Muhammad Iqbal Ramadhan
5025221274

Cloud Computing
EF4704

Tugas 1

Pendahuluan

Tugas ini membahas implementasi konsep containerisasi menggunakan Docker. Docker merupakan sebuah platform yang memungkinkan aplikasi dijalankan di dalam *container*, yaitu lingkungan terisolasi yang berisi seluruh dependensi yang dibutuhkan sehingga aplikasi dapat berjalan secara konsisten di berbagai sistem. Dengan pendekatan ini, proses pengembangan, deployment, dan pemeliharaan aplikasi menjadi lebih efisien, terstandarisasi, dan mudah direplikasi.

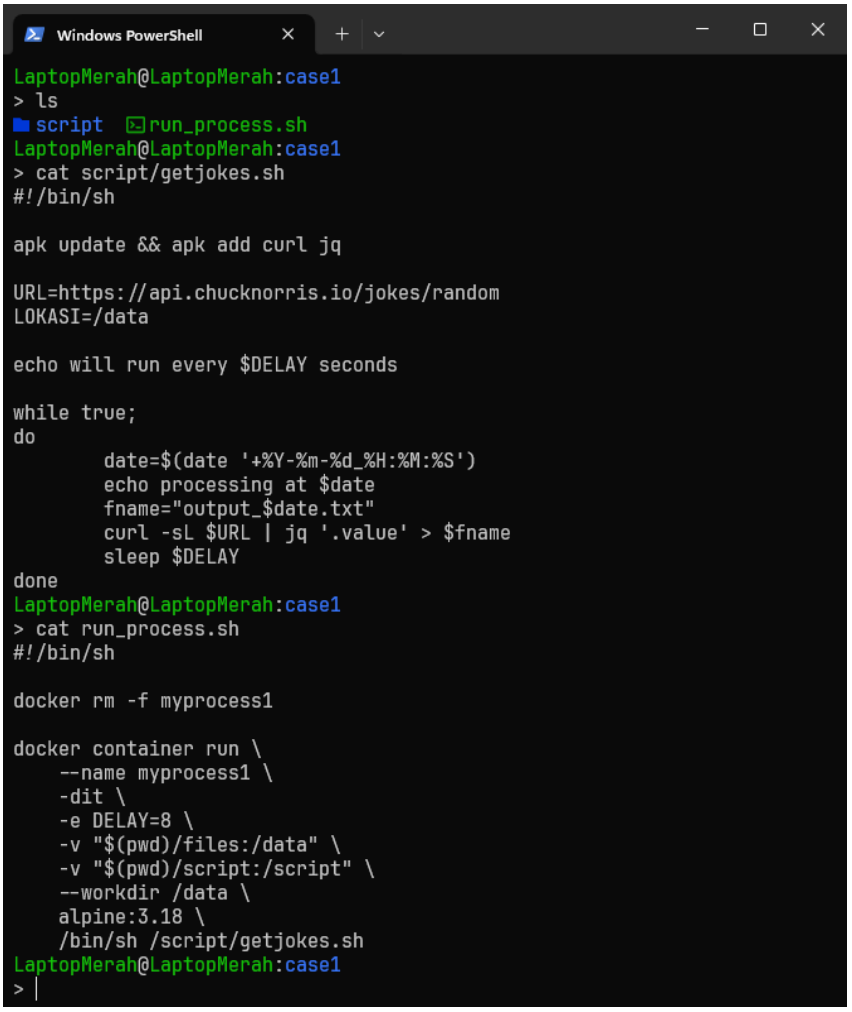
Tujuan dari tugas ini adalah untuk mempraktikkan konsep-konsep dasar Docker melalui tiga kasus yang disediakan, meliputi menjalankan container dari sebuah image, mengelola volume untuk menjaga data agar tetap persisten, serta menyiapkan environment yang stabil dan saling terhubung. Laporan ini akan menjelaskan langkah-langkah pengerjaan, konfigurasi yang digunakan, serta bukti hasil implementasi untuk setiap kasus yang dikerjakan.

Tugas yang Diberikan

- Menjalankan 3 Case yang disediakan pada repository github. Setiap case, diperlukan dokumentasi yang mencakup cara menjalankan, screenshot hasil, dan penjelasan proses.
- Membuat Case baru (case 4) berbasis pada case sebelumnya. Proyek ini harus dilengkapi dengan:
    - Penjelasan skenario atau kondisi di mana kreasi tersebut penting atau cocok untuk digunakan.
    - Gambar arsitektur sistem yang dibuat.
    - Script pendukung dan screenshot hasil serta penjelasannya.

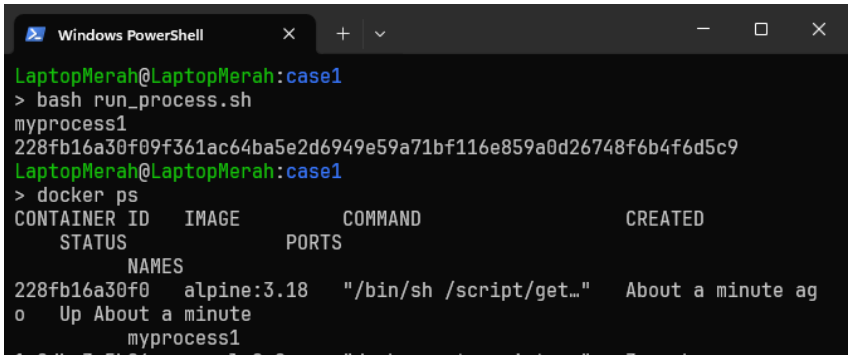
Case 1:

Pada case1 ini terdapat 2 file yang tersedia, yaitu `script/getjokes.sh` dan `run_process.sh`.

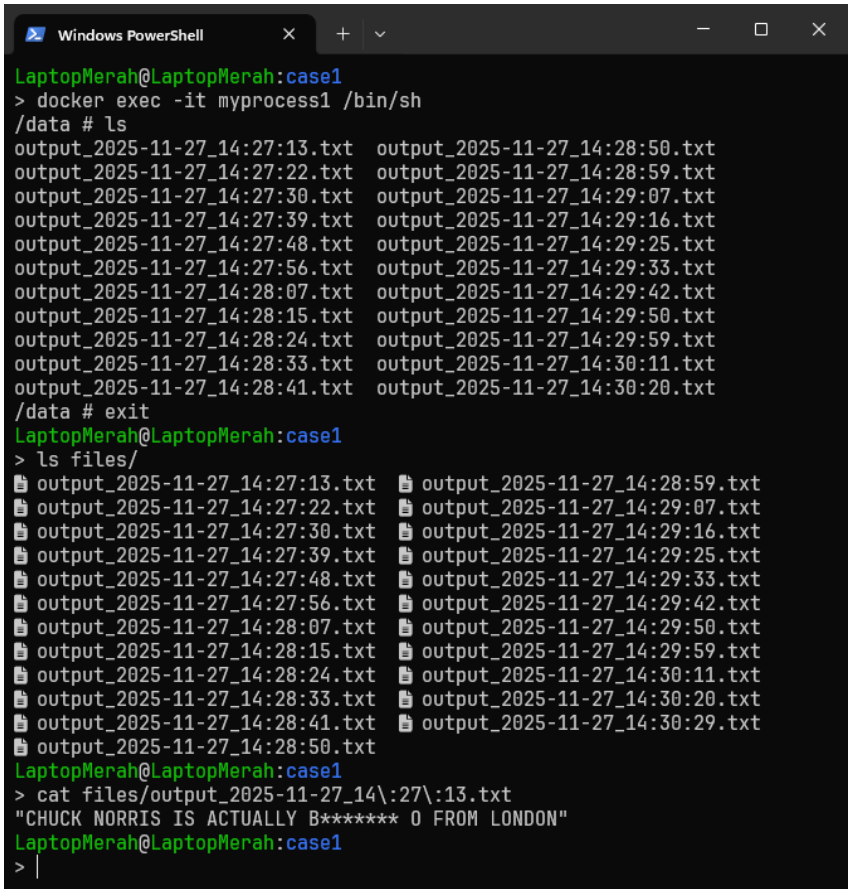


`script/getjokes.sh` merupakan script untuk mendapatkan jokes-jokes Chuck Norris yang disediakan pada API https://api.chucknorris.io/jokes/random, dan script ini akan berjalan dalam sebuah *loop* (`while true`) selama 8 detik sekali (berdasarkan env yang di setup pada docker container). Jokes yang didapatkan akan disimpan dalam bentuk file .txt di direktori `/data`.

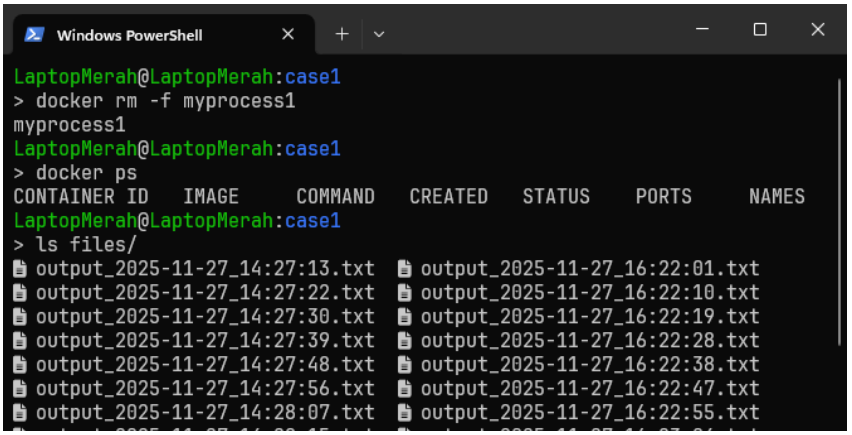
`run_process.sh` merupakan script untuk membuat docker container dengan nama myprocess1 yang dijalankan dalam mode latar belakang -dit (Detached, Interactive, TTY) menggunakan image alpine:3.18. Container ini menetapkan variabel $DELAY=8 dan melakukan mounting -v untuk memetakan folder /files pada host ke /data yang ada di container dan direktori /script pada host ke /script di container. Lalu menjalankan /bin/sh /script/getjokes.sh, untuk pengambilan jokes Chuck Norris secara otomatis.



untuk menjalankan script gunakan `bash run_process.sh`, dan terlihat myprocess1 sudah berjalan. Dengan `docker ps` dapat memastikan kembali docker yang sedang berjalan, dan terbukti dengan adanya `myprocess1`.



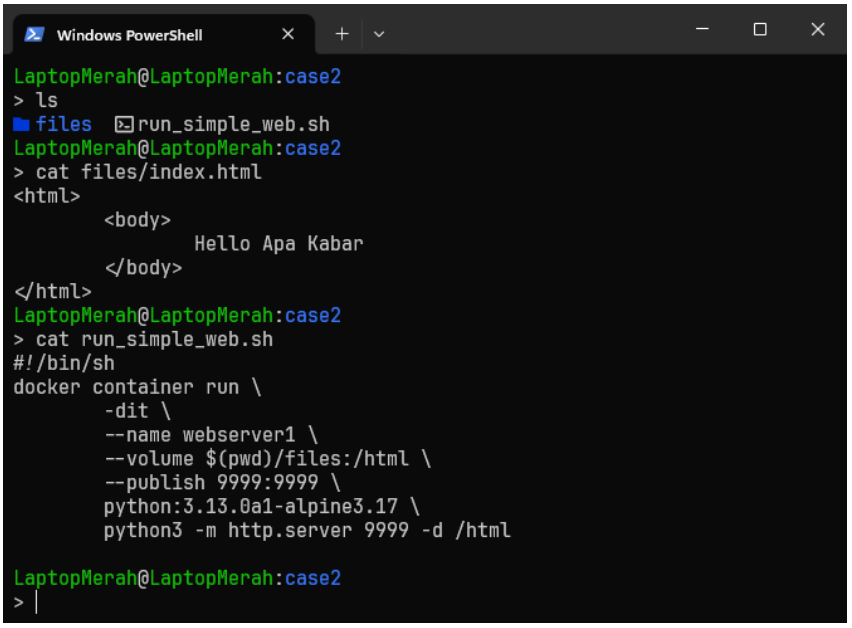
diatas merupakan bukti hasil dari berjalannya script tersebut untuk menghasilkan file yang berisikan jokes Chuck Norris. Di dalam /data yang berada di dalam docker berisikan sama dengan yang ada di /files di host



Gambar diatas adalah bukti dari metode mounting folder host ke dalam docker agar segala file yang terbuat didalam docker tersimpan di host. docker sudah di remove dan tidak ada docker yang ada, file file yang dihasilkan dari docker tersebut masih tersimpan di folder /files tersebut

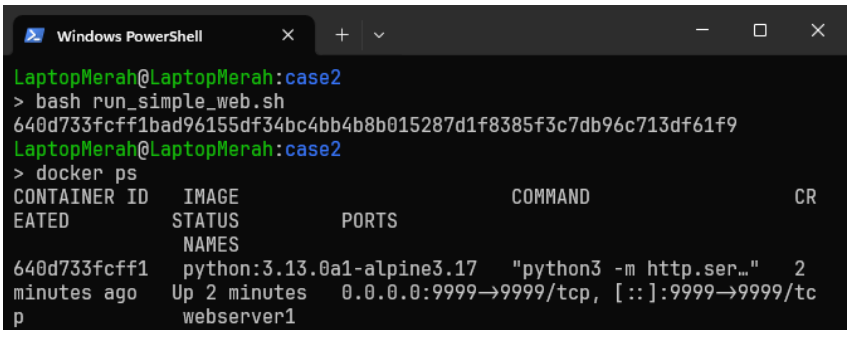
Case 2:

Pada case2 ini terdapat 2 file yang tersedia, yaitu `files/index.html` dan `run_simple_web.sh`.

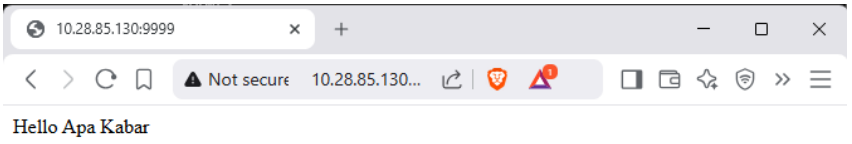


`files/index.html` merupakan html simple yang akan digunakan untuk web yang akan kita buat.

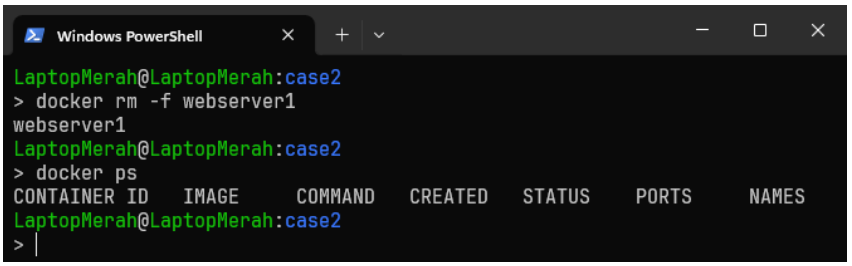
`run_simple_web.sh` merupakan script untuk membuat docker container dengan nama webserver1 yang dijalankan dalam mode latar belakang -dit (Detached, Interactive, TTY) menggunakan image python:3.13.0a1-alpine3.17. Container ini melakukan mounting –volume untuk memetakan folder /files pada host ke /html. Dan mapping port 9999 host ke 9999 pada container. pada Lalu menjalankan python3 -m http.server 9999 -d /html, untuk mengaktifkan *web server* HTTP bawaan Python pada port 9999, dan direktori /html di dalam *container* sebagai *root directory* untuk menyajikan berkas.



untuk menjalankan script gunakan `bash run_simple_web.sh`, dan terlihat sudah berjalan. Dengan `docker ps` dapat memastikan kembali docker yang sedang berjalan, dan terbukti dengan adanya webserver1.



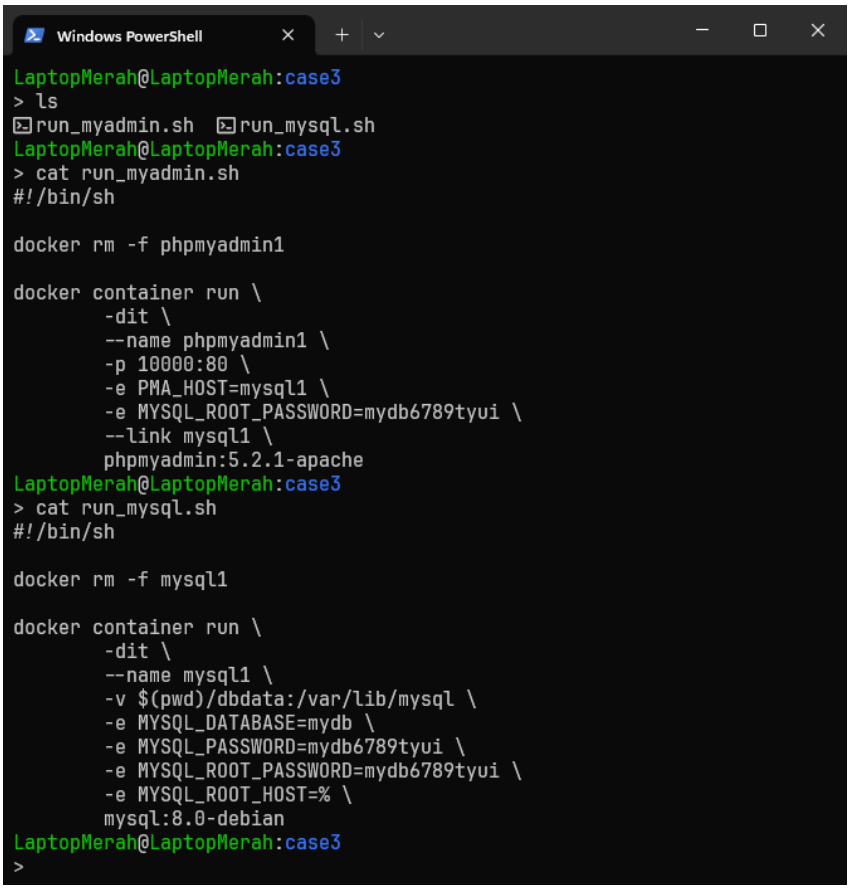
Sudah terlihat bahwa pada port 9999 yang berjalan di ip 10.28.85.130 sudah tersajikan web yang ada pada files/index.html dan di mount ke /html tadi sudah bisa berjalan dan tersajikan.



Gambar diatas memastikan bahwa docker tersebut sudah di remove untuk melanjutkan ke case selanjutnya.

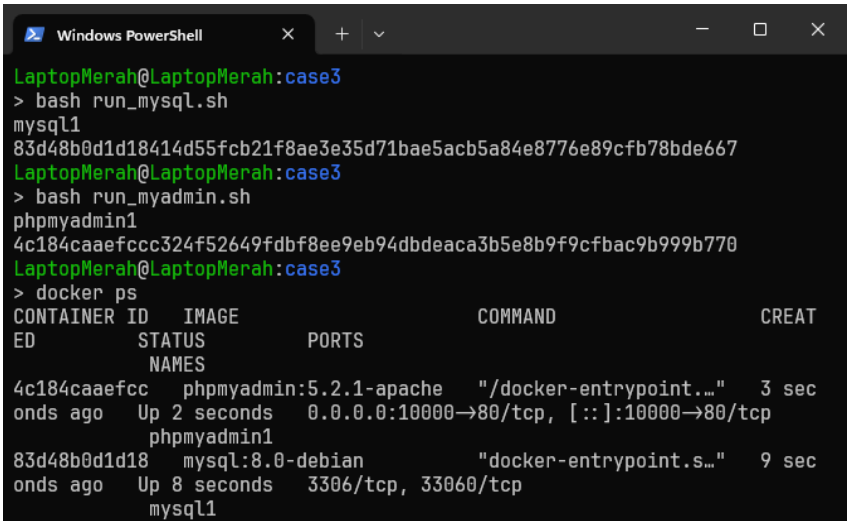
Case 3:

Pada case3 ini terdapat 2 file yang tersedia, yaitu `run_myadmin.sh` dan `run_mysql.sh`.

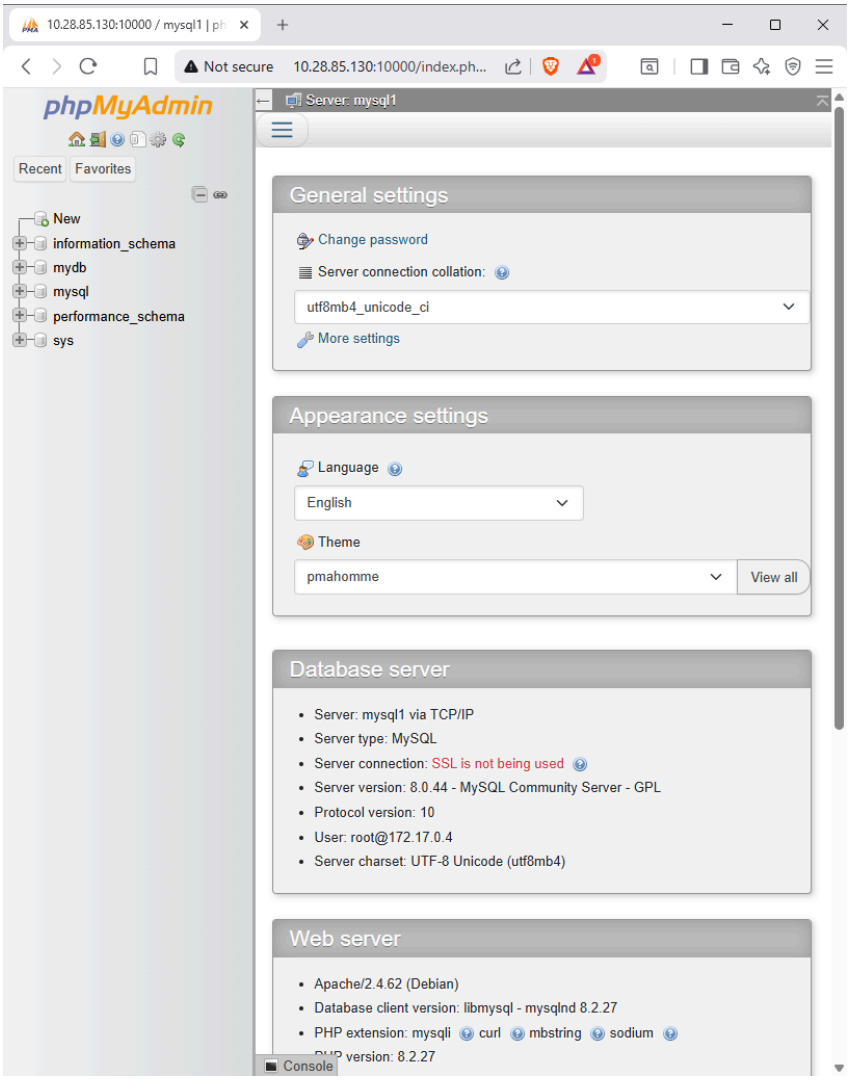


`run_myadmin.sh` merupakan script untuk membuat docker container dengan nama phpmyadmin1 yang dijalankan dalam mode latar belakang -dit (Detached, Interactive, TTY) menggunakan image phpmyadmin:5.2.1-apache .Dan mapping port 10000 host ke 80 pada container. Dengan environment PMA_HOST dan MYSQL_ROOT_PASSWORD. Serta menghubungkan dengan container mysql1.

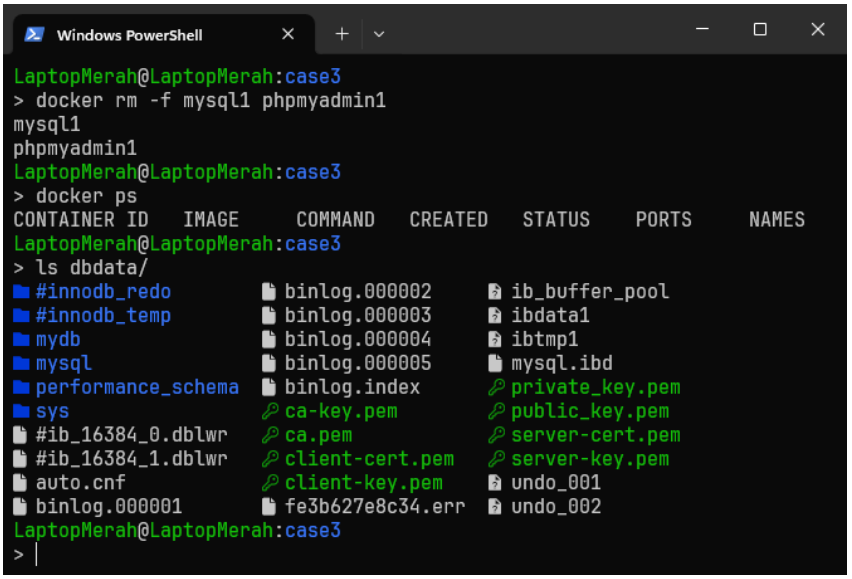
`run_mysql.sh` merupakan script untuk membuat docker container dengan nama mysql1 yang dijalankan dalam mode latar belakang -dit (Detached, Interactive, TTY) menggunakan image mysql:8.0-debian. Container ini menetapkan environment MYSQL_DATABASE, MYSQL_PASSWORD, MYSQL_ROOT_PASSWORD, dan MYSQL_ROOT_HOST yang dibutuhkan oleh mysql. Dan melakukan mounting -v untuk memetakan folder $(pwd)/dbdata pada host ke /var/lib/mysql yang ada di container.



Gambar diatas adalah cara menjalankan script dengan menggunakan `bash run_mysq.sh` dan `bash run_mysq.sh`. Dengan docker ps dapat memastikan kembali docker yang sedang berjalan, dan terbukti dengan adanya phpmyadmin1 dan mysql1.

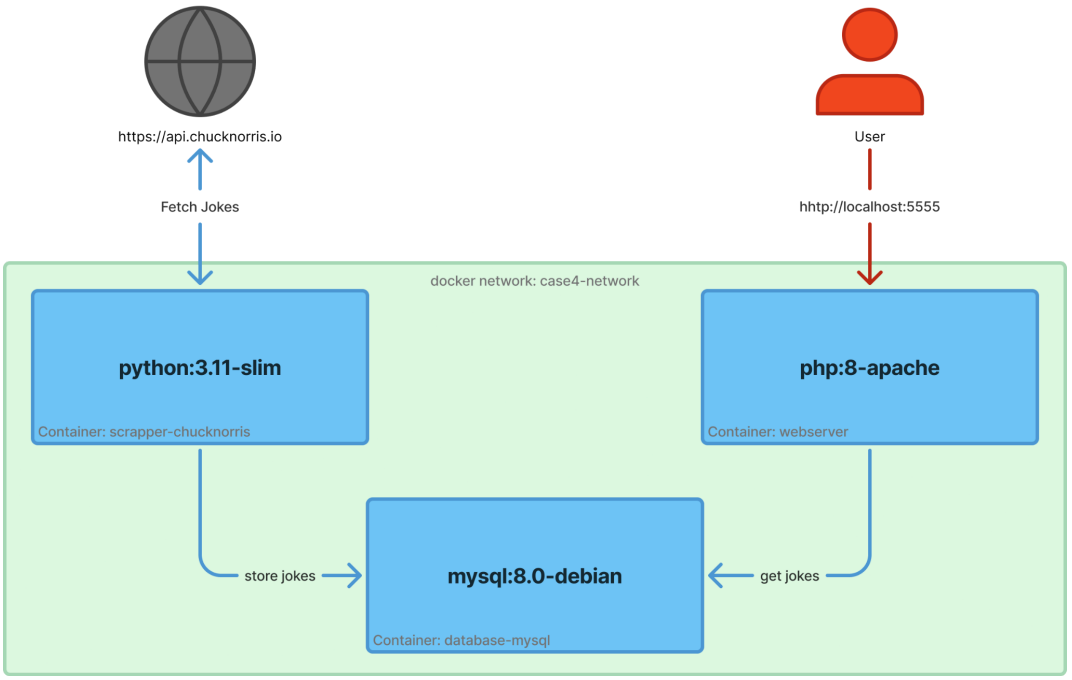


Sudah terlihat bahwa pada port 10000 yang berjalan di ip 10.28.85.130 sudah tersajikan phpmyadmin yang sudah bisa berjalan dan tersajikan. dengan login dengan username ‘root’ dan password mydb6789tyui sesuai env yang sudah di set pada docker tersebut.

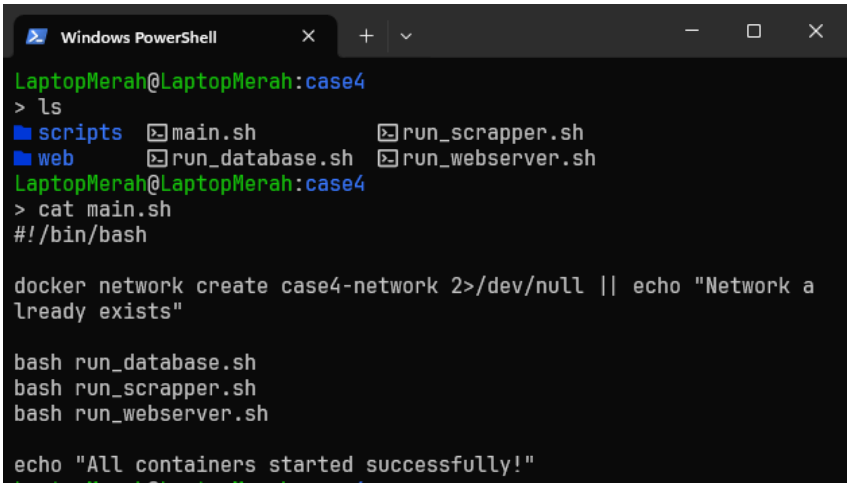


Terlihat pada gambar diatas bahwa data yang ada di /var/lib/mysql akan tetap tersimpan di dbdata/ karena sudah di mounting, yang berguna untuk mencegah file hilang ketika docker crash.

Case 4:



Skenario container yang saya buat ini penting untuk dipelajari karena menggambarkan penerapan konsep dasar arsitektur berbasis microservices yang umum digunakan dalam pengembangan sistem modern. Setiap container memiliki peran yang terpisah namun saling terhubung melalui satu Docker network: container Python bertugas melakukan pengambilan data dari API eksternal dan menyimpannya ke dalam database, container MySQL berfungsi sebagai penyimpanan utama data, sedangkan container PHP–Apache menampilkan data tersebut ke pengguna melalui web. Pemisahan fungsi seperti ini membuat sistem lebih mudah dikelola, fleksibel, dan tidak saling bergantung secara langsung, sehingga apabila satu bagian mengalami gangguan, komponen lain tetap dapat berjalan dengan stabil. Latihan ini juga relevan karena menggabungkan beberapa kemampuan yang telah saya pelajari sebelumnya—mulai dari fetching API, menjalankan web server, hingga menggunakan database—dalam satu alur kerja yang realistis dan menyerupai lingkungan produksi.



pada case 4 ini saya membuat 4 shell script utama `main.sh`, `run_database.sh`, `run_scrapper.sh`, dan `run_webserver.sh`. Lalu ada `scripts/scrapper.py` untuk

scraping dari Chuck Norris Jokes API dan `web/index.php` untuk tampilan dari web servernya. main.sh memuat pembuatan network agar setiap container dapat berkomunikasi, dan akan menjalan ketika container tersebut.

```
LaptopMerah@LaptopMerah:case4
> cat run_database.sh
#!/bin/bash

docker rm -f database-mysql

docker run \
    -dit \
    --name database-mysql \
    --network case4-network \
    -v "$(pwd)/DB-data:/var/lib/mysql" \
    -e MYSQL_DATABASE=db_case4 \
    -e MYSQL_PASSWORD=passdb_case4 \
    -e MYSQL_ROOT_PASSWORD=mydb6789tyui \
    -e MYSQL_ROOT_HOST=% \
    -e MYSQL_USER=case4 \
    mysql:8.0-debian

echo "Database MySQL started!"
```

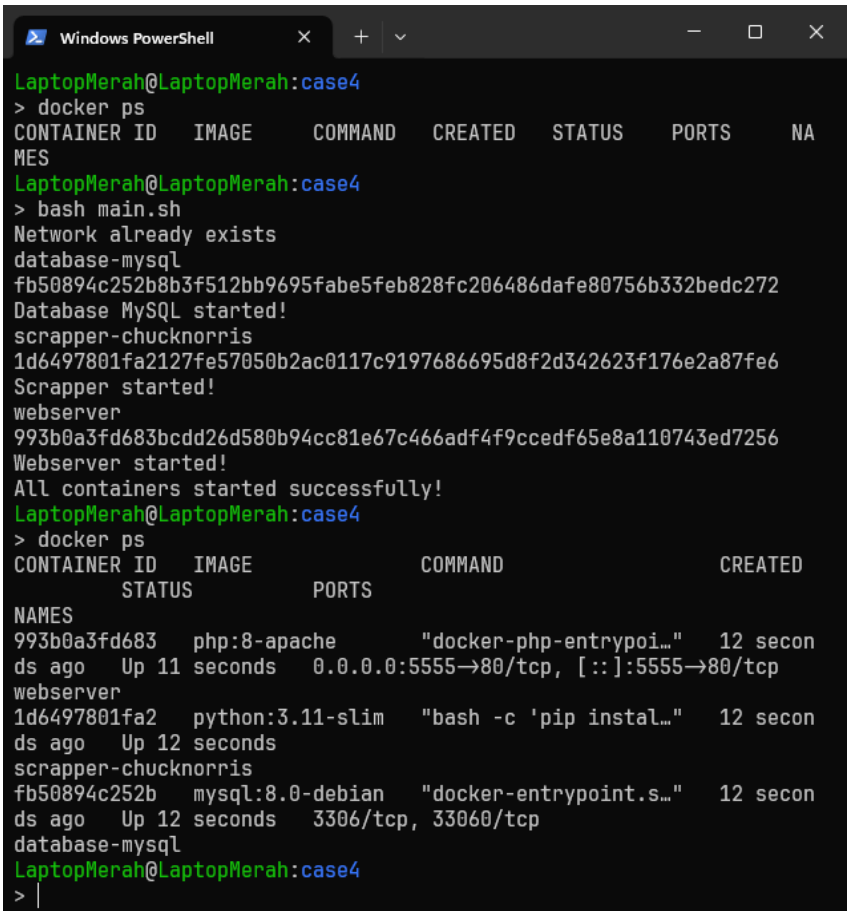
`run_database.sh` merupakan script untuk membuat docker container dengan nama *database-mysql* yang terhubung ke *case4-network,* menggunakan image mysql:8.0-debian. Menggunakan volume lokal agar data MySQL tetap tersimpan, serta menyetel berbagai environment variable seperti nama database, username, dan password.

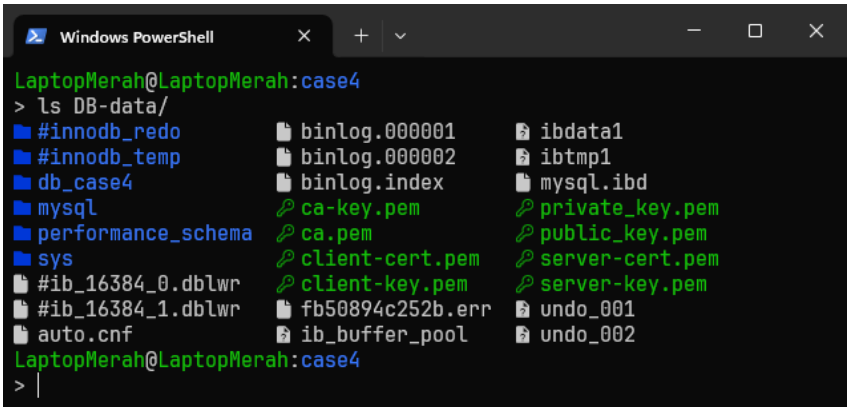
```
LaptopMerah@LaptopMerah:case4
> cat run_scrapper.sh
#!/bin/bash

docker rm -f scrapper-chucknorris

# Run the scrapper container directly with Python image
docker run \
    -dit \
    --name scrapper-chucknorris \
    --network case4-network \
    -e DB_HOST=database-mysql \
    -e DB_USER=case4 \
    -e DB_PASSWORD=passdb_case4 \
    -e DB_NAME=db_case4 \
    -e API_URL=https://api.chucknorris.io/jokes/random \
    -e INTERVAL=10 \
    -v "$(pwd)/scripts:/app" \
    python:3.11-slim \
    bash -c "pip install requests mysql-connector-python && python
 -u /app/scrapper.py"

echo "Scrapper started!"
```

`run_scrapper.sh` merupakan script untuk membuat container *scrapper-chucknorris* yang terhubung ke *case4-network*, menggunakan image *python:3.11-slim*, memberikan environment variable seperti konfigurasi database dan URL API, serta me-mount folder lokal berisi script Python agar container dapat menjalankan proses pengambilan data dan menyimpannya ke database. Script ini juga mengeksekusi perintah instalasi dependensi Python sebelum akhirnya menjalankan file *scrapper.py* di dalam container.

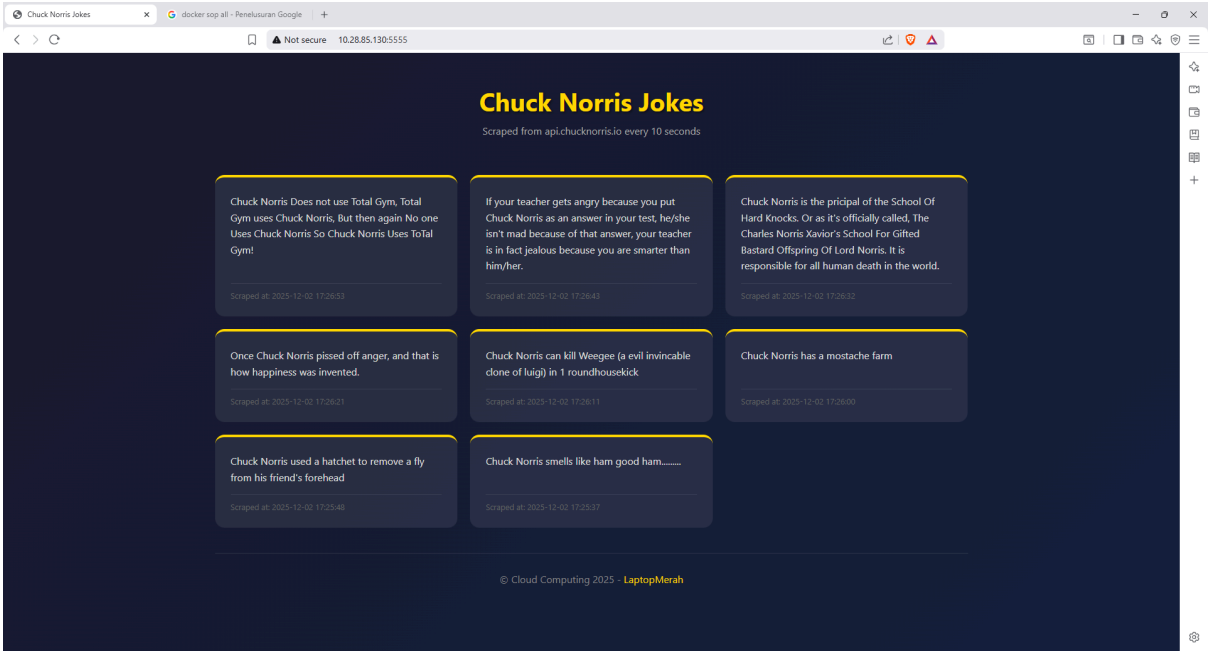
```
LaptopMerah@LaptopMerah:case4
> cat run_webserver.sh
#!/bin/bash

docker rm -f webserver

docker run \
    -dit \
    --name webserver \
    --network case4-network \
    -e DB_HOST=database-mysql \
    -e DB_USER=case4 \
    -e DB_PASSWORD=passdb_case4 \
    -e DB_NAME=db_case4 \
    -v "$(pwd)/web:/var/www/html" \
    --publish 5555:80 \
    php:8-apache \
    bash -c "docker-php-ext-install mysqli && apache2-foreground"

echo "Webserver started!"
```

`run_webserver.sh` merupakan script untuk membuat container *webserver* yang menggunakan image *php:8-apache*, terhubung ke *case4-network*, serta diberikan konfigurasi environment untuk mengakses database MySQL, sekaligus me-mount folder lokal sebagai direktori *web root* PHP. Script ini juga menambahkan ekstensi mysqli di dalam container sebelum menjalankan Apache, sehingga webserver dapat mengambil dan menampilkan data dari database.





Bukti bahwa script `main.sh` sudah berhasil dijalakan dan menghasil 3 container, yaitu webserver, scrapper-chucknorris, dan database-mysql. Serta database yang tersimpan akan berada di host /DB-data.



Ini adalah tampilan website yang berjalan yang berisikan kumpulan Chuck Norris Jokes yang berhasil di-scrapping setiap 10 detik sekali. Berjalan pada port 5555

Kesimpulan

Dengan tugas ini yang mempelajari docker kontainerisasi, mulai dari mengambil data dari API secara berkala, menjalankan webserver sederhana, hingga sistem database dengan mySQL dan PHPMyAdmin. Dari tugas yang dilakukan menghasilkan sebuah pipeline *scraping* berbasis arsitektur *microservice* yang berjalan sepenuhnya di dalam lingkungan ter-containerisasi. Melalui proses ini, saya mempelajari bagaimana setiap layanan—mulai dari database MySQL, *scrapper* berbasis Python, hingga webserver PHP—dapat dijalankan secara terpisah namun tetap saling terhubung melalui satu jaringan Docker. Penyusunan script otomatis membantu memastikan bahwa setiap layanan dapat dijalankan secara konsisten, terisolasi, dan mudah direplikasi tanpa konfigurasi ulang manual.