Tugas 1 Cloud Computing

Rafhi Sadipta – 5025221014

Link Repository: https://github.com/RafhiSadipta/cloud-computing-2025/tree/main/tugas1

Spesifikasi Laptop

Merek	Acer
Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz (2.42 GHz)
RAM	16 GB
System Type	64-bit operating system, x64-based processor

Spesifikasi Virtual Machine (Ubuntu)

Virtualizer	VMware Workstation
RAM	4 GB
CPU	2
Disk	20 GB
Operation System	Ubuntu 64-bit
IP	192.168.88.128/24

1. Pendahuluan

1.1 Latar Belakang

Dalam era komputasi modern, teknologi container seperti Docker menjadi solusi efektif untuk menjalankan aplikasi secara konsisten di berbagai lingkungan. Docker memungkinkan pengembang mengemas aplikasi dan dependensinya dalam satu wadah yang ringan, portabel, dan mudah dikelola. Melalui pembelajaran ini, mahasiswa diperkenalkan pada konsep dasar containerization dan penerapannya dalam pengembangan serta deployment aplikasi berbasis cloud.

1.2 Tujuan

Tugas ini bertujuan untuk.

1. Mempelajari konsep dasar dan penggunaan Docker.
2. Menjalankan dan memahami setiap case pada repository yang diberikan.
3. Mengembangkan satu skenario tambahan sebagai bentuk penerapan konsep Docker
4. Mendokumentasikan hasil percobaan dan analisis secara sistematis.

2. Tugas Yang Diberikan

Tugas ini berfokus pada penerapan konsep containerization menggunakan Docker dengan menjalankan tiga case yang tersedia pada repository https://github.com/rm77/cloud2023/tree/master/containers/docker. Setiap case memiliki skenario berbeda yang dirancang untuk memperkenalkan berbagai aspek dasar penggunaan Docker, mulai dari pembuatan dan konfigurasi container, pengelolaan image, hingga komunikasi antar-container.

Selain menjalankan ketiga case tersebut, kita juga diminta untuk mengembangkan satu skenario tambahan sebagai Case 4 berdasarkan kreativitas sendiri. Skenario tambahan ini diharapkan dapat

menunjukkan pemahaman kita dalam merancang arsitektur sederhana berbasis Docker yang relevan dengan kebutuhan nyata. Semua hasil percobaan, termasuk langkah-langkah eksekusi, cuplikan kode, tangkapan layar, dan penjelasan, didokumentasikan dalam satu laporan akhir berbentuk PDF.

3. Implementasi dan Penjelasan

3.1 Case 1

- Tujuan Case 1
  Case ini bertujuan untuk mempelajari cara menjalankan proses otomatis di dalam container Docker dengan menggunakan skrip shell. Container akan menjalankan proses secara berulang dengan jeda waktu tertentu (delay) untuk mengambil data dari API eksternal, dalam hal ini API Chuck Norris Jokes (https://api.chucknorris.io/jokes/random), lalu menyimpannya ke file di direktori lokal host.

- Cuplikan Kode / Script dan Penjelasan
  - 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 ini menunjukkan isi dari file run_process.sh, yaitu script utama yang berfungsi untuk menjalankan container baru bernama myprocess1. Script ini menghapus container lama (jika ada), lalu menjalankan container dengan image alpine:3.18. Parameter -e DELAY=8 mengatur agar proses di dalam container berjalan dengan jeda 8 detik antar-eksekusi, sedangkan opsi -v digunakan untuk melakukan mount direktori host (files dan script) agar dapat diakses di dalam container. Pada akhirnya, container akan mengeksekusi file getjokes.sh di direktori /script.

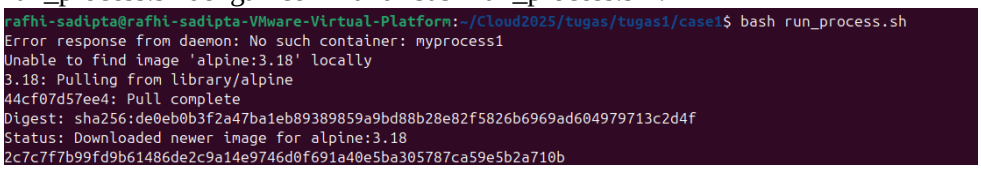
- getjokes.sh

```
#!/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
~
~
```

Gambar ini memperlihatkan isi dari **getjokes.sh**, yaitu script yang dijalankan di dalam container. Script ini pertama-tama memperbarui *package index* dan menginstal curl serta

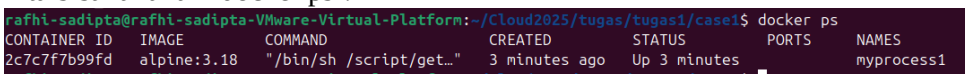
jq, dua utilitas penting untuk mengambil dan memproses data dari API. Variabel URL berisi alamat API *Chuck Norris Jokes*, dan setiap beberapa detik (sesuai nilai $DELAY), script akan mengambil data dari API tersebut. Hasil lelucon disimpan dalam file teks dengan nama berdasarkan waktu eksekusi (output_YYYY-MM-DD_HH:MM:SS.txt). Proses ini berjalan terus-menerus di dalam container.

- Cara Menjalankan
  - Install docker terlebih dahulu (jika sebelumnya belum pernah menginstall), disini saya melakukan `sudo apt-get update` terlebih dahulu lalu `sudo apt install docker.io` untuk menginstall docker ke dalam ubuntu dengan versi 28.2.2. Lalu karena disini saya melakukan `sudo usermod -aG $USER` agar user yang dipakai punya izin untuk mengakses docker daemon.
  - Jika docker sudah terinstall dan mendapata izin akses, selanjutkan jalankan file run_process.sh dengan command `bash run_process.sh`.



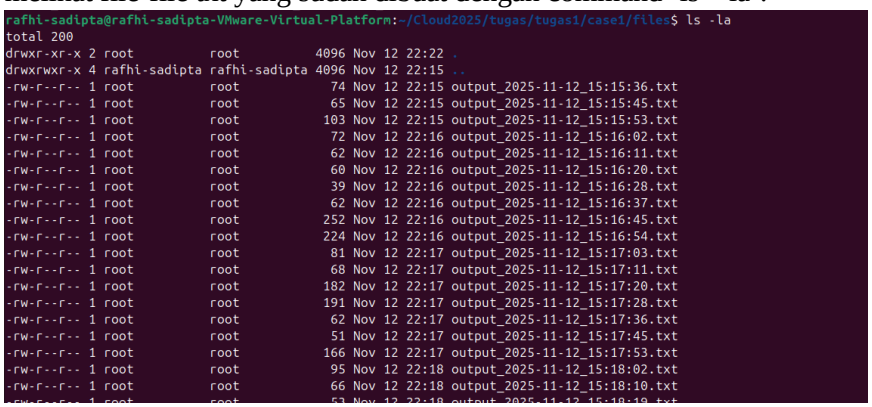
Gambar di atas menunjukkan proses eksekusi perintah bash run_process.sh yang berhasil dijalankan di terminal. Pada awalnya, Docker memberikan pesan bahwa container bernama myprocess1 belum ada, sehingga sistem kemudian melakukan penarikan (pull) image baru yaitu alpine:3.18 dari repositori Docker Hub. Setelah proses download selesai, sistem menampilkan informasi bahwa image telah berhasil diunduh dan siap digunakan. Ini menandakan bahwa Docker berfungsi dengan baik serta mampu mengambil image dari registry publik untuk menjalankan container baru.

  - Lalu untuk mencoba verifikasi ulang apakah docker benar-benar sudah berjalan, kita bisa lakukan `docker ps`.

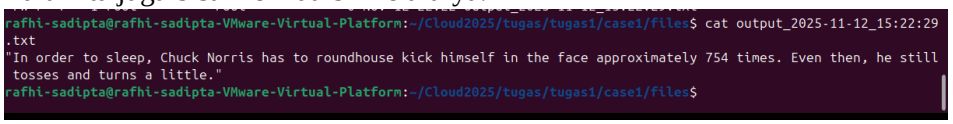


Melalui perintah ini, pengguna dapat melihat informasi penting seperti ID container, nama image yang digunakan, status container, serta port yang terbuka. Jika daftar terlihat kosong, berarti saat ini belum ada container yang sedang berjalan. Perintah ini berguna untuk memantau aktivitas container secara langsung dalam lingkungan Docker.

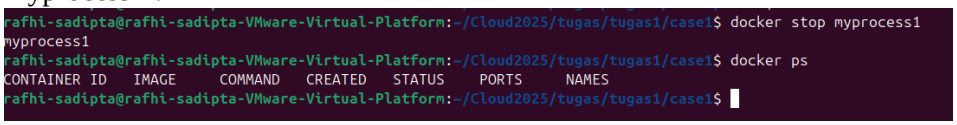
  - Karena sudah berjalan maka kita langsung bisa beralih ke folder files untuk melihat file-file txt yang sudah dibuat dengan command `ls – la`.



Informasi yang ditampilkan mencakup hak akses (permission), jumlah link, pemilik file (owner), grup, ukuran file, tanggal modifikasi terakhir, dan nama file atau folder. Perintah ini membantu pengguna memahami struktur dan detail isi direktori secara lebih mendalam, terutama ketika sedang memeriksa file yang digunakan dalam proses container Docker.
Lalu kita juga bisa melihat isi file txtnya.



- Jika kita ingin memberhentikan prosesnya, bisa dengan `docker stop myprocess1`.



Kita lihat, ketika proses docker sudah berhasil diberhentikan, ketika kita melakukan `docker ps` sudah tidak ada proses docker lagi yang terlihat sedang berjalan.

3.2 Case 2

- Tujuan Case 2
  Case 2 bertujuan untuk mendemonstrasikan proses deployment sederhana sebuah web server menggunakan Docker dengan memanfaatkan Docker Volume. Dalam skenario ini, sebuah container Docker dibuat untuk menjalankan web server berbasis Python, di mana konten (file index.html) disajikan langsung dari direktori lokal pada host ($(pwd)/files/html) yang dipasang (mount) sebagai volume (/html) di dalam container. Konfigurasi ini sangat penting karena memungkinkan pemisahan antara aplikasi (container) dan data (volume), sehingga perubahan pada konten website dapat dilakukan dengan mudah di host tanpa perlu membangun ulang image atau restart container. Selain itu, web server ini diakses melalui port 9999 pada host, yang dipetakan ke port 9999 di dalam container (-p 9999:9999).

- Cuplikan Kode / Script dan Penjelasan
  - 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
```

Script di atas digunakan untuk menjalankan sebuah container web server sederhana berbasis Python.

- -d -t menjalankan container dalam mode detached (latar belakang) dengan terminal pseudo-TTY.
- --name webserver1 memberi nama container agar mudah dikenali.
- --volume $(pwd)/files/html:/html melakukan mount direktori lokal files/html ke direktori /html di dalam container, sehingga file HTML lokal dapat diakses oleh server.

- --publish 9999:9999 membuka port 9999 agar web server di container bisa diakses dari host melalui port yang sama.
- python3 -m http.server 9999 -d /html menjalankan web server bawaan Python dan menyajikan konten dari direktori /html.

- index.html

```
<html>
        <body>
                Hello Apa Kabar
        </body>
</html>
```

File ini berfungsi sebagai halaman utama (index page) yang akan disajikan oleh web server di dalam container. Saat server berjalan, membuka alamat http://localhost:9999 di browser akan menampilkan teks sederhana “Hello Apa Kabar”, yang menandakan bahwa container dan konfigurasi volume telah berfungsi dengan benar.

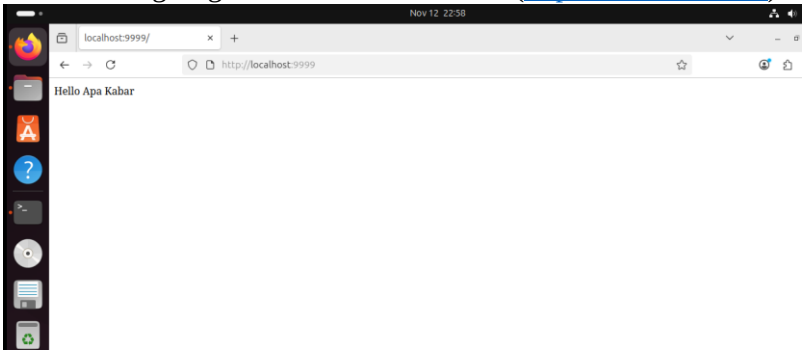
- Cara Menjalankan
  - Pertama kita masuk terlebih dahulu ke folder case2. Setelah itu kita lakukan `bash run_simple_web.sh` jika user udah ada izin akses docker, jika tidak bisa tambahkan `sudo` diawal untuk menjalankan script.

```
rafhi-sadipta@rafhi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas1/case2$ sudo bash run_simple_web.sh
[sudo] password for rafhi-sadipta:
Unable to find image 'python:3.13.0a1-alpine3.17' locally
3.13.0a1-alpine3.17: Pulling from library/python
9398808236ff: Pull complete
512e36fc6765: Pull complete
9889d7b0a730: Pull complete
ace557181643: Pull complete
cba9fa7309c5: Pull complete
Digest: sha256:7002aeb85bc6b2eaf3b8893248accda89fe4420e8606c9c6d9f53d3a170c7f0e
Status: Downloaded newer image for python:3.13.0a1-alpine3.17
2ff013dfe513f1ed223e697e8c75eb1fac603517133dcdf8c429b2451f7a8c98
```

  Kita bisa cek apakah docker sudah berhasil jalan atau belum dengan command `sudo docker ps`.

```
rafhi-sadipta@rafhi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas1/case2$ sudo docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED         STATUS         PORTS
                         NAMES
2ff013dfe513   python:3.13.0a1-alpine3.17   "python3 -m http.ser…"   2 minutes ago   Up 2 minutes   0.0.0.0:9999->9999/t
cp, [::]:9999->9999/tcp   webserver1
rafhi-sadipta@rafhi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas1/case2$
```

  Disini terlihat bahwa container bernama webserver1 telah berjalan dengan status Up dan siap melayani permintaan HTTP melalui port 9999 pada mesin host, membuktikan bahwa script run_simple_web.sh telah dieksekusi dengan benar.
  - Kita bisa langsung coba akses melalui host (http://localhost:9999) di browser.



  Disini terlihat bahwa hasil dari index.html sudah bisa diakses.

3.3 Case 3

- Tujuan Case 3

Case 3 bertujuan untuk mendemonstrasikan deployment aplikasi multi-container yang saling terhubung (Multi-Container Application) menggunakan fitur Docker Networking dan Docker Volume. Dalam skenario ini, dua layanan penting dijalankan.

- MySQL Server: Sebagai database utama untuk menyimpan data aplikasi.
- phpMyAdmin: Sebagai antarmuka web grafis (Graphical User Interface/GUI) untuk mengelola database MySQL tersebut.

Tujuan utama adalah menunjukkan bagaimana container dapat berkomunikasi satu sama lain menggunakan linking (--link mysql1) dan bagaimana data database (/var/lib/mysql) dapat dipertahankan secara persistent di host melalui Docker Volume (-v $(pwd)/dbdata:/var/lib/mysql), sehingga data tidak hilang ketika container dihentikan atau dihapus.

- Cuplikan Kode / Script dan Penjelasan

- 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
```

Script ini digunakan untuk menjalankan container MySQL versi 8.0 berbasis image Debian.

- docker rm -f mysql1 memastikan container lama dengan nama mysql1 dihapus sebelum membuat yang baru, agar tidak terjadi konflik.
- -v $(pwd)/dbdata:/var/lib/mysql melakukan mount direktori lokal dbdata untuk menyimpan data MySQL secara persisten (tidak hilang saat container dihentikan).
- Variabel -e digunakan untuk mengatur environment variable, seperti nama database, password user, dan password root.
- MYSQL_ROOT_HOST=% mengizinkan koneksi dari semua host (berguna untuk phpMyAdmin agar dapat terhubung).
- -d -t menjalankan container dalam mode detached dan terminal interaktif.

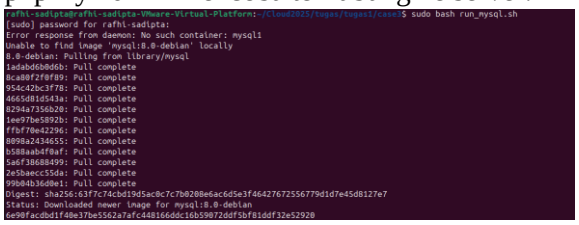
- run_myadmin.sh

```
#!/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
```

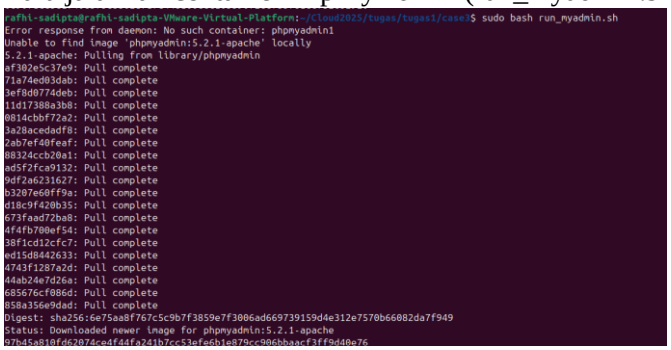
Script run_myadmin.sh menjalankan container phpMyAdmin. Sama seperti sebelumnya, container lama dihapus dengan docker rm -f phpmyadmin1. Elemen kunci di sini adalah penghubungan container dengan flag --link mysql1. Flag ini menciptakan tautan jaringan internal yang memungkinkan phpMyAdmin menggunakan nama alias mysql1 untuk mengakses database. Variabel lingkungan PMA_HOST=mysql1 memanfaatkan tautan

ini. Port 10000 pada host (-p 10000:80) dipublikasikan untuk mengakses antarmuka web phpMyAdmin.

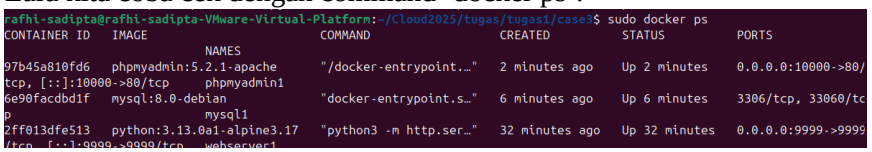
- Cara Menjalankan
  - Jalankan container MySQL (run_mysql.sh) terlebih dahulu sebelum phpMyAdmin mencoba terhubung ke server.



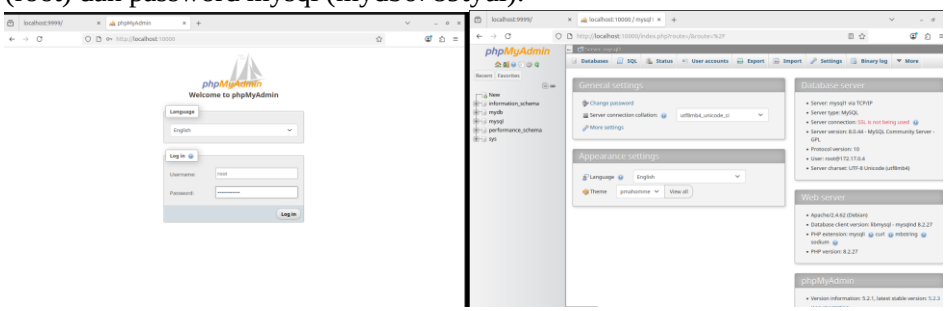
  - Lalu jalankan container PhpMyAdmin (run_myadmin.sh).



  - Lalu kita coba cek dengan command `docker ps`.



  - Kita coba akses PhpMyAdmin di http://localhost:10000 dan memasukan user (root) dan password mysql (mydb6789tyui).

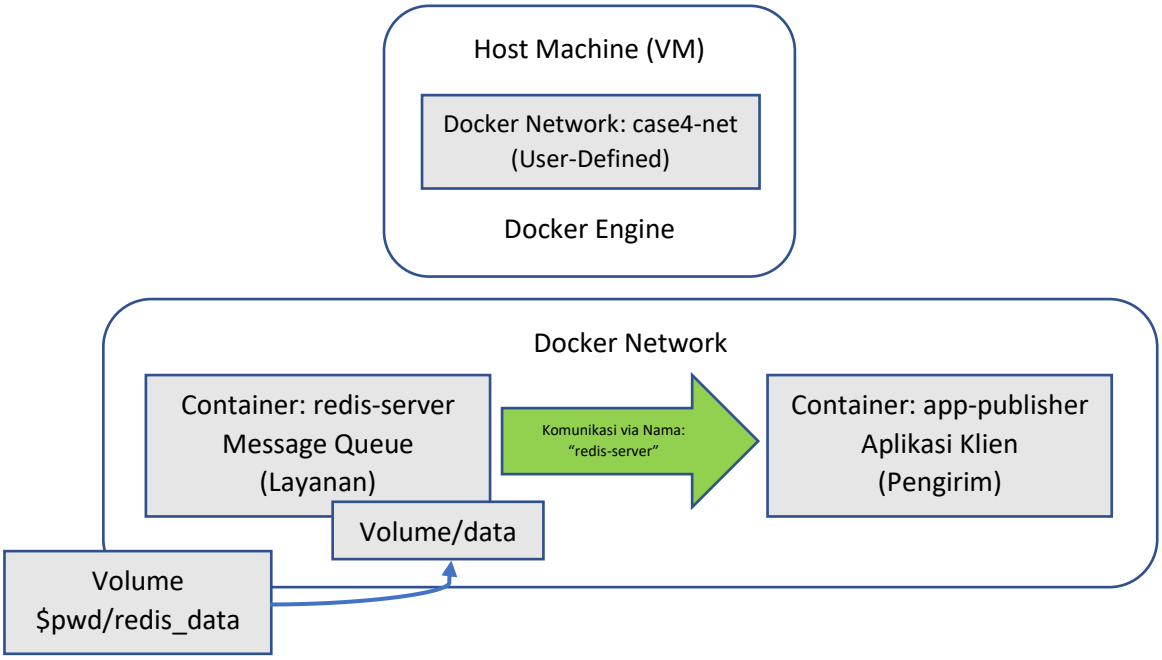


3.4 Case 4 (Aplikasi Web dan Message Queue Redis)

- Penjelasan Skenario (Kapan Ini Cocok?)
  Skenario deployment ini sangat penting dan cocok dilakukan untuk aplikasi yang mengadopsi arsitektur mikroservis atau memiliki tugas asinkron yang memakan waktu lama.
  - Pemisahan Tugas (Decoupling): Aplikasi web utama (app-worker) bertugas merespons permintaan pengguna dengan cepat (misalnya, pengguna menekan tombol "Kirim Email"). Alih-alih menunggu email terkirim, aplikasi hanya mengirim tugas tersebut ke Message Queue (Redis).

- Peningkatan Performa dan Skalabilitas: Aplikasi utama tidak terhambat oleh tugas berat. Kita bisa menskalakan container app-worker secara independen dari container redis-server sesuai kebutuhan beban.
- Ketahanan (Resilience): Jika worker pemroses tugas crash, data tugas tetap aman di queue Redis yang memiliki persistent volume, sehingga dapat diproses setelah worker pulih.

- Gambar Arsitektur



Pada gambar Arsitektur di atas memperlihatkan dua container yang berjalan di atas Docker Engine dan saling terhubung melalui jaringan internal bernama case4-net. Container pertama adalah redis-server, yang berfungsi sebagai message queue dan menyimpan data antrian secara persisten melalui volume $(pwd)/redis_data:/data. Container kedua adalah app-publisher, yaitu aplikasi klien yang mengirimkan pesan ke queue melalui perintah RPUSH. Keduanya tidak perlu membuka port ke host karena komunikasi dilakukan secara internal menggunakan hostname redis-server. Arsitektur ini menunjukkan pemisahan tugas (decoupling) antara layanan penyimpanan antrian dan aplikasi pengirim pesan, sekaligus memastikan data tetap aman melalui penggunaan volume.

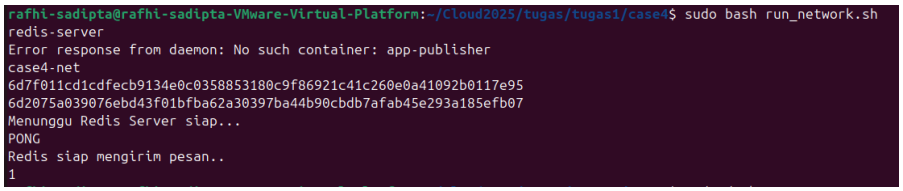
- Cuplikan Kode / Script dan Penjelasan
  - run_network.sh

```
#!/bin/sh

docker rm -f redis-server app-publisher
docker network rm case4-net

docker network create case4-net

docker run \
        -d --name redis-server \
        --network case4-net \
        -v $(pwd)/redis_data:/data \
        redis:7.0-alpine \
        redis-server --appendoly yes

docker run \
        --rm \
        --name app-publisher \
        --network case4-net \
        redis:7.0-alpine \
        redis-cli -h redis-server RPUSH task_queue "PROSES_EMAIL_USER_123"
~
~
```

Skrip ini diawali dengan perintah pembersihan (docker rm -f dan docker network rm) untuk menjamin deployment dilakukan dari kondisi bersih. Langkah vital berikutnya adalah docker network create case4-net, yang membuat jaringan internal khusus. Jaringan ini memastikan kedua container dapat saling berkomunikasi secara terisolasi tanpa perlu mengekspos port tambahan.

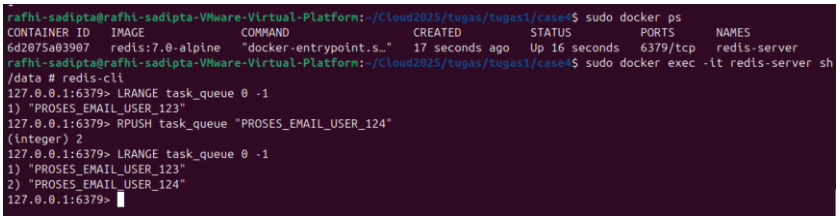
Selanjutnya, container redis-server dijalankan di latar belakang (-d) dan dihubungkan ke jaringan (--network case4-net). Bagian krusial adalah konfigurasi persistence melalui volume -v $(pwd)/redis_data:/data dan perintah redis-server --appendonly yes, yang memastikan semua data queue (pesan) disimpan secara permanen di host dan tidak hilang saat container di-restart. Terakhir, container app-publisher diluncurkan untuk simulasi Aplikasi Klien. Container ini menjalankan perintah redis-cli -h redis-server RPUSH task_queue "PROSES_EMAIL_USER_123", yang secara eksplisit mengirimkan pesan ke antrian bernama task_queue di container redis-server. Penggunaan hostname redis-server membuktikan keberhasilan komunikasi melalui User-Defined Network. Container klien ini bersifat sementara (--rm), menunjukkan pemisahan tugas di mana aplikasi hanya hidup untuk mengirim pesan dan kemudian selesai.

- Cara Menjalankan
    - Pertama kita hanya perlu menjalankan scrip run_network.sh yang sudah dibuat,



```
rafhi-sadipta@rafhi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas1/case4$ sudo bash run_network.sh
redis-server
Error response from daemon: No such container: app-publisher
case4-net
6d7f011cd1cdfecb9134e0c0358853180c9f86921c41c260e0a41092b0117e95
6d2075a039076ebd43f01bfba62a30397ba44b90cbdb7afab45e293a185efb07
Menunggu Redis Server siap...
PONG
Redis siap mengirim pesan..
1
```

    eks Menunggu Redis Server siap... diikuti oleh PONG menandakan bahwa Aplikasi Klien telah berhasil mengirim perintah ping ke Redis Server melalui jaringan case4-net, dan Server membalas dengan status siap (PONG). Ini mengonfirmasi koneksi dan inisialisasi layanan berhasil. Baris Redis siap mengirim pesan.. diikuti oleh angka 1. Angka 1 ini adalah output dari perintah RPUSH task_queue "..." yang dijalankan oleh container app-publisher. Angka tersebut menandakan bahwa container klien berhasil menambahkan satu item ke dalam queue bernama task_queue di container Redis Server.
    - Sekarang kita melakukan verifikasi apakah docker sudah berhasil jalan atau tidak alau kita coba untuk masuk ke dalam container redis-server dengan command `sudo docker exec -it redis-server sh`,



Setelah berhasil masuk ke shell container redis-server dan menjalankan redis-cli, perintah verifikasi awal LRANGE task_queue 0 -1 menghasilkan 1) "PROSES_EMAIL_USER_123". Output ini secara langsung mengonfirmasi keberhasilan pengiriman pesan pertama oleh container app-publisher melalui jaringan case4-net. Selanjutnya, untuk membuktikan fungsionalitas dinamis queue, perintah RPUSH task_queue "PROSES_EMAIL_USER_124" dijalankan, yang segera mengembalikan nilai (integer) 2, menandakan penambahan item kedua. Verifikasi akhir dengan LRANGE sekali lagi menampilkan kedua pesan tersebut secara berurutan, menegaskan bahwa Layanan Redis Message Queue berfungsi dengan baik, menerima, mengelola, dan mempertahankan data antrian (yang didukung oleh Persistent Volume) sesuai dengan tujuan arsitektur decoupling.

4. Kesimpulan

Melalui tugas ini, saya mempelajari konsep dasar penggunaan Docker melalui empat skenario berbeda. Pada Case 1, saya memahami cara menjalankan proses otomatis di dalam container untuk mengambil data secara berkala. Case 2 memperkenalkan cara menjalankan web server sederhana dengan memanfaatkan volume agar konten dapat dikelola langsung dari host. Pada Case 3, saya menjalankan dua container yang saling terhubung (MySQL dan phpMyAdmin), sehingga saya memahami penggunaan networking dan volume untuk membangun layanan yang lebih kompleks.

Case 4 menjadi penerapan tambahan yang menggabungkan konsep-konsep sebelumnya melalui implementasi message queue berbasis Redis. Dalam skenario ini, container redis-server berfungsi sebagai penyimpan antrian, sementara container app-publisher bertugas mengirimkan pesan. Hal ini menunjukkan bagaimana beberapa container dapat berkomunikasi dengan baik dalam satu jaringan yang terisolasi.

Secara keseluruhan, tugas ini membantu saya memahami dasar-dasar containerization, termasuk cara kerja container, volume, port mapping, environment variable, hingga interaksi antar-container. Pengalaman ini menjadi fondasi penting untuk mempelajari teknologi Docker yang lebih lanjut.