KOMPUTASI AWAN (T)

TUGAS 1

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 1

Repository Github: https://github.com/hilmifawwazsaad/CloudComp_Class_2025/tree/main/task-1

Pendahuluan

Tugas ini disusun untuk memberikan pemahaman yang lebih mendalam mengenai konsep containerization menggunakan Docker melalui penerapan beberapa studi kasus yang telah disediakan, ditambah satu studi kasus baru yang dirancang secara mandiri lengkap dengan arsitektur, script, serta dokumentasinya. Docker sendiri merupakan platform kontainerisasi yang memungkinkan aplikasi dan dependensinya dikemas dalam lingkungan terisolasi sehingga dapat dijalankan secara konsisten pada berbagai sistem. Sejak diperkenalkan pada tahun 2013 oleh Solomon Hykes, Docker telah menjadi solusi populer untuk mempermudah proses pengembangan, pengujian, dan deployment aplikasi tanpa harus khawatir terhadap perbedaan konfigurasi antar mesin.

Tugas yang Diberikan

Tugas yang diberikan adalah menjalankan semua studi kasus (cases) yang ada pada repository https://github.com/rm77/cloud2023/tree/master/containers/docker. Setiap studi kasus harus dijalankan menggunakan Docker dengan konfigurasi serta pengelolaan container sesuai kebutuhan layanan. Lingkup pengerjaan mencakup tiga 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. Menjalankan program background process pada Docker yang berfungsi mengambil atau menyimpan Chuck Norris jokes dari API dan menyimpannya ke dalam folder `files/` pada host.
2. Menjalankan static webserver berbasis Python untuk menyajikan file HTML yang berada pada host.
3. Menjalankan layanan database MySQL dan phpMyAdmin di dalam Docker sehingga keduanya saling terhubung dan phpMyAdmin dapat diakses melalui antarmuka web.
4. [CASE TAMBAHAN] Mengembangkan program background process dengan mekanisme penyimpanan data ke dalam database container alih-alih menyimpan langsung ke host.

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 terdapat dua file utama yang digunakan. File pertama adalah `run_process.sh`, yaitu script untuk menjalankan container Docker yang akan mengambil Chuck Norris jokes dari API setiap 8 detik dan menyimpannya ke direktori pada host. File kedua adalah `script/getjokes.sh`, yaitu script yang dieksekusi di dalam container untuk melakukan proses pengambilan dan penyimpanan jokes tersebut.

```
hfwxyz@VM-5-14-ubuntu:~/cc25/task-1/case1$ cat run_process.sh
#!/bin/sh
docker rm -f myprocess1

docker container run \
        --name myprocess1 \
        -dit \
        -e DELAY=8 \
        -v $(pwd)/files:/data \
        -v $(pwd)/script:/script \
        --workdir /data \
        alpine:3.18 \
        /bin/sh /script/getjokes.sh
```

Gambar 1. Script run_process.sh

Ketika script ini dijalankan, langkah pertama yang dilakukan adalah menghapus container lama bernama `myprocess1` jika container tersebut masih ada. Setelah itu, script membuat container baru menggunakan image alpine:3.18 dan mengeksekusi file `/script/getjokes.sh` di dalamnya. Container juga diberikan environment variable DELAY=8 sebagai interval waktu untuk pengambilan jokes. Selanjutnya, dilakukan proses mounting direktori, yaitu `$(pwd)/files` ke `/data` sebagai lokasi penyimpanan output, serta `$(pwd)/script` ke `/script` agar script `getjokes.sh` dapat dijalankan di dalam container. Terakhir, container dijalankan dalam mode background menggunakan opsi `-dit`.

```
hfwxyz@VM-5-14-ubuntu:~/cc25/task-1/case1$ cat script/getjokes.sh
#!/bin/sh

apk update && apk add curl jq

URL=https://api.chucknorris.io/jokes/random
LOKASI=/data

echo will run every $DELAY seconds

while true;
do
        date=$(date '+%Y-%m-%d_%H:%M:%S')
        echo processing at $date
        fname="output_$date.txt"
        curl -sL $URL | jq '.value' > $fname
        sleep $DELAY
```

Gambar 2. Script getjokes.sh

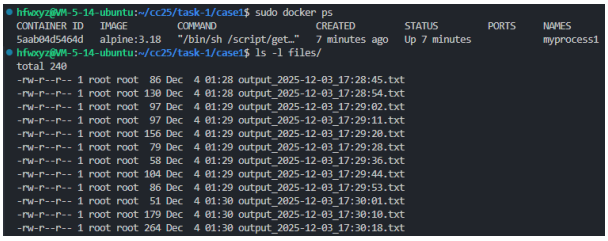
Di dalam container, script tersebut terlebih dahulu menginstal dependensi yang dibutuhkan, yaitu `curl` untuk melakukan permintaan ke API dan `jq` untuk memproses data JSON. Setelah itu, script menetapkan URL API dan folder output `/data` sebagai lokasi penyimpanan hasil. Program kemudian masuk ke dalam infinite loop yang berjalan terus-menerus. Pada setiap iterasi, script mengambil tanggal dan waktu saat ini untuk membentuk nama file, misalnya `output_2025-12-03_14:22:10.txt`. Selanjutnya, script melakukan permintaan API menggunakan curl ke endpoint https://api.chucknorris.io/jokes/random dan mengekstrak hanya field `.value`, yaitu teks joke-nya, menggunakan `jq`. Hasil tersebut disimpan ke dalam file di direktori `/data`. Setelah proses penyimpanan selesai, script menunggu selama `$DELAY` detik (secara default 8 detik) sebelum mengulangi langkah yang sama.

```
hfwxyz@VM-5-14-ubuntu:~/cc25/task-1/case1$ ./run_process.sh
bash: ./run_process.sh: Permission denied
hfwxyz@VM-5-14-ubuntu:~/cc25/task-1/case1$ chmod +x run_process.sh
hfwxyz@VM-5-14-ubuntu:~/cc25/task-1/case1$ ./run_process.sh
permission denied while trying to connect to the docker API at unix:///var/run/docker.sock
docker: permission denied while trying to connect to the docker API at unix:///var/run/dock
er.sock

Run 'docker run --help' for more information
hfwxyz@VM-5-14-ubuntu:~/cc25/task-1/case1$ sudo ./run_process.sh
[sudo] password for hfwxyz:
Error response from daemon: No such container: myprocess1
Unable to find image 'alpine:3.18' locally
3.18: Pulling from library/alpine
44cf07d57ee4: Pull complete
Digest: sha256:de0eb0b3f2a47ba1eb89389859a9bd88b28e82f5826b6969ad604979713c2d4f
Status: Downloaded newer image for alpine:3.18
5aab04d5464d7084abbb8e099cfd059eb8fb72c34feaa3174e4176d2a377dbe3
```

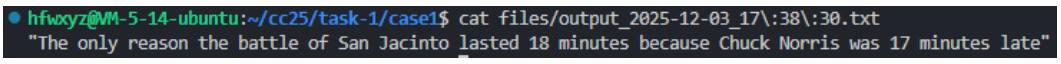
Gambar 3. Proses eksekusi script run_process.sh

Saat menjalankan `run_process.sh`, muncul error permission denied karena file belum memiliki izin eksekusi. Setelah diberi izin menggunakan `chmod +x`, script dapat dijalankan, tetapi muncul error baru karena user tidak memiliki akses ke Docker daemon. Hal ini menyebabkan Docker tidak bisa dijalankan tanpa `sudo`, sehingga script dieksekusi ulang dengan `sudo`. Setelah itu Docker melaporkan bahwa container `myprocess1` tidak ditemukan dan kemudian mencoba membuat container baru menggunakan image `alpine:3.18`. Karena image tersebut belum tersedia di sistem, Docker otomatis melakukan pull dari Docker Hub hingga image siap digunakan.



Gambar 4. Menjalankan container

Gambar tersebut menunjukkan bahwa container Docker berhasil berjalan dengan nama `myprocess1`, terlihat melalui perintah `sudo docker ps`. Container ini kemudian secara otomatis menghasilkan file output di dalam folder `/files` pada host.

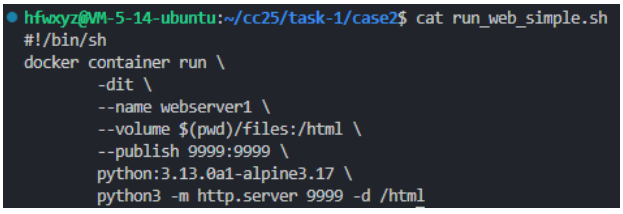


Gambar 5. Isi salah satu file txt

Jika file `.txt` sudah terisi dengan joke Chuck Norris yang diambil oleh script, artinya container berfungsi dengan baik. Proses ini berjalan otomatis di latar belakang tanpa perlu tindakan tambahan apa pun dari kita.

- ***Case 2***

Pada studi kasus ini terdapat dua file utama yang digunakan. File pertama adalah `run_web_simple.sh`, yaitu sebuah script yang bertugas menjalankan web server statis menggunakan Python di dalam container Docker untuk menyajikan konten web. File kedua adalah `/files/index.html`, yaitu halaman HTML yang akan ditampilkan oleh web server tersebut.

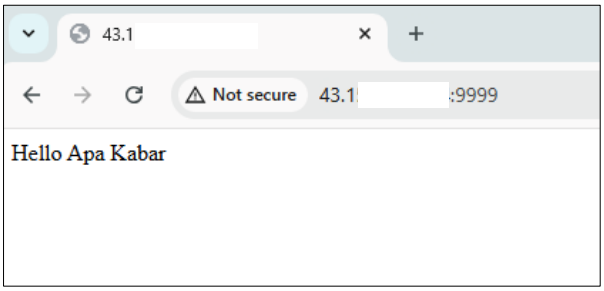


Gambar 6. Script run_web_simple.sh

Ketika script ini dijalankan, ia akan membuat container baru bernama `webserver1` menggunakan image `python:3.13.0a1-alpine3.17`. Container tersebut kemudian menjalankan web server bawaan Python dengan perintah `python3 -m http.server 9999 -d /html`. Port 9999 pada container dipetakan ke port 9999 pada host sehingga layanan dapat diakses melalui `http://localhost:9999`. Selain itu, dilakukan bind mount dari direktori `$(pwd)/files` pada host ke `/html` di dalam container.

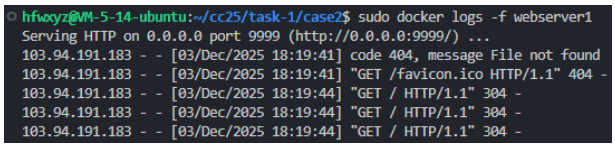
Gambar 7. Menjalankan container

Sama seperti sebelumnya, file `runwebsimple.sh` terlebih dahulu harus diberi izin eksekusi menggunakan perintah `chmod +x`. Setelah itu, script dijalankan dengan menggunakan `sudo` agar memiliki akses penuh ke Docker daemon. Pada output terlihat bahwa Docker berhasil menarik image Python dan menjalankan container baru yang memetakan port 9999 pada host ke port 9999 di dalam container. Container tersebut berjalan dengan nama `webserver1`.



Gambar 8. Hasil localhost pada port 9999

Untuk memverifikasi bahwa container telah berjalan dengan benar, halaman web dapat diakses melalui `localhost` atau, dalam kasus ini, melalui `IP VPS` pada port 9999. Jika container berfungsi sebagaimana mestinya, maka akan tampil pesan “Hello Apa Kabar” pada browser.

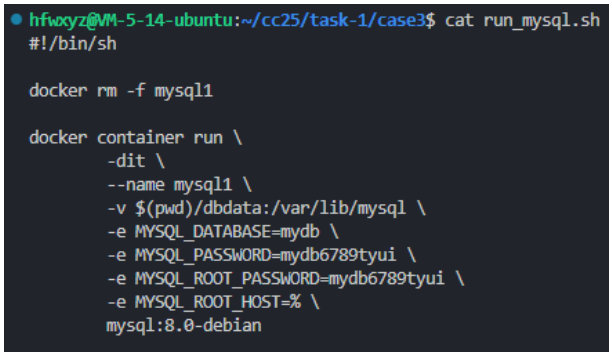


Gambar 9. Log dari container webserver1

Sebagai tambahan, kita juga dapat melihat log dari container `webserver1` untuk memastikan bahwa halaman tersebut benar-benar diakses. Log ini menampilkan setiap permintaan yang masuk ke web server, sehingga membantu memverifikasi aktivitas dan status container.

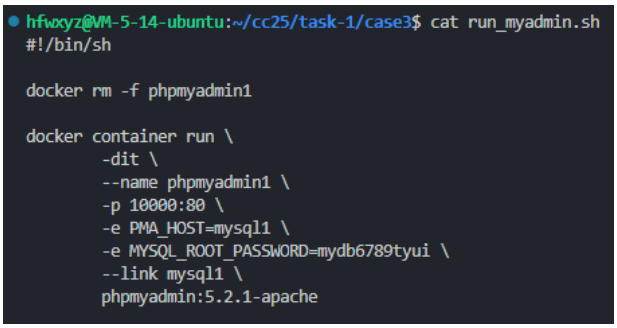
- ***Case 3***

Pada studi kasus ini terdapat dua file utama yang digunakan. File pertama adalah `run_mysql.sh`, yaitu script untuk menjalankan layanan database MySQL di dalam container. File kedua adalah `run_myadmin.sh`, yaitu script yang digunakan untuk menjalankan phpMyAdmin sebagai antarmuka GUI untuk mengelola database MySQL tersebut.



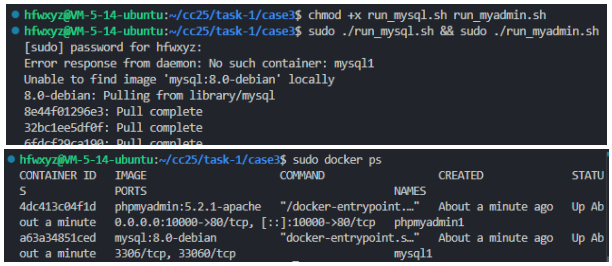
Gambar 10. Script run_mysql.sh

Ketika script ini dijalankan, langkah pertama yang dilakukan adalah menghapus container lama bernama `mysql1` apabila masih ada. Setelah itu, script membuat container MySQL baru menggunakan image `mysql:8.0-debian`. Container diberi nama `mysql1` dan data MySQL disimpan secara permanen melalui bind mount `$(pwd)/dbdata:/var/lib/mysql`, sehingga data tetap aman meskipun container dihapus. Script juga mengatur environment variable untuk kebutuhan konfigurasi database, serta mengizinkan akses root dari host mana pun melalui `MYSQL_ROOT_HOST=%`.



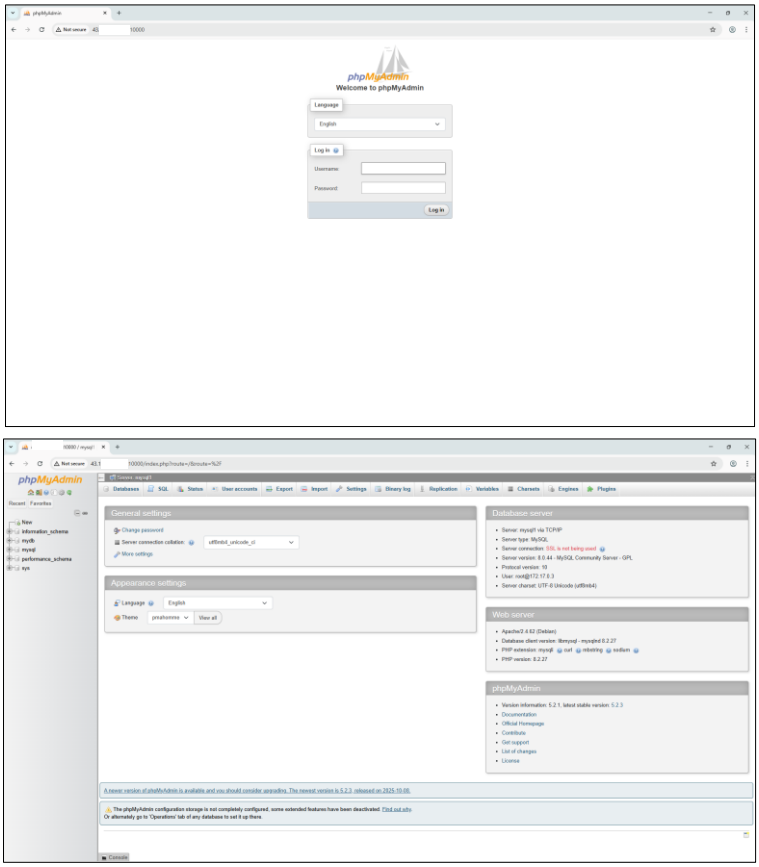
Gambar 11. Script run_myadmin.sh

Script ini berfungsi untuk menjalankan layanan phpMyAdmin sebagai antarmuka GUI bagi database MySQL yang telah dibuat sebelumnya. Pertama, script menghapus container lama bernama `phpmyadmin1` jika masih ada. Selanjutnya, script membuat container baru menggunakan image `phpmyadmin:5.2.1-apache` dan memetakan port 10000 pada host ke port 80 di dalam container sehingga phpMyAdmin dapat diakses melalui browser pada alamat `http://localhost:1000`. Konfigurasi `PMAHOST=mysql1` digunakan agar phpMyAdmin mengetahui bahwa ia harus terhubung ke container MySQL bernama `mysql1`, sementara `MYSQLROOT_PASSWORD` disetting agar phpMyAdmin dapat melakukan autentikasi menggunakan password root yang sama dengan MySQL. Selain itu, opsi `--link mysql1` memastikan bahwa container phpMyAdmin terhubung langsung dengan container MySQL agar komunikasi antar layanan dapat berjalan dengan lancar.



Gambar 12. Menjalankan container

Sebelum dijalankan, file `run_mysql.sh` dan `run_myadmin.sh` perlu diberi izin eksekusi menggunakan `chmod +x`, lalu dieksekusi dengan `sudo` agar dapat mengakses Docker daemon. Berdasarkan output, Docker tidak menemukan container sebelumnya sehingga otomatis melakukan pull image `mysql:8.0-debian` dan `phpmyadmin:5.2.1-apache` dari Docker Hub. Setelah image selesai diunduh, kedua container berhasil dijalankan, dengan konfigurasi `mysql1` pada port 3306 dan `phpmyadmin1` pada port 10000.

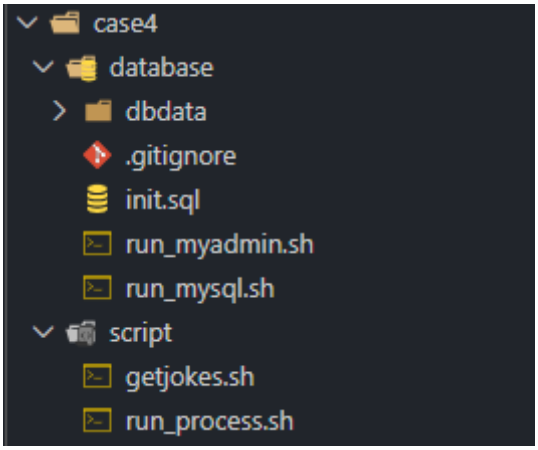


Gambar 13. Tampilan phpMyAdmin sebelum dan setelah login

Ketika diakses melalui localhost pada port 10000, akan muncul tampilan seperti pada Gambar 13 bagian pertama. Selanjutnya, pengguna dapat masuk ke phpMyAdmin dengan memasukkan kredensial yang telah dikonfigurasi, yaitu username “root” dan password “mydb6789tyui”. Jika proses login berhasil, halaman utama phpMyAdmin akan ditampilkan.

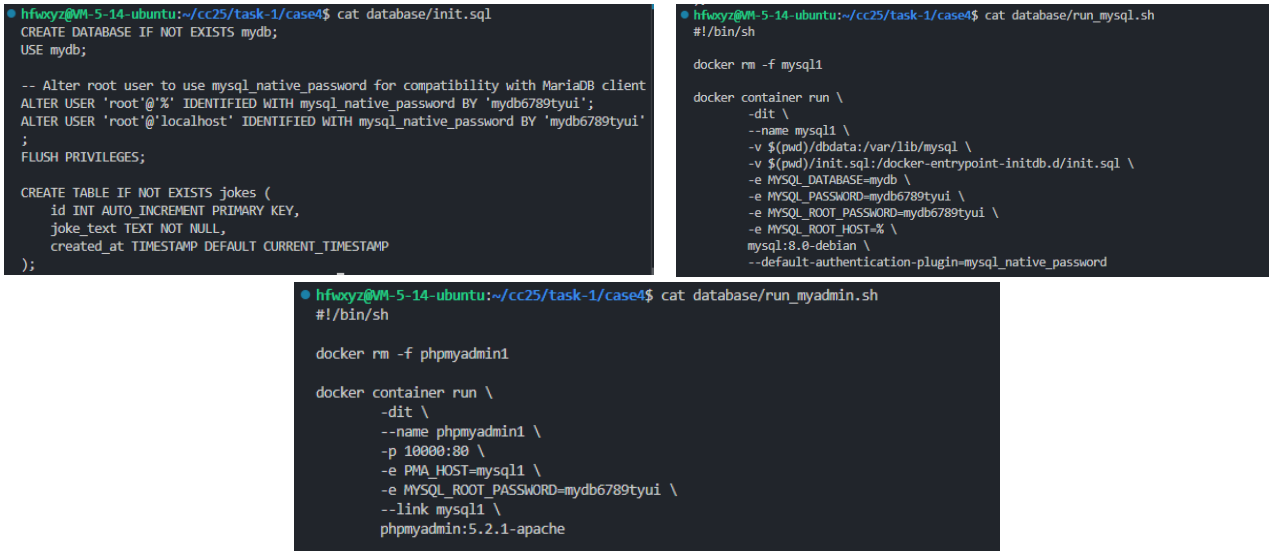
- ***Case 4***

Pada studi kasus eksplorasi kali ini, saya terinspirasi dari Case 1 dan Case 3 yang sebelumnya telah dilakukan. Pada Case 1, jokes Chuck Norris yang di-download disimpan langsung ke dalam host yang berpotensi membuat direktori menjadi kurang rapi seiring bertambahnya file. Berdasarkan hal tersebut, pendekatannya kemudian dikembangkan seperti pada Case 3, yaitu memanfaatkan database sebagai media penyimpanan yang lebih terstruktur.



Gambar 14. Directory case4

Untuk mendukung penerapan ini, langkah awal dilakukan dengan menyiapkan struktur direktori seperti pada Gambar 14, kemudian menyesuaikan konfigurasi script agar proses pengambilan data dan penyimpanan ke database aman.



Gambar 15. File pada directory /database

File `init.sql` berfungsi sebagai skrip inisialisasi yang otomatis dijalankan saat container MySQL pertama kali dibuat. Skrip ini membuat database `mydb` jika belum ada, mengatur autentikasi user root ke `mysql_native_password` untuk kompatibilitas dengan MariaDB dan aplikasi lama, menetapkan password mydb6789tyui, lalu menjalankan FLUSH PRIVILEGES. Terakhir, skrip membuat tabel `jokes` dengan kolom `id`, `joke_text`, dan `created_at`.

Script `run_mysql.sh` bertugas membuat container MySQL yang baru. Script ini menghapus container `mysql1` jika sudah ada, lalu menjalankan container baru dengan volume untuk penyimpanan data dan volume untuk menjalankan file `init.sql` secara otomatis. Selain itu, script mengatur environment variables seperti MYSQL_DATABASE dan MYSQL_ROOT_PASSWORD, sekaligus memastikan metode autentikasi default memakai `mysql_native_password`.

Script `run_myadmin.sh` menjalankan phpMyAdmin dengan menghapus container `phpmyadmin1` yang lama dan membuat container baru. phpMyAdmin dipetakan ke port 10000 sehingga dapat diakses melalui browser, dan dihubungkan ke container MySQL melalui PMA_HOST=mysql1, MYSQL_ROOT_PASSWORD, serta flag `--link mysql1` agar keduanya dapat berkomunikasi.

```
hfwxyz@VM-5-14-ubuntu:~/cc25/task-1/case4$ cat script/run_process.sh
#!/bin/bash
docker rm -f myprocess1

# Get absolute path of script directory
SCRIPT_DIR="$(cd "$(dirname "$0")" && pwd)"

docker container run \
        --name myprocess1 \
        -dit \
        -e DELAY=8 \
        --link mysql1 \
        -v "$SCRIPT_DIR":/script \
        alpine:3.18 \
        /bin/sh /script/getjokes.sh
```

```
hfwxyz@VM-5-14-ubuntu:~/cc25/task-1/case4$ cat script/getjokes.sh
#!/bin/sh

apk update && apk add curl jq mysql-client

URL=https://api.chucknorris.io/jokes/random
DB_HOST=mysql1
DB_USER=root
DB_PASS=mydb6789tyui
DB_NAME=mydb

echo will run every $DELAY seconds
echo waiting for MySQL to be ready...
sleep 10

while true;
do
        date=$(date '+%Y-%m-%d_%H:%M:%S')
        echo processing at $date

        # Get joke from API
        joke=$(curl -sL $URL | jq -r '.value')

        # Escape single quotes for SQL
        joke_escaped=$(echo "$joke" | sed "s/'/''/g")

        # Insert to MySQL database
        mysql -h"$DB_HOST" -u"$DB_USER" -p"$DB_PASS" "$DB_NAME" \
                -e "INSERT INTO jokes (joke_text) VALUES ('$joke_escaped');"

        echo "Joke saved to database: $joke"
        sleep $DELAY
```

Gambar 16. File pada directory /script

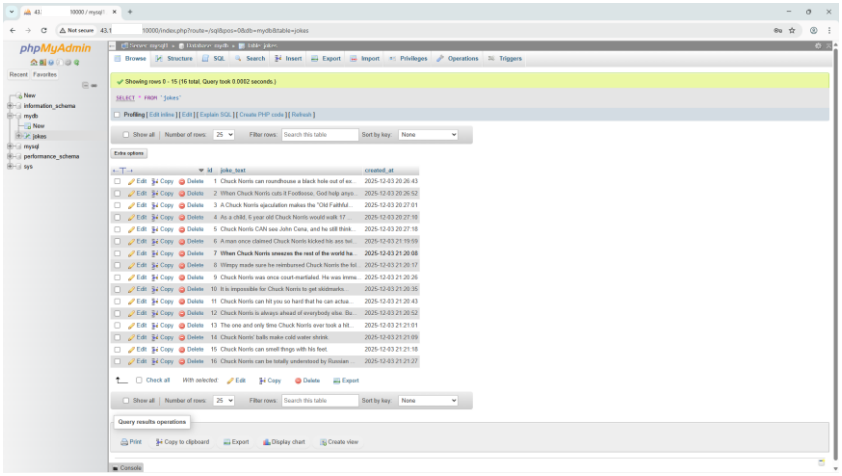
Script `run_process.sh` menjalankan container Alpine untuk proses pengambilan joke. Script menghapus container lama, menentukan path direktori script agar volume mount tepat, lalu menjalankan container dalam mode detached dengan DELAY=8, terhubung ke MySQL melalui `--link mysql1`, dan memetakan folder lokal ke `/script`. Sama seperti sebelumya, script `getjokes.sh` menjadi worker yang mengambil joke dari API dan menyimpannya ke database. Script menginstal `curl`, `jq`, dan `mysql-client`, menyiapkan konfigurasi, lalu menunggu 10 detik hingga MySQL siap. Selanjutnya, script terus berjalan dalam loop dengan mengambil joke, mem-parsing JSON, melakukan escaping karakter berbahaya, lalu memasukkan hasilnya ke tabel `jokes`.

Apabila semua script sudah siap, bisa mulai untuk menjalankan `run_mysql.sh`, `run_myadmin.sh`, dan `run_process.sh` secara berurutan, yang nantinya akan ada tiga container seperti gambar berikut.

```
hfwxyz@VM-5-14-ubuntu:~/cc25/task-1/case4$ sudo docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                                          NAMES
215434748a8f   alpine:3.18            "/bin/sh /script/get…"   13 seconds ago   Up 12 seconds                                                  myprocess1
e19f63d0625a   phpmyadmin:5.2.1-apache   "/docker-entrypoint.…"   16 seconds ago   Up 16 seconds   0.0.0.0:10000->80/tcp, [::]:10000->80/tcp      phpmyadmin1
3df581b67688   mysql:8.0-debian       "docker-entrypoint.s…"   32 seconds ago   Up 31 seconds   3306/tcp, 33060/tcp                            mysql1
```

Gambar 17. Container yang running saat ini

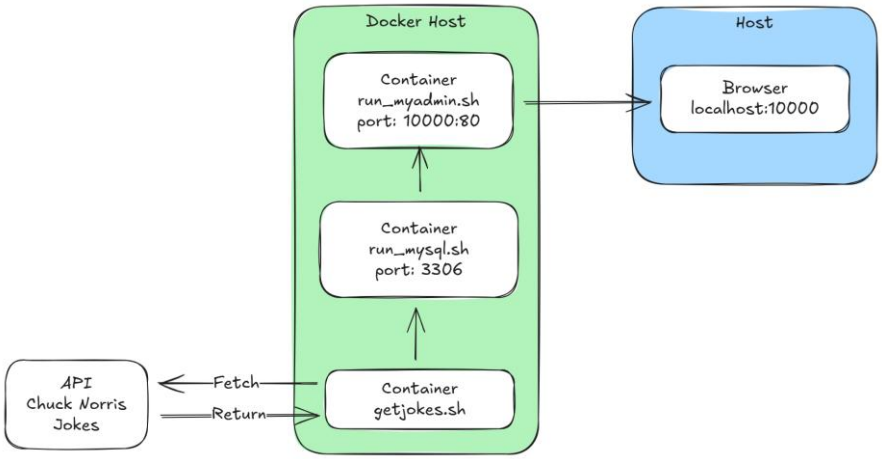
Dan apabila berhasil berjalan, maka nantinya apabila kita buka pada localhost port 10000 akan seperti gambar berikut.



Gambar 18. Data pada tabel jokes

Container `getjokes.sh` secara berkala mengambil joke dari API Chuck Norris dan menyimpannya ke container MySQL (port 3306). Container phpMyAdmin (port 10000) terhubung ke MySQL dan

menyediakan antarmuka web di `localhost:10000` untuk melihat dan mengelola data. Untuk lebih jelasnya bisa melihat gambar di bawah ini.



Gambar 19. Arsitektur Case 4

Spesifikasi komputer/VM yang digunakan meliputi 4 GB RAM, 2 vCPU, 60 GB storage, dengan IP Address 43.xxx.xxx.xxx. Port yang digunakan adalah 3306 untuk MySQL server dan 10000 untuk phpMyAdmin.

Skenario ini cocok diterapkan ketika aplikasi membutuhkan pemisahan tugas antara proses pengambilan data dan penyajian data. Arsitektur ini ideal untuk sistem yang secara berkala mengumpulkan data dari sumber eksternal, seperti API sensor IoT, harga crypto, cuaca, atau social media metrics lalu menyimpannya di database dan menampilkannya melalui antarmuka web.

Kesimpulan

Tugas ini berhasil menunjukkan penerapan Docker melalui empat studi kasus yang mencakup background process, web server statis, database management, dan integrasi multi-container. Case 1 mendemonstrasikan proses otomatis pengambilan joke menggunakan container Alpine dengan volume untuk persistensi. Case 2 memperlihatkan betapa mudahnya menjalankan static web server dalam container. Case 3 menampilkan integrasi MySQL dan phpMyAdmin dalam arsitektur multi-container dan Case 4 menggabungkan konsep sebelumnya dengan penyimpanan data ke database untuk arsitektur yang lebih modern dan terstruktur. Secara keseluruhan, tugas ini membuktikan fleksibilitas Docker dalam membangun lingkungan aplikasi yang konsisten, terisolasi, mudah dideploy.