / X25519 / RSASSA-PSS
* ALPN: server accepted http/1.1
* Server certificate:
*  subject: C=ID; ST=Some-State; O=Internet Widgits Pty Ltd; CN=*.rm-dev.my.id
*  start date: Nov  6 20:58:54 2023 GMT
*  expire date: Nov  5 20:58:54 2024 GMT
*  issuer: C=ID; ST=Some-State; O=Internet Widgits Pty Ltd; CN=*.rm-dev.my.id
```

Gambar 2.2 Hasil curl https

Pada Gambar 2.2 dijalankan perintah **docker compose up -d** dan berhasil menjalankan 1 container bernama **case2-webserver-1**. Seperti yang terlihat pada gambar 2.2, cert berhasil terpasang pada webserver dengan **Common Name** *.rm-dev.my.id ini menandakan pemasangan SSL pada nginx telah berhasil.

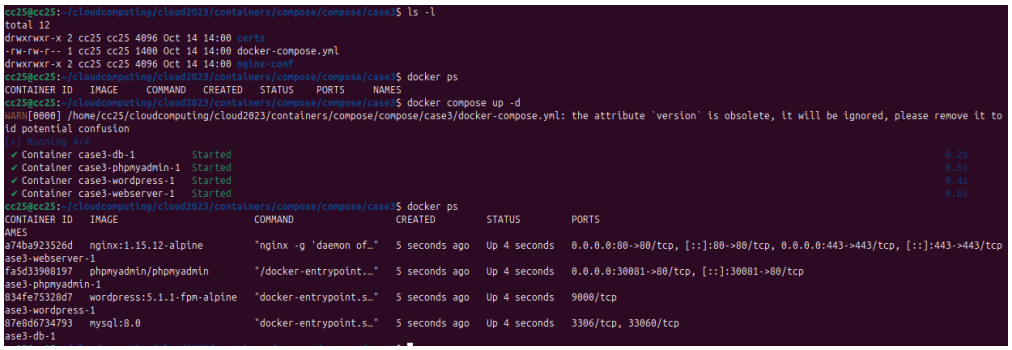
Moch. Avin
5025221061

Case 3: Wordpress

Pada case 3 ini dilakukan instalasi wordpress dengan persistent data dengan mysql, phpmyadmin, nginx.
Isi case3/docker-compose.yml:

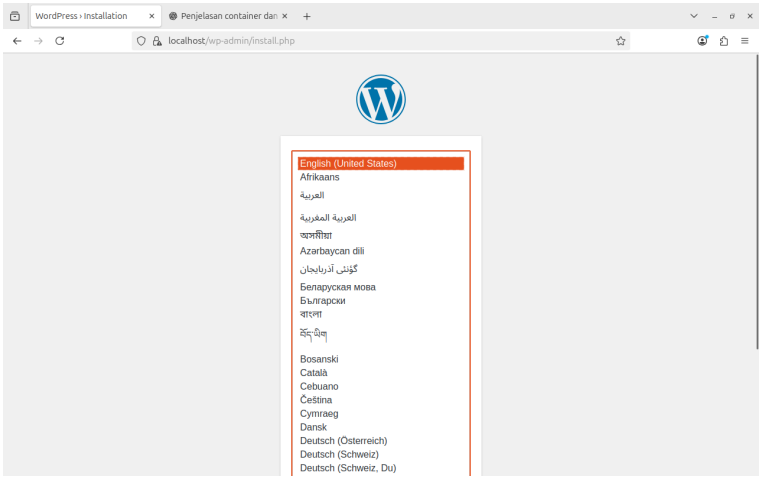
```
version: '3'

services:
  db:
    image: mysql:8.0
    restart: unless-stopped
    env_file: .env
    environment:
      - MYSQL_DATABASE=$WORDPRESS_DB
      - MYSQL_PASSWORD=$MYSQL_ROOT_PASSWORD
      - MYSQL_ROOT_PASSWORD=$MYSQL_ROOT_PASSWORD
      - MYSQL_ROOT_HOST=%
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - ./dbdata:/var/lib/mysql
    networks:
      - app-network
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
      - "$PHPMYADMIN_PORT:80"
    links:
      - db
    env_file: .env
    environment:
      - PMA_HOST=db
      - MYSQL_ROOT_PASSWORD=$MYSQL_ROOT_PASSWORD
    networks:
      - app-network
  wordpress:
    depends_on:
      - db
    links:
      - db
    image: wordpress:5.1.1-fpm-alpine
    restart: unless-stopped
    env_file: .env
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=root
      - WORDPRESS_DB_PASSWORD=$MYSQL_ROOT_PASSWORD
      - WORDPRESS_DB_NAME=$WORDPRESS_DB
    volumes:
      - ./wp_vol:/var/www/html
    networks:
      - app-network
  webserver:
    depends_on:
      - wordpress
    image: nginx:1.15.12-alpine
    restart: unless-stopped
    ports:
      - "443:443"
      - "80:80"
    volumes:
      - ./wp_vol:/var/www/html
      - ./certs:/certs
      - ./nginx-conf:/etc/nginx/conf.d
    networks:
      - app-network

networks:
  app-network:
    driver: bridge
```

Docker Compose ini membuat environment WordPress yang terdiri dari empat service utama: **db** (MySQL 8.0) sebagai database, **phpmyadmin** sebagai interface untuk mengelola database, **wordpress** (FPM Alpine) sebagai aplikasi utama, dan **webserver** (Nginx Alpine) sebagai *reverse proxy* yang menangani trafik HTTP dan HTTPS. Semua service terhubung melalui satu network bernama *app-network*. Volume digunakan untuk menyimpan data MySQL (./dbdata), file WordPress (./wp_vol), sertifikat SSL (./certs), dan konfigurasi Nginx (./nginx-conf) agar data serta konfigurasi tetap persisten meskipun container dihentikan. File .env digunakan untuk menyimpan environment variable seperti kredensial database dan port untuk phpMyAdmin.



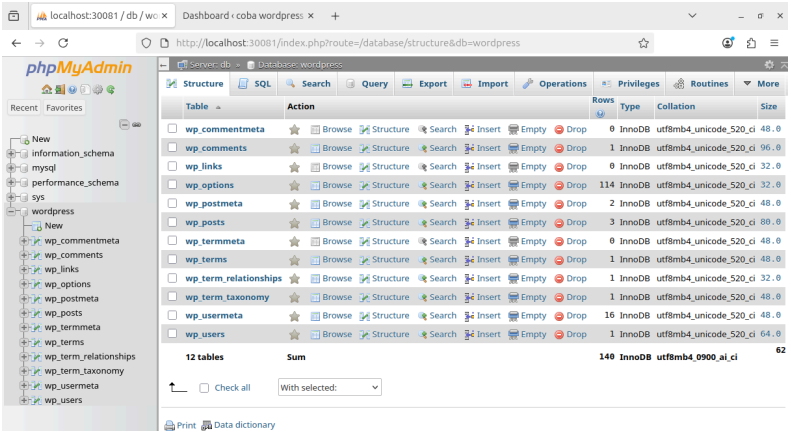
Gambar 3.1 Proses menjalankan docker compose

Setelah melakukan command **docker compose up -d,** 4 container berjalan sesuai dengan yang telah didefinisikan dalam docker-compose.yml terlihat pada gambar 3.1. Untuk wordpress dapat diakses melalui port 80 atau **https://localhost,** untuk phpmyadmin bisa diakses melalui port 30081, 4 *services* tersebut tersambung pada satu *network* bernama *app-network*.



Gambar 3.2 setup wordpress

Gambar 3.2 menunjukkan bahwa pada port 80 yakni https://localhost wordpress telah running dan bisa mulai setup wordpress.



Gambar 3.3 phpmyadmin

phpmyadmin dapat diakses pada port 30081 sesuai .env, setelah setup wordpress muncul database bernama wordpress dan juga tables di dalamnya.

Moch. Avin
5025221061

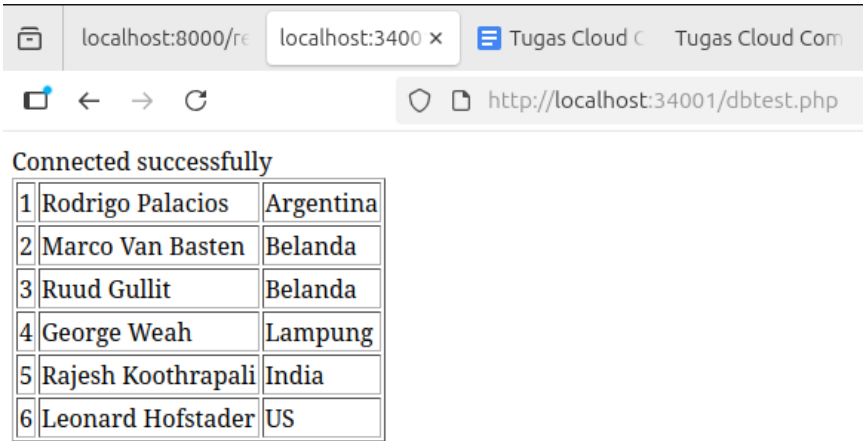
Case 4: PHP env development

```yaml
version: "3"

services:
  mysql1:
    image: mysql:5.7-debian
    restart: unless-stopped
    environment:
      - MYSQL_DATABASE=mydb
      - MYSQL_PASSWORD=mydb6789tyui
      - MYSQL_ROOT_PASSWORD=mydb6789tyui
      - MYSQL_ROOT_HOST=%
    volumes:
      - ./dbdata:/var/lib/mysql
      - ./dbscripts:/dbscripts
    networks:
      - example1-network
  app:
    restart: unless-stopped
    build:
      context: ./platform
      dockerfile: Dockerfile
    ports:
      - "34001:80"
    environment:
      - MODE=development
      - MYSQL_DATABASE=mydb
      - MYSQL_PASSWORD=mydb6789tyui
      - MYSQL_ROOT_PASSWORD=mydb6789tyui
      - MYSQL_ROOT_HOST=%
    volumes:
      - ./src:/var/www/localhost/htdocs/
      - ./platform/httpd.conf:/etc/apache2/httpd.conf
      - ./platform/ssl.conf:/etc/apache2/conf.d/ssl.conf
      - ./platform/log:/var/log/
      - ./platform/php.ini:/etc/php7/php.ini
    extra_hosts:
      - "test:10.199.2.29"
    networks:
      - example1-network
  alpine:
    image: alpine:3.18
    command:
      - /bin/sh
      - -c
      - "while true;do sleep 10000;done"
    networks:
      - example1-network
  phpmyadmin:
    image: phpmyadmin:5.2.1-apache
    ports:
      - "10000:80"
    links:
      - mysql1
    environment:
      - PMA_HOST=mysql1
      - MYSQL_ROOT_PASSWORD=mydb6789tyui
    networks:
      - example1-network

networks:
  example1-network:
    driver: bridge
```

Docker Compose di atas mendefinisikan empat *service* yang terhubung dalam satu *bridge network* bernama **example1-network**. *Service* **mysql1** menjalankan image **mysql:5.7-debian** dengan konfigurasi *environment* untuk *database* **mydb**, serta menyimpan data dan script pada direktori lokal **./dbdata** dan **./dbscripts**. *Service* **app** membangun *image* dari Dockerfile di direktori **./platform**, expose port **34001:80**, dan menggunakan beberapa *volume* untuk *source code*, konfigurasi Apache, SSL, PHP, serta log. *Service* ini juga memiliki variabel *environment* yang sama dengan database agar dapat terhubung. *Service* **alpine** menggunakan image **alpine:3.18** dan menjalankan loop *sleep* untuk menjaga container tetap hidup tidak ada fungsi secara khusus. *Service* **phpmyadmin** untuk mengelola database MySQL melalui port **10000**, dengan koneksi langsung ke *service* **mysql1**.

Gambar 4.1 localhost port 34001

Pada port 34001 di host terlihat php version sesuai dari volume yang didefinisikan pada **./platform/Dockerfile**. Hasil terlihat pada Gambar 4.1, php version yang jalan adalah versi 7.2.33 sesuai dengan yang ada di dalam Dockerfile.Jika ingin mengubah platform, maka perlu untuk melakukan build ulang dengan cara **docker compose build** dan **docker compose up -d**. Untuk shutdown project gunakan **docker compose down**.



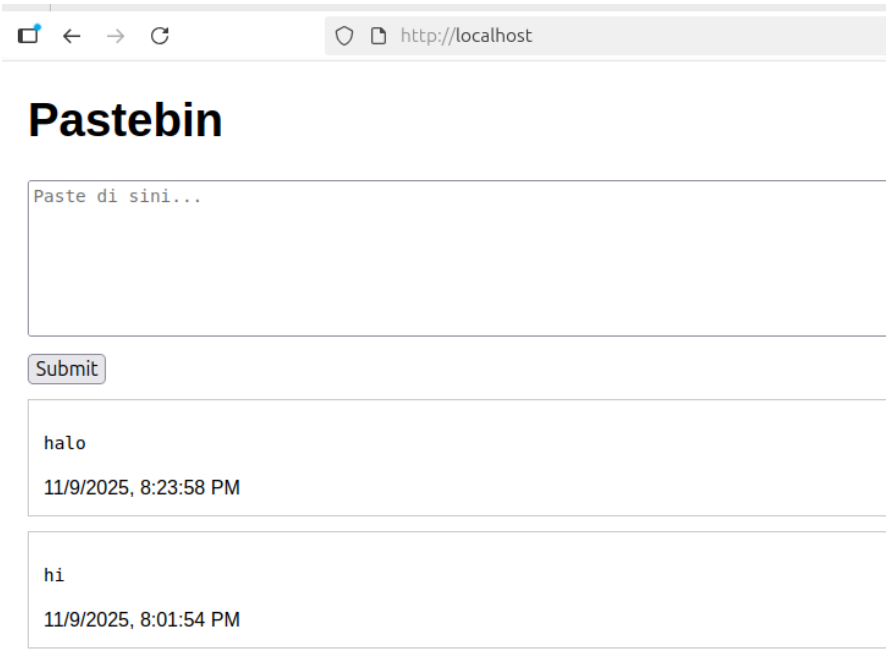
Gambar 4.3 Tes koneksi DB

Lakukan restore database dengan cara **docker compose exec -it mysql1 sh /dbscripts/restoredb.sh**, akan memunculkan list data dari ./dbscripts/backup.sql dan hasilnya dapat dilihat pada gambar 4.3. Untuk melakukan backup existing data lakukan command **docker compose exec -it mysql1 sh /dbscripts/backupdb.sh** maka akan mengoutputkan backup.sql di dalam folder /dbscript.

Moch. Avin
5025221061

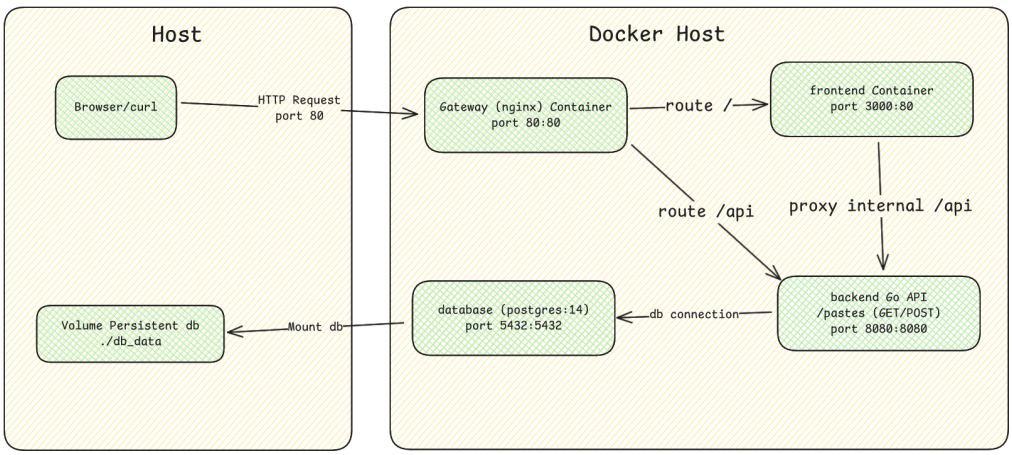
Case 5: Simple Microservice

Pada case 5 ini akan dibuat aplikasi seperti pastebin sederhana dimana text disimpan dalam persistent database postgreSQL dimana ini menunjukkan *proof of concept* dari arsitektur *microservice* menggunakan docker compose. Untuk spesifikasi VM yang digunakan pada case ini antara lain: 8192 MB Memory, 4 vCPU, 20GB Disk Size, IP address 10.21.85.107. Port yang digunakan pada case ini utamanya port 80 namun docker host juga ekspos port 3000, 5432, dan 8080.



Gambar 5.1 User Interface localhost port 80

Gambar 5.1 menunjukkan *frontend* pada aplikasi pastebin yang bisa diakses pada localhost port 80. User dapat melihat list yang telah disubmit dan user juga dapat menambahkan text baru.



Gambar 5.2 Arsitektur *simple microservice*

Gambar 5.2 menunjukkan arsitektur *simple microservice*, di mana terdapat empat container, yakni container untuk Nginx, frontend, backend, dan database. Gateway nginx meneruskan request dari user melalui port 80 dan meneruskannya ke frontend jika *route request*-nya "/", serta meneruskannya ke backend jika *route request*-nya berawalan dengan "/api". Container frontend juga menggunakan base Nginx untuk melakukan proxy internal ke backend melalui "/api", dan hanya berisi file HTML sederhana

untuk menampilkan UI yang dapat diakses dari browser pada port 3000, atau melalui proxy Nginx container dari port 80 dengan route “/”. Container backend menggunakan Golang, di mana di dalamnya hanya terdapat dua endpoint, yaitu **GET /pastes** untuk mengambil data text yang telah dipaste, dan **POST /pastes** untuk menambahkan text ke dalam database. Container ini bergantung pada database karena membutuhkan koneksi database untuk dapat berjalan. Container database menggunakan image **postgres:14-alpine**, di mana container ini menggunakan volume host pada **$(pwd)/db_data** agar datanya persisten. Database mengekspose port 5432 untuk dapat diakses oleh backend.

```yaml
version: "3.8"

services:
  db:
    image: postgres:14-alpine
    environment:
      POSTGRES_DB: pastebin
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    volumes:
      - ./db_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  backend:
    build: ./backend
    depends_on:
      - db
    environment:
      DB_HOST: db
      DB_PORT: 5432
      DB_USER: user
      DB_PASSWORD: password
      DB_NAME: pastebin
    ports:
      - "8080:8080"

  frontend:
    build: ./frontend
    ports:
      - "3000:80"

  gateway:
    build: ./gateway
    depends_on:
      - frontend
      - backend
    ports:
      - "80:80"
```

Code di atas merupakan isi dari docker-compose.yml pada direktori root case5. Pada docker compose ini ada empat *services* seperti yang telah dijelaskan sebelumnya. Pada *services* backend, frontend, dan backend, di dalam direktorinya terdapat Dockerfile masing-masing yang digunakan untuk *build container* dari masing-masing *service*.

Gambar 5.3 Struktur folder pada case5

Gambar 5.3 menunjukkan bagaimana struktur folder pada case ini, dimana seperti yang telah dijelaskan selanjutnya untuk backend, frontend, dan gateway masing masing terdapat Dockerfile untuk mengatur bagaimana container masing-masing dijalankan. Terdapat pula **db_data** digunakan sebagai *persistent data* yang digunakan oleh container database.

Kapan Skenario ini cocok dilakukan?

Arsitektur *microservice* sederhana ini sangat cocok untuk proyek baru skala kecil hingga menengah yang membutuhkan fleksibilitas untuk tumbuh di masa depan tanpa langsung terjun ke kompleksitas sistem orkestrasi skala besar seperti Kubernetes. Skenario ini menjadi pilihan pada skenario berikut:

- ketika ada pemisahan tim atau teknologi yang jelas (seperti frontend dan backend) yang memungkinkan *develop* aplikasi secara independen,
- ketika ada prediksi bahwa satu komponen aplikasi (misalnya API) akan membutuhkan penskalaan yang berbeda dari komponen lainnya.

Kesimpulan

Dari seluruh rangkaian tugas, saya menyimpulkan bahwa penggunaan Docker Compose memberikan kemudahan dalam mengelola dan menjalankan aplikasi yang terdiri dari banyak *services*. Dengan satu file konfigurasi berbasis YAML, saya dapat mengatur berbagai *service* seperti web server, database, PHP *environment*, hingga *microservice* secara konsisten dan mudah direproduksi di berbagai *environment*. Setiap case yang saya jalankan menunjukkan implementasi Docker Compose dalam konteks yang berbeda, mulai dari konfigurasi Nginx tanpa dan dengan SSL, implementasi WordPress dengan MySQL dan phpMyAdmin, hingga pembuatan environment pengembangan PHP. Pada case terakhir, yaitu *simple microservice*, saya berhasil mendemonstrasikan konsep *microservice* melalui pemisahan fungsi antar container (frontend, backend, gateway, dan database).

Melalui seluruh *study case* ini, saya menyadari bahwa Docker Compose berperan penting dalam menyederhanakan proses *deployment* sekaligus meningkatkan efisiensi pengujian dan *maintain* aplikasi. Dengan mengelola beberapa *container* secara terpusat, saya dapat membangun sistem yang fleksibel dan mudah dikembangkan.