

Laporan Tugas 1 Komputasi Awan

Nathaniel Ryo Kurniadi – 5025221019

Link Repository : <https://github.com/rororyo/komputasi-awan-2025/tree/main/Tugas%201>

Pendahuluan

Docker merupakan platform open-source yang memungkinkan pengembang untuk mengemas aplikasi dan seluruh dependensinya ke dalam suatu kontainer yang portabel dan konsisten di berbagai lingkungan. Platform ini pertama kali dirilis pada tahun 2013 oleh Solomon Hykes. Docker membantu memastikan bahwa aplikasi dapat berjalan dengan baik di berbagai sistem operasi dan konfigurasi tanpa perlu melakukan penyesuaian besar dengan teknologi kontainerisasi yang ditawarkan.

Tugas yang Diberikan

Pada tugas kali ini, kita diminta untuk menjalankan seluruh case yang terdapat pada repositori berikut: <https://github.com/rm77/cloud2023/tree/master/containers/docker>. Pada repositori itu terdapat tiga case utama yang harus dijalankan, didokumentasikan langkah-langkahnya, dan disertai dengan screenshot dan penjelasan cara kerjanya. Selanjutnya, kita diminta mengembangkan satu case tambahan, dinamai case 4 sebagai hasil kreasi sendiri yang merupakan pengembangan dari case-case di repositori tersebut. Case 4 disertai gambar arsitektur sistem, script pendukung, serta screenshot hasil implementasi dan penjeleasnya.

Case 1

Pada case ini kita diminta untuk menjalankan script **getjokes.sh** setiap 5 detik dengan menggunakan container berbasis Alpine:3.18 untuk mengumpulkan Chuck Norris jokes dari API <https://api.chucknorris.io/jokes/random>. Sebelum menjalankan script tersebut, container perlu melakukan update repository apk dan menginstal package jq.

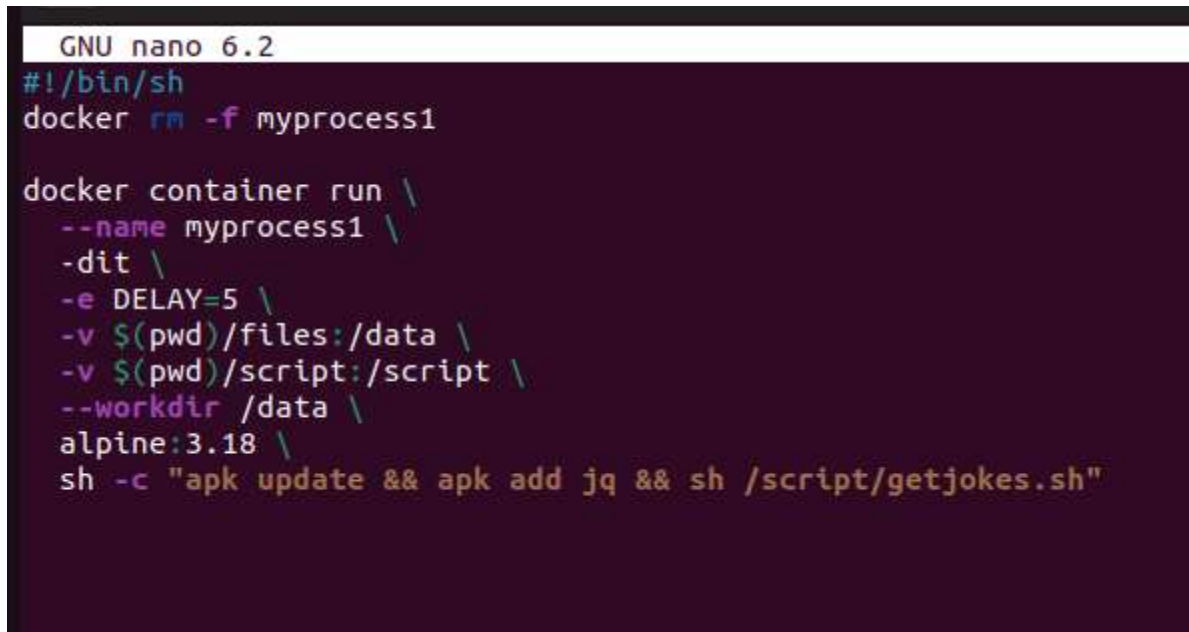
Script getjokes.sh harus dimount ke dalam container, di mana:

- di dalam container akan berada di direktori `/script`
- di host (komputer lokal) akan berada di direktori `./script`

File hasil pengambilan joke akan disimpan di direktori **/data** di dalam container yang dipetakan (mounted) ke direktori `./files` di host

Setelah proses berjalan di background, kita dapat mengecek direktori **./files** di host untuk melihat kumpulan joke yang telah dikoleksi.

Langkah pertama adalah update script **run_process.sh**. Kita ubah delay 8 detik, menjadi 5 detik dan tambahkan kode untuk update apk repo dan install jq. Untuk perubahan update apk repo maka bisa gunakan perintah **“apk update”**, dan untuk install jq tambahkan perintah **“apk add jq”**. Sehingga hasil akhir dari **run_process.sh** adalah sebagai berikut:



```
GNU nano 6.2
#!/bin/sh
docker rm -f myprocess1

docker container run \
  --name myprocess1 \
  -dit \
  -e DELAY=5 \
  -v $(pwd)/files:/data \
  -v $(pwd)/script:/script \
  --workdir /data \
  alpine:3.18 \
  sh -c "apk update && apk add jq && sh /script/getjokes.sh"
```

Penjelasan kode dari script berikut adalah sebagai berikut:

1. **“docker rm -f myprocess1”** : Pertama kita hapus container yang bernama myprocess1, fungsinya adalah memungkinkan kita menjalankan script ini berulang kali tanpa harus hapus container lama secara manual.
2. **“docker container run”**: Baris ini menjalankan container dengan spesifikasi yang telah ditentukan :
 - **Nama container**: myprocess1
 - **dit**: detached, jadi berjalan di background, sehingga terminal dapat kita gunakan untuk hal lain
 - **e (environment variable)**: memasukkan environment variable bernama DELAY dengan nilai 5 untuk menentukan berapa detik sekali getjokes.sh dijalankan.
 - **Volume**: kita akan mount folder dari host ke container agar dapat di akses di dalam container. Terdapat 2 folder yang akan kita mount /mapping ke dalam container. Yang pertama adalah folder files di mapping ke folder data di dalam container. Folder ini berfungsi untuk melihat output dari getjokes.sh. Lalu, yang ke 2 adalah folder script di host di mapping ke folder script di container. Fungsinya adalah agar di dalam container dapat tetap menjalankan getjokes.sh
 - **“--workdir data”**: Menetapkan folder data sebagai folder utama, ketika mengakses container, maka langsung diarahkan ke folder tersebut.
 - Container akan menggunakan alpine:3.18

- `"sh -c 'apk update && apk add jq && sh /script/getjokes.sh'"` : Perintah ini berguna untuk update daftar package alpine, install jq, dan menjalankan script `getjokes.sh`.

Setelah script `run_process.sh` di modifikasi, maka langkah selanjutnya adalah menjalankannya dengan perintah `"sh run_process.sh"`. Selanjutnya, kita dapat mengecek apakah container berhasil dijalankan dengan menggunakan perintah `"docker ps"`. Hasilnya dapat dilihat pada gambar berikut:

```
ryo@ryo-virtual-machine:~/tugas-cc-1-container/case1$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
42df0f7ccfde	alpine:3.18	"sh -c 'apk update &..."	17 seconds ago	Up 16 seconds		myprocess1

Bila status nya sudah `"Up"`, maka script yang dijalankan benar, dan container docker telah berhasil dijalankan. Untuk mengecek apakah container berjalan sesuai fungsi (get chuck norris joke dan menulis ke dalam file txt), maka dapat kita akses folder `files` di host, karena tadi kita sudah memetakannya.

```
ryo@ryo-virtual-machine:~/tugas-cc-1-container/case1/files$ ls -al
```

```
total 32
drwxr-xr-x 2 root root 4096 Okt 15 21:00 .
drwxrwxr-x 4 ryo ryo 4096 Okt 15 21:00 ..
-rw-r--r-- 1 root root 41 Okt 15 21:00 output_2025-10-15_14:00:30.txt
-rw-r--r-- 1 root root 42 Okt 15 21:00 output_2025-10-15_14:00:36.txt
-rw-r--r-- 1 root root 72 Okt 15 21:00 output_2025-10-15_14:00:41.txt
-rw-r--r-- 1 root root 149 Okt 15 21:00 output_2025-10-15_14:00:47.txt
-rw-r--r-- 1 root root 141 Okt 15 21:00 output_2025-10-15_14:00:52.txt
-rw-r--r-- 1 root root 107 Okt 15 21:00 output_2025-10-15_14:00:58.txt
```

Dapat terlihat pada gambar di atas, container berhasil menyimpan file txt ke dalam folder `files` yang berada di host. Selanjutnya, untuk melihat file-file tersebut di dalam container kita, kita dapat masuk ke dalam container dengan perintah `"docker exec -it myprocess1 sh"`. Lalu, menjalankan perintah `"ls -al"` untuk melihat isi folder data.

```
ryo@ryo-virtual-machine:~/tugas-cc-1-container/case1/files$ sudo docker exec -it myprocess1 sh
```

```
/data # ls -al
```

```
total 92
drwxr-xr-x 2 root root 4096 Oct 15 14:02 .
drwxr-xr-x 1 root root 4096 Oct 15 14:00 ..
-rw-r--r-- 1 root root 41 Oct 15 14:00 output_2025-10-15_14:00:30.txt
-rw-r--r-- 1 root root 42 Oct 15 14:00 output_2025-10-15_14:00:36.txt
-rw-r--r-- 1 root root 72 Oct 15 14:00 output_2025-10-15_14:00:41.txt
-rw-r--r-- 1 root root 149 Oct 15 14:00 output_2025-10-15_14:00:47.txt
-rw-r--r-- 1 root root 141 Oct 15 14:00 output_2025-10-15_14:00:52.txt
-rw-r--r-- 1 root root 107 Oct 15 14:00 output_2025-10-15_14:00:58.txt
-rw-r--r-- 1 root root 103 Oct 15 14:01 output_2025-10-15_14:01:03.txt
```

Pada folder data di container juga sudah berhasil terbuat file-file jokes chuck norris. Untuk mengecek apakah isi dari filenya sudah benar, maka dapat menggunakan perintah `"nano <nama_file>"`. Berikut adalah salah satu contoh isi file

```
GNU nano 6.2
"Chuck Norris is too legit to quit."
```

Bila file txt sudah terisi dengan joke chuck norris, maka container berhasil dijalankan dengan baik.

Case 2

Pada case ini kita diminta untuk menjalankan sebuah web server sederhana berbasis modul `http.server` dari Python. Web server ini akan melayani (serve) file-file yang berada di direktori `./files` pada host, yang dipetakan (mounted) ke direktori `/html` di dalam container. Di dalam direktori `./files` sudah terdapat file **index.html** yang akan ditampilkan sebagai halaman utama. Web server dijalankan pada port **9999** dan port tersebut diekspos ke host pada port yang sama (**9999**). Setelah container dijalankan, kita dapat mengakses web server melalui browser dengan membuka alamat **http://[host-ip]:9999**, serta memeriksa log container yang sedang berjalan untuk melihat aktivitas web server tersebut.

Berikut adalah script yang telah disediakan di case2:

```
GNU nano 6.2
#!/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
```

Penjelasannya adalah sebagai berikut:

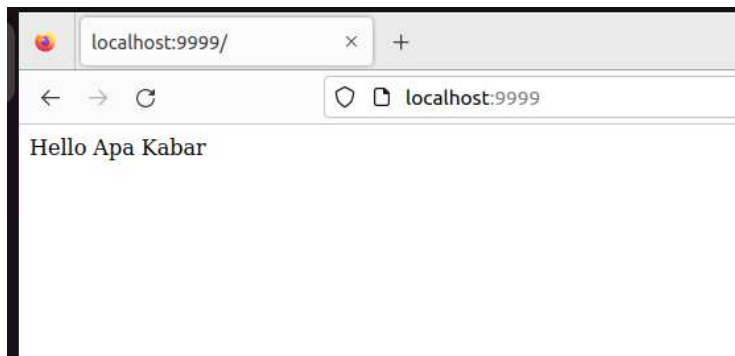
1. **“docker container run”**: Perintah ini menjalankan docker container dengan spesifikasi berikut:
 - **dit**: detached, jadi berjalan di background, sehingga terminal dapat kita gunakan untuk hal lain
 - **Nama container**: `webserver1`
 - **Volume**: Kita akan mapping folder `files` di host dengan `html` di dalam container, sehingga kita dapat mengakses folder `html` melalui folder `files` di host.
 - **“publish 9999:9999”**: Perintah ini melakukan pemetaan port dari container ke host, kita akan memetakan port `9999` dari container agar dapat diakses di port `9999` pada host.

- **“python:3.13.0a1-alpine3.17”**: Ini adalah image yang akan dipakai untuk menjalankan webserver, bila image tidak ada di dalam sistem kita, maka akan otomatis di pull terlebih dahulu
- **“python3 -m http.server 9999 -d /html”**: Kode ini menjalankan server python dengan folder html sebagai root nya.

Karena *run_simple_web.sh* sudah memenuhi kriteria case, maka cukup dijalankan dengan perintah **“sh run_simple_web.sh”**.

```
ryo@ryo-virtual-machine:~/tugas-cc-1-container/case1$ sudo docker ps -all
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
4885c01ab1d4   python:3.13.0a1-alpine3.17         "python3 -m http.ser..." 55 seconds ago Up 54 seconds 0.0.0.0:9999->9999/tcp, [::]:9999->9999/tcp  webserver1
ryo@ryo-virtual-machine:~/tugas-cc-1-container/case1$
```

Sama seperti case sebelumnya, bila container berhasil dijalankan, maka status “Up” dan nama container akan muncul. Lalu, untuk menguji apakah webserver berhasil dijalankan dan di mapping di host di port 9999, kita dapat akses <http://localhost:9999> , bila berhasil, maka akan tampil seperti gambar berikut:



Ketika <http://localhost:9999>, maka akan tampil isi dari folder html.

```
ryo@ryo-virtual-machine:~/tugas-cc-1-container/case2$ sudo docker logs -f webserver1
Serving HTTP on 0.0.0.0 port 9999 (http://0.0.0.0:9999/) ...
172.17.0.1 - - [15/Oct/2025 14:33:18] "GET / HTTP/1.1" 200 -
```

Request pun dapat dilihat pada log container webserver 1.

Case 3

Pada case ini, kita diminta untuk menjalankan MySQL dan phmMyAdmin di dua container terpisah. Data MySQL disimpan secara persisten di host dalam direktori **./dbdata**.

Berikut adalah script yang telah disediakan di case 3

run_mysql.sh

```
GNU nano 6.2 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
```

Penjelasan kodenya adalah sebagai berikut:

1. **“docker rm -f mysql1”** : Pertama, kita hapus container yang bernama mysql1, fungsinya adalah memungkinkan kita menjalankan script ini berulang kali tanpa harus hapus container lama secara manual.
2. **“docker container run”**: Perintah ini menjalankan docker container dengan spesifikasi:
 - o dit: dittached, jadi berjalan di background, sehingga terminal dapat kita gunakan untuk hal lain
 - o Nama container: mysql1
 - o **Volume**: mapping/mount **/var/lib/mysql** di container agar dapat di akses di **/dbdata** di host.
 - o **e**: Set environment variables mengenai database (nama db,password,host,dll)
 - o **“mysql: 8.0-debian”**: Menggunakan image mysql:8.0.

run_myadmin.sh

```
GNU nano 6.2 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
```

Penjelasan kodenya adalah sebagai berikut:

1. **“docker rm -f phpmyadmin1”** : Pertama, kita hapus container yang bernama phpmyadmin1, fungsinya adalah memungkinkan kita menjalankan script ini berulang kali tanpa harus hapus container lama secara manual.
2. **“docker container run”**: Perintah ini menjalankan docker container dengan spesifikasi:

- dit: detached, jadi berjalan di background, sehingga terminal dapat kita gunakan untuk hal lain.
- Nama container: phpmyadmin1
- **“-p 10000:80”**: Memetakan port 80 di container ke port 10000 di host, agar host dapat mengakses container pada port 10000.
- **e**: set environment variables untuk kredensial phpmyadmin
- **“link mysql1”**: membuat container phpmyadmin1 terhubung dengan mysql1
- **“phpmyadmin : 5.2.1-apache”**: docker image yang akan digunakan adalah phpmyadmin versi 5.2.1.

Untuk menyelesaikan case ini, hal pertama yang dilakukan adalah menjalankan script **run_mysql.sh** dengan perintah **“sh run_mysql.sh”**.

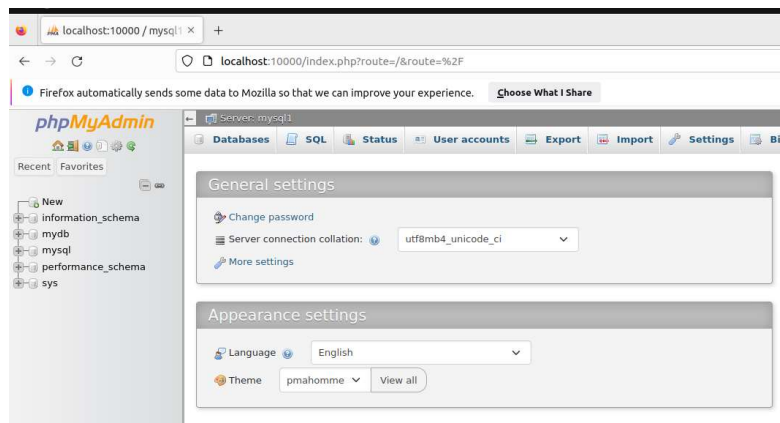
```
ryo@ryo-virtual-machine: ~/tugas-cc-1-container/case3$ sudo docker ps -all
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c3bdd8eb5a3c	mysql:8.0-debian	"docker-entrypoint.s..."	About a minute ago	Up About a minute	3306/tcp, 33060/tcp	mysql1

Setelah berhasil di run, maka image akan ter pull dan docker container berjalan. Setelah itu, langkah selanjutnya adalah menjalankan script **run_myadmin.sh** dengan perintah **“sh run_myadmin.sh”**. Setelah sudah, maka kedua container akan berjalan, seperti pada gambar berikut:

```
ryo@ryo-virtual-machine: ~/tugas-cc-1-container/case3$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ad8567950ff2	phpmyadmin:5.2.1-apache	"/docker-entrypoint.s..."	7 seconds ago	Up 6 seconds	0.0.0.0:10000->80/tcp, [::]:10000->80/tcp	phpmyadmin1
942be5d7af11	mysql:8.0-debian	"docker-entrypoint.s..."	About a minute ago	Up About a minute	3306/tcp, 33060/tcp	mysql1



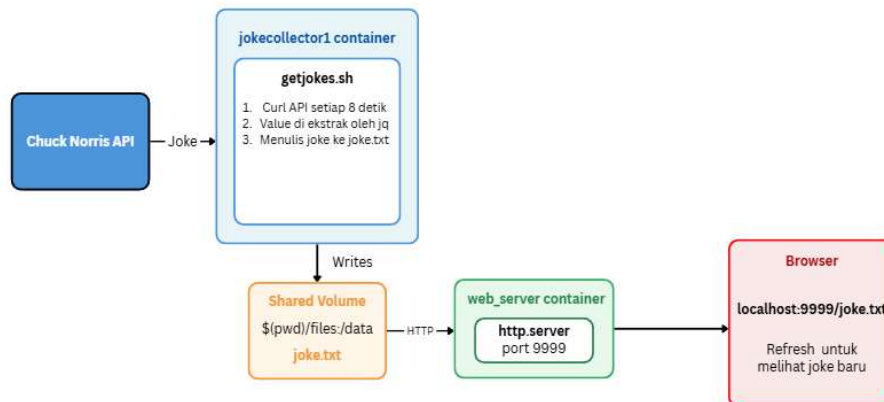
```
ryo@ryo-virtual-machine: ~/tugas-cc-1-container/case3$ ls
```

auto.cnf	binlog.000003	ca-key.pem	client-key.pem	ib_buffer_pool	private_key.pem	server-key.pem	undo_002
binlog.000001	binlog.index	ca.pem	#ib_16384_0.dblwr	ibdata1	public_key.pem	server-cert.pem	undo_001
binlog.000002	c3bdd8eb5a3c.err	client-cert.pem	#ib_16384_1.dblwr	ibtmp1	performance_schema		

Setelah kedua container tersebut berhasil dijalankan, maka host dapat mengakses <http://localhost:10000> dan masuk ke phpmyadmin dengan username= root, dan password = mydb6789tyui untuk mengakses panel phpmyadmin. Setelah itu, dapat dicek folder db data, dan terlihat data tersimpan secara persistent.

Case 4

Pada case4 , akan dibuat modifikasi dari case1 dan 2, dimana kita akan membuat 2 buah container, 1 bertugas untuk mendapatkan joke, dan 1 lagi untuk menampilkan joke tersebut di web server. Berikut adalah gambar arsitektur yang dipakai untuk case4



Cara kerjanya adalah **getjokes.sh** akan mengambil joke dari chuck norris API setiap 8 detik, lalu di tulis ke dalam **files/joke.txt**. Karena sharing volume, maka **web_server_container** juga dapat mengakses file **joke.txt**, lalu ditampilkan ke halaman browser di alamat <http://localhost:9999/joke.txt>. Untuk spek VM case4 adalah sebagai berikut:

1. Memory: 4096 MB
2. vCPU: 2
3. Disk Size: 20GB
4. IP address: dinamis

Port yang digunakan hanya port 9999 untuk host web server. Langkah berikutnya adalah memodifikasi script-script yang telah ada agar sesuai dengan tujuan case4.


```
GNU nano 6.2 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="joke.txt"
    curl -sL $URL | jq '.value' > $fname
    sleep $DELAY
done
```

Gambar di atas adalah modifikasi dari script case1 , dimana fname diganti dengan **joke.txt**, agar tidak lagi membuat file baru sesuai dengan tanggal, namun hanya mengganti konten **joke.txt** dengan joke baru.

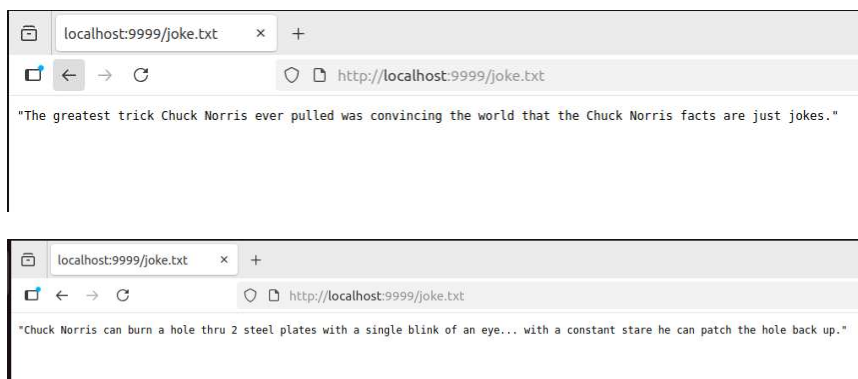
```
GNU nano 6.2 run_process.sh
#!/bin/sh
docker rm -f jokecollector1

docker container run \
    --name jokecollector1 \
    -dit \
    -e DELAY=8 \
    -v $(pwd)/files:/data \
    -v $(pwd)/script:/script \
    --workdir /data \
    alpine:3.18 \
    /bin/sh /script/getjokes.sh
```

Gambar di atas adalah hasil modifikasi dari script case1. Dimana nama container di ubah menjadi **jokecollector1** dan memetakan folder **data** di container dengan folder **files** di host.

```
GNU nano 6.2                                web_server.sh
#!/bin/sh
docker rm -f webserver1
docker run -dit \
  --name webserver1 \
  -v $(pwd)/files:/data \
  -p 9999:9999 \
  python:3.13.0a1-alpine3.17 \
  python3 -m http.server 9999 -d /data
```

Gambar di atas adalah hasil modifikasi dari script case2. Dimana kita memetakan folder **data** di container dengan folder **files** di host, dan mengubah penyajian folder **html** dengan folder **data** di container. Setelah semuanya berhasil dijalankan, maka hasilnya akan seperti berikut:



Joke akan diperbarui setiap 8 detik sekali dengan melakukan refresh pada page secara berkala.

Kapan Skenario ini Cocok Digunakan?

Skenario ini cocok digunakan untuk aplikasi yang membutuhkan pemisahan tugas antara pengumpulan data dan penyajian data secara efisien, seperti:

- Aplikasi yang secara berkala mengambil data dari sumber eksternal (seperti API) dan menampilkannya secara real-time di web.
- Penerapan arsitektur microservice sederhana

Kesimpulan

Dari tugas ini, saya mempelajari dasar penggunaan Docker pada Case 1–3 serta pengembangannya pada Case 4 yang memisahkan proses pengambilan dan penyajian data ke dua container terhubung. Melalui semua case, saya memahami dasar konsep dasar docker dan pengaplikasian sederhananya untuk containerisasi.