Laporan Tugas 3 Cloud Computing

Nathaniel Ryo Kurniadi – 5025221019

Link Repository: https://github.com/rororyo/komputasi-awan-2025/tree/main/Tugas%203

Pendahuluan

Tugas 3 ini bertujuan untuk memperdalam pemahaman mahasiswa dalam membangun, mengelola, dan menjalankan layanan berbasis Docker Image. Melalui tugas ini, mahasiswa dapat memahami proses pembuatan image menggunakan Dockerfile, membangun container dari image tersebut, serta menjalankan berbagai layanan berbasis web dan aplikasi melalui lingkungan Docker yang terisolasi. Mahasiswa juga diminta untuk mendokumentasikan langkah-langkah implementasi, dan memberikan penjelasan detail terkait proses dari repositori https://github.com/rm77/cloud2023/tree/master/containers/compose/images. Selain itu, mahasiswa juga ditugaskan untuk mengembangkan satu kasus tambahan (Case 5) berdasarkan kreativitas masing-masing, yang mencakup desain arsitektur, skrip pendukung, serta dokumentasi hasil implementasi.

Deskripsi Tugas

Tugas ini terdiri dari empat kasus yang perlu dijalankan menggunakan Docker dan Docker Compose. Untuk penjelasan tiap kasusnya adalah sebagai berikut:

- **Case** 1: Membuat Docker Image dan menjalankan web server Nginx yang menyajikan layanan aplikasi PHP.
- **Case 2**: Membuat Docker Image dan menjalankan aplikasi Linux sederhana pada browser.
- **Case 3**: Membuat Docker Image dan menjalankan web server Nginx menggunakan protokol HTTP.
- **Case 4**: Membuat Docker Image dan menjalankan web server Nginx untuk menampilkan aplikasi PHP.
- **Case 5 (variasi pribadi)**: membuat Docker Image yang menjalankan aplikasi PHP sederhana dengan koneksi ke database menggunakan MySQL dan caching dengan redis.

Arsitektur dan Script yang Digunakan

Case 1

1. **Deskripsi**
   Pada Case ini, kita diminta untuk membuat image **mywebserver:1.0** menggunakan Dockerfile. Image ini adalah aplikasi PHP yang menampilkan tulisan “Hello World” dan detail dependensi yang digunakan.

2. **Langkah-langkah**
   Untuk menjalankan case ini, maka cukup jalankan *build.sh* di direktori */platform* untuk membangun Docker Image, lalu jalankan *run-mywebserver.sh* yang berada di direktori */runcontainer* untuk menjalankan container dengan konfigurasi yang telah diatur.
3. **Penjelasan**
   Ketika script dijalankan, maka aplikasi PHP dapat diakses melalui http://localhost:9999 yang menampilkan dependensi yang digunakan dan pesan "Hello World".

Case 2

1. **Deskripsi**
   Pada case ini, kita menggunakan Dockerfile untuk membuat sebuah image bernama **mylinux**. Aplikasi Linux akan berjalan pada browser menggunakan VNC.
2. **Langkah-langkah**
   Untuk menjalankan Case 2, maka kita jalankan *build.sh* pada direktori */platform* untuk membuat image, lalu jalankan *run-mylinux.sh* pada direktori */runcontainer* untuk menjalankan container.
3. **Penjelasan**
   Aplikasi Linux menggunakan VNC dapat diakses pada http://localhost:11111. Ketika diakses, maka tampilan desktop Linux sederhana dapat terlihat yang memungkinkan akses ke terminal dan browser.

Case 3

1. **Deskripsi**
   Pada Case ini, kita diminta untuk membuat image **mywebserver:2.0** yang menampilkan template bootstrap dengan Nginx menggunakan HTTP.
2. **Langkah-langkah**
   Untuk menjalankan Case 3, maka kita buat dulu imagenya dengan menjalankan *build.sh*, lalu untuk menjalankan container, kita jalankan *run-server.sh*.
3. **Penjelasan**
   Ketika script-script tersebut sudah berjalan dengan benar, maka hasilnya berupa sebuah layanan yang menyajikan template bootstrap yang tersedia pada http://localhost:9999.

Case 4

1. **Deskripsi**
   Pada Case 4, kita diminta untuk membuat image **mywebserver:2.1** dan menampilkan berbagai template bootstrap menggunakan Nginx web server, dan terdapat aplikasi php pada path /*test.php.*
2. **Langkah-langkah**

Seperti case-case sebelumnya, pertama Docker Image dibuat terlebih dahulu dengan menjalankan *build.sh*, lalu untuk menjalankan container dari image yang terbuat, maka kita jalankan *run-server.sh*.

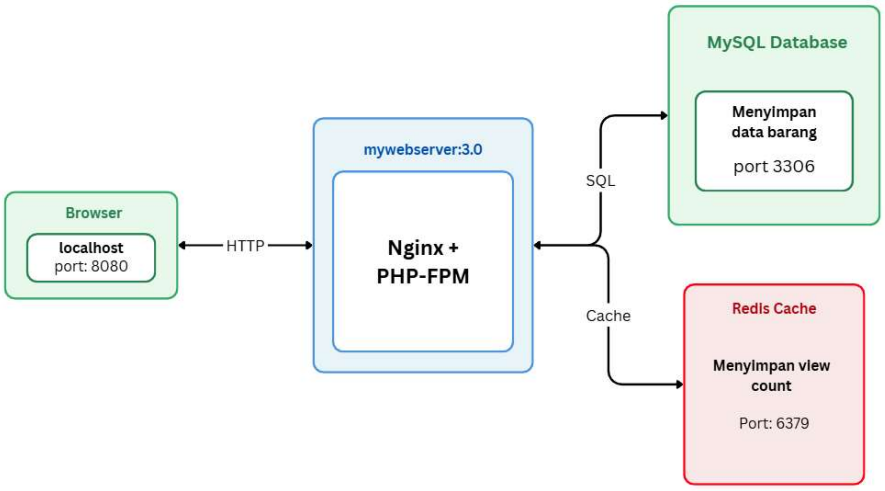
3. **Penjelasan**
   Ketika container sudah berhasil dibuat, maka web akan menampilkan berbagai template bootstrap dan aplikasi PHP dengan protokol HTTP. Untuk melihat hasilnya, maka dapat akses http://localhost:9999, maka akan ditamplikan template bootstrap. Untuk melihat hasil aplikasi PHP maka dapat diakses pada http://localhost:9999/test.php.

Case 5

1. **Deskripsi**
   Pada case ini, kita akan menyediakan sebuah aplikasi toko sederhana yang terintegrasi dengan database MySQL untuk penyimpanan data persistent dan Redis untuk sistem caching. Terdapat 4 halaman utama yang ada pada case ini,yaitu:
   - Homepage (index.php): Tampilan utama ketika mengakses aplikasi, dan menyediakan navigasi ke laman lainnya.
   - Database Test (db-test.php): Halaman untuk menguji koneksi database dan menampilkan data dari MySQL.
   - Cache Test (cache-test.php): Halaman untuk menguji fungsionalitas caching menggunakan redis, menampilkan counter views yang bertambah setiap refresh.
   - PHP info (info.php): Menampilkan konfigurasi PHP dan dependensi yang dipakai.
2. **Langkah-langkah**

   Untuk menjalankan case 5,maka cukup menjalankan *build.sh* pada direktori *platform*, dan *run-app.sh* pada direktori *runcontainers*. Aplikasi akan berjalan pada port 8080.

3. **Penjelasan**
   Bila script-script tersebut berhasil dijalankan, maka akan terbuat tiga container untuk webserver, database, dan caching. Layanan dapat diakses pada http://localhost:8080.
4. **Arsitektur**
   Berikut adalah arsitektur yang digunakan pada Case 5



Pertama, kita membuat terlebih dahulu image untuk layanan webserver yang akan digunakan menggunakan Dockerfile.

```
FROM nginx:1.19-alpine

RUN apk add --no-cache \
    php7 \
    php7-fpm \
    php7-mysqli \
    php7-pdo \
    php7-pdo_mysql \
    php7-json \
    php7-openssl \
    php7-curl \
    php7-zlib \
    php7-xml \
    php7-phar \
    php7-intl \
    php7-dom \
    php7-xmlreader \
    php7-ctype \
    php7-session \
    php7-mbstring \
    php7-redis

RUN sed -i 's/listen = 127.0.0.1:9000/listen = 9000/g' /etc/php7/php-fpm.d/www.conf

COPY nginx-conf/default.conf /etc/nginx/conf.d/default.conf

RUN mkdir -p /var/www/html

COPY html/ /var/www/html/

RUN chown -R nginx:nginx /var/www/html

RUN echo '#!/bin/sh' > /start.sh && \
    echo 'php-fpm7 &' >> /start.sh && \
    echo 'nginx -g "daemon off;"' >> /start.sh && \
    chmod +x /start.sh

EXPOSE 80

CMD ["/start.sh"]
```

Dockerfile di atas membangun image berbasis Nginx Alpine dan menambahkan dukungan PHP-FPM dan dependensi yang dibutuhkan (MySQL, Redis, dll). Bagian **RUN SED** mengubah konfigurasi PHP-FPM agar mendengarkan di port 9000, sementara COPY digunakan untuk menyalin file konfigurasi Nginx (default.conf) dan file aplikasi web ke dalam direktori */var/www/html*.
Selanjutnya, dibuat skrip */start.sh* untuk menjalankan PHP-FPM dan Nginx secara bersamaan dalam container, dengan port 80 diekspos agar layanan web dapat diakses

dari luar. Dockerfile ini kemudian dijalankan oleh script *build.sh* untuk membuat suatu image bernama mywebserver:3.0.

Lalu, kita membuat 3 container, yang pertama adalah MySQL untuk databasenya, dengan spesifikasi berikut:

- Nama: mysql-db
- Image: mysql:8.0
- Ports: 3306, ini port agar aplikasi kita dapat mengakses database, kita harus mengekspos port tersebut karena aplikasi berjalan pada container terpisah
- Volumes: kita melakukan mapping direktori *mysql-data* pada host ke direktori *var/lib/mysql* di dalam container. Lalu, kita juga mapping data inisial ke dalam container agar data sampel terisi.

Lalu, untuk container redis sendiri dibuat dengan spesifikasi berikut:

- Nama: redis-cache
- Image: redis:7-alpine
- Ports: 6379, port 6379 perlu di ekspos agar dapat diakses oleh container web server agar dapat menyimpan data webserver menggunakan redis.
- Volumes: pemetaan direktori *redis-data* di host ke direktori *data* di dalam container. Tujuan dari langkah ini adalah agar data yang disimpan oleh Redis tetap dapat diakses dan dipertahankan di sisi host, sehingga tidak hilang ketika container dihentikan atau dihapus.

Terakhir, adalah container webservernya sendiri untuk menyajikan layanan dengan Nginx dengan spesifikasi berikut:

- Nama: mywebserver3
- Image: mywebserver:3.0 (image yang telah di build sebelumnya).
- Ports: 8080:80, kita memetakan port 80 di dalam kontainer agar dapat di akses pada port 8080 pada host, sehingga host dapat mengakses port tersebut untuk melihat hasil dari webserver
- Depends On: Container ini mempunyai dependensi dengan container mysql dan redis untuk berjalan, jadi kedua container tersebut dijalankan,baru container mywebserver3 mulai diaktifkan.
- Environments: Untuk environment variables, kredensial database MySQL dan redis disimpan di environment agar nantinya dapat melakukan koneksi pada container MySQL dan redis.

Setelah pembuatan container selesai, maka aplikasi dapat diakses pada http://localhost:8080. Di sana, terdapat beberapa tombol untuk navigasi.

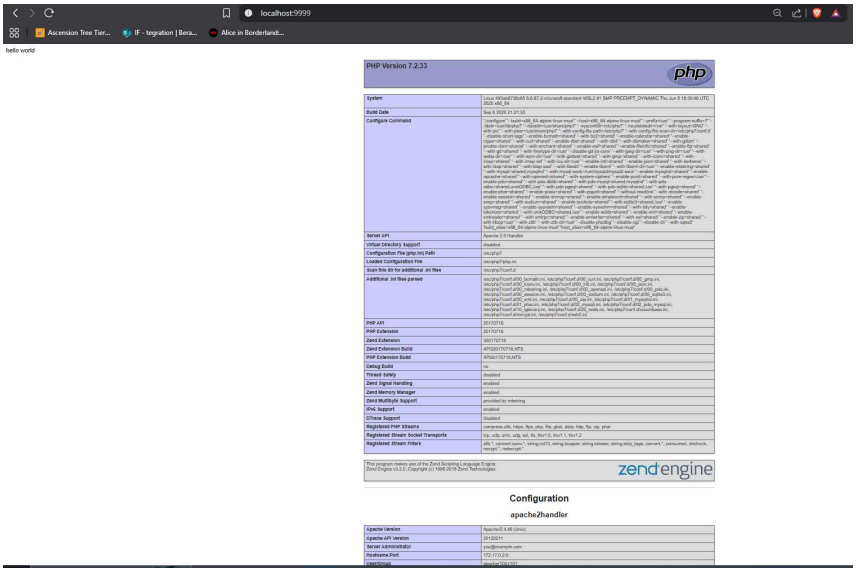
5. Kapan Skenario Ini Cocok Digunakan?

Arsitektur ini cocok digunakan pada beberapa skenario berikut:

- Penerapan aplikasi fullstack microservice dengan penyimpanan dan caching untuk menambah performance.
- Meningkatkan skalabilitas, karena tiap container mempunyai tanggung jawab masing-masing, sehingga dapat di scale dengan mudah.
- Belajar dan Eksperimen Arsitektur Layanan.

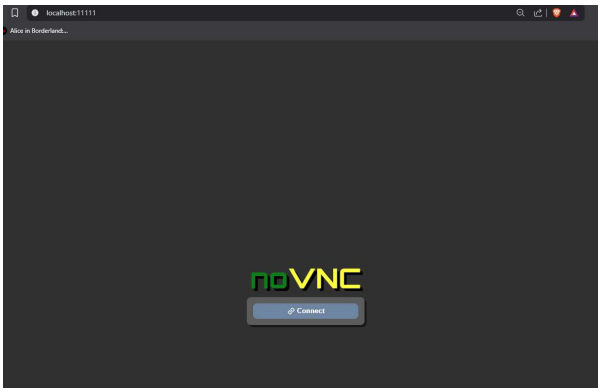
Screenshot dan Penjelasan

Case 1



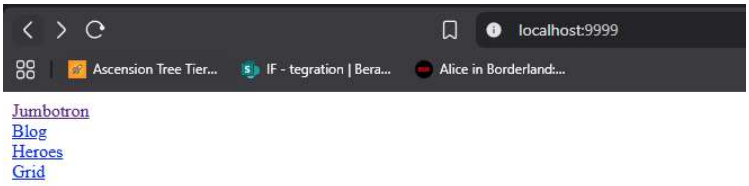
Penjelasan: Gambar di atas merupakan laman web ketika docker image berhasil dibuat dan container berhasil di run dengan menjalankan *run-mywebserver.sh*. Dapat dilihat, bahwa ketika semua langkah berhasil dilakukan, maka pesan ”Hello world” akan muncul di kiri atas layar dan terlihat dependensi PHP yang digunakan. Untuk mengaksesnya, dapat menggunakan http://localhost:9999

Case 2



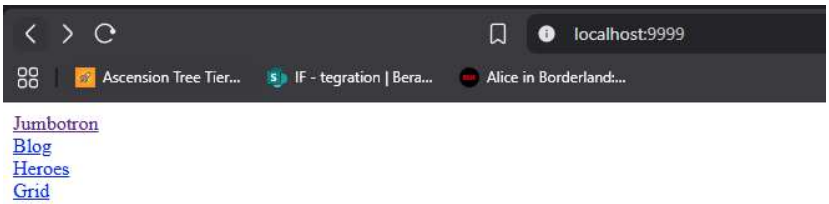
Penjelasan: Gambar di atas merupakan laman web ketika image berhasil dibuat, dan container berhasil dijalankan. Pada laman ini, kita dapat menggunakan dekstop Linux pada browser lewat VNC. Laman ini dapat diakses pada http://localhost:11111

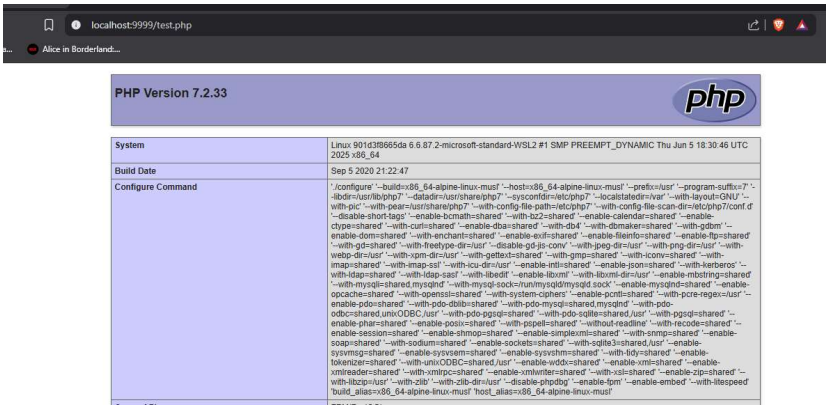
Case 3



Penjelasan: Setelah pembuatan image **mywebserver:2.0** berhasil dan container berhasil dijalankan, maka tampilan yang keluar adalah template elemen-elemen bootstrap seperti yang ditunjukkan pada gambar di atas. Layanan web server menggunakan Nginx dan dapat di akses pada http://localhost:9999.

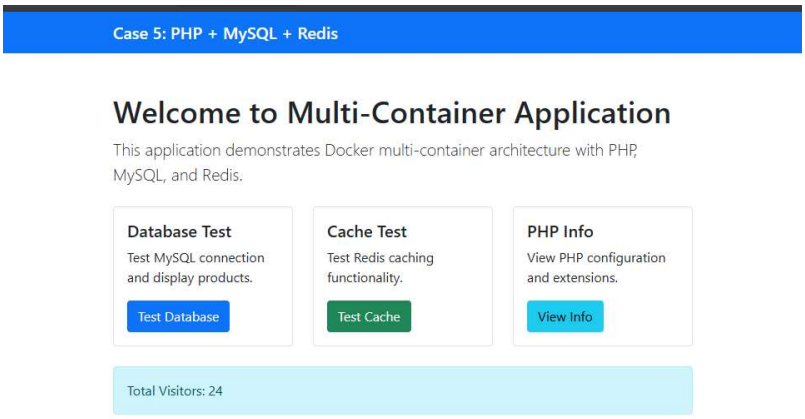
Case 4



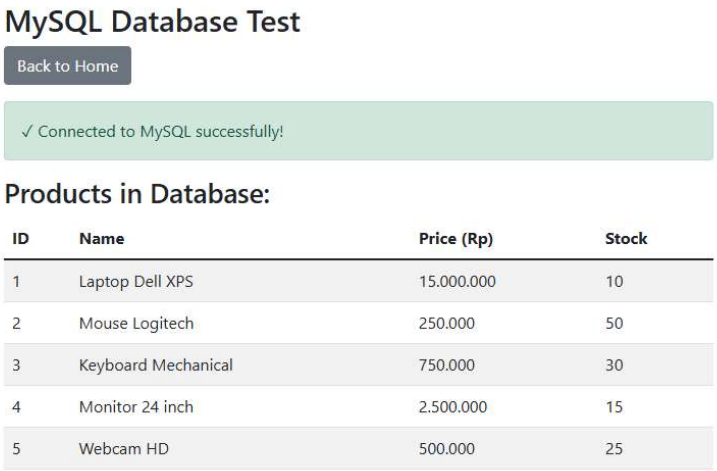


Penjelasan: Kedua gambar di atas adalah tampilan bila container berhasil dijalankan. Dapat dilihat, bahwa case4 merupakan pengembangan dari case3, dimana pada http://localhost:9999 terdapat template bootstrap yang disajikan. Lalu, terdapat tambahan path */test.php* yang menampilkan depedensi aplikasi PHP yang dapat diakses pada http://localhost:9999/test.php.

Case 5



Gambar 1. http://localhost:8080



Gambar 2. http://localhost:8080/db-test.php

Gambar 3. http://localhost:8080/cache-test.php

Gambar 4. http://localhost:8080/info.php

Penjelasan: Keempat gambar di atas adalah tampilan dari hasil implementasi Case 5 yang menggambarkan fungsi utama dari setiap komponen aplikasi. Gambar 1 menampilkan laman utama (homepage) aplikasi yang menyediakan navigasi ke halaman-halaman lain. Gambar 2 menunjukkan hasil pengujian koneksi ke database MySQL melalui halaman db-test.php, di mana ditampilkan daftar produk sampel yang telah di inject saat inisialisasi. Gambar 3 adalah pengujian fungsi caching menggunakan Redis melalui halaman cache-test.php, yang menampilkan jumlah view count yang terus meningkat setiap kali halaman di-refresh tanpa harus mengakses database MySQL. Terakhir, Gambar 4 menampilkan halaman info.php yang berisi informasi konfigurasi PHP dan dependensi ekstensi yang digunakan oleh aplikasi dalam container webserver.

Kesimpulan

Dari tugas yang diberikan kali ini, mahasiswa mempelajari secara menyeluruh bagaimana membangun, mengelola, dan menjalankan layanan berbasis Docker secara modular dan terisolasi. Tiap casenya memberikan pemahaman terhadap cara membuat Image dan menjalankannya. Lalu, pada variasi ke 5, saya mencoba menggabungkan pembuatan Docker Image dengan Docker Compose yang telah dipelajari pada tugas sebelumnya untuk membangun

arsitektur multi-container dengan integrasi database MySQL dan redis caching. Penerapan konsep containerization ini menunjukkan efisiensi dalam pengembangan aplikasi karena tiap layanannya dapat dijalankan secara independen namun tetap saling terhubung melalui jaringan virtual Docker. Selain itu, penggunaan volume juga memastikan data tetap persistent meskipun container dihentikan. Secara keseluruhan, tugas ini memperkuat pemahaman mahasiswa terhadap konsep container orchestration, service integration, dan environment isolation.