

**LAPORAN KOMPUTASI AWAN**  
**TUGAS 1**



**DEPARTEMEN INFORMATIKA**

**Muhammad Wafi Zaki Hanif**  
**5025221039**

## 1. Pendahuluan

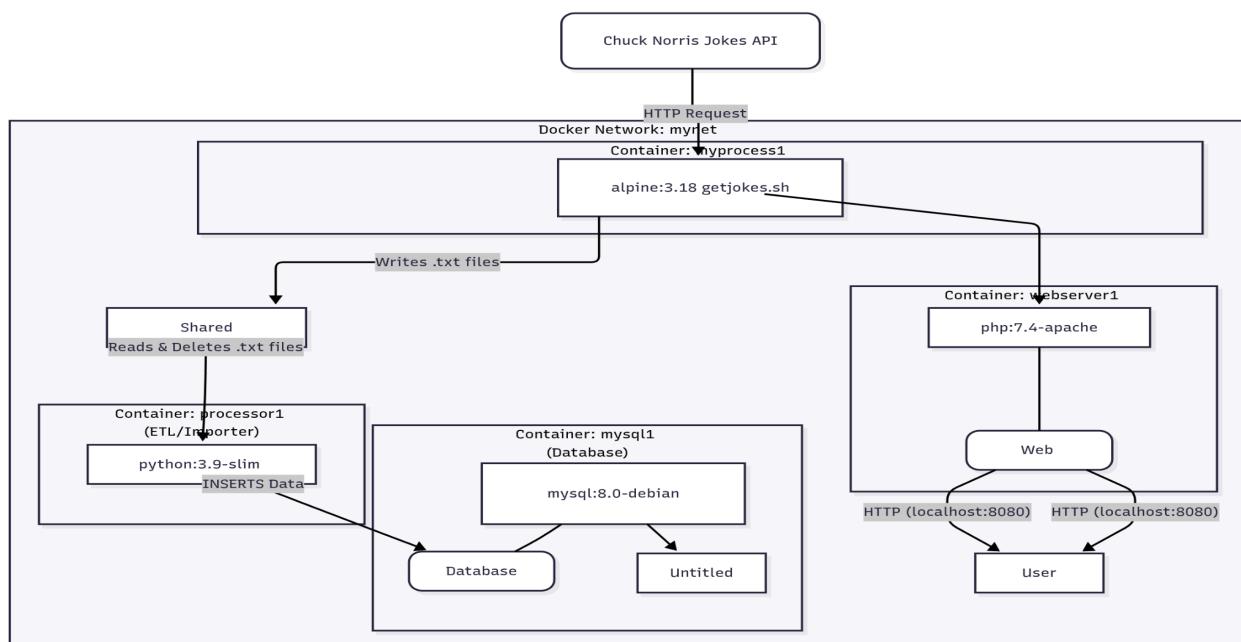
Proyek ini bertujuan untuk mendemonstrasikan konsep arsitektur Microservices dan data pipeline otomatis menggunakan teknologi *container* Docker. Skenario yang diangkat adalah **Automated Content Aggregator**, sebuah sistem yang secara mandiri mengambil konten dari *API External*, menyimpannya dalam *Database*, dan menyajikannya melalui antarmuka web.

Sistem ini digunakan dalam skenario di mana sebuah entitas membutuhkan suplai data/konten secara periodik, tetapi tidak ingin terjadi *single point of failure*. Dengan memisahkan tugas menjadi layanan-layanan kecil (Microservices), proses pengambilan data, pengolahan, dan penyajian dapat dilakukan secara independen. Misalnya, jika API eksternal gagal diakses, data yang sudah tersimpan di database masih bisa ditampilkan oleh webserver.

## 2. Arsitektur dan Data Pipeline

Arsitektur proyek ini dibagi menjadi empat layanan utama yang berjalan secara independen di dalam satu jaringan (mynet). Data mengalir secara sekuensial dari kiri ke kanan, membentuk sebuah pipeline.

### Diagram Arsitektur



### Penjelasan Arsitektur

Arsitektur ini terdiri dari empat lapisan utama. Lapisan **Collector** bertanggung jawab untuk mendapatkan data dari sumber eksternal (API Jokes Chuck Norris) dan menuliskannya sebagai file teks mentah ke volume bersama (files/). Lapisan **Processor** (Python) kemudian bertindak sebagai mekanisme Extract, Transform, dan Load (ETL), membaca file teks tersebut, memastikan data unik, dan memindahkannya ke Lapisan **Database** yang menggunakan MySQL (mysql1). Terakhir, lapisan **Frontend** (PHP Webserver) bertugas untuk menampilkan data yang sudah terstruktur di database ke pengguna akhir melalui *web dashboard* di **port 8080**.

### 3. Pembahasan Tugas

#### a. Pembahasan Case 1

Pada case ini dilakukan pemanggilan API Chuck Norris Joke secara berulang. Kemudian hasil pemanggilan akan disimpan pada directory files/. Hal ini dapat terjadi dengan membuat *script* yang memanfaatkan perulangan *while*. *Script* dapat dilihat pada gambar berikut.

```
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 3.1: Script Pemanggilan API

Selanjutnya, *script* dijalankan dalam suatu docker container. Dengan parameter “-dit” memastikan kontainer berjalan dalam mode *Detached* (di latar belakang), mode *Interactive*, dan mengaktifkan TTY untuk menjaga proses tetap hidup. Kemudian *container* menggunakan image Alpine Linux yang sangat ringan, meminimalkan kebutuhan sumber daya. Kemudian dilakukan *Volume Mapping* pada kontainer yaitu folder files di host dipetakan ke /data di container, menjadikannya Shared Volume tempat data mentah disimpan. Folder script dipetakan ke /script agar Docker dapat mengeksekusi getjokes.sh.

```
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 3.2: Pengaturan Container Docker

Pada Case 1 ini, untuk menjalankan cukup dengan menjalankan perintah “./run\_process.sh” untuk mengeksekusi pemanggilan API Chuck Norris Joke. Berikut tampilan hasil setelah dijalankan.

```
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case1$ sudo ./run_process.sh
Error response from daemon: No such container: myprocess1
12eee0d8c13d6198aa1ba8e0277a9da677e9d39293660d6104042d820dd41032
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case1$ sudo docker logs myprocess1
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/community/x86_64/APKINDEX.tar.gz
v3.18.12-140-g3d8a58a5f14 [https://dl-cdn.alpinelinux.org/alpine/v3.18/main]
v3.18.12-132-ge2ea6f61358 [https://dl-cdn.alpinelinux.org/alpine/v3.18/community]
OK: 20070 distinct packages available
(1/10) Installing ca-certificates (20241121-r1)
(2/10) Installing brotli-libs (1.0.9-r14)
(3/10) Installing libunistring (1.1-r1)
(4/10) Installing libidn2 (2.3.4-r1)
(5/10) Installing nghttp2-libs (1.57.0-r0)
(6/10) Installing libpsl (0.21.5-r0)
(7/10) Installing libcurl (8.12.1-r0)
(8/10) Installing curl (8.12.1-r0)
(9/10) Installing oniguruma (6.9.8-r1)
(10/10) Installing jq (1.6-r4)
Executing busybox-1.36.1-r7.trigger
Executing ca-certificates-20241121-r1.trigger
OK: 13 MiB in 25 packages
will run every 8 seconds
processing at 2025-11-25 11:13:27
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case1$ ls files/
output_2025-11-25_11:13:27.txt output_2025-11-13:36.txt
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case1$ cat files/output_2025-11-25_11:13:27.txt
output_2025-11-25_11:13:27.txt output_2025-11-25_11:13:36.txt output_2025-11-25_11:13:45.txt
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case1$ cat files/output_2025-11-25_11:13:27.txt
output_2025-11-25_11:13:27.txt output_2025-11-25_11:13:36.txt output_2025-11-25_11:13:45.txt output_2025-11-25_11:13:53.txt
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case1$ cat files/output_2025-11-25_11:13:27.txt
output_2025-11-25_11:13:27.txt output_2025-11-25_11:13:36.txt output_2025-11-25_11:13:45.txt output_2025-11-25_11:13:53.txt
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case1$ cat files/output_2025-11-25_11:13:27.txt
output_2025-11-25_11:13:27.txt output_2025-11-25_11:13:36.txt output_2025-11-25_11:13:45.txt output_2025-11-25_11:13:53.txt
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case1$ cat files/output_2025-11-25_11:13:45.txt
In soviet Russia Ian's (Smosh) pizza eats Ian and Anthony. SMOSHTIME WITH LUNCH Chuck Norris"
```

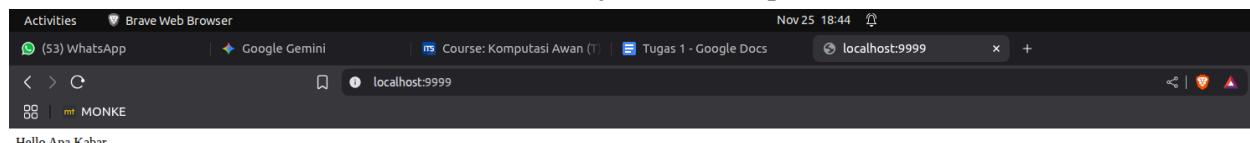
Gambar 3.3: Menjalankan Script Case 1

### b. Pembahasan Case 2

Pada Case 2 ini, diberikan sebuah script untuk menjalankan container webserver. Kontainer ini dirancang sebagai file server statis sederhana menggunakan Python. Kontainer ini hanya menjalankan file apapun yang ada di folder mount (index.html yang ada di files/). Dengan menjalankan perintah “./run\_simple\_web.sh”, kontainer akan berjalan dan akan menghasilkan tampilan website. Berikut gambar untuk menjalankan serta tampilan websitenya.

```
Run 'docker run --help' for more information
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case2$ sudo ./run_simple_web.sh
0be8a05e9250feeb944ff498b685074b3883a6bd5425c431c58c8b3c21c8a485
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case2$
```

Gambar 3.4: Menjalankan Script Case 2



Gambar 3.5: Tampilan Website

### c. Pembahasan Case 3

Pada Case 3 ini, diberikan 2 buah *script* untuk menjalankan Database MySQL serta Dashboard phpMyAdmin.

```
dbdata run_myadmin.sh run_mysql.sh
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/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 3.6: Script Container phpMyAdmin**

Kontainer phpmyadmin1 adalah alat bantu berbasis web (GUI) yang digunakan untuk mengelola dan memonitor database MySQL. Dalam konteks arsitektur Microservices, alat ini berfungsi sebagai lapisan debugging dan verifikasi. Konfigurasi skrip yang pertama dilakukan adalah *Port Mapping* yang dipublikasikan ke port 10000. Kemudian konfigurasi koneksi dihubungkan dengan kontainer mysql1 agar phpMyAdmin mengetahui harus terhubung kemana dalam jaringan Docker. Terakhir, untuk memberi tahu Docker *hostname resolution* digunakan “*–link mysql1*” agar phpmyadmin dan mysql1 terhubung.

```
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/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
```

**Gambar 3.7: Script Container MySQL**

Kontainer mysql1 adalah layanan penyimpanan data relasional. Konfigurasi “*-v \$(pwd)/dbdata:/var/lib/mysql*”: Ini adalah konfigurasi Persistensi Data. Folder /var/lib/mysql di dalam kontainer adalah lokasi penyimpanan data MySQL. Dengan memetakan folder ini ke dbdata/ di host, data database akan tetap tersimpan meskipun kontainer mysql1 dimatikan atau dihapus. Kemudian “*-e MYSQL\_DATABASE=mydb*”: Mendefinisikan database awal bernama mydb yang akan dibuat secara otomatis saat kontainer berjalan. Kemudian “*-e MYSQL\_ROOT\_PASSWORD=mydb6789tyui*”: Menetapkan password. Terakhir, digunakan image “mysql:8.0-debian:” yaitu image resmi MySQL versi 8.0.

#### d. Pembahasan Case 4

##### 1. Database Service (mysql1)

Kontainer ini bertanggung jawab untuk menyimpan data jokes. Konfigurasi menggunakan *volume mapping* untuk menyimpan data persisten di folder dbdata/ host.

```
2025-11-25 09:13:00+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.44-1debian12 started.
2025-11-25 09:13:00+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2025-11-25 09:13:00+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.44-1debian12 started.
2025-11-25T09:13:01.013707Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.44) starting as process 1
2025-11-25T09:13:01.022302Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2025-11-25T09:13:01.240121Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2025-11-25T09:13:01.454845Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
2025-11-25T09:13:01.454881Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
2025-11-25T09:13:01.460003Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.
2025-11-25T09:13:01.478607Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
2025-11-25T09:13:01.478646Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.44' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.
```

**Gambar 4.1:** Terminal saat MySQL pertama kali dijalankan

Pada saat pertama kali dijalankan, mysql1 secara otomatis mengeksekusi file init.sql melalui *entrypoint* Docker. Ini memastikan tabel jokes terbentuk di database mydb sebelum aplikasi lain mencobanya. Langkah ini sangat penting untuk menjamin konsistensi skema data.

##### 2. Collector Service (myprocess1)

Layanan ini adalah bot pencari data yang berjalan di container Alpine. Tugasnya adalah mengambil data dari API eksternal secara periodik.

```
index.php
wafizakih@wafizakih-swift:~/KULIAH/Semester_7/cloud2023/containers/docker/case4/files$ ls
index.php  output_2025-11-25_09:53:09.txt
```

**Gambar 4.2:** Isi folder files setelah myprocess1 berjalan

Seperti yang terlihat pada tangkapan layar, kontainer ini menjalankan skrip getjokes.sh yang mengambil jokes dan menyimpannya sebagai file teks (output\_\*.txt) ke folder bersama (files/). Dengan

variabel lingkungan `DELAY=8`, proses ini berulang setiap 8 detik, menyediakan aliran data mentah yang konstan untuk diproses oleh layanan berikutnya.

### 3. Processor Service (processor1)

Layanan ini adalah inti dari ETL pipeline. Berjalan di environment Python, tugas utamanya adalah membersihkan data dari folder files/ dan memindahkannya ke database.

```
Collecting mysql-connector-python
  Downloading mysql_connector_python-9.4.0-cp39-cp39-manylinux_2_28_x86_64.whl (33.9 MB) ━━━━━━━━━━━━━━━━ 33.9/33.9 MB 225.3 kB/s eta 0:00:00
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-9.4.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv

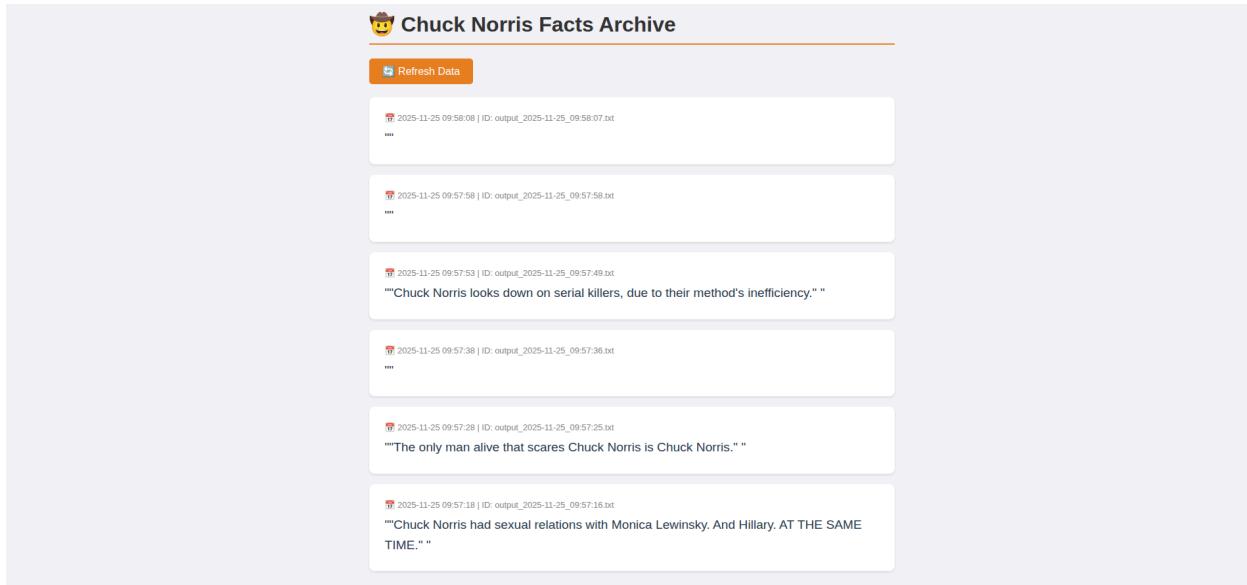
[notice] A new release of pip is available: 23.0.1 -> 25.3
[notice] To update, run: pip install --upgrade pip
Processor Service Berjalan...
Sukses import: output_2025-11-25_09:14:50.txt
Sukses import: output_2025-11-25_09:14:39.txt
Sukses import: output_2025-11-25_09:14:29.txt
Sukses import: output_2025-11-25_09:14:10.txt
Sukses import: output_2025-11-25_09:15:01.txt
Sukses import: output_2025-11-25_09:13:50.txt
```

Gambar 4.3: Log Terminal processor1 saat memproses data

Log yang ditampilkan di atas menunjukkan bahwa skrip `importer.py` telah berhasil terhubung ke MySQL melalui jaringan mynet. Setiap kali ia mendeteksi file .txt baru dari `collector`, ia akan mengeksekusi perintah SQL `INSERT`, melakukan `db.commit()`, dan kemudian **menghapus file sumber** (`os.remove(filepath)`). Langkah penghapusan ini krusial untuk mencegah duplikasi data dan menjaga kebersihan *volume* bersama.

### 4. Frontend Service (webserver1)

Layanan ini berfungsi sebagai antarmuka utama pengguna (dashboard). Berjalan menggunakan image `php:7.4-apache`, ia mampu memproses logika *server-side* untuk berinteraksi dengan database.



**Gambar 4.4: Tampilan Dashboard Aplikasi Web**

Dashboard yang diakses melalui `http://localhost:8080` tidak lagi menampilkan konten statis. Ia menjalankan kode PHP yang secara langsung terkoneksi ke MySQL dan menampilkan 10 jokes terbaru. Ini menunjukkan keberhasilan komunikasi *service-to-service* dari frontend ke backend database.

#### 4. Penutup

Proyek ini berhasil mengimplementasikan *data pipeline* otomatis menggunakan arsitektur Microservices berbasis Docker. Keberhasilan koneksi antar-kontainer (terutama antara PHP Webserver dan MySQL) melalui Docker Network (mynet) dan otomatisasi pemrosesan data (ETL) menggunakan kontainer Python membuktikan pemahaman yang solid terhadap konsep **Orchestration** dan **Service Discovery** dalam konteks *Cloud Computing*.