

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.

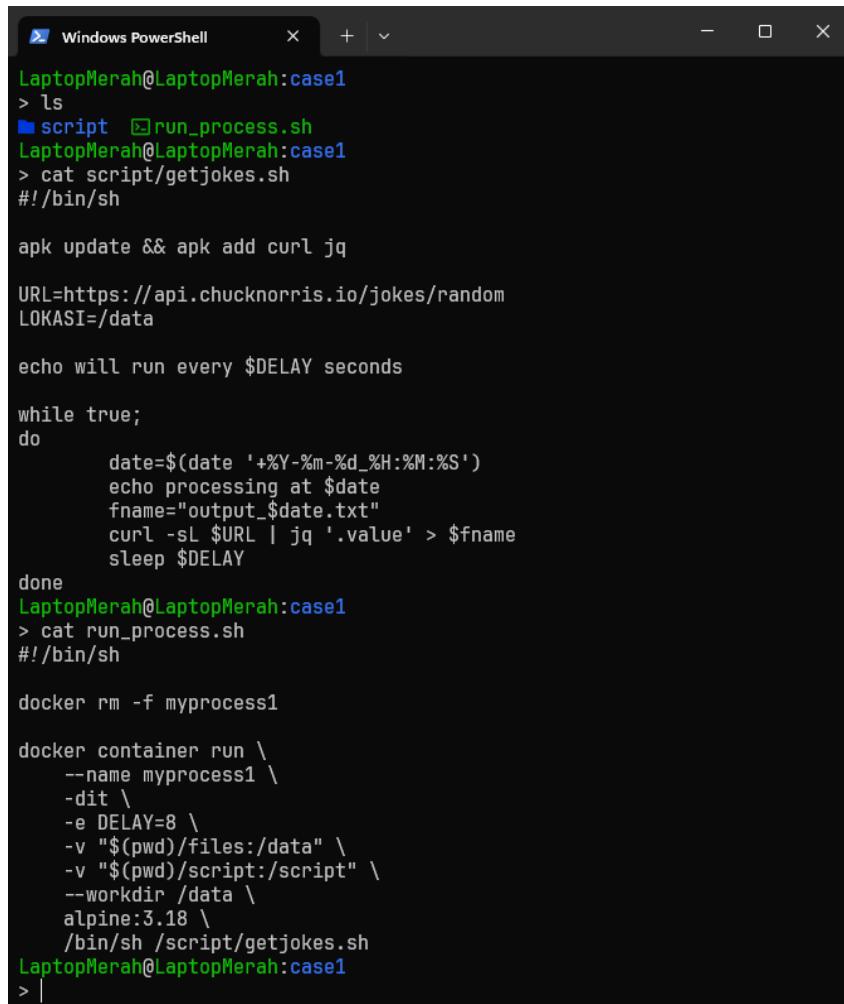
Tujuan dari tugas ini adalah untuk mempraktikkan konsep-konsep dasar Docker melalui tiga kasus yang disediakan, meliputi menjalankan container dari sebuah image, mengelola volume untuk menjaga data agar tetap persisten, serta menyiapkan environment yang stabil dan saling terhubung. Laporan ini akan menjelaskan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



```
LaptopMerah@LaptopMerah:case1
> ls
  script  run_process.sh
LaptopMerah@LaptopMerah: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
done
LaptopMerah@LaptopMerah: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
LaptopMerah@LaptopMerah:case1
> |
```

`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`.

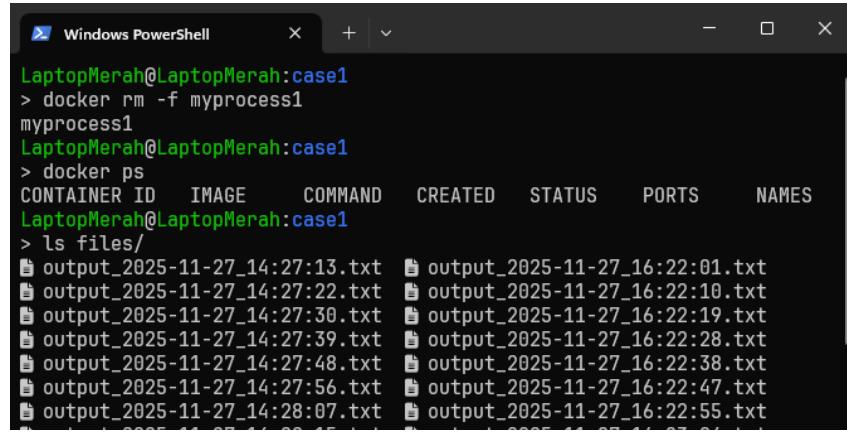
`run_process.sh` merupakan script untuk membuat docker container dengan nama `myprocess1` yang dijalankan dalam mode latar belakang `-dit` (Detached, Interactive, TTY) menggunakan image `alpine:3.18`. Container ini menetapkan variabel `$DELAY=8` dan melakukan mounting `-v` untuk memetakan folder `/files` 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.

```
Windows PowerShell
LaptopMerah@LaptopMerah:case1
> bash run_process.sh
myprocess1
228fb16a30f09f361ac64ba5e2d6949e59a71bf116e859a0d26748f6b4f6d5c9
LaptopMerah@LaptopMerah:case1
> docker ps
CONTAINER ID   IMAGE      COMMAND           CREATED
STATUS          NAMES
228fb16a30f0   alpine:3.18  "/bin/sh /script/get..."  About a minute ago
o   Up About a minute          myprocess1
```

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.

```
Windows PowerShell
LaptopMerah@LaptopMerah:case1
> docker exec -it myprocess1 /bin/sh
/data # ls
output_2025-11-27_14:27:13.txt  output_2025-11-27_14:28:50.txt
output_2025-11-27_14:27:22.txt  output_2025-11-27_14:28:59.txt
output_2025-11-27_14:27:30.txt  output_2025-11-27_14:29:07.txt
output_2025-11-27_14:27:39.txt  output_2025-11-27_14:29:16.txt
output_2025-11-27_14:27:48.txt  output_2025-11-27_14:29:25.txt
output_2025-11-27_14:27:56.txt  output_2025-11-27_14:29:33.txt
output_2025-11-27_14:28:07.txt  output_2025-11-27_14:29:42.txt
output_2025-11-27_14:28:15.txt  output_2025-11-27_14:29:50.txt
output_2025-11-27_14:28:24.txt  output_2025-11-27_14:29:59.txt
output_2025-11-27_14:28:33.txt  output_2025-11-27_14:30:11.txt
output_2025-11-27_14:28:41.txt  output_2025-11-27_14:30:20.txt
/data # exit
LaptopMerah@LaptopMerah:case1
> ls files/
output_2025-11-27_14:27:13.txt  output_2025-11-27_14:28:59.txt
output_2025-11-27_14:27:22.txt  output_2025-11-27_14:29:07.txt
output_2025-11-27_14:27:30.txt  output_2025-11-27_14:29:16.txt
output_2025-11-27_14:27:39.txt  output_2025-11-27_14:29:25.txt
output_2025-11-27_14:27:48.txt  output_2025-11-27_14:29:33.txt
output_2025-11-27_14:27:56.txt  output_2025-11-27_14:29:42.txt
output_2025-11-27_14:28:07.txt  output_2025-11-27_14:29:50.txt
output_2025-11-27_14:28:15.txt  output_2025-11-27_14:29:59.txt
output_2025-11-27_14:28:24.txt  output_2025-11-27_14:30:11.txt
output_2025-11-27_14:28:33.txt  output_2025-11-27_14:30:20.txt
output_2025-11-27_14:28:41.txt  output_2025-11-27_14:30:29.txt
output_2025-11-27_14:28:50.txt
LaptopMerah@LaptopMerah:case1
> cat files/output_2025-11-27_14\27\13.txt
"CHUCK NORRIS IS ACTUALLY B***** O FROM LONDON"
LaptopMerah@LaptopMerah:case1
> |
```

diantar 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

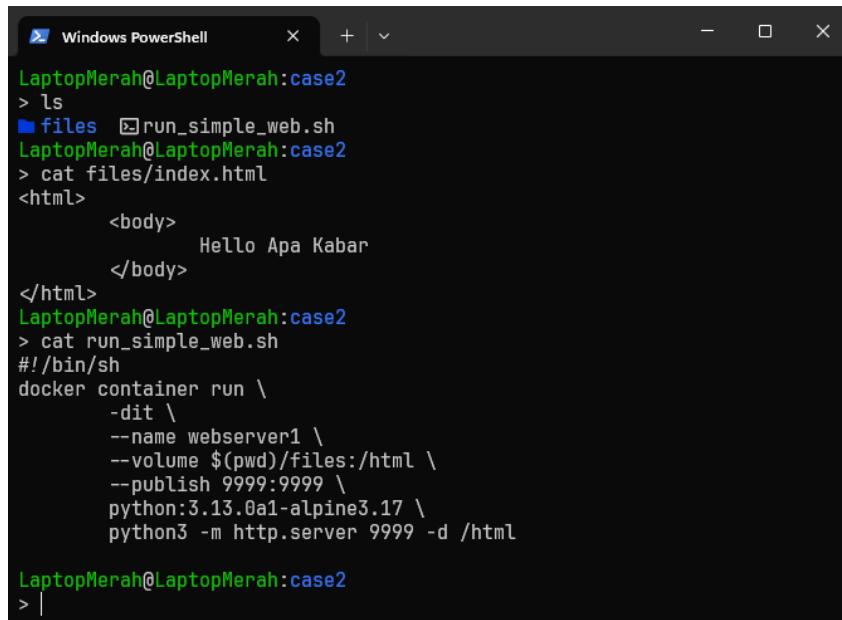


```
LaptopMerah@LaptopMerah:case1
> docker rm -f myprocess1
myprocess1
LaptopMerah@LaptopMerah:case1
> docker ps
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS    PORTS     NAMES
LaptopMerah@LaptopMerah:case1
> ls files/
📄 output_2025-11-27_14:27:13.txt 📄 output_2025-11-27_16:22:01.txt
📄 output_2025-11-27_14:27:22.txt 📄 output_2025-11-27_16:22:10.txt
📄 output_2025-11-27_14:27:30.txt 📄 output_2025-11-27_16:22:19.txt
📄 output_2025-11-27_14:27:39.txt 📄 output_2025-11-27_16:22:28.txt
📄 output_2025-11-27_14:27:48.txt 📄 output_2025-11-27_16:22:38.txt
📄 output_2025-11-27_14:27:56.txt 📄 output_2025-11-27_16:22:47.txt
📄 output_2025-11-27_14:28:07.txt 📄 output_2025-11-27_16:22:55.txt
```

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`.

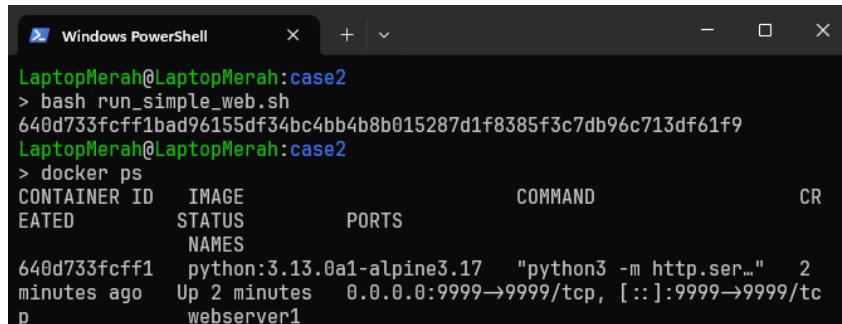


```
LaptopMerah@LaptopMerah:case2
> ls
  files run_simple_web.sh
LaptopMerah@LaptopMerah:case2
> cat files/index.html
<html>
  <body>
    Hello Apa Kabar
  </body>
</html>
LaptopMerah@LaptopMerah:case2
> cat run_simple_web.sh
#!/bin/sh
docker container run \
  -dit \
  --name webserver1 \
  --volume $(pwd)/files:/html \
  --publish 9999:9999 \
  python:3.13.0a1-alpine3.17 \
  python3 -m http.server 9999 -d /html

LaptopMerah@LaptopMerah:case2
> |
```

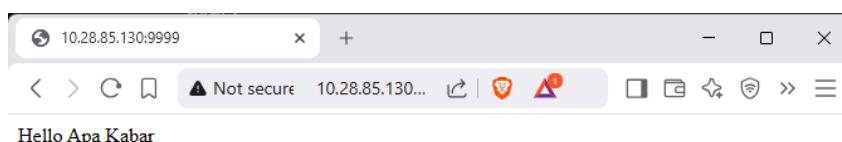
`files/index.html` merupakan html simple yang akan digunakan untuk web yang akan kita buat.

`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. 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.

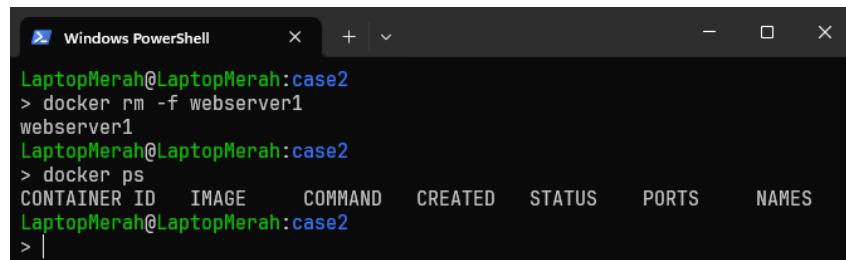


```
LaptopMerah@LaptopMerah:case2
> bash run_simple_web.sh
640d733fcff1bad96155df34bc4bb4b8b015287d1f8385f3c7db96c713df61f9
LaptopMerah@LaptopMerah:case2
> docker ps
CONTAINER ID   IMAGE          COMMAND
EATED          STATUS        NAMES
640d733fcff1  python:3.13.0a1-alpine3.17 "python3 -m http.ser..." 2
minutes ago   Up 2 minutes  0.0.0.0:9999→9999/tcp, [::]:9999→9999/tc
p              webserver1
```

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.



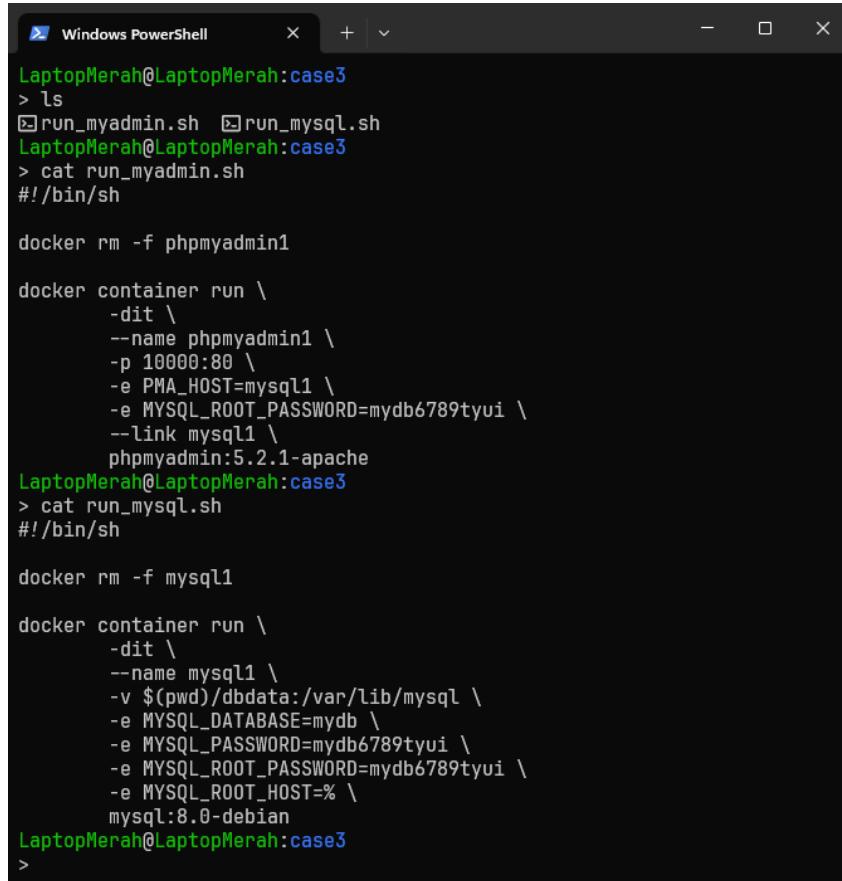
A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command history shows:

```
LaptopMerah@LaptopMerah:case2
> docker rm -f webserver1
webserver1
LaptopMerah@LaptopMerah:case2
> docker ps
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS    PORTS      NAMES
LaptopMerah@LaptopMerah:case2
>
```

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`.



```
LaptopMerah@LaptopMerah:case3
> ls
run_myadmin.sh  run_mysql.sh
LaptopMerah@LaptopMerah:case3
> cat run_myadmin.sh
#!/bin/sh

docker rm -f phpmyadmin1

docker container run \
-dit \
--name phpmyadmin1 \
-p 10000:80 \
-e PMA_HOST=mysql1 \
-e MYSQL_ROOT_PASSWORD=mydb6789tyui \
--link mysql1 \
phpmyadmin:5.2.1-apache
LaptopMerah@LaptopMerah:case3
> cat run_mysql.sh
#!/bin/sh

docker rm -f mysql1

docker container run \
-dit \
--name mysql1 \
-v $(pwd)/dbdata:/var/lib/mysql \
-e MYSQL_DATABASE=mydb \
-e MYSQL_PASSWORD=mydb6789tyui \
-e MYSQL_ROOT_PASSWORD=mydb6789tyui \
-e MYSQL_ROOT_HOST=% \
mysql:8.0-debian
LaptopMerah@LaptopMerah:case3
>
```

`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.

```

Windows PowerShell
LaptopMerah@LaptopMerah:case3
> bash run_mysql.sh
mysql1
83d48b0d1d18414d55fcb21f8ae3e35d71bae5acb5a84e8776e89cfb78bde667
LaptopMerah@LaptopMerah:case3
> bash run_myadmin.sh
phpmyadmin1
4c184caaefccc324f52649fdbf8ee9eb94dbdeaca3b5e8b9f9cfbac9b999b770
LaptopMerah@LaptopMerah:case3
> docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED
ED             STATUS              PORTS                 NAMES
4c184caaefcc  phpmyadmin:5.2.1-apache "/docker-entrypoint..."  3 seconds ago
                   Up 2 seconds      0.0.0.0:10000→80/tcp, [::]:10000→80/tcp
                   phpmyadmin1
83d48b0d1d18  mysql:8.0-debian  "docker-entrypoint.s..."  9 seconds ago
                   Up 8 seconds     3306/tcp, 33060/tcp
                   mysql1

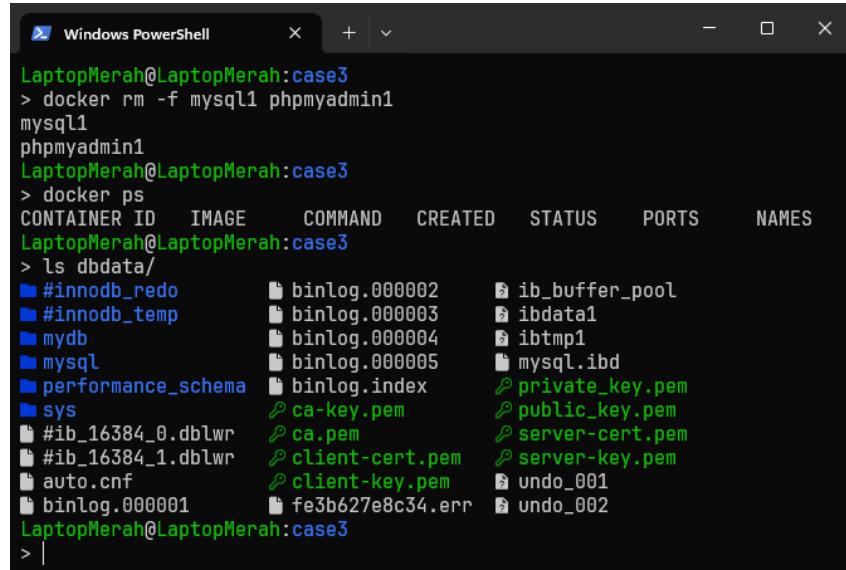
```

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

The screenshot shows the phpMyAdmin interface running on port 10000. The left sidebar lists databases: New, information_schema, mydb, mysql, performance_schema, and sys. The main content area is divided into several panels:

- General settings:** Includes options for Change password, Server connection collation (set to utf8mb4_unicode_ci), and More settings.
- Appearance settings:** Includes Language (set to English) and Theme (set to pmahomme).
- Database server:** Displays server details: Server: mysql1 via TCP/IP, Server type: MySQL, Server connection: SSL is not being used, Server version: 8.0.44 - MySQL Community Server - GPL, Protocol version: 10, User: root@172.17.0.4, and Server charset: UTF-8 Unicode (utf8mb4).
- Web server:** Displays web server details: Apache/2.4.62 (Debian), Database client version: libmysql - mysqlnd 8.2.27, PHP extension: mysqli, curl, mbstring, sodium, and PHP version: 8.2.27.

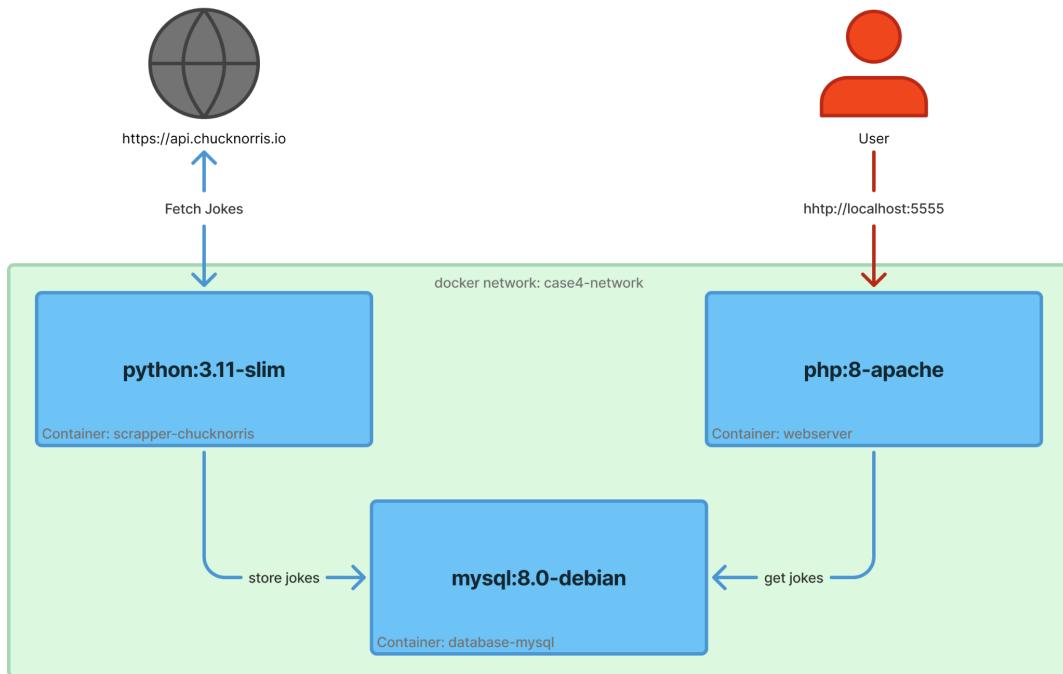
Sudah terlihat bahwa pada port 10000 yang berjalan di ip 10.28.85.130 sudah tersajikan phpmyadmin yang sudah bisa berjalan dan tersajikan. dengan login dengan username 'root' dan password mydb6789tyui sesuai env yang sudah di set pada docker tersebut.



```
LaptopMerah@LaptopMerah:case3
> docker rm -f mysql1 phpmyadmin1
mysql1
phpmyadmin1
LaptopMerah@LaptopMerah:case3
> docker ps
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS    PORTS      NAMES
LaptopMerah@LaptopMerah:case3
> ls dbdata/
  ■ #innodb_redo          ■ binlog.000002      ■ ib_buffer_pool
  ■ #innodb_temp          ■ binlog.000003      ■ ibdata1
  ■ mydb                  ■ binlog.000004      ■ ibtmp1
  ■ mysql                 ■ binlog.000005      ■ mysql.ibd
  ■ performance_schema    ■ binlog.index      ■ private_key.pem
  ■ sys                   ■ ca-key.pem        ■ public_key.pem
  ■ #ib_16384_0 dblwr     ■ ca.pem           ■ server-cert.pem
  ■ #ib_16384_1 dblwr     ■ client-cert.pem  ■ server-key.pem
  ■ auto.cnf              ■ client-key.pem   ■ undo_001
  ■ binlog.000001         ■ fe3b627e8c34.err  ■ undo_002
LaptopMerah@LaptopMerah:case3
> |
```

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 realistik dan menyerupai lingkungan produksi.

```
Windows PowerShell
LaptopMerah@LaptopMerah:case4
> ls
scripts  main.sh      run_scrapper.sh
web      run_database.sh  run_webserver.sh
LaptopMerah@LaptopMerah:case4
> cat main.sh
#!/bin/bash

docker network create case4-network 2>/dev/null || echo "Network already exists"

bash run_database.sh
bash run_scrapper.sh
bash run_webserver.sh

echo "All containers started successfully!"
```

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 menjalankan 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.

```

Windows PowerShell
LaptopMerah@LaptopMerah:case4
> docker ps
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS      PORTS     NAMES
MES
LaptopMerah@LaptopMerah:case4
> bash main.sh
Network already exists
database-mysql
fb50894c252b8b3f512bb9695fabe5feb828fc206486dafe80756b332bedc272
Database MySQL started!
scrapper-chucknorris
1d6497801fa2127fe57050b2ac0117c9197686695d8f2d342623f176e2a87fe6
Scrapper started!
webserver
993b0a3fd683bcdd26d580b94cc81e67c466adf4f9ccedf65e8a110743ed7256
Webserver started!
All containers started successfully!
LaptopMerah@LaptopMerah:case4
> docker ps
CONTAINER ID   IMAGE      COMMAND      CREATED
           STATUS      PORTS
NAMES
993b0a3fd683   php:8-apache      "docker-php-entrypoi..."   12 seconds ago   Up 11 seconds   0.0.0.0:5555→80/tcp, [::]:5555→80/tcp
webserver
1d6497801fa2   python:3.11-slim   "bash -c 'pip instal..."   12 seconds ago   Up 12 seconds
scrapper-chucknorris
fb50894c252b   mysql:8.0-debian   "docker-entrypoint.s..."   12 seconds ago   Up 12 seconds   3306/tcp, 33060/tcp
database-mysql
LaptopMerah@LaptopMerah:case4
>

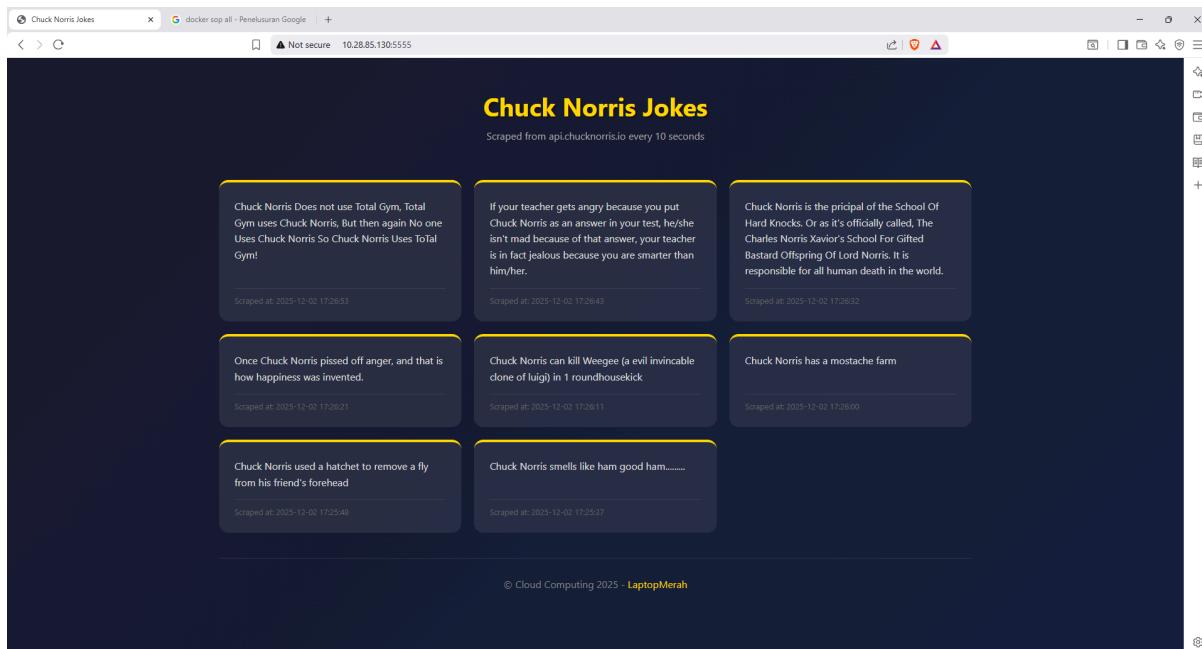
```

```

Windows PowerShell
LaptopMerah@LaptopMerah:case4
> ls DB-data/
  #innodb_redo      binlog.000001      ibdata1
  #innodb_temp      binlog.000002      ibtmp1
  db_case4          binlog.index
  mysql             ca-key.pem       private_key.pem
  performance_schema ca.pem          public_key.pem
  sys               client-cert.pem  server-cert.pem
  #ib_16384_0 dblwr  client-key.pem  server-key.pem
  #ib_16384_1 dblwr  fb50894c252b.err
  auto.cnf          ib_buffer_pool   undo_001
  LaptopMerah@LaptopMerah:case4
> |

```

Bukti bahwa script `main.sh` sudah berhasil dijalankan dan menghasilkan 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 berisi 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.