Tugas 3 Cloud Computing

Rafhi Sadipta – 5025221014

Link Repository: https://github.com/RafhiSadipta/cloud-computing-2025/tree/main/tugas3

Spesifikasi Laptop

Merek	Acer
Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz (2.42 GHz)
RAM	16 GB
System Type	64-bit operating system, x64-based processor

Spesifikasi Virtual Machine (Ubuntu)

Virtualizer	VMware Workstation
RAM	4 GB
CPU	2
Disk	20 GB
Operation System	Ubuntu 64-bit
IP	192.168.88.128/24

1. Pendahuluan

1.1 Latar Belakang

Dalam era komputasi modern, kecepatan dan konsistensi deployment aplikasi menjadi faktor krusial. Teknologi containerization, khususnya Docker, telah muncul sebagai solusi standar industri yang efektif untuk mengatasi tantangan ini. Docker memungkinkan pengembang untuk mengemas aplikasi, beserta semua pustaka, dependensi, dan konfigurasinya, ke dalam satu unit yang disebut Docker Image. Unit yang terisolasi dan portabel ini, ketika dijalankan, menjadi Docker Container, yang menjamin bahwa aplikasi akan berjalan secara konsisten di lingkungan pengembangan, pengujian, hingga produksi, terlepas dari perbedaan infrastruktur host. Melalui tugas ini, mahasiswa diperkenalkan secara praktis pada konsep dasar pembuatan packaging aplikasi menggunakan Docker Images dan penerapan skenario container dalam lingkungan berbasis cloud. Repository yang disediakan menjadi basis pembelajaran untuk menganalisis berbagai pola penggunaan dan konfigurasi Docker.

1.2 Tujuan

Tugas ini bertujuan untuk.

1. Memahami dan menerapkan konsep packaging aplikasi menggunakan Docker Images dan containerization.
2. Menjalankan dan mendokumentasikan seluruh skenario yang terdapat dalam repositori yang diberikan.
3. Mengembangkan skenario tambahan sebagai bentuk kreasi mandiri dan penerapan konsep Docker yang lebih mendalam, termasuk perancangan arsitektur, pembuatan script, dan justifikasi kegunaannya.
4. Mendokumentasikan hasil percobaan dan analisis secara sistematis..

2. Tugas Yang Diberikan

Tugas ini berfokus pada penerapan dan dokumentasi konsep packaging aplikasi menggunakan Docker Images melalui eksplorasi repository yang tersedia pada https://github.com/rm77/cloud2023/tree/master/containers/compose/images.

Tugas utama terbagi menjadi dua bagian. Pertama, implementasi dan dokumentasi terhadap empat kasus (case 1 hingga case 4) yang telah tersedia dalam repositori. Setiap case harus dijalankan, didokumentasikan langkah-langkahnya, dan dilengkapi dengan screenshot serta penjelasannya. Kedua, pengembangan mandiri berupa pembuatan satu skenario tambahan yang dinamakan Case 5. Pengembangan ini harus mencakup perancangan gambar arsitektur baru, penulisan script pendukung, penjelasan relevansi skenario tersebut, dan dokumentasi hasilnya. Semua hasil implementasi, analisis, dan spesifikasi teknis (termasuk arsitektur dan spesifikasi komputer) harus dikumpulkan dalam satu laporan akhir berbentuk PDF dengan batasan maksimal 10 halaman.

3. Implementasi dan Penjelasan

3.1 Case 1

- Tujuan Case 1

Tujuan dari Case 1 adalah untuk memahami proses pembuatan Docker Image yang menjalankan layanan web server dengan dukungan PHP menggunakan Apache HTTP Server. Skenario ini memperlihatkan bagaimana sebuah aplikasi web berbasis PHP (index.php) di-containerize dengan dependensi, konfigurasi, dan script startup yang diperlukan (platform), kemudian dijalankan (runcontainer). Hal ini bertujuan untuk memastikan aplikasi PHP dapat berjalan secara konsisten di lingkungan container yang terisolasi.

- Cuplikan Kode / Script dan Penjelasan

  \- Dockerfile

```
FROM alpine:3.9

RUN apk update && apk add --no-cache apache2 curl supervisor socat php7-intl php7-openssl php7-pecl-redis php7-sqlite3 php7-pdo_mysql php7-mbstring apache2-ssl php7-apache2 php7-intl php7-openssl php7-sqlite3 php7-pdo_mysql php7-mbstring apache2-ssl php7-apache2 apache2 apache2-ctl php7-pecl-ssh2 php7-bcmath php7-curl php7-json php7-pecl-couchbase php7-zip php7-mysqli mysql-client php7-pecl-mcrypt php7-pecl-redis supervisor
RUN apk add composer php7-gmp php7-sodium  php7-xml supervisor
RUN rm -rf /tmp/* /var/cache/apk/*

ADD start.sh /tmp
ADD supervisord.conf /tmp/supervisord.conf

EXPOSE 80
EXPOSE 9999

ENV SESSIONAGE 60
ENV REAUTHENTICATION 60
ENV IPEXPIRE 120
VOLUME ["/var/www/localhost/htdocs"]

ENTRYPOINT ["sh","-C","/tmp/start.sh"]
```

  File ini adalah blueprint untuk Image, dimulai dari basis Alpine Linux yang ringan. Tugas utamanya adalah menginstal `Apache2` dan `PHP 7` beserta ekstensi yang dibutuhkan, menyalin konfigurasi (supervisord.conf dan start.sh), mendeklarasikan port yang akan digunakan, dan menetapkan script `/tmp/start.sh` sebagai entrypoint utama ketika container dijalankan.

  \- run-mywebserver.sh

```
docker rm -f mywebserver

docker run \
        -dit \
        --name mywebserver \
        -p 9999:80 \
        -v $(pwd)/html:/var/www/localhost/htdocs/ \
        mywebserver:1.0
```

  Script ini berfungsi untuk memulai container. Ia terlebih dahulu menghapus container lama bernama mywebserver jika ada, kemudian menjalankan container baru dari Image `mywebserver:1.0`. Konfigurasi kunci yang dilakukan adalah Port Mapping (-p 9999:80) untuk membuat layanan dapat diakses di port 9999 host, dan Volume Mounting (-v ...) yang menghubungkan direktori kode aplikasi host ke direktori web root di dalam container, memungkinkan pembaruan kode tanpa rebuild image.

- Cara Menjalankan
  - Jalankan file `build.sh` yang terletak di dalam folder /platform dengan perintah `sudo bash build.sh` untuk membangun image, lalu kita beralih ke folder /runcontainer dan menjalankan file `run-mywebserver.sh` dengan perintah `sudo bash run-mywebserver.sh` untuk menjalan container.

```
rafhi-sadipta@rafhi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas3/case1/runcontainer$ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED              STATUS              PORTS
                          NAMES
bc116c7488ad   mywebserver:1.0   "sh -C /tmp/start.sh"    About a minute ago   Up About a minute   9999/tcp, 0.0.0.0:9999
->80/tcp, [::]:9999->80/tcp   mywebserver
```

  - Akses layanan melalui browser pada alamat `http://localhost:9999`.

  Gambar ini memverifikasi keberhasilan implementasi Case 1. Setelah container dijalankan menggunakan `run-mywebserver.sh`, web browser diakses melalui port yang dipetakan (http://localhost:9999). Tampilan ini menunjukkan dua hal, yaitu pertama tulisan "hello world" berhasil ditampilkan, menunjukkan bahwa Volume Mounting kode (index.php) ke dalam container berhasil, lalu kedua, ditampilkan halaman informasi PHP Version 7.2.33, yang menegaskan bahwa web server Apache telah berfungsi dengan baik dan modul PHP yang diinstal melalui Dockerfile telah aktif dan siap memproses skrip PHP.

3.2 Case 2

- Tujuan Case 2

  Tujuan dari Case 2 adalah untuk membuat Docker Image dasar Linux yang sangat kaya fitur. Image ini tidak hanya berisi utilitas dasar, tetapi juga menginstal lingkungan Remote Desktop menggunakan `Xvfb`, ` x11vnc `, dan `Openbox`, serta `Firefox` sebagai browser dan noVNC (berbasis VNC melalui web) untuk akses antarmuka grafis. Selain itu, Image ini menyertakan sertifikat keamanan kustom. Ini adalah skenario yang cocok untuk menyediakan lingkungan Virtual Desktop ringan berbasis container yang dapat diakses melalui browser.

- Cuplikan Kode / Script dan Penjelasan

  \- Dockerfile

```
FROM alpine:3.18


RUN apk update && apk add xvfb x11vnc openbox ttf-dejavu lxterminal firefox supervisor git bash


RUN git clone --depth 1 https://github.com/novnc/noVNC.git /opts/noVNC && \
    git clone --depth 1 https://github.com/novnc/websockify /opts/noVNC/utils/websockify && \
    rm -rf /opts/noVNC/.git && \
    rm -rf /opts/noVNC/utils/websockify/.git && \
    sed -i -- "s/ps -p/ps -o pid | grep/g" /opts/noVNC/utils/novnc_proxy


RUN rm -rf /tmp/* /var/cache/apk/*
RUN cd /opts/noVNC && cp vnc.html index.html
RUN mkdir -p /home/ && mkdir -p /etc/xdg/openbox



ADD certs/My* /tmp/
ADD start.sh /tmp
ADD supervisord.conf /tmp/supervisord.conf
ADD menu.xml /etc/xdg/openbox/menu.xml

RUN adduser user1 -D ; echo 'user1:user1qwerty' | chpasswd


EXPOSE 5910
EXPOSE 6666
EXPOSE 11111


ENTRYPOINT ["sh","-C","/tmp/start.sh"]
```

  Dockerfile ini memulai dengan Image dasar `Alpine Linux 3.18`. Bagian instalasi paket sangat intensif, menambahkan tools seperti `Xvfb`, `x11vnc`, `Openbox`, `Firefox`, `Supervisor`, `Git`, dan `Bash` untuk membangun lingkungan desktop minimal. Bagian selanjutnya mengkloning repositori noVNC dan websockify yang memungkinkan akses VNC melalui HTTP/WebSocket, serta melakukan penyesuaian script internal. Terakhir, ia menyalin sertifikat kustom (certs/My*), konfigurasi layanan (supervisord.conf, menu.xml), membuat user baru (user1), dan menetapkan port 5910 (VNC), 6666, dan 11111 untuk diakses.

  \- run-mylinux.sh

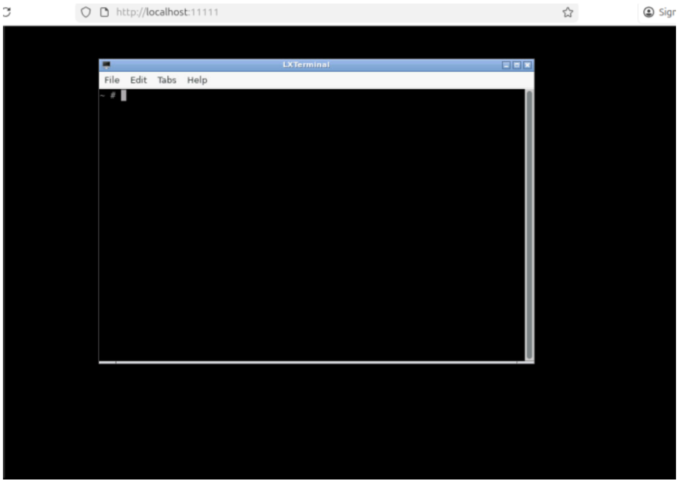
```
docker rm -f mylinux

docker run \
        -dit \
        --name mylinux \
        -p 12111:5910 \
        -p 11111:11111  \
        mylinux:1.0
```

  Script ini mengelola eksekusi container. Pertama, ia memastikan tidak ada container lama bernama `mylinux` yang berjalan dengan perintah `docker rm -f mylinux`. Selanjutnya, ia menjalankan container baru dari Image `mylinux:1.0` dalam mode detached (-dit). Bagian krusial adalah Port Mapping: ia memetakan port 12111 host ke port 5910 container (port VNC), dan port 11111 host ke port 11111 container. Pemetaan ini memungkinkan pengguna mengakses lingkungan desktop melalui VNC atau noVNC pada port yang ditentukan.

- Cara Menjalankan
  - Jalankan file `build.sh` yang terletak di dalam folder /platform dengan perintah `sudo bash build.sh` untuk membangun image, lalu kita beralih ke folder /runcontainer dan menjalankan file `run-mylinux.sh` dengan perintah `sudo bash run-mylinux.sh` untuk menjalan container.

```
rafhi-sadipta@rafhi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas3/case2/runcontainer$ docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS         PORTS
                                                                                        NAMES
60e03d6e2b46   mylinux:1.0   "sh -C /tmp/start.sh"    9 seconds ago   Up 8 seconds   6666/tcp, 0.0.0.0:11111->11111/tcp,
[::]:11111->11111/tcp, 0.0.0.0:12111->5910/tcp, [::]:12111->5910/tcp    mylinux
rafhi-sadipta@rafhi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas3/case2/runcontainer$
```

  - Lalu kita coba untuk membuka `http://localhost:11111`.



Tangkapan layar ini memverifikasi keberhasilan implementasi Case 2 sebagai lingkungan Remote Desktop berbasis container. Gambar menunjukkan bahwa setelah menjalankan container dan mengakses host pada port yang dipetakan (misalnya, http://localhost:11111 melalui noVNC), antarmuka grafis ringan LXTerminal berhasil ditampilkan. Hal ini menegaskan bahwa semua komponen desktop (Xvfb, VNC Server, Openbox, dan noVNC) di dalam container berfungsi, menyediakan akses terminal interaktif yang terisolasi.

3.3 Case 3

- Tujuan Case 3
  Tujuan dari Case 3 adalah membangun Docker Image Nginx Web Server yang bersifat kustom. Pada kasus ini ditunjukkan cara menggunakan base image Nginx yang lebih spesifik, yaitu `nginx:1.15.12-alpine`, serta memanfaatkan file konfigurasi Nginx dan konten HTML yang berasal dari direktori eksternal selama proses build.

- Cuplikan Kode / Script dan Penjelasan
  - Dockerfile

```
FROM nginx:1.15.12-alpine

ADD nginx-conf/nginx.conf /etc/nginx/conf.d
COPY nginx-conf/nginx.conf /etc/nginx/conf.d/default.conf
ADD html/ /var/www/html/
```

  Dockerfile ini memulai dengan Image dasar `nginx:1.15.12-alpine`, yang menyediakan web server Nginx dalam lingkungan Alpine Linux yang ringan. Perintah `ADD` dan `COPY` adalah inti di sini, di mana file konfigurasi `nginx.conf` dan konten web statis `html/` disalin ke lokasi standar di dalam Image Nginx (/etc/nginx/conf.d dan /var/www/html/). Proses ini menghasilkan Image yang sudah dikemas dengan konfigurasi dan konten aplikasi yang siap melayani permintaan.

  - run-server.sh

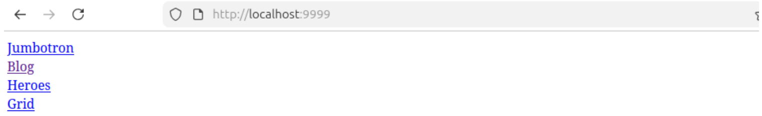
```
docker rm -f webserver2
docker run -dit \
        --name webserver2 \
        -p 9999:80 \
        mywebserver:2.0
```

  Script ini bertanggung jawab untuk menjalankan container dari Image yang telah dibuat. Sama seperti kasus sebelumnya, ia menghapus container lama (webserver2), kemudian menjalankan container baru dalam mode detached (-dit). Ia melakukan Port Mapping yaitu `-p 9999:80`, yang berarti layanan Nginx yang berjalan di port 80 di dalam container akan dapat diakses melalui port 9999 pada host Anda.

- Cara Menjalankan
  - Sesuaikan symlink untuk case1 ke path yang benar, disini saya mengubah yang path awal adalah `../../compose/case1` menjadi `../../tugas2/case1`.
  - Jalankan file `build.sh` dengan perintah `sudo bash build.sh` untuk membangun image, lalu kita menjalankan file `run-server.sh` dengan perintah `sudo bash run-server.sh` untuk menjalan container.

```
rafhi-sadipta@rafhi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas3/case3$ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS         PORTS
          NAMES
3dd3eb36347e   mywebserver:2.0   "nginx -g 'daemon of…"   8 seconds ago   Up 7 seconds   0.0.0.0:9999->80/tcp, [::]:9999
->80/tcp   webserver2
```

- Lalu kita coba untuk membuka `http://localhost:9999`.



Tangkapan layar tersebut menunjukkan hasil akses ke `http://localhost:9999`, yaitu port yang dipetakan ke web server Nginx di dalam container webserver2. Tampilan halaman dengan tautan navigasi Jumbotron, Blog, Heroes, dan Grid menandakan bahwa konten HTML statis berhasil dimuat, yang sebelumnya telah disalin ke dalam Image melalui Dockerfile. Keberhasilan ini memverifikasi bahwa proses build yang menggunakan konteks dari luar direktori (misalnya case1/ melalui symlink) mampu menarik aset yang dibutuhkan, serta memastikan bahwa web server Nginx berbasis `nginx:1.15.12-alpine` berjalan dengan benar di dalam container dan melayani konten dari web root yang telah dikonfigurasi.

3.4 Case 4

- Tujuan Case 4
  Tujuan dari Case 4 adalah untuk mendemonstrasikan bagaimana mengemas aplikasi web lengkap (Nginx dan PHP) yang membutuhkan beberapa layanan untuk berjalan secara simultan dalam satu container. Dalam skenario ini, Nginx berfungsi sebagai web server utama, yang kemudian meneruskan permintaan PHP ke PHP-FPM. Supervisord digunakan sebagai process entrypoint untuk mengelola dan memastikan kedua layanan (nginx dan php-fpm7) selalu berjalan di latar belakang secara terus menerus.

- Cuplikan Kode / Script dan Penjelasan
  - Dockerfile

```
FROM nginx:1.15.12-alpine

RUN apk update && apk add php7-fpm supervisor git curl

ADD nginx.conf /etc/nginx/conf.d
COPY nginx.conf /etc/nginx/conf.d/default.conf
ADD html/ /var/www/html/
ADD start.sh /
ADD supervisord.conf /tmp


ENTRYPOINT ["/bin/sh","/start.sh"]
```

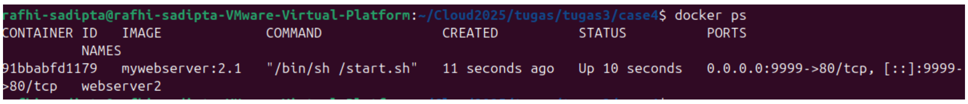
`Dockerfile` ini menggunakan Image dasar `nginx:1.15.12-alpine` yang ringan. Baris `RUN` menginstal paket-paket penting, yaitu `php7-fpm` (untuk memproses skrip PHP), `superviso`r (manajer proses), `git`, dan `curl`. Selanjutnya, file-file konfigurasi Nginx, konten web statis/dinamis (html/), script startup (start.sh), dan konfigurasi Supervisord (supervisord.conf) disalin ke dalam Image. /bin/sh /start.sh ditetapkan sebagai `ENTRYPOINT` utama.

- supervisord.conf

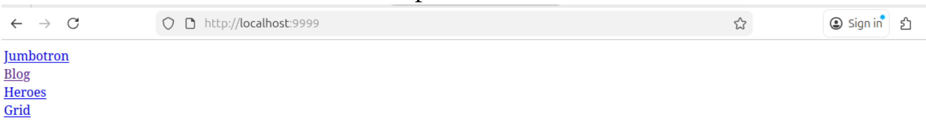
```
[supervisord]
nodaemon=true
logfile = /tmp/supervisord.log
logfile_maxbytes = 50MB
pidfile = /tmp/supervisord.pid
minfds = 1024
minprocs = 200
directory = /tmp
childlogdir = /tmp

[program:nginx]
command=nginx -g 'daemon off;'
autorestart=true
priority=100

[program:php7]
command=/usr/sbin/php-fpm7 -F
autorestart=true
priority=400
```

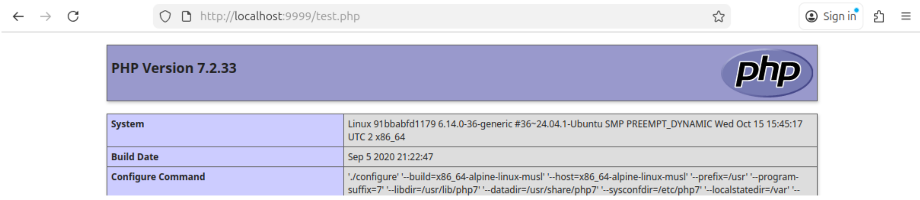
File konfigurasi ini merupakan komponen inti dari Case 4. Pada bagian `[supervisord]`, konfigurasi `nodaemon=true` memastikan Supervisord berjalan tanpa mode daemon, sehingga container tetap aktif selama Supervisord beroperasi. Selanjutnya, bagian `[program:nginx]` dan `[program:php7]` mendefinisikan dua proses terpisah yang dijalankan dan diawasi secara otomatis oleh Supervisord. Proses nginx dijalankan dengan perintah `nginx -g 'daemon off;'` agar tetap berjalan di foreground di bawah pengawasan Supervisord, sedangkan proses php7 dijalankan menggunakan perintah `/usr/sbin/php-fpm7 -F` sehingga layanan PHP-FPM juga berjalan di foreground dan dapat dikelola secara terpusat oleh Supervisord.

- Cara Menjalankan
  - Jalankan file `build.sh` dengan perintah `sudo bash build.sh` untuk membangun image, lalu kita menjalankan file `run-server.sh` dengan perintah `sudo bash run-server.sh` untuk menjalankan container.



  - Lalu kita coba untuk membuka `http://localhost:9999`.

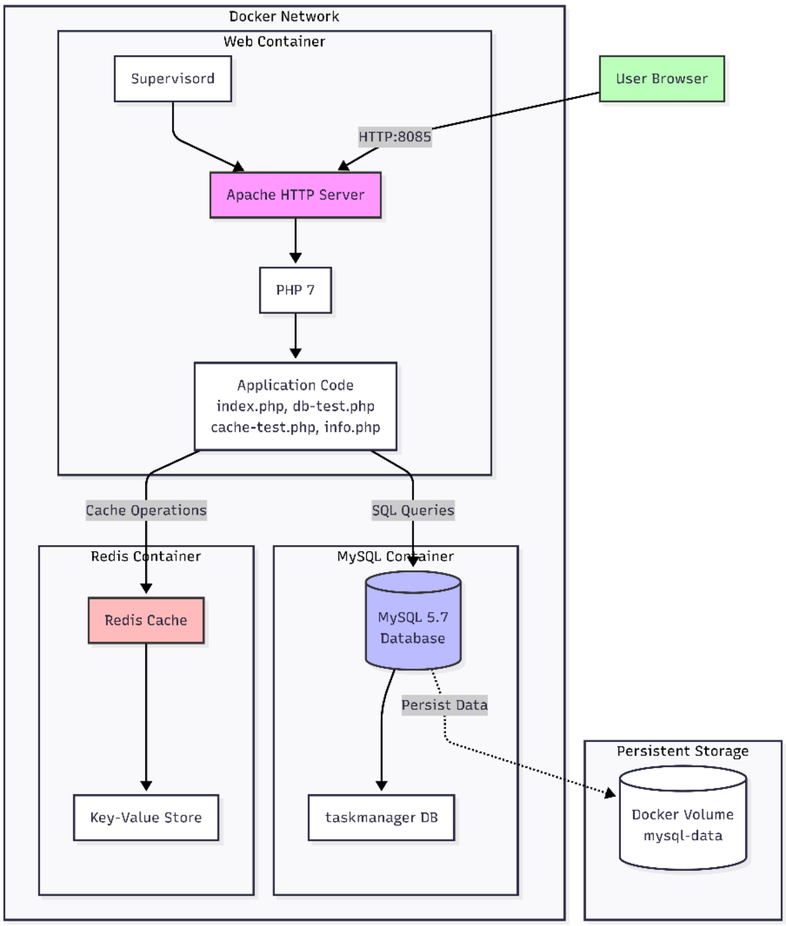




Hasil ini menunjukkan keberhasilan koneksi ke port `9999` pada host, yang dipetakan ke web server Nginx di dalam container. Tampilan halaman statis dengan tautan navigasi (Jumbotron, Blog, dll.) memverifikasi bahwa Nginx berhasil melayani konten statis. Lebih lanjut, ketika mengakses `test.php`, halaman informasi `PHP Version 7.2.33` berhasil ditampilkan, yang membuktikan bahwa Supervisord telah berhasil menjalankan dan mengelola dua layanan penting secara bersamaan (nginx dan php7-fpm).

3.5 Case 5 (Multi-Stack E-commerce Service (Web, Database, Cache))

- Gambar Arsitektur



Pada gambar Arsitektur Multi-Stack di atas, diperlihatkan tiga container utama yang saling terhubung melalui jaringan internal yang diorkestrasi oleh Docker Compose. Arsitektur ini bertujuan untuk mendemonstrasikan pemisahan tanggung jawab dan persistensi data untuk aplikasi web yang membutuhkan database dan caching.

Container pertama adalah Web Container yang berisi Apache HTTP Server dan PHP 7, berfungsi sebagai layer aplikasi utama yang melayani User Browser melalui port `8085`. Container kedua adalah MySQL Container, yang bertindak sebagai service Persistensi Data. Aplikasi di Web Container mengirim SQL Queries ke MySQL, dan data dipertahankan menggunakan Docker Volume yang terhubung ke Persistent Storage di host. Terakhir, container ketiga adalah Redis Container, yang melayani Cache Operations dan menyediakan Key-Value Store. Komunikasi di dalam stack ini dilakukan secara internal menggunakan service names (misalnya dari PHP ke `mysql` dan `redis`), memastikan setiap layanan berfungsi secara independen dan efisien.

- Cuplikan Kode / Script dan Penjelasan
  - docker-compose.yaml

```yaml
version: '3.8'

▷Run All Services
services:
  ▷Run Service
  web:
    build: ./platform
    container_name: taskmanager-web
    ports:
      - "8085:80"
    environment:
      - DB_HOST=mysql
      - DB_USER=root
      - DB_PASS=rootpassword
      - DB_NAME=taskmanager
      - REDIS_HOST=redis
      - REDIS_PORT=6379
    depends_on:
      - mysql
      - redis
    networks:
      - taskmanager-network
```

```yaml
  mysql:
    image: mysql:5.7
    container_name: taskmanager-mysql
    environment:
      - MYSQL_ROOT_PASSWORD=rootpassword
      - MYSQL_DATABASE=taskmanager
      - MYSQL_ROOT_HOST=%
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - taskmanager-network
    ports:
      - "3306:3306"

  ▷Run Service
  redis:
    image: redis:alpine
    container_name: taskmanager-redis
    ports:
      - "6379:6379"
    networks:
      - taskmanager-network

volumes:
  mysql-data:

networks:
  taskmanager-network:
    driver: bridge
```

File `docker-compose.yml` adalah blueprint orkestrasi, yang mendefinisikan dan menghubungkan tiga layanan (container) yang membentuk arsitektur web lengkap, yaitu web, mysql, dan redis. Layanan web dibangun dari `Dockerfile` lokal dan diekspos melalui port `8085` di Host ke port `80` Container (8085:80). Selain itu, service `mysql` dan `redis` juga memetakan port standar mereka (3306:3306 dan 6379:6379) ke Host, meskipun aplikasi web terhubung melalui jaringan internal menggunakan service names. Seluruh layanan dihubungkan oleh jaringan `taskmanager-network`, di mana Web Container menggunakan service names (mysql dan redis) sebagai hostname untuk koneksi. File ini juga mendefinisikan Docker Volume (mysql-data) untuk memastikan data database tersimpan secara persisten di host.

- Dockerfile

```dockerfile
FROM alpine:3.9 (last pushed 5 years ago)

RUN apk update && apk add --no-cache \
    apache2 \
    curl \
    supervisor \
    php7-apache2 \
    php7-intl \
    php7-openssl \
    php7-pecl-redis \
    php7-sqlite3 \
    php7-pdo_mysql \
    php7-mbstring \
    php7-json \
    php7-curl \
    php7-mysqli \
    mysql-client \
    apache2-ssl

RUN rm -rf /tmp/* /var/cache/apk/*

ADD start.sh /tmp/start.sh
ADD supervisord.conf /tmp/supervisord.conf
ADD httpd.conf /etc/apache2/httpd.conf
ADD php.ini /etc/php7/php.ini

RUN chmod +x /tmp/start.sh

EXPOSE 80

COPY html /var/www/localhost/htdocs

ENTRYPOINT ["sh", "/tmp/start.sh"]
```

`Dockerfile` ini memulai dengan Image `alpine:3.9` dan bertanggung jawab membangun Image Web Server utama. Ia menginstal Apache2 dan Supervisor, serta mengintegrasikan PHP 7 dengan semua ekstensi yang diperlukan, termasuk ekstensi koneksi MySQL (php7-pdo_mysql, php7-mysqli) dan Redis (php7-pecl-redis). File-file konfigurasi kustom (httpd.conf, php.ini, supervisord.conf) dan kode aplikasi disalin ke dalam Image, dan Container disiapkan untuk dieksekusi melalui script entrypoint yang menjalankan Supervisord.

- run-taskmanager.sh

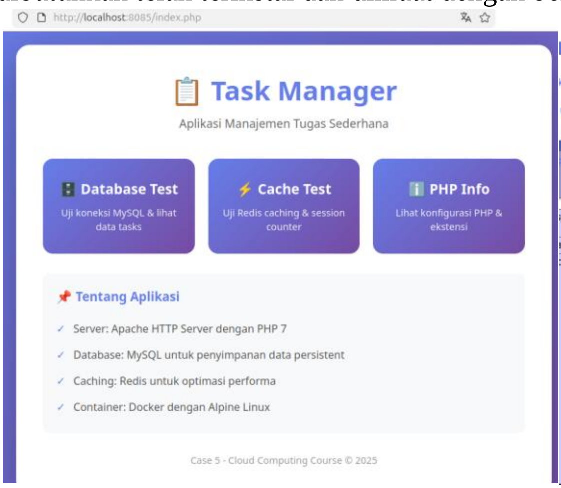
```
#!/bin/sh

docker run -d \
  --name taskmanager \
  -p 8085:80 \
  -e DB_HOST=mysql \
  -e DB_USER=root \
  -e DB_PASS=password \
  -e DB_NAME=taskmanager \
  -e REDIS_HOST=redis \
  -e REDIS_PORT=6379 \
  mytaskmanager:latest

echo "Task Manager container started!"
echo "Access the app at: http://localhost:8085"
```

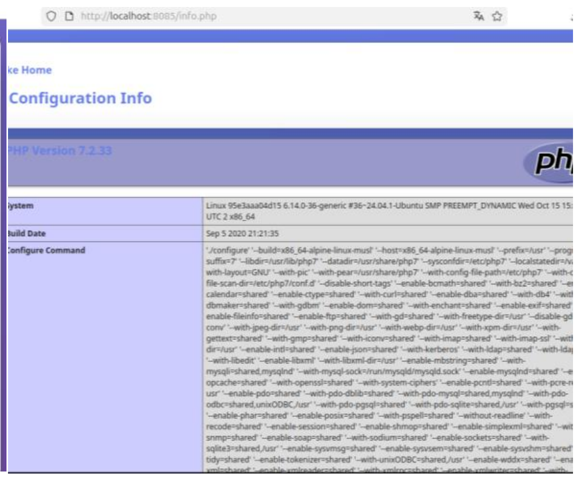
Meskipun script ini awalnya menggunakan perintah `docker run` tunggal yang tidak mencerminkan arsitektur multi-stack, tujuan fungsional script ini adalah untuk mengotomatisasi deployment seluruh sistem Case 5. Idealnya, script ini menjalankan perintah `docker-compose up --build -d`, yang berfungsi sebagai otomatisasi satu langkah untuk membangun Image web, membuat, menjalankan, dan menghubungkan ketiga container (web, mysql, redis) ke jaringan yang sama, sehingga aplikasi dapat diakses dengan cepat di port `8085`.

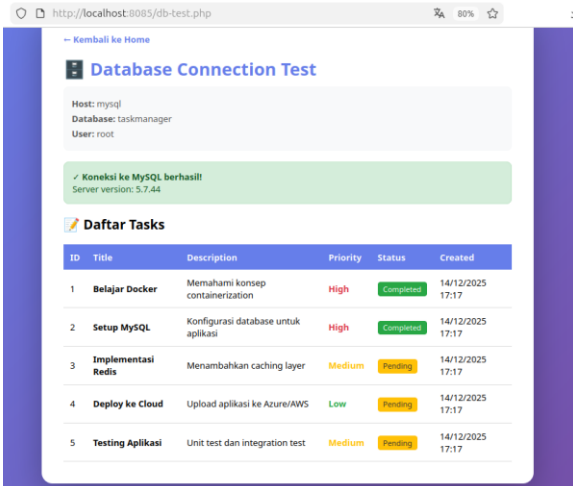
- Cara Menjalankan
  - Pertama kita hanya perlu menjalankan `docker-compose up -d`, lalu menjalankan `sudo bash build.sh` yang berada di folder /platform untuk ngebuild image, lalu terakhir kita coba menjalankan `sudo bash run-taskmanager.sh` untuk menjalankan container. Jika kita ingin mengecek apakah sudah berjalan dengan baik atau belum, kita tinggal menjalankan `docker ps` atau `docker ps -a`.

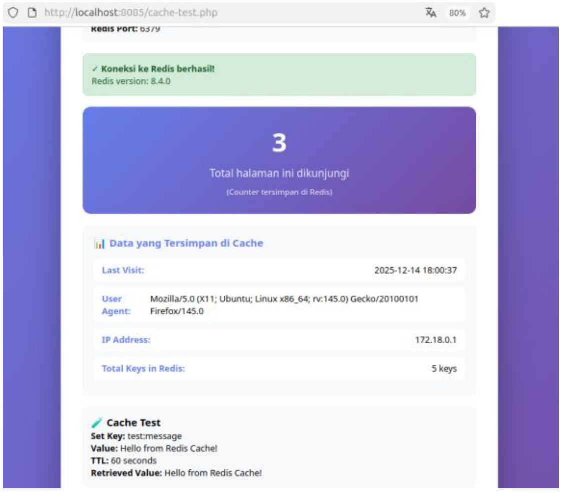
```
rafhi-sadipta@rafhi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas3/case5/runcontainer$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS
                NAMES
dd9965876b85   case5_web      "sh /tmp/start.sh"       56 seconds ago   Up 56 seconds   0.0.0.0:8085->80/tcp, [::]:8085-
>80/tcp         taskmanager-web
7267974e5c3f   redis:alpine   "docker-entrypoint.s…"   57 seconds ago   Up 57 seconds   0.0.0.0:6379->6379/tcp, [::]:637
9->6379/tcp     taskmanager-redis
8a5bb87b8c1e   mysql:5.7      "docker-entrypoint.s…"   57 seconds ago   Up 57 seconds   0.0.0.0:3306->3306/tcp, [::]:330
6->3306/tcp, 33060/tcp   taskmanager-mysql
```

  - Proses menjalankan arsitektur Multi-Service Case 5 diotomatisasi melalui script `run-taskmanager.sh`. Script ini memicu Docker Compose untuk membangun Image aplikasi web, dan kemudian menjalankan tiga container yang saling terhubung, yaitu, Web Server, Database (MySQL), dan Cache (Redis), serta mengatur Docker Volume untuk persistensi data `MySQL`. Setelah layanan aktif, aplikasi dapat diakses melalui URL `http://localhost:8085`, yang menampilkan `index.php` sebagai halaman navigasi utama, dan `info.php` sebagai laman diagnostik untuk memverifikasi bahwa semua ekstensi PHP yang dibutuhkan telah terinstal dan dimuat dengan benar di dalam Web Container.









Verifikasi keberhasilan integrasi arsitektur tiga tingkat dilakukan dengan mengakses halaman pengujian fungsional. Laman `db-test.php` digunakan untuk membuktikan bahwa Web Container berhasil terhubung, membaca, dan menulis data ke MySQL Container, memverifikasi layanan Persistensi Data. Sementara itu, laman `cache-test.php` memverifikasi koneksi dan fungsionalitas ke Redis Container, membuktikan keberhasilan implementasi caching layer yang dapat digunakan untuk meningkatkan kecepatan dan mengurangi beban database.

- Kapan Skenario ini cocok untuk dilakukan
  Skenario Multi-Stack Service menggunakan Docker Compose ini sangat cocok untuk aplikasi web dinamis tingkat menengah hingga kompleks yang memiliki persyaratan data dan kinerja yang spesifik. Pendekatan ini adalah transisi yang realistis dari container sederhana ke arsitektur microservices karena memisahkan tanggung jawab (Web, DB, Cache) ke dalam unit yang terisolasi. Pilihan arsitektur ini menjadi penting ketika:
  - Skalabilitas dan Kinerja Tinggi Diperlukan ketika aplikasi sering mengalami beban tinggi.
  - Persistensi Data Mutlak Diperlukan ketika data aplikasi harus dijamin tetap ada dan tidak hilang, bahkan saat container database diperbarui atau dihapus.
  - Service Isolation Dibutuhkan ketika setiap komponen harus dikembangkan, di-deploy, atau di-scale secara independen.
  - Aplikasi Production-Like Diperlukan untuk meniru lingkungan production modern di mana database dan web server dipisahkan ke dalam server atau container berbeda.

4. Kesimpulan

Dari seluruh rangkaian tugas ini, dapat disimpulkan bahwa Docker Image berperan sebagai blueprint utama dalam arsitektur container. Image mendefinisikan bagaimana sebuah aplikasi dijalankan, mulai dari kode aplikasi, runtime, hingga seluruh dependensi sistem yang dibutuhkan. Dengan Image yang baik, aplikasi dapat dijalankan secara konsisten di berbagai lingkungan tanpa perbedaan konfigurasi.

Setiap case memperkuat pemahaman mengenai pentingnya Docker Image yang beragam. Case 1 mengajarkan konsep membangun Image untuk single-service dan Case 2 menunjukkan fleksibilitas dari Docker untuk meng-container-kan lingkungan non-web yang kompleks, seperti antarmuka grafis (GUI) beserta semua dependensi layanannya. Lebih lanjut, Case 4 memperlihatkan kompleksitas Image yang harus menginstal dan mengintegrasikan beberapa proses daemon (Nginx, PHP-FPM) sekaligus di dalam container tunggal.

Secara spesifik, Case 3 menyoroti tantangan build time dan manajemen Build Context, di mana masalah akses aset eksternal harus diselesaikan agar proses `docker build` berhasil menyalin dan mengintegrasikan konfigurasi yang diperlukan ke dalam Image akhir. Sementara itu, Case 5 menjadi contoh Image yang paling kaya dependensi, di mana Image aplikasi web harus secara eksplisit dibangun dengan ekstensi koneksi MySQL dan Redis. Setelah Image yang stabil dibangun, Docker Compose kemudian mengambil peran penting sebagai alat penyederhana deployment yang mengelola dan mengintegrasikan semua Image yang telah terisolasi dan stabil tersebut, mengatur jaringan dan persistensi data.

Secara keseluruhan, studi case ini menunjukkan bahwa kemampuan untuk secara konsisten membangun Docker Image yang teruji adalah prasyarat utama, memberikan fondasi kuat untuk arsitektur containerized yang kompleks.