Moch. Avin
5025221061

Laporan Tugas 1 Komputasi Awan
Link repo: https://github.com/mochavin/komputasi-awan
Deskripsi tugas: Tugas 1

Pendahuluan

Laporan ini membahas implementasi *cases* yang berkaitan dengan *containerization* menggunakan Docker. Docker adalah platform yang memungkinkan pembuatan, pengujian, dan penerapan aplikasi dengan cepat dalam *isolated environment* yang disebut container. Dengan container, aplikasi dapat dijalankan secara konsisten di berbagai lingkungan komputasi, mengatasi masalah "it works on my machine".

Tujuan dari tugas ini adalah untuk mempraktikkan konsep-konsep dasar Docker melalui serangkaian studi kasus yang telah ditentukan. Konsep-konsep tersebut mencakup cara menjalankan container dari sebuah image, manajemen volume untuk *persistent* data agar data tidak hilang saat container berhenti, pemetaan port untuk mengakses layanan dari mesin host, serta interaksi antar-container. Laporan ini akan menjelaskan langkah-langkah pengerjaan untuk setiap studi kasus yang diberikan, disertai dengan penjelasan, screenshot sebagai bukti eksekusi.

Tugas yang Diberikan

Tugas ini berfokus pada studi kasus menggunakan Docker, dengan rincian sebagai berikut:

1. **Menjalankan Studi Kasus yang Ada:** Menjalankan 3 (tiga) studi kasus yang telah disediakan pada repositori github. Untuk setiap kasus, diperlukan dokumentasi yang mencakup cara menjalankan, screenshot hasil, dan penjelasan proses.
2. **Mengembangkan Studi Kasus Baru:** Mengembangkan sebuah proyek kreasi sendiri (disebut Case 4) yang berbasis pada repositori yang sama. 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 untuk menjalankan container.
    - Screenshot hasil dan penjelasannya.

Moch. Avin
5025221061

Case 1: Save persistent data chuck norris joke

Pada kasus ini, mula mula terdapat file `run_process.sh` dan juga folder script yang berisi `getjokes.sh`

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case1$ ls -l
total 12
drwxr-xr-x 2 root root 4096 Oct 14 14:17 files
-rw-rw-r-- 1 cc25 cc25  219 Oct 15 22:28 run_process.sh
drwxrwxr-x 2 cc25 cc25 4096 Oct 14 14:00 script
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case1$ chmod +x run_process.sh
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case1$ ls -l
total 12
drwxr-xr-x 2 root root 4096 Oct 14 14:17 files
-rwxrwxr-x 1 cc25 cc25  219 Oct 15 22:28 run_process.sh
drwxrwxr-x 2 cc25 cc25 4096 Oct 14 14:00 script
```

Gambar 1.1 List direktori case 1

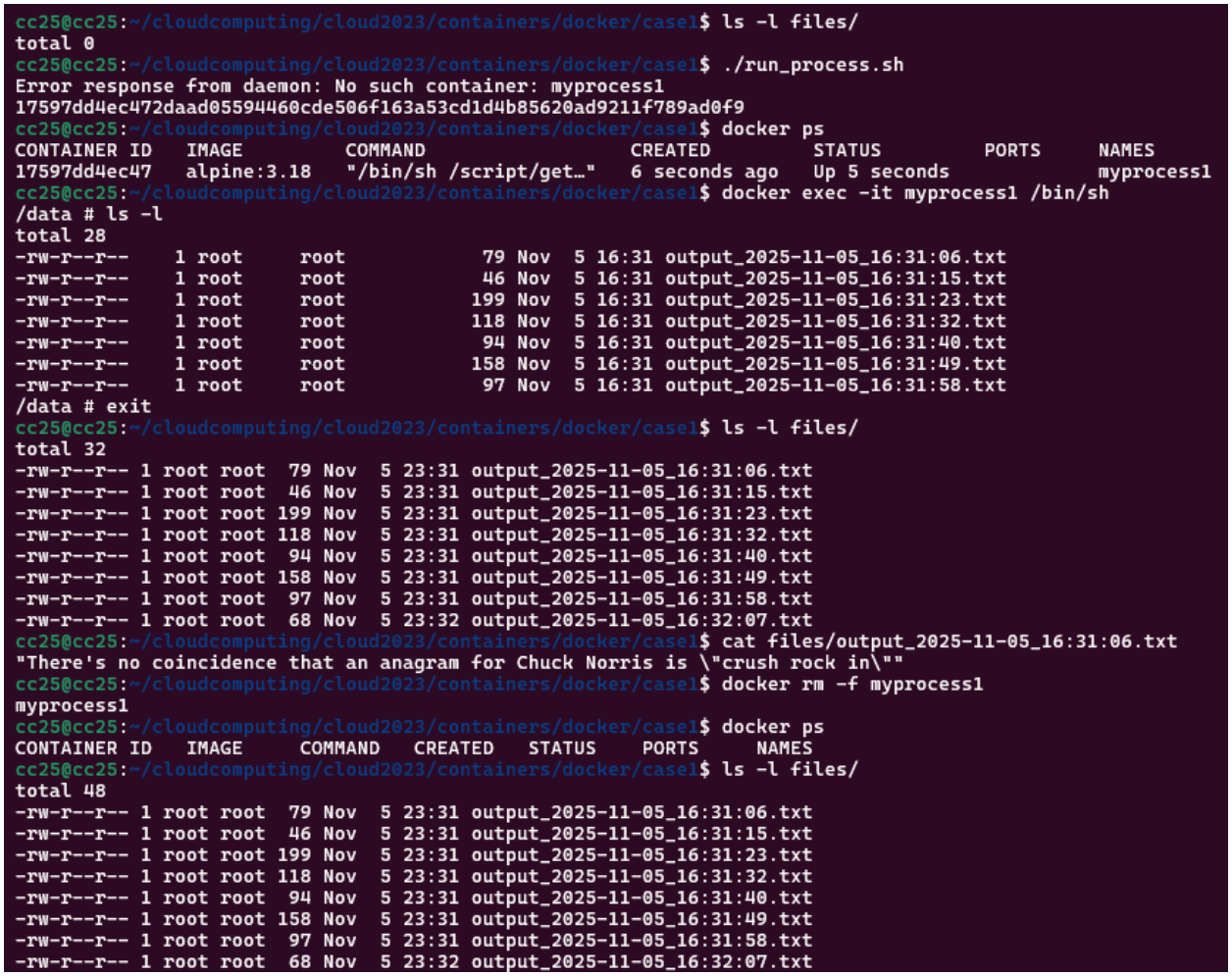
Pada Gambar 1.1, terlihat file `run_process.sh` masih belum mempunyai akses untuk *execute,* untuk menjalankan `run_process.sh` perlu untuk change mode dengan cara `chmod +x run_process.sh` lalu akan terlihat tambahan *permission* untuk eksekusi.

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/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


cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/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 $DELAY
done
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case1$
```

Gambar 1.2 Isi code *shell* pada *case* 1

Pada kasus ini terdapat 2 *shell script* yang bisa dijalankan, Seperti yang terlihat pada Gambar 1.2 terdapat `run_process.sh` berisi script imperatif untuk run docker container di dalam image *alpine* dengan tag volume dari direktori `files` dan direktori `script` dimana artinya data akan *persist* di direktori *host*-nya meskipun containternya mati. Container tersebut akan *init* run script `getjokes.sh`.

Pada `getjokes.sh` berisi *shell script* untuk fetch jokes dari API dan simpan datanya pada /data pada container yang mana nyambung dengan direktori `files` di *host*-nya. `getjokes.sh` ini akan fetch jokesnya setiap $DELAY seconds, dimana dalam kasus ini diatur pada tag -e pada script `run_process.sh` yakni 8 seconds sekali



Gambar 1.3 Menjalankan container

Ketika run script `run_process.sh` maka akan membuat container baru yang bisa dicek menggunakan *command* `docker ps`, seperti yang terlihat pada Gambar 1.3, hasil *fetch* jokes terlihat pada workdir /data di dalam container dan juga tersimpan pada direktori /files pada *host*-nya dan juga meskipun container sudah dihapus dan sudah tidak running, data masih ada dan tidak terhapus di *disk host*-nya.

Case 2: Web Server

Pada case 2 akan dijalankan container untuk web server sederhana menggunakan di port 9999 baik di hostnya maupun di container-nya.

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case2$ ls -l
total 8
drwxrwxr-x 2 cc25 cc25 4096 Oct 14 14:00 files
-rw-rw-r-- 1 cc25 cc25  185 Oct 15 22:28 run_simple_web.sh
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case2$ ls -l files/
total 4
-rw-rw-r-- 1 cc25 cc25 50 Oct 14 14:00 index.html
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/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

cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case2$ cat files/index.html
<html>
        <body>
                Hello Apa Kabar
        </body>
</html>
```

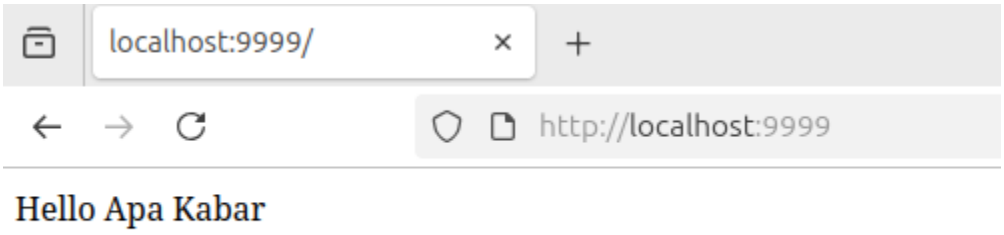
Gambar 2.1 Isi dari case 2

Seperti yang terlihat pada Gambar 2.1, terdapat *shell script* `run_simple_web.sh` untuk menjalankan container dengan volumes di /files hostnya, menggunakan container python:3.13.0a10alpine3.17 yang expose port 9999 dan init *command* nya adalah untuk start http server di dalam direktori /html dalam container. Pada /html di container nyambung dengan direktori files host yang berisi *simple html* dengan string “Hello Apa Kabar”.

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case2$ chmod +x run_simple_web.sh

cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case2$ ./run_simple_web.sh
50ba615dad19202421e186cc3ba8b8b8678f1d0d0491240d540e2d17f09def91
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case2$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STA
TUS         PORTS                                       NAMES
50ba615dad19   python:3.13.0a1-alpine3.17   "python3 -m http.ser…"   3 seconds ago   Up
3 seconds   0.0.0.0:9999->9999/tcp, [::]:9999->9999/tcp   webserver1
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case2$ docker exec -it webserver1
 /bin/sh
/ # cd /html
/html # cat index.html
<html>
        <body>
                Hello Apa Kabar
        </body>
</html>
```

Gambar 2.2 Run *shell script*

Untuk bisa eksekusi *shell script,* perlu untuk dilakukan `chmod +x run_simple_web.sh`, kemudian run *shell script*-nya. Pada Gambar 2.2, terlihat container yang run di port 9999 dengan nama `webserver1` dan juga jika dilihat di dalam direktori /html pada containernya terdapat index.html sama halnya dengan .html yang ada di ./files pada hostnya.



Gambar 2.3 Hasil localhost pada port 9999

Setelah running container web server pada port 9999, untuk memverifikasi bisa menggunakan browser dengan cara mengunjungi `localhost:9999`, dan terlihat pada Gambar 2.3 hasil dari file .html yang ada di container yang telah di-ekspos menggunakan http server sederhana.

Case 3: mysql

Pada case 3, mula mula di hanya terdapat 2 *shell script* yang keduanya berisi *shell script.*

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ ls -l
total 8
-rw-rw-r-- 1 cc25 cc25 215 Oct 15 22:28 run_myadmin.sh
-rw-rw-r-- 1 cc25 cc25 264 Oct 15 22:28 run_mysql.sh
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ cat run_mysql.sh
#!/bin/sh

docker rm -f mysql1

docker container run \
        -dit \
        --name mysql1 \
        -v $(pwd)/dbdata:/var/lib/mysql \
        -e MYSQL_DATABASE=mydb \
        -e MYSQL_PASSWORD=mydb6789tyui \
        -e MYSQL_ROOT_PASSWORD=mydb6789tyui \
        -e MYSQL_ROOT_HOST=% \
        mysql:8.0-debian
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ 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
```

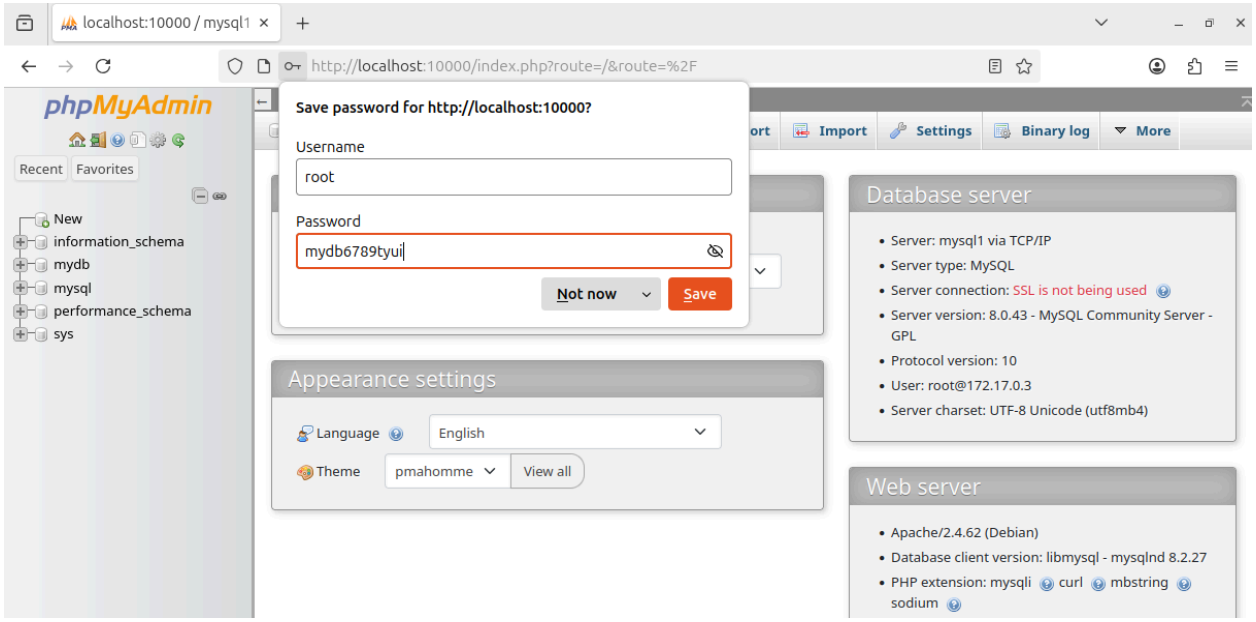
Gambar 3.1 Isi mula mula dari case 3

Pada Gambar 3.1 terlihat `run_mysql.sh` dan `run_myadmin.sh`, `run_mysql.sh` merupakan script untuk menjalankan container dengan image mysql:8.0-debian dengan tag -v untuk *persist volume* di ./dbdata pada *host*-nya dan tag -e untuk atur akun mysqlnya. Sedangkan `run_mysql.sh` merupakan script untuk menjalankan container dengan image phpmyadmin:5.2.1-apache dengan expose port 10000 dan tag -e untuk atur envnya dan –link untuk supaya bisa link ke container `mysql1`.

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ chmod +x run_mysql.sh
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ chmod +x run_myadmin.sh
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ ./run_mysql.sh
Error response from daemon: No such container: mysql1
344d5da6cf8cae299fc6fda3934f9138286bb25c2a678728b333500a21d43c7b
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ ./run_myadmin.sh
Error response from daemon: No such container: phpmyadmin1
a38756635435de8c3de26fa26359dfab7316ff587dd39403118abb8fec9213c
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                                          NAMES
a38756635435   phpmyadmin:5.2.1-apache   "/docker-entrypoint.…"   5 seconds ago    Up 4 seconds    0.0.0.0:10000->80/tcp, [::]:10000->80/tcp      phpmyadmin1
344d5da6cf8c   mysql:8.0-debian          "docker-entrypoint.s…"   9 seconds ago    Up 8 seconds    3306/tcp, 33060/tcp                            mysql1
```

Gambar 3.2 Proses menjalankan kedua container

Seperti yang terlihat pada Gambar 3.2, setelah run kedua *shell script* yang tersedia, maka akan muncul 2 container dengan nama `phpmyadmin1` untuk manajemen database mysql, dan container dengan nama `mysql1` untuk databasenya. `phpmyadmin1` mengekspos port 10000 ke hostnya, sehingga bisa diakses melalui `localhost:10000`



Gambar 3.3 Tampilan phpmyadmin setelah login

Masuk dashboard phpmyadmin dengan port 10000 dengan username “root” dan password “mydb6789tyui” sesuai dengan env yang telah di set pada `run_myadmin.sh`.

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ cat ./run_mysql.sh
#!/bin/sh

docker rm -f mysql1

docker container run \
        -dit \
        --name mysql1 \
        -v $(pwd)/dbdata:/var/lib/mysql \
        -e MYSQL_DATABASE=mydb \
        -e MYSQL_PASSWORD=mydb6789tyui \
        -e MYSQL_ROOT_PASSWORD=mydb6789tyui \
        -e MYSQL_ROOT_HOST=% \
        mysql:8.0-debian
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ ls dbdata/
 344d5da6cf8c.err   ca.pem              ibdata1         mysql.ibd            sys
 auto.cnf           client-cert.pem     ibtmp1          performance_schema   undo_001
 binlog.000001      client-key.pem     '#innodb_redo'   private_key.pem      undo_002
 binlog.000002     '#ib_16384_0.dblwr' '#innodb_temp'   public_key.pem
 binlog.index      '#ib_16384_1.dblwr'  mydb            server-cert.pem
 ca-key.pem         ib_buffer_pool      mysql           server-key.pem
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ docker rm -f mysql1 phpmy
admin1
mysql1
phpmyadmin1
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case3$ ls dbdata/
 344d5da6cf8c.err   ca.pem              ibdata1         mysql.ibd            sys
 auto.cnf           client-cert.pem     ibtmp1          performance_schema   undo_001
 binlog.000001      client-key.pem     '#innodb_redo'   private_key.pem      undo_002
 binlog.000002     '#ib_16384_0.dblwr' '#innodb_temp'   public_key.pem
 binlog.index      '#ib_16384_1.dblwr'  mydb            server-cert.pem
 ca-key.pem         ib_buffer_pool      mysql           server-key.pem
```

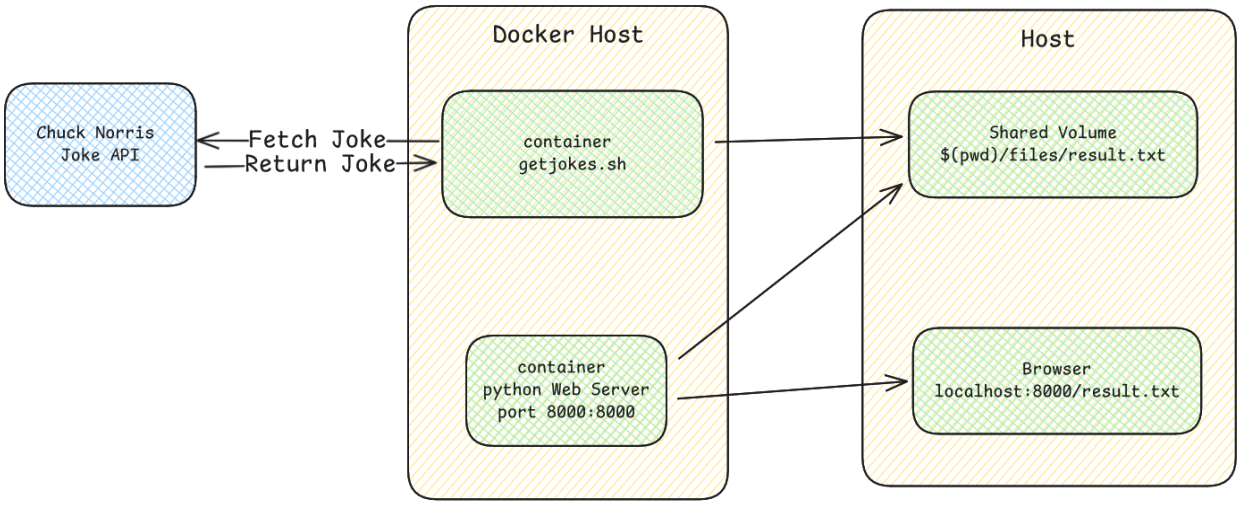
Gambar 3.4 Data pada case 3

Terlihat pada Gambar 3.4, data tetap tersimpan di hostnya di (pwd)/dbdata supaya jika container mati maka data tidak hilang.

Moch. Avin
5025221061

Case 4: Web Server Dynamic Data

Pada case 4 ini terinspirasi dari case 1 dan case 2 dimana tujuannya adalah untuk membuat *web server* dengan sumber data dari external API dan *persistent data* dengan memanfaatkan *bind volume.*



Gambar 4.1 Arsitektur Case 4

Pada Gambar 4.1 merupakan arsitektur yang akan digunakan pada case ini, dimana terdapat 2 container yakni container getjokes untuk fetch dari external API dan container web server yang digunakan untuk serve hasil joke yang telah difetch. Port yang akan diekspos dari docker host adalah port 8000 sehingga Host dapat mengakses hasil jokes nya di `localhost:8000/result.txt`.

Untuk spesifikasi VM yang digunakan pada case ini antara lain: 8192 MB Memory, 4 vCPU, 20GB Disk Size, Ip address 10.21.85.107. Port yang digunakan pada case ini hanya port 8000 yang bisa diakses melalui localhost:8000 atau 10.21.85.107 untuk web server.

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case4$ ls
run_process.sh  script  web_server.sh
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case4$ ls script/
getjokes.sh
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case4$ 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="result.txt"
        curl -sL $URL | jq '.value' > $fname
        sleep $DELAY
done
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case4$ cat run_process.sh
#!/bin/sh
docker rm -f jokefetcher1

docker container run \
        --name jokefetcher1 \
        -dit \
        -e DELAY=8 \
        -v $(pwd)/files:/data \
        -v $(pwd)/script:/script \
        --workdir /data \
        alpine:3.18 \
        /bin/sh /script/getjokes.sh
```

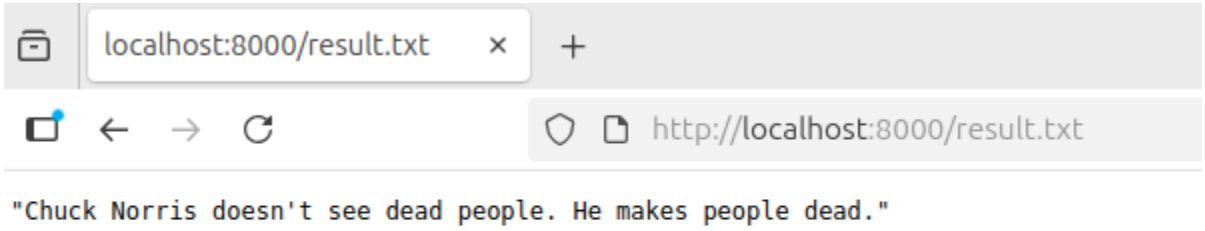
Gambar 4.1 isi case 4

Terlihat pada Gambar 4.1 terdapat `getjokes.sh` untuk fetch API dari chucknorris.io untuk disimpan dalam “result.txt” di dalam $(pwd)/files setiap 8 detik sesuai tag -e pada `run_process.sh`. `run_process.sh` digunakan untuk buat container bernama “jokefetcher1” yang dimana container ini menjalankan code dari `getjokes.sh` dan menyimpan pada $(pwd)/files/result.txt di hostnya.

```
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case4$ ./run_process.sh
jokefetcher1
01f4c20f4d3defc2654dac43abbb37140a15e165232c208877de642f97525ecf
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case4$ cat web_server.sh
#!/bin/sh
docker container run \
       -dit \
       --name webserver1 \
       --volume $(pwd)/files:/data \
       --publish 8000:8000 \
       python:3.13.0a1-alpine3.17 \
       python3 -m http.server 8000 -d /data

cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case4$ ./web_server.sh
8ea39b9864346b36f2f3fcbe41008874038e998688a45b56cc1e9d8e52ba67c7
cc25@cc25:~/cloudcomputing/cloud2023/containers/docker/case4$ docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS                                       NAMES
8ea39b986434   python:3.13.0a1-alpine3.17   "python3 -m http.ser…"   2 seconds ago    Up 2 seconds    0.0.0.0:8000->8000/tcp, [::]:8000->8000/tcp   webserver1
01f4c20f4d3d   alpine:3.18                "/bin/sh /script/get…"   7 seconds ago    Up 6 seconds                                                jokefetcher1
```

Gambar 4.2 menjalankan web server

Selanjutnya run 2 *shell code* pada case 4 ini. `web_server.sh` adalah *shell code* untuk run container web server dengan volume $(pwd)/files pada hostnya dan expose port 8000 sehingga container bisa membaca hasil fetch yang dilakukan oleh container `jokefetcher1`. Pada Gambar 4.2 terlihat terdapat 2 container yang sedang running.



Gambar 4.2 hasil localhost:8000/result.txt

Setelah kedua container running, hasil fetch bisa dilihat dalam `localhost:8000/result.txt` yang mana artinya container webserver1 berhasil untuk menampilkan data pada $(pwd)/files/result.txt yang mana merupakan hasil fetch dari container `jokefetcher1`. Hasil dari result.txt ini akan update untuk setiap 8 detik sekali sesuai dengan tag -e pada `jokefetcher1`.

Kapan Skenario ini cocok dilakukan?
Ini cocok untuk pemisahan tanggung jawab satu untuk *data producer* (fetch API dan simpan data), satu untuk *data consumer* (serve ke user). Ini meniru pola *producer-consumer* antar service. Selain itu, Cocok juga ketika ingin isolasi proses dan kemudahan deployment. Container terpisah membuat penggantian salah satu (misal ganti API source atau web framework) tidak memengaruhi yang lain.

Kesimpulan

Melalui tugas ini, saya telah mempraktikkan berbagai skenario umum penggunaan Docker. Saya belajar cara menjalankan *container* untuk mengambil data dari API secara berkala, *running web server* sederhana, hingga membangun sistem yang lebih kompleks dengan dua *container* yang saling terhubung, yaitu MySQL dan phpMyAdmin. Pada case 4, saya merancang skenario sendiri yang mengombinasikan konsep-konsep tersebut, di mana satu container bertugas sebagai *producer* yang mengambil data, dan container lainnya sebagai *consumer* yang menyajikan data tersebut kepada *user*.

Fokus pada tugas ini adalah pada konsep-konsep inti Docker. Saya memahami kegunaan manajemen volume untuk memastikan data (seperti *joke*, file website, dan database) tetap aman dan *persistent* meskipun *container* dimatikan. Saya juga mempraktikkan pemetaan port agar dapat diakses dari luar. Selain itu, saya belajar bagaimana *running* aplikasi yang terdiri dari beberapa bagian melalui interaksi antar-container, baik melalui *linking* maupun volume bersama, sehingga setiap komponen dapat berjalan secara terisolasi namun tetap terhubung.