KOMPUTASI AWAN (T)

TUGAS 3

Dosen Pengampu:

Royyana Muslim Ijtihadie, S.Kom.,M.Kom., Ph.D.

Disusun Oleh:

Hilmi Fawwaz Sa’ad

NRP. 5025221103

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS

DEPARTEMEN TEKNIK INFORMATIKA

SURABAYA

2025/2026

Tugas 3

Repository Github: https://github.com/hilmifawwazsaad/CloudComp_Class_2025/tree/main/task-3

Pendahuluan

Containerization telah menjadi standar industri dalam pengembangan dan deployment aplikasi modern karena kemampuannya menyediakan environment yang konsisten, terisolasi, dan mudah direplikasi di berbagai platform. Docker, sebagai salah satu teknologi containerization terpopuler, memungkinkan developer untuk membuat, mengelola, dan mendistribusikan aplikasi beserta seluruh dependensinya dalam bentuk container yang portable dan lightweight. Docker Images merupakan template read-only yang berisi sistem operasi, aplikasi, dependencies, dan konfigurasi yang diperlukan untuk menjalankan container, sedangkan Dockerfile adalah file teks berisi serangkaian instruksi yang mendefinisikan bagaimana sebuah Docker Image dibangun secara otomatis dan reproducible. Penggunaan Docker Images yang dibuat melalui Dockerfile memberikan kontrol penuh terhadap konfigurasi environment, memastikan bahwa aplikasi berjalan dengan cara yang sama di mesin development, staging, maupun production.

Tugas yang Diberikan

Tugas yang diberikan adalah menjalankan semua studi kasus (cases) yang ada pada repository https://github.com/rm77/cloud2023/tree/master/containers/compose/images. Setiap studi kasus harus dijalankan dengan konfigurasi dan pengelolaan layanan yang berbeda, termasuk pembuatan Docker Image. Lingkup pengerjaan mencakup empat studi kasus utama dari repository tersebut, ditambah satu studi kasus tambahan sebagai bentuk eksplorasi dan pengembangan berdasarkan referensi yang sama. Adapun penjelasan masing-masing studi kasus adalah sebagai berikut:

1. Membuat Docker Image dan menjalankan web server Nginx yang meng-host layanan aplikasi PHP
2. Membuat Docker Image dan menjalankan aplikasi Linux sederhana pada browser menggunakan VNC
3. Membuat Docker Image dan menjalankan web server Nginx menggunakan protokol HTTP
4. Membuat Docker Image dan menjalankan web server Nginx dengan layanan aplikasi PHP yang ditampilkan melalui browser
5. [CASE TAMBAHAN] Membuat Docker Image dan menjalankan aplikasi QR Code Generator berbasis Flask

Setelah seluruh studi kasus berhasil dijalankan, dilakukan proses dokumentasi yang mencakup penjelasan terhadap script/code yang digunakan, langkah menjalankan tiap case, alur proses yang terjadi, serta hasil yang diperoleh dari masing-masing program.

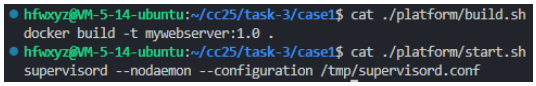
Implementasi dan Pembahasan

- ***Case 1***

Pada studi kasus ini, pembuatan image dilakukan menggunakan Dockerfile dengan nama `mywebserver:1.0`. Proses pengerjaan melibatkan dua komponen utama selain Dockerfile itu sendiri. Pertama, script `platform/build.sh` yang digunakan untuk membangun Docker Image. Kedua, script `runcontainer/run-mywebserver.sh` yang berfungsi untuk melakukan konfigurasi dan menjalankan container.

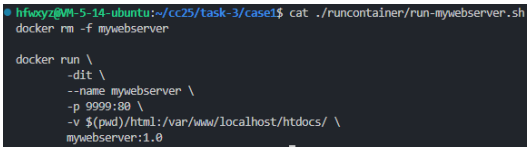
Gambar 1. Script Dockerfile

Dockerfile ini menggunakan Alpine linux 3.9 sebagai base image dan menginstall Apache2 sebagai web server beserta PHP 7 dengan berbagai extensions lengkap, serta utilities pendukung seperti Composer, mysql-client, socat, dan Supervisor untuk menjalankan multiple services secara bersamaan. Container dikonfigurasi untuk membuka port 80 untuk web server dan port 9999 untuk layanan tambahan, dengan environment variables untuk mengatur session management. Directory `/var/www/localhost/htdocs` dijadikan volume yang memungkinkan update kode tanpa rebuild image, dan saat container dijalankan, script `start.sh` akan dieksekusi sebagai entry point untuk menginisialisasi Supervisor yang kemudian menjalankan Apache beserta service-service lainnya.



Gambar 2. Script build.sh dan start.sh

Script `build.sh` merupakan shortcut untuk membangun Docker image dengan menjalankan perintah `docker build -t mywebserver:1.0 .` yang akan mencari Dockerfile di current directory dan menghasilkan image bernama `mywebserver` dengan tag `1.0`. Sementara itu, script `start.sh` berfungsi sebagai entry point utama container yang menjalankan perintah `supervisord --nodaemon --configuration /tmp/supervisord.conf` untuk menginisialisasi Supervisor sebagai proses foreground yang mengelola multiple services sekaligus.



Gambar 3. Script run-mywebserver.sh

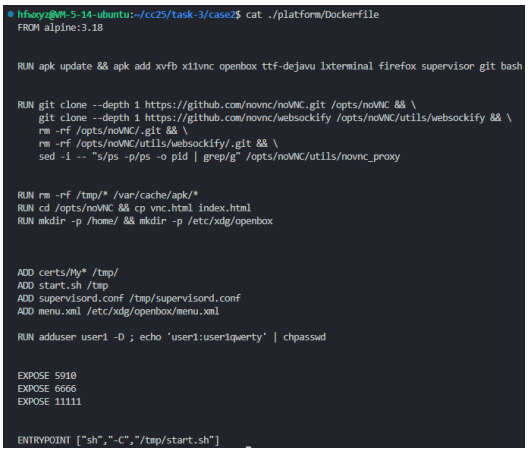
Script `run-mywebserver.sh` pertama-tama menghapus container lama bernama `mywebserver`, kemudian menjalankan container baru dengan `docker run -dit --name mywebserver -p 9999:80 -v $(pwd)/html:/var/www/localhost/htdocs/ mywebserver:1.0` yang akan membuat container berjalan di background, melakukan port mapping dari port 80 di container ke port 9999 di host sehingga web server dapat diakses melalui `http://localhost:9999`, serta melakukan volume mapping yang mengikat folder `html/` di host ke `/var/www/localhost/htdocs/` di container sebagai document root Apache.

Gambar 4. Hasil localhost pada port 9999

Setelah menjalankan script `build.sh` dan kemudian `run-mywebserver.sh`, layanan dapat diakses melalui `http://localhost:9999`. Pada halaman tersebut terlihat bahwa container berhasil berjalan, ditandai dengan munculnya pesan “Hello World” di bagian kiri atas layar serta tampilan informasi mengenai dependensi PHP yang digunakan.

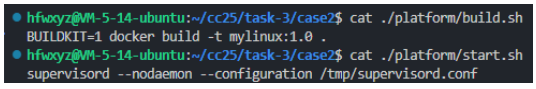
- ***Case 2***

Pada studi kasus ini, pembuatan image dilakukan menggunakan Dockerfile dengan nama `mylinux`. Proses pengerjaan melibatkan dua komponen utama selain Dockerfile itu sendiri. Pertama, script `platform/build.sh` yang digunakan untuk membangun Docker Image. Kedua, script `runcontainer/run-mylinux.sh` yang berfungsi untuk melakukan konfigurasi dan menjalankan container.



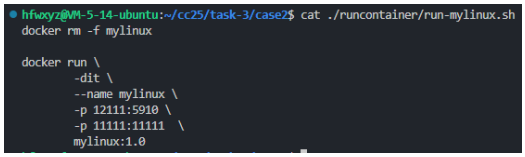
Gambar 5. Script Dockerfile

Dockerfile ini membangun lingkungan remote desktop Linux berbasis Alpine 3.18 yang dapat diakses melalui browser menggunakan VNC dan noVNC. Container ini menginstal komponen utama seperti Xvfb untuk virtual display, x11vnc sebagai VNC server, Openbox sebagai window manager, Firefox, serta terminal dan font pendukung, sekaligus melakukan clone noVNC dan websockify untuk menyediakan akses VNC berbasis HTML5. Container membuat user `user1` dengan password `user1qwerty` dan membuka port 5910, 6666, serta 11111 untuk layanan VNC, websockify, dan noVNC. Seluruh proses Xvfb, Openbox, x11vnc, websockify, dan Firefox dijalankan dan dikelola oleh Supervisor melalui entry point `start.sh`.



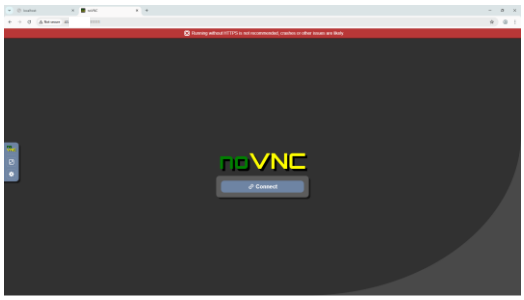
Gambar 6. Script build.sh dan start.sh

Script `build.sh` menggunakan perintah `BUILDKIT=1 docker build -t mylinux:1.0.` untuk membangun Docker image dengan mengaktifkan Docker BuildKit yang membuat proses build lebih cepat, efisien, dan menghasilkan image lebih kecil melalui optimasi layer dan parallel build, kemudian menyimpan hasilnya sebagai image bernama `mylinux` dengan tag `1.0`. Script `start.sh` memiliki fungsi yang sama dengan case sebelumnya, yaitu menjalankan Supervisor sebagai main process container, namun kali ini mengelola service-service untuk desktop environment.



Gambar 7. Script run-mylinux.sh

Script `run-mylinux.sh` berfungsi untuk menjalankan container baru dari image `mylinux:1.0`. Pertama, script menghapus container lama bernama `mylinux` (jika ada). Setelah itu, script membuat container baru dengan nama yang sama, berjalan di background, dan memetakan port 5910 di container ke 12111 di host untuk akses VNC, serta port 11111 di container ke 11111 di host untuk akses noVNC atau service lain. Container kemudian dijalankan menggunakan image `mylinux:1.0`.



Gambar 8. Hasil localhost pada port 11111

Setelah menjalankan script `build.sh` dan kemudian `run-mylinux.sh`, layanan dapat diakses melalui `http://localhost:11111`. Pada halaman tersebut terlihat bahwa container berhasil berjalan, ditandai dengan munculnya tampilan VNC untuk menjalankan desktop environment Linux lengkap dengan Firefox dan terminal melalui browser.

- ***Case 3***

Pada studi kasus ini, pembuatan image dilakukan menggunakan Dockerfile dengan nama `mywebserver:2.0`. Proses pengerjaan melibatkan dua komponen utama selain Dockerfile itu sendiri. Pertama, script `build.sh` yang digunakan untuk membangun Docker Image. Kedua, script `run-server.sh` yang berfungsi untuk melakukan konfigurasi dan menjalankan container.

```
hfwxyz@VM-5-14-ubuntu:~/cc25/task-3/case3$ cat ./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/
```

Gambar 9. Script Dockerfile

Dockerfile ini menggunakan image nginx:1.15.12-alpine sebagai base. File konfigurasi `nginx.conf` dari folder `nginx-conf/` disalin ke dalam container. Pada script ini terdapat dua instruksi penyalinan, yaitu menggunakan ADD ke `/etc/nginx/conf.d/` dan menggunakan COPY ke `/etc/nginx/conf.d/default.conf`. Selanjutnya, folder `html/` disalin ke `/var/www/html/`, yang menjadi directory untuk file yang akan di-serve oleh Nginx.

```
hfwxyz@VM-5-14-ubuntu:~/cc25/task-3/case3$ cat build.sh
docker build -t mywebserver:2.0 -f Dockerfile ../../task-2/case1/
```

Gambar 10. Script build.sh

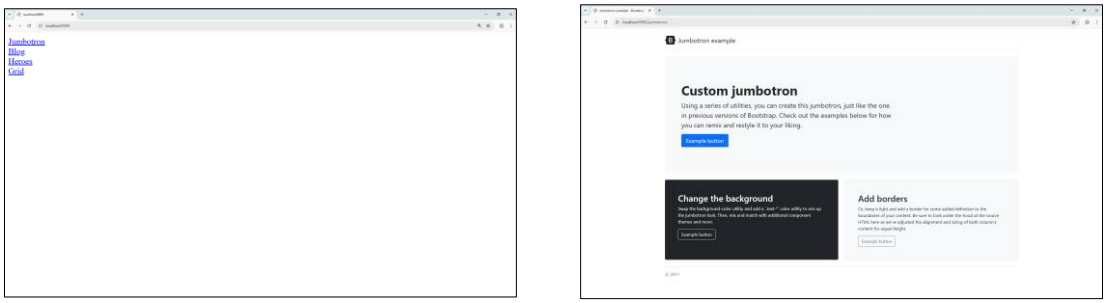
Script build.sh berfungsi untuk membangun Docker Image `mywebserver:2.0`. Dockerfile berada di folder `task-3/case3/`, namun proses build menggunakan build context dari folder `task-2/case1/` karena file-file yang dibutuhkan Dockerfile seperti folder `html/` dan `nginx-conf/` berada di folder tersebut. Script ini juga saya konfigurasi sendiri agar pathnya bisa sesuai.

```
hfwxyz@VM-5-14-ubuntu:~/cc25/task-3/case3$ cat run-server.sh
docker rm -f webserver2
docker run -dit \
        --name webserver2 \
        -p 9999:80 \
        mywebserver:2.0
```

Gambar 11. Script run-server.sh

Script `run-server.sh` ini digunakan untuk menjalankan container web server. Pertama, script menghapus container lama bernama `webserver2` (jika ada). Setelah itu, script menjalankan container baru dengan nama `webserver2` dan berjalan di background serta memetakan port 80 di dalam

container ke port 9999 di host sehingga web server dapat diakses melalui `localhost:9999`. Container tersebut dijalankan menggunakan image `mywebserver:2.0`.



Gambar 12. Hasil localhost pada port 9999

Setelah menjalankan script `build.sh` dan kemudian `run-server.sh`, layanan dapat diakses melalui `http://localhost:9999`. Pada halaman tersebut terlihat bahwa container berhasil berjalan, ditandai dengan munculnya tampilan `index.html` dan template bootstrap yang dilayani oleh web server Nginx.

- ***Case 4***

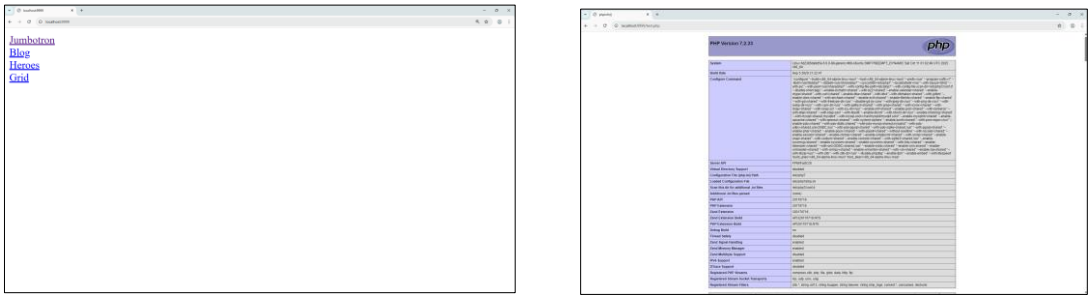
Pada studi kasus ini, pembuatan image dilakukan menggunakan Dockerfile dengan nama `mywebserver:2.1`. Sebenarnya untuk studi kasus keempat ini hampir sama dengan studi kasus sebelumnya, hanya saja yang terdapat penambahan aplikasi PHP pada path tertentu.

```
hfwxyz@VM-5-14-ubuntu:~/cc25/task-3/case4$ cat ./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"]
```

Gambar 13. Script Dockerfile

Dockerfile ini membangun image berbasis nginx:1.15.12-alpine yang menjalankan Nginx + PHP-FPM dalam satu container menggunakan Supervisor. Proses dimulai dengan instalasi php7-fpm, supervisor, git, dan curl melalui apk. File konfigurasi `nginx.conf` disalin ke `/etc/nginx/conf.d/`, folder `html/` ke `/var/www/html/`, serta `start.sh` dan `supervisord.conf` untuk mengatur eksekusi proses. Supervisor memastikan Nginx dan PHP-FPM berjalan bersamaan, dengan `start.sh` sebagai entry point yang dijalankan saat container dimulai.

Untuk konfigurasi `build.sh` dan `run-server.sh` hanya berbeda pada bagian image yang digunakan yaitu `mywebserver:2.1`.

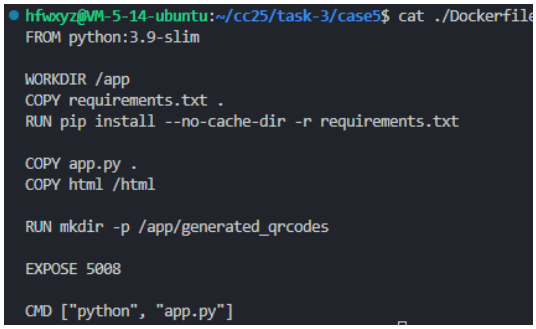


Gambar 14. Hasil localhost port 9999 dan /test.php

Setelah menjalankan script `build.sh` dan kemudian `run-server.sh`, layanan dapat diakses melalui `http://localhost:9999`. Pada halaman tersebut terlihat bahwa container berhasil berjalan, ditandai dengan munculnya tampilan `index.html` dan template bootstrap yang dilayani oleh web server Nginx. Selain itu, terdapat pada aplikasi PHP dan detail dependensinya yang berjalan di `http://localhost:9999/test.php`.

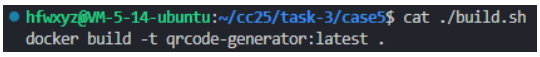
- ***Case 5***

Pada studi kasus eksplorasi ini, saya mengembangkan aplikasi QR Code Generator berbasis Flask yang dijalankan dengan Dockerfile dengan nama `qrcode-generator:latest`. Studi kasus ini menggunakan antarmuka yang disediakan oleh `html/index.html` untuk menerima input teks atau URL dari pengguna, lalu diproses oleh Flask melalui `app.py`.



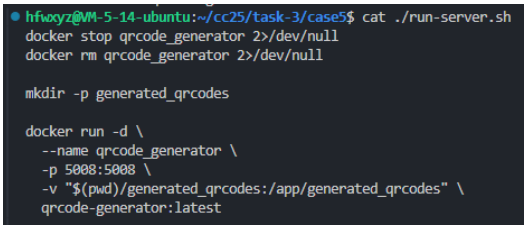
Gambar 15. Script Dockerfile

Dockerfile ini membuat image aplikasi Python 3.9 berbasis python:3.9-slim yang berfungsi sebagai generator QR Code. Working directory diset ke `/app`, kemudian file `requirements.txt` disalin dan dependencies diinstall tanpa cache menggunakan `pip` agar image tetap ringan. File utama aplikasi `app.py` dan folder `html/` (frontend) disalin ke dalam container, lalu folder `/app/generated_qrcodes` dibuat untuk menyimpan hasil QR Code. Aplikasi di-expose pada port 5008 dan dijalankan menggunakan perintah `python app.py` sebagai CMD default saat container dimulai.



Gambar 16. Script build.sh

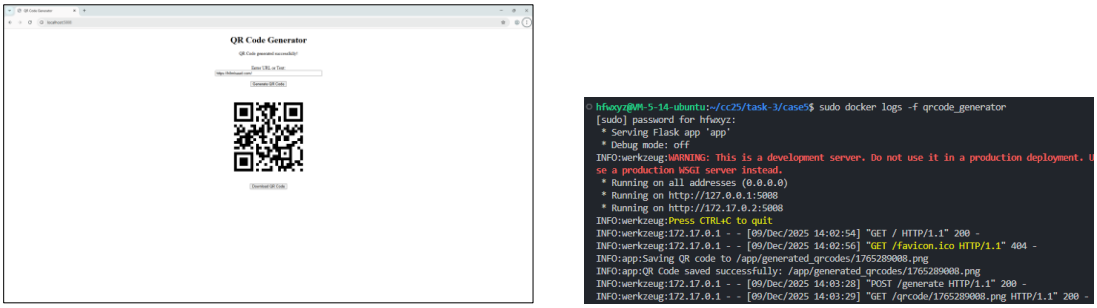
Script `build.sh` ini menjalankan perintah `docker build -t qrcode-generator:latest .` yang berfungsi untuk membangun Docker Image dari Dockerfile di direktori saat ini dengan nama `qrcode-generator` dan tag `latest` sebagai penanda versi terbaru.



Gambar 17. Script run-server.sh

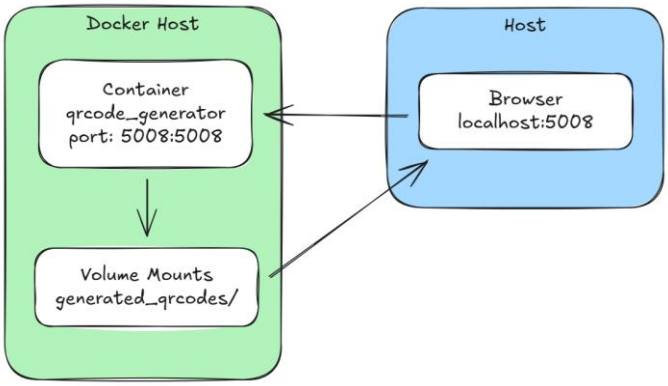
Script `run-server.sh` digunakan untuk menyiapkan dan menjalankan container aplikasi QR Code generator. Pertama, script menghentikan dan menghapus container bernama `qrcode_generator` jika sebelumnya sudah ada, dan kedua perintah tersebut dibungkam sehingga tidak menampilkan error apabila container belum pernah dibuat. Setelah itu, script membuat folder `generated_qrcodes` di host sebagai tempat menyimpan hasil QR Code. Selanjutnya, perintah docker run menjalankan container baru dengan nama `qrcode_generator` secara background, memetakan port 5008 di container ke port 5008 di host, dan melakukan mount volume dari folder lokal `generated_qrcodes`

ke `/app/generated_qrcodes` di dalam container. Container dijalankan menggunakan image `qrcode-generator:latest` yang telah dibangun sebelumnya.



Gambar 18. Halaman localhost:5008 (kiri) dan logs dari container (kanan)

Setelah menjalankan script `build.sh` dan kemudian `run-server.sh`, layanan dapat diakses melalui `http://localhost:5008`. Pada halaman tersebut terlihat bahwa container berhasil berjalan, ditandai dengan munculnya tampilan `index.html` dan bisa generate QR Code serta dapat dilihat melalui logs container.



Gambar 19. Arsitektur Case 5

Untuk alurnya sendiri, pengguna mengakses aplikasi melalui browser di `localhost:5008`, yang meneruskan request ke container `qrcode_generator` untuk diproses oleh Flask. Setelah QR code dibuat, file hasil disimpan melalui volume mounts (`generated_qrcodes/`) yang menghubungkan direktori container dengan host, sehingga output tersimpan secara persisten dan dapat diakses dari luar container.

Komputer atau VM yang digunakan memiliki spesifikasi 4 GB RAM, 2 vCPU, dan 60 GB storage, dengan IP Address 43.xxx.xxx.xxx atau localhost. Pada studi kasus ini, port yang digunakan adalah 5008 sebagai port akses aplikasi Flask di dalam container Docker, yang kemudian di forward ke host sehingga dapat dibuka melalui browser pada alamat localhost:5008.

Penggunaan Docker Images dengan Dockerfile pada aplikasi QR Code Generator ini diterapkan untuk memastikan konsistensi environment di berbagai sistem operasi, mengingat aplikasi memiliki dependencies Python yang spesifik dan memerlukan versi Python 3.9 yang terisolasi. Pendekatan containerization ini memudahkan proses deployment ke environment production maupun development tanpa memerlukan konfigurasi manual berulang, serta mendukung portabilitas aplikasi ketika didistribusikan kepada tim atau dipindahkan ke server berbeda. Penggunaan volume mounts pada direktori `generated_qrcodes/` memungkinkan persistensi data output di luar container, sehingga hasil QR code tetap tersimpan meskipun container dihentikan atau di-restart. Selain itu, Dockerfile memfasilitasi version control terhadap konfigurasi environment dan mempermudah

maintenance jangka panjang, karena setiap perubahan dependencies atau konfigurasi dapat dilakukan dengan rebuild image tanpa mengubah setup di host machine, sehingga mendukung proses CI/CD yang lebih efisien dan skalabilitas aplikasi di masa mendatang.

Kesimpulan

Dalam laporan ini, berhasil diimplementasikan empat studi kasus dari repository yang diberikan serta dikembangkan Case 5 sebagai eksplorasi tambahan, yaitu layanan Flask untuk generator QR Code menggunakan Docker Images dan Dockerfile. Laporan mencakup implementasi web server Nginx dengan PHP (Case 1 dan 4), remote desktop Linux berbasis VNC (Case 2), web server Nginx dengan HTTP (Case 3), dan aplikasi QR Code Generator berbasis Flask (Case 5). Setiap studi kasus mendemonstrasikan penggunaan Dockerfile untuk membangun image dengan konfigurasi spesifik, pengelolaan multi-service menggunakan Supervisor, serta implementasi port mapping dan volume mounts untuk persistensi data dan aksesibilitas layanan. Seluruh script, diagram alur, dan screenshot yang disertakan menunjukkan bagaimana setiap layanan dikonfigurasi dan dijalankan dengan benar. Pendekatan containerization terbukti efektif dalam memastikan konsistensi environment, isolasi dependencies, kemudahan deployment, dan portabilitas aplikasi lintas platform, sehingga mendukung proses development yang lebih efisien, maintenance jangka panjang, serta skalabilitas aplikasi di masa mendatang dengan implementasi CI/CD yang optimal.