

Tugas 2 Cloud Computing

Rafhi Sadipta – 5025221014

Link Repository: <https://github.com/RafhiSadipta/cloud-computing-2025/tree/main/tugas2>

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

Teknologi containerization menggunakan Docker telah menjadi pilar utama dalam pengembangan dan deployment aplikasi modern. Docker menawarkan solusi untuk mengemas aplikasi dan dependensi secara terisolasi, menjamin portabilitas dan konsistensi di berbagai lingkungan. Tugas ini berfokus pada penggunaan Docker Compose, sebuah alat yang krusial untuk mengelola aplikasi yang terdiri dari banyak container (aplikasi multi-container). Dengan mengeksplorasi repository yang diberikan, mahasiswa akan secara praktis menguasai konfigurasi dan orkestrasi layanan terdistribusi, sebuah kemampuan dasar yang wajib dikuasai dalam arsitektur berbasis cloud.

1.2 Tujuan

Tugas ini bertujuan untuk:

1. Mempelajari konsep dasar Docker Compose dan penggunaannya untuk mengelola aplikasi multi-container.
2. Menganalisis, mengimplementasikan, dan mendokumentasikan keempat case yang terdapat pada repository.
3. Mengembangkan dan mendokumentasikan satu skenario kasus tambahan secara mandiri, lengkap dengan desain arsitektur dan script pendukung.
4. Mendokumentasikan hasil percobaan dan analisis secara sistematis.

2. Tugas Yang Diberikan

Tugas ini berfokus pada penerapan konsep containerization menggunakan Docker Compose melalui eksplorasi repository yang tersedia pada <https://github.com/rm77/cloud2023/tree/master/containers/compose/compose>.

Tugas utama terbagi menjadi dua bagian. Pertama, implementasi dan dokumentasi terhadap empat kasus (case 1 hingga case 4) yang telah tersedia dalam repository. Setiap case harus dijalankan, didokumentasikan langkah-langkahnya, dan dilengkapi dengan screenshot serta penjelasannya. Kedua, pengembangan mandiri berupa pembuatan satu skenario tambahan yang dinamakan Case 5. Pengembangan ini harus mencakup perancangan gambar arsitektur baru, penulisan script pendukung, penjelasan relevansi skenario tersebut, dan dokumentasi hasilnya. Semua hasil implementasi, analisis, dan spesifikasi teknis (termasuk arsitektur dan spesifikasi komputer) harus dikumpulkan dalam satu laporan akhir berbentuk PDF dengan batasan maksimal 10 halaman.

3. Implementasi dan Penjelasan

3.1 Case 1

- Tujuan Case 1

Menerapkan konsep dasar Docker Compose untuk menjalankan sebuah web server statis tunggal. Tujuan utama adalah memastikan container Nginx dapat berjalan, membaca file konfigurasi kustom, dan menyajikan konten HTML dari direktori lokal. Menggunakan image Nginx versi 1.15.12-alpine sebagai web server berkinerja tinggi, dan Docker Compose untuk orkestrasi.

- Cuplikan Kode / Script dan Penjelasan

- nginx.conf

```
server {
    listen 80;
    listen [::]:80;

    server_name _;

    index index.html index.htm;

    root /var/www/html;

    location = /favicon.ico {
        log_not_found off; access_log off;
    }
    location = /robots.txt {
        log_not_found off; access_log off; allow all;
    }
    location ~* \.(css|gif|ico|jpeg|jpg|js|png)$ {
        expires max;
        log_not_found off;
    }
}
```

Gambar ini menunjukkan konfigurasi yang membuat server listen pada port 80 di dalam container, menetapkan direktori root pada /var/www/html sesuai dengan volume yang dipetakan dari lokal, serta menambahkan pengaturan caching menggunakan expires max dan menonaktifkan logging untuk file statis seperti CSS dan gambar guna meningkatkan efisiensi.

- docker-compose.yml

```
version: '3'

services:
  webserver:
    image: nginx:1.15.12-alpine
    restart: unless-stopped
    ports:
      - "9999:80"
    volumes:
      - ./html:/var/www/html
      - ./nginx-conf:/etc/nginx/conf.d
    networks:
      - app-network
  networks:
    app-network:
      driver: bridge
~
```

Gambar diatas menunjukkan bahwa layanan webserver ini menggunakan image Nginx versi alpine yang ringan dengan konfigurasi restart: unless-stopped agar container otomatis berjalan kembali kecuali dihentikan manual. Port 80 di dalam container yang menjalankan Nginx dipetakan ke port 9999 pada host sehingga dapat diakses melalui http://localhost:9999. Dua volume digunakan: ./html:/var/www/html untuk memuat konten web statis dari folder lokal, dan ./nginx-conf:/etc/nginx/conf.d untuk menerapkan konfigurasi kustom Nginx. Selain itu, layanan ini berjalan pada network bernama app-network dengan driver bridge untuk memberikan isolasi antar layanan.

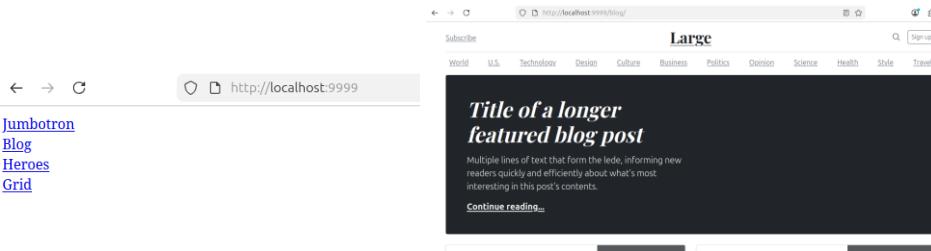
- Cara Menjalankan

- Jalankan perintah `docker-compose up -d` untuk membuat dan menjalankan container. Lalu coba verifikasi status container dengan menjalankan `docker-compose ps`.

```
raffi-sadipta@raffi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas2/case1$ docker-compose up -d
Pulling webserver (nginx:1.15.12-alpine)...
1.15.12-alpine: Pulling from library/nginx
e7c96db7181b: Pull complete
264026bbe255: Pull complete
a71634c55d29: Pull complete
5595887beb81: Pull complete
Digest: sha256:57a226fb6ab6823027c0704a9346a890ff80cacde06bc19bbc234c8720673555
Status: Downloaded newer image for nginx:1.15.12-alpine
Creating case1_webserver_1 ... done
raffi-sadipta@raffi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas2/case1$ docker-compose ps
     Name          Command   State    Ports
-----
```

Name	Command	State	Ports
case1_webserver_1	nginx -g daemon off;	Up	0.0.0.0:9999->80/tcp, :::9999->80/tcp

- Akses layanan melalui browser pada alamat `http://localhost:9999`



Gambar ini menunjukkan keberhasilan deployment Nginx melalui port 9999 di host. Tampilan `localhost:9999` memperlihatkan empat tautan sesuai dengan index.html. Hal ini memverifikasi Volume Mapping berhasil dilakukan. Ketika salah satu tautan diklik (contoh: Blog), user diarahkan ke path baru, menyajikan konten index.html dari folder terkait (misalnya, /blog/), membuktikan fungsionalitas hosting web statis berjalan sukses.

3.2 Case 2

- Tujuan Case 2

Tujuan Case 2 ini adalah mengatur web server Nginx agar mendukung koneksi aman melalui protokol HTTPS (SSL/TLS) dengan membuka dua port sekaligus—port 80 untuk HTTP dan port 443 untuk HTTPS—serta memasang sertifikat SSL/TLS self-signed. Selain itu, konfigurasi ini menerapkan redirect otomatis dari HTTP ke HTTPS untuk memastikan seluruh akses selalu terenkripsi, dengan memanfaatkan Nginx, Docker Compose, dan sertifikat SSL/TLS.

- Cuplikan Kode / Script dan Penjelasan

- docker-compose.yml

```
version: '3'
services:
  webserver:
    image: nginx:1.15.12-alpine
    restart: unless-stopped
    privileged: true
    ports:
      - "443:443"
      - "80:80"
    volumes:
      - ./certs:/certs
      - ./html:/var/www/html
      - ./nginx-conf/nginx.secure.conf:/etc/nginx/conf.d/nginx.conf
    networks:
      - app-network
networks:
  app-network:
    driver: bridge
```

Port 80 dan 443 dipetakan langsung ke host untuk mengaktifkan HTTP dan HTTPS. Volume `./certs:/certs` digunakan agar Nginx dapat membaca sertifikat `MyCert.crt` dan kunci privat `MyPrivate.key` dari folder lokal. Konfigurasi default Nginx kemudian diganti dengan file kustom `nginx.secure.conf` yang berisi pengaturan SSL.

- nginx.secure.conf

```
1 server {
2   listen 80 default_server;
3   server_name .;
4   return 301 https://$host$request_uri;
5 }
6
7
8
9 server {
10   listen 443 ssl;
11
12   server_name .;
13   ssl_certificate      /certs/MyCert.crt;
14   ssl_certificate_key  /certs/MyPrivate.key;
15   ssl_protocols TLSv1.2;
16   ssl_ciphers  'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384';
17   ssl_prefer_server_ciphers on;
18   ssl_session_cache shared:SSL:10m;
19
20   client_max_body_size      0;
21   chunked_transfer_encoding on;
22
23   index index.html index.htm;
24
25   root /var/www/html;
26
27   location = /favicon.ico {
28     log_not_found off; access_log off;
29   }
30   location = /root.txt {
31     log_not_found off; access_log off; allow all;
32   }
33   location ~ \.(css|js|gif|jpg|png|js|json)$ {
34     expires max;
35     log_not_found off;
36   }
37 }
```

Blok server pada port 80 digunakan untuk redirect 301 ke HTTPS agar semua akses otomatis dialihkan ke koneksi aman. Blok server pada port 443 mengaktifkan SSL, dengan `ssl_certificate` dan `ssl_certificate_key` menunjuk ke file sertifikat dan kunci privat di `./certs` yang telah dipetakan melalui volume.

- Cara Menjalankan

- Jalankan `docker-compose up -d`, lalu bisa sekalian kita cek juga apakah sudah berhasil dijalankan dengan `docker ps`.

```
rafhil-sadipta@rafhil-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas2/case2$ docker-compose up -d
Creating network "case2_app-network" with driver "bridge"
Creating case2_webserver_1 ... done
rafhil-sadipta@rafhil-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas2/case2$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
77a7d2c49129      nginx:1.15.12-alpine   "nginx -g 'daemon off;'"  3 minutes ago     Up 3 minutes      0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp   case2_webserver_1
```

- Lalu kita coba untuk membuka `http://localhost:443`.



Tampilan browser menunjukkan error `400 Bad Request: The plain HTTP request was sent to HTTPS port` saat mengakses `http://localhost:443`. Error ini membuktikan keberhasilan aktivasi SSL/TLS pada port 443 yang menolak koneksi tidak terenkripsi. Atau kita bisa mencoba `curl -v --insecure <https://localhost>` di terminal untuk melihat apa cert berhasil terpasang atau tidak.

3.3 Case 3

- Tujuan Case 3

Tujuan Case ini adalah menerapkan arsitektur multi-container yang terdistribusi untuk menjalankan WordPress secara dinamis, melibatkan empat layanan utama: db sebagai database MySQL, wordpress sebagai container PHP-FPM yang menjalankan inti WordPress, phpmyadmin sebagai antarmuka pengelolaan database, dan webserver (Nginx) sebagai reverse proxy sekaligus server HTTPS yang meneruskan permintaan PHP ke container wordpress. Teknologi yang digunakan mencakup MySQL 8.0, WordPress 5.1.1-fpm-alpine, phpMyAdmin, dan Nginx 1.15.12-alpine.

- Cuplikan Kode / Script dan Penjelasan

- docker-compose.yml

```
version: '3'
services:
  db:
    image: mysql:8.0
    restart: unless-stopped
    env_file: .env
    environment:
      - MYSQL_DATABASE=$WORDPRESS_DB
      - MYSQL_PASSWORD=$MYSQL_ROOT_PASSWORD
      - MYSQL_ROOT_PASSWORD=$MYSQL_ROOT_PASSWORD
      - MYSQL_ROOT_HOST=%
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - ./dbdata:/var/lib/mysql
    networks:
      app-network
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
      - "$PHPMYADMIN_PORT:80"
    links:
      - db
    env_file: .env
    environment:
      - PMA_HOST=db
      - MYSQL_ROOT_PASSWORD=$MYSQL_ROOT_PASSWORD
    networks:
      app-network
  wordpress:
    depends_on:
      - db
```

Script ini memastikan semua layanan terhubung melalui satu network bernama app-network, sehingga tiap service dapat saling mengakses menggunakan nama servicenya sebagai hostname, misalnya phpMyAdmin dan WordPress menggunakan `PMA_HOST=db` dan `WORDPRESS_DB_HOST=db`. Dependensi antar layanan diatur dengan `depends_on` agar container berjalan dalam urutan yang benar. Data juga dibuat persisten melalui volume seperti `./dbdata` untuk MySQL dan `./wp_vol` untuk file WordPress, sehingga data tetap aman meskipun container dihentikan.

- nginx.conf

```

1    server {
2        listen 80 default_server;
3        server_name _;
4        return 301 https://$host$request_uri;
5    }
6
7
8
9    server {
10       listen 443 ssl;
11
12       server_name _;
13       ssl_certificate      /certs/myCert.crt;
14       ssl_certificate_key  /certs/myPrivate.key;
15       ssl_protocols TLSv1.2;
16       ssl_ciphers ECDHE-RSA-CHACHA256-ECDSA-CHACHA256-EDH;
17       ssl_prefer_server_ciphers on;
18       ssl_session_cache shared:SSL:10m;
19
20       client_max_body_size 0;
21       chunked_transfer_encoding on;
22
23       index index.php index.html index.htm;
24
25       root /var/www/html;
26
27       location / {
28           try_files $uri $uri/ /index.php$is_args$args;
29       }
30
31       location ~ \.php$ {
32           try_files $uri =404;
33           fastcgi_split_path_info ^(.+\.php)(/.+)$;
34           fastcgi_pass wordpress:9000;
35       }
36   }
37
38   fastcgi_index index.php;
39
40   include fastcgi_params;
41   fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
42   fastcgi_param PATH_INFO $fastcgi_path_info;
43
44   location ~ /\.ht {
45       deny all;
46   }
47
48   location = /favicon.ico {
49       log_not_found off; access_log off;
50   }
51   location = /robots.txt {
52       log_not_found off; access_log off; allow all;
53   }
54   location ~* \.(css|gif|ico|jpeg|jpg|js|png)$ {
55       expires max;
56       log_not_found off;
57   }
58 }

```

Nginx menangani file PHP melalui blok `location ~ \.php\$`, di mana ia tidak memproses PHP sendiri tetapi bertindak sebagai reverse proxy dengan meneruskan permintaan ke `fastcgi_pass wordpress:9000`. Di sini, Nginx menggunakan nama service wordpress sebagai hostname dan port 9000 sebagai port standar PHP-FPM, sehingga setiap permintaan dinamis diproses langsung oleh container PHP-FPM tersebut.

• Cara Menjalankan

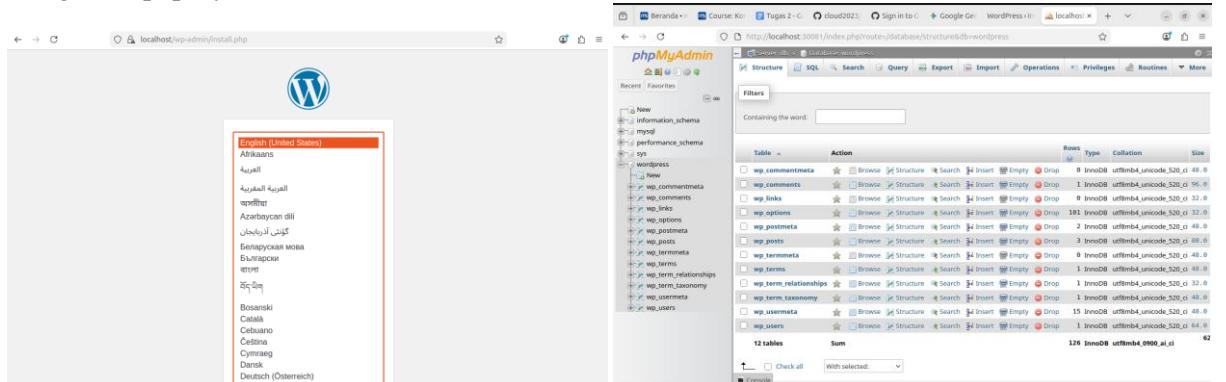
- Jalankan `docker-compose up -d` dan `docker ps` jika ingin mengecek kembali.

```

Status: Downloaded newer image for wordpress:5.1.1-fpm-alpine
Creating case3_db_1 ... done
Creating case3_phpmyadmin_1 ... done
Creating case3_wordpress_1 ... done
Creating case3_webserver_1 ... done
rafhi-sadipta@rafhi-sadipta-Virtual-Platform:~/Cloud2025/tugas/tugas2/case3$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
8a3f3f8f7bcd        nginx:1.15.12-alpine   "nginx -g 'daemon of..."  31 seconds ago   Up 31 seconds   0.0.0.0:80->80/tcp
r_1:[::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp   case3_webserver_1
596a296cf470        wordpress:5.1.1-fpm-alpine   "docker-entrypoint.s..."  32 seconds ago   Up 32 seconds   9000/tcp
                                         case3_wordpress_1
fe3873b8998e        phpmyadmin:1.0.0       "/docker-entrypoint..."  32 seconds ago   Up 32 seconds   0.0.0.0:30081->80/
tcp, [::]:30081->80/tcp
afec5056fedf        mysql:8.0            "docker-entrypoint.s..."  36 seconds ago   Up 34 seconds   3306/tcp, 3306/tdc
p

```

- Lalu kita coba buka `http://localhost` untuk mengakses WordPress dan `http://localhost:30081` untuk mengakses phpMyAdmin.



Implementasi Case 3 berhasil mengorkestrasikan empat container (MySQL, WordPress PHP-FPM, phpMyAdmin, dan Nginx) dalam satu network. Output terminal mengonfirmasi keempat layanan Up dan berjalan. Akses ke `https://localhost` menampilkan halaman setup WordPress, yang membuktikan Nginx berhasil menjadi reverse proxy dan container WordPress berhasil terhubung ke database `db`. Selain itu, layanan phpMyAdmin yang diakses melalui port host 30081 (sesuai dengan variabel \$PHPMYADMIN_PORT) juga berfungsi dan berhasil menampilkan halaman login. Hal ini memverifikasi bahwa semua komponen multi-container, termasuk tool administrasi database, telah terintegrasi dan berfungsi dengan sukses.

3.4 Case 4

- Tujuan Case 4

Tujuan Case 4 adalah mendemonstrasikan orkestrasi aplikasi multi-container di mana salah satu layanannya dibangun secara kustom melalui Dockerfile lokal, bukan dari image Docker Hub. Empat layanan digunakan: server Apache/PHP hasil custom build, MySQL sebagai database, phpMyAdmin sebagai alat administrasi, serta container utilitas berbasis Alpine. Teknologi yang terlibat meliputi MySQL 5.7, Apache/PHP 7 (custom build), phpMyAdmin, dan Alpine.

- Cuplikan Kode / Script dan Penjelasan

- docker-compose.yml

```

1  version: '3'
2
3  > Run All Services
4  > services:
5    > mysql1:
6      > image: mysql:5.7-debian
7      > restart: unless-stopped
8      > environment:
9        - MYSQL_DATABASE=mydb
10       - MYSQL_PASSWORD=mydb6789tyui
11       - MYSQL_ROOT_PASSWORD=mydb6789tyui
12       - MYSQL_ROOT_HOST=%
13       > volumes:
14         - ./dbdata:/var/lib/mysql
15         - ./dbscripts:/dbscripts
16       > networks:
17         example1-network
18   > app:
19     > restart: unless-stopped
20     > build:
21       > context: ./platform
22       > dockerfile: Dockerfile
23     > ports:
24       - "34001:80"
25     > environment:
26       - MODE=development
27       - MYSQL_DATABASE=mydb
28       - MYSQL_PASSWORD=mydb6789tyui
29       - MYSQL_ROOT_PASSWORD=mydb6789tyui
30       - MYSQL_ROOT_HOST=%
31     > volumes:
32       - ./src:/var/www/localhost/htdocs
33       - ./platform/httpd.conf:/etc/apache2/httpd.conf
34
35       > extra_hosts:
36         - "test:10.199.2.29"
37       > networks:
38         - example1-network
39   > Run Service
40   > alpine:
41     > image: alpine:3.18
42     > command:
43       - /bin/sh
44       - -c
45       - "while true;do sleep 10000;done"
46   > networks:
47     - example1-network
48   > phpmyadmin:
49     > image: phpmyadmin:5.2.1-apache
50     > ports:
51       - "10000:80"
52     > links:
53       - mysql1
54     > environment:
55       - PMA_HOST=mysql1
56       - MYSQL_ROOT_PASSWORD=mydb6789tyui
57   > networks:
58     - example1-network
59
60
61   > networks:
62     - example1-network
63     > driver: bridge

```

Layanan app dibangun menggunakan konfigurasi build yang mengambil Dockerfile dari folder `./platform`, bukan dari image Docker Hub. Aplikasi kustom tersebut dapat diakses melalui port 34001, sementara phpMyAdmin tersedia di port 10000. Sebuah container alpine disertakan sebagai utility container untuk debugging atau menjalankan perintah one-off dalam network yang sama. Selain itu, fitur extra_hosts digunakan untuk menambahkan entry statis ke `/etc/hosts` di dalam container app, sehingga nama host kustom seperti test dapat dikenali.

- Dockerfile

```

FROM alpine:3.9

RUN apk update && apk add --no-cache apache2 curl supervisor socat php7-intl php7-openssl php7-pecl-redis php7-sqlite3 p
hp7-pdo_mysql php7-mbstring apache2-ssl php7-apache2 php7-intl php7-openssl php7-sqlite3 php7-pdo_mysql php7-mbstring ap
ache2-ssl php7-apache2 apache2 apache2-ctl php7-pecl-ssh2 php7-bcmath php7-curl php7-json php7-pecl-couchbase php7-zip p
hp7-mysqli mysql-client php7-pecl-mcrypt php7-pecl-redis supervisor

VOLUME ["/var/www/localhost/htdocs"]

RUN apk add composer php7-gmp php7-sodium php7-xml supervisor

RUN rm -rf /tmp/* /var/cache/apk/*
ADD start.sh /tmp
ADD supervisord.conf /tmp/supervisord.conf
EXPOSE 80

ENV SESSIONAGE 60
ENV REAUTHENTICATION 60
ENV IPEXPIRE 120

#ENTRYPOINT [ "apachectl","-D",""FOREGROUND"]
ENTRYPOINT [ "sh","-C","/tmp/start.sh"]

```

Dockerfile ini membangun image berbasis Alpine 3.9 dengan instalasi Apache 2, PHP 7 beserta berbagai extension penting seperti pdo_mysql, curl, dan json, serta MySQL client dan supervisor. Proses startup tidak menggunakan `apachectl` secara langsung, melainkan dijalankan melalui script `start.sh` kustom yang berfungsi sebagai entrypoint untuk menginisialisasi Apache dan supervisor.

- Cara Menjalankan

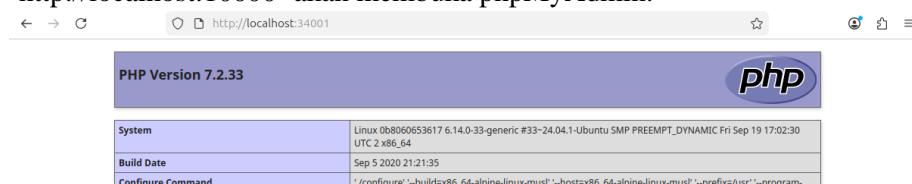
- Jalankan `docker-compose up -d --build` dan `docker ps` jika ingin mengecek kembali.

```

Successfully built 5922b7bc753
Successfully tagged case4_app:latest
Creating case4_mysql1_1 ... done
Creating case4_alpine1_1 ... done
Creating case4_app1_1 ... done
Creating case4_phpmyadmin1_1 ... done
rafhi-sadipta@rafhi-sadipta-VMware-Virtual-Platform:~/Cloud2025/tugas/tugas2/cases$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
a04b108b33a9 phpmyadmin:5.2.1-apache "/docker-entrypoint..." 5 seconds ago Up 4 seconds 0.0.0.0:10000->80/tcp,
[::]:10000->80/tcp case4_phpmyadmin1_1
a0261b20d96d alpine:3.18 "/bin/sh -c 'while t..." 6 seconds ago Up 5 seconds
case4_alpine1_1
0b8060653617 case4_app1_1 "sh -C /tmp/start.sh" 6 seconds ago Up 5 seconds 0.0.0.0:34001->80/tcp,
[::]:34001->80/tcp case4_app1_1
bbe948f7b9a3 mysql:5.7-debian "docker-entrypoint.s..." 6 seconds ago Up 5 seconds 3306/tcp, 33060/tcp
case4_mysql1_1

```

- Jika kita akses `http://localhost:34001`, maka akan membuka web server Apache/php jika akses `http://localhost:10000` akan membuka phpMyAdmin.



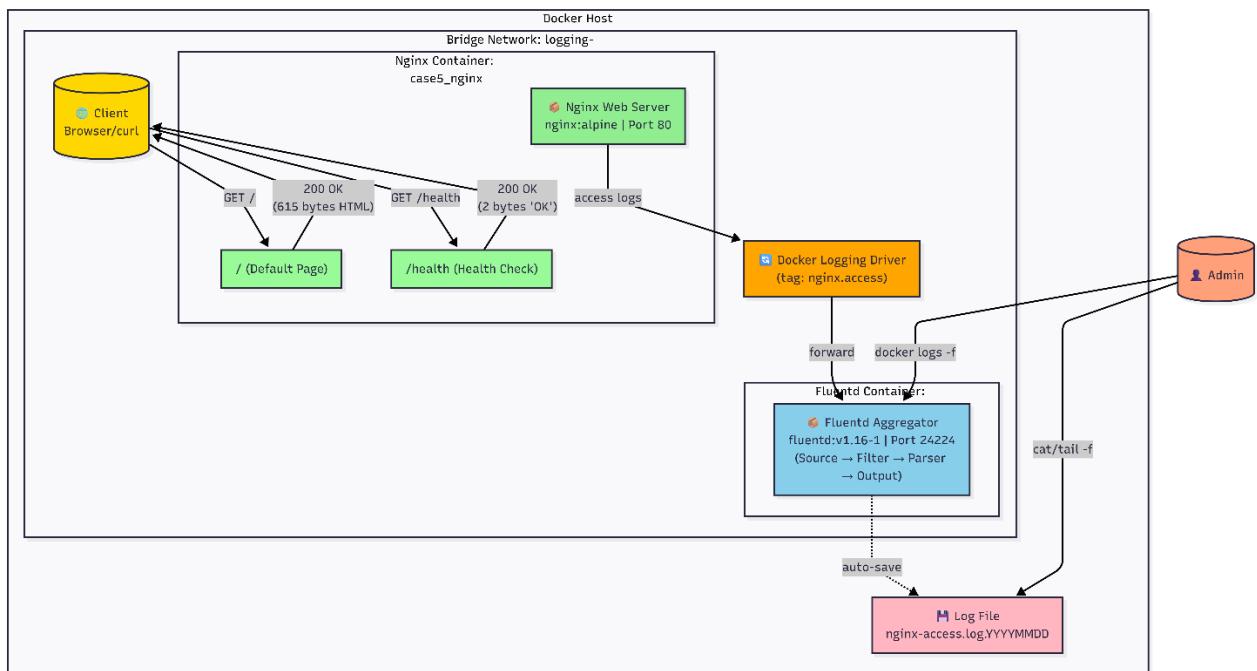
- Lalu jika kita menjalankan `docker-compose exec mysql1 sh /dbscripts/restoredb.sh` maka akan menambahkan tabel `mydata` di db `mydb` dan menambahkan row sesuai dengan yang ada pada file `backup.sql`, Serta jika kita beralih ke `http://localhost:34001/dbtest.php` maka akan muncul list tablenya.

The screenshot shows a MySQL database interface. On the left, the database structure is displayed with the 'mydb' database selected, containing tables like 'New', 'mydata', 'mysql', 'performance_schema', and 'sys'. On the right, a table named 'mydata' is shown with the following data:

	id	nama	alamat
1	Rodrigo Palacios	Argentina	
2	Marco Van Basten	Belanda	
3	Ruud Gullit	Belanda	
4	George Weah	Lampung	
5	Rajesh Koothrapali	India	
6	Leonard Hofstader	US	

3.5 Case 5 (Log Aggregation dengan Fluentd dan Nginx)

- Gambar Arsitektur



Pada gambar Arsitektur Log Aggregation di atas, diperlihatkan dua container utama yang saling terhubung melalui jaringan internal bernama `logging-network`. Arsitektur ini bertujuan untuk mendemonstrasikan bagaimana log aplikasi dari Web Server dapat dikumpulkan secara terpusat.

Container pertama adalah webserver (Nginx) yang berfungsi sebagai penghasil log (Log Generator). Konfigurasi Nginx di dalamnya diatur untuk menggunakan Docker Logging Driver jenis fluentd. Ini berarti, setiap access log (seperti permintaan HTTP yang masuk) dan error log yang dihasilkan Nginx tidak disimpan secara lokal, melainkan secara otomatis diteruskan melalui jaringan internal ke service Fluentd di port 24224. Container kedua adalah fluentd yang berfungsi sebagai pengumpul / agregator log. Fluentd listening port 24224 dan dikonfigurasi untuk menerima log yang ditandai nginx.access. Setelah menerima log, Fluentd memprosesnya dan mengarahkan output ke console (stdout) container itu sendiri. Arsitektur ini menunjukkan pemisahan tanggung jawab (Separation of Concerns) dimana Nginx berfokus melayani web, sementara Fluentd berfokus mengelola log. Komunikasi dilakukan secara internal menggunakan nama service (dari Nginx ke fluentd:24224), memastikan log berjalan lancar tanpa harus membuka port ke host untuk log itu sendiri.

- Cuplikan Kode / Script dan Penjelasan
 - docker-compose.yaml

```

1 events {
2     worker_connections 1024;
3 }
4
5 http {
6     include      /etc/nginx/mime.types;
7     default_type application/octet-stream;
8
9     # Custom log format untuk logging
10    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
11        '$status $body_bytes_sent "$http_referer" '
12        '"$http_user_agent" "$http_x_forwarded_for"';
13
14    # Access log akan ditangkap oleh Fluentd melalui Docker logging driver
15    access_log /dev/stdout main;
16    error_log /dev/stderr;
17
18    sendfile     on;
19    keepalive_timeout 65;
20}
21
22 server {
23     listen       80;
24     server_name  localhost;
25
26     location / {
27         root   /usr/share/nginx/html;
28         index  index.html index.htm;
29     }
30
31     # Halaman welcome sederhana
32     location = /health {
33         access_log /dev/stdout main;
34         return 200 "healthy\n";
35         add_header Content-Type text/plain;
36     }
37
38     error_page  500 502 503 504  /50x.html;
39     location = /50x.html {
40         root   /usr/share/nginx/html;
41     }
42 }
43

```

File `docker-compose.yaml` mendefinisikan arsitektur Log Aggregation sederhana dengan dua layanan di atas `logging-network`. Layanan fluentd berfungsi sebagai pengumpul log dengan port 24224 terbuka, sementara webserver (Nginx) berfungsi sebagai penghasil log. Kunci utamanya adalah konfigurasi logging driver pada webserver, yaitu fluentd dengan option `fluentd-address: localhost:24224` dan `tag: nginx.access`, yang memaksa semua log Nginx diteruskan langsung ke container fluentd. Penggunaan `depends_on: fluentd` memastikan pipeline siap sebelum log mulai dikirim, dan volume mount digunakan untuk konfigurasi serta persistensi data log.

- fluent.conf

```

1 <source>
2   type forward
3   port 24224
4   bind 0.0.0.0
5 </source>
6
7 <# Only pass lines that look like nginx access logs to the parser
8 <filter nginx=>>
9   type filter
10  regexps>
11   key log
12   pattern ^[^\n]+\d+\.\d+\.\d+\.\d+- /$ />
13 </regexps>
14 </filter>
15
16 <filter json=>>
17   type filter
18   key log
19   reserve_data true
20 </parse>
21   type record
22   pattern <?(<remote>\S+)> - (<user>\S+)> \[(?<time>\S+)\]> (?<method>\S+) (?<path>[^"]*) (?<protocol>[^"]*) (?<code>\d{3}) (?<size>\d+)? (?<referers>[^"]*) (?<agents>[^"]*) (?<os>[^"]*)?>?>/>
23   time_format %d/%b/%Y:%H:%M:%S %Z
24 </parse>
25 </filter>
26
27 <match stdout=>>
28   type copy
29 </match>
30 <match file=>>
31   type file
32   path /fluentd/log/nginx-access.log
33   append true
34 </match>
35 <format>
36   <type single_value>
37   message_key log
38   </type>
39 </format>
40 <buffer>
41   chunk_size 1000
42   chunk_limit_size 50
43   <buffer>
44   </buffer>
45 </buffer>
46 </match>
47
48 <filter file=>>
49   type filter
50   path /fluentd/log/nginx-access.log
51   flush_interval 10s
52   chunk_limit_size 50
53   <buffer>
54   </buffer>
55 </match>

```

Konfigurasi Fluentd ini merinci processing pipeline log yang masuk. Section `<source>` menggunakan type `forward` untuk mendengarkan di port 24224, menerima log dari Docker logging driver. Log yang masuk kemudian disaring (filter) dan di-parse menggunakan regular expression yang kompleks untuk mengubah raw log Nginx menjadi data JSON terstruktur, mengekstrak field penting seperti IP dan status code. Akhirnya, section `<match>` menduplikasi (copy) output log ke dua tujuan: type `stdout` untuk monitoring real-time via `docker logs -f`, dan type file untuk persistent storage di host machine.

- nginx.fluentd.conf

```
1 events {
2     worker_connections 1024;
3 }
4
5 http {
6     include         /etc/nginx/mime.types;
7     default_type application/octet-stream;
8
9     # Custom log format untuk logging
10    log_format main '$remote_addr $remote_user [$time_local] "$request"
11                      $status $body_bytes_sent "$http_referer"
12                      "$http_user_agent" "$http_x_forwarded_for"';
13
14    # Access log akan diketahui oleh Fluentd melalui Docker logging driver
15    access_log /dev/stdout main;
16    error_log /dev/stderr;
17
18    sendfile          on;
19    keepalive_timeout 65;
20
21    server {
22        listen          80;
23        server_name    localhost;
24
25        location / {
26            root   /usr/share/nginx/html;
27            index  index.html index.htm;
28        }
29
30        # Halaman welcome sederhana
31        location = /health {
32            access_log /dev/stdout main;
33            return 200 "healthy\n";
34            add_header Content-Type text/plain;
35        }
36
37        error_page  500 502 503 504  /50x.html;
38        location = /50x.html {
39            root   /usr/share/nginx/html;
40        }
41    }
42 }
```

Konfigurasi Nginx ini sangat penting untuk memastikan log berhasil dikirim. Blok http mendefinisikan `log_format main` yang menentukan format access log (wajib cocok dengan parser di Fluentd). Directive `access_log /dev/stdout main` mengarahkan semua access log ke stdout container, yang kemudian secara otomatis dicegat oleh Docker logging driver untuk diteruskan ke Fluentd. Blok server mendengarkan di port 80 dan menyediakan dua endpoint: root (/) dan `/health` check, di mana setiap akses ke endpoint ini akan menghasilkan access log yang dikirim ke pipeline Fluentd.

• Cara Menjalankan

- Pertama kita hanya perlu menjalankan `docker-compose up -d`, dan kita bisa coba cek dengan `docker ps`.

```
ruihi-sadipta@rafini-sadipta-VMware-Virtual-Platform:~/Cloud2025/tuges/tugas2/case5$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS NAMES
585ed9b9593b nginx:alpine "/docker-entrypoint..." 4 seconds ago Up 4 seconds 0.0.0.0:80->80/tcp, ::1:8080->8080/tcp cases5_ginx
a52d780ea9 fluent/fluentd:v1.16.1 "tini -- /bin/entryp..." 7 seconds ago Up 6 seconds $140/tcp, 0.0.0.0:42424->42424/tcp [::]:42424->42424/tcp [::]:24224->24224/udp [::]:24224->24224/udp cases5_Fluentd
```

- Sekarang kita coba mengakses 2 webpage, yaitu default `http://localhost`, dan checkpoint health `http://localhost/health`, lalu coba untuk di refresh beberapa kali agar bisa masuk ke log yang bisa diakses atau dilihat di dalam file log `nginx-access.log.YYYYMMDD` atau dengan menjalankan perintah `docker logs -f case5_fluentd`

Perbedaan kedua endpoint terlihat dari ukuran response: root path (/) menghasilkan 615 bytes HTML, sedangkan /health hanya 2 bytes ("OK"). Dari screenshot terlihat log dalam format JSON yang menampilkan field hasil parsing seperti IP address, HTTP method, path, status code, dan timestamp, membuktikan pipeline Fluentd berhasil memproses log dari Nginx secara real-time.

- Kapan Skenario ini cocok untuk dilakukan

Skenario Log Aggregation dengan Fluentd sangat cocok untuk aplikasi production atau sistem dengan banyak container yang membutuhkan monitoring terpusat. Pendekatan ini penting ketika debugging langsung ke setiap container sudah tidak praktis, atau ketika log harus disimpan secara persisten di luar container yang sifatnya ephemeral. Fluentd menjadi pilihan ideal pada kondisi seperti:

- Aplikasi terdiri dari banyak service (microservices) dan log perlu dianalisis secara bersamaan.
 - Ada kebutuhan compliance atau audit yang mengharuskan penyimpanan log terstruktur dalam jangka panjang.
 - Dibutuhkan real-time monitoring atau alerting berdasarkan pola/anomali pada log.

- Sistem berjalan di platform orkestrasi seperti Kubernetes, di mana pod bersifat sementara sehingga log harus diekspor ke storage eksternal.

4. Kesimpulan

Dari seluruh rangkaian tugas, saya menyimpulkan bahwa penggunaan Docker Compose memberikan kemudahan dan konsistensi yang sangat baik dalam mengelola aplikasi yang terdiri dari berbagai layanan. Dengan mendefinisikan seluruh infrastruktur dalam satu file YAML, saya dapat menjalankan, mengatur dependensi, serta mengelola jaringan dan konfigurasi lingkungan dengan cara yang jauh lebih terstruktur dan mudah direproduksi.

Setiap case yang dikerjakan memberikan pemahaman yang semakin luas. Pada Case 1 dan 4, saya berhasil menerapkan konsep fundamental seperti port mapping, volume mapping, serta membangun custom image untuk lingkungan full-stack PHP/Apache dan MySQL. Case ini memperkuat pemahaman saya tentang bagaimana container bekerja, berkomunikasi, dan menyimpan data secara persisten. Case 2 dan 3 memberikan pengalaman dalam membangun arsitektur yang lebih modern. Pada Case 2, saya dapat mengonfigurasi layanan dengan dukungan HTTPS, sementara Case 3 memungkinkan saya untuk mengintegrasikan beberapa layanan sekaligus dalam satu orkestrasi WordPress yang lengkap. Keduanya memperlihatkan bagaimana Docker Compose mampu menangani sistem multi-container secara efisien. Pada Case 5, saya memperdalam pemahaman mengenai centralized logging dengan Fluentd. Saya berhasil membangun pipeline log yang mengambil output dari Docker Logging Driver, memprosesnya melalui tahap filtering dan parsing, kemudian mengirimkannya ke beberapa output. Case ini memperkenalkan konsep observability dan praktik DevOps yang umum digunakan pada sistem modern.

Secara keseluruhan, seluruh study case ini menunjukkan bagaimana Docker Compose dapat menyederhanakan deployment, maintenance, serta pengelolaan beberapa container secara terintegrasi. Keberhasilan menyelesaikan semua case ini memberikan fondasi yang kuat untuk memahami arsitektur containerized dan mempersiapkan pengembangan ke arah microservices yang lebih besar dan kompleks.