

**IMPLEMENTASI SISTEM DISTRIBUSI DATA PRODUK  
MENGUNAKAN POSTGRESQL, REDIS, DAN MONGODB  
BERBASIS DOCKER**

**UAS PEMROSESAN DATA TERDISTRIBUSI**

Oleh:

**MUHAMMAD HASBI AS'ARI**  
**10222175**



**PROGRAM STUDI INFORMATIKA  
SEKOLAH TINGGI TEKNOLOGI CIPASUNG  
TASIKMALAYA  
2025**

# **BAB I**

## **PENDAHULUAN**

### **1.1 Latar Belakang**

Dalam pengembangan sistem modern, kebutuhan untuk mengelola data yang besar, beragam, dan real-time menjadi semakin penting. Sistem informasi yang mengandalkan satu jenis database seringkali tidak mampu memenuhi semua kebutuhan performa, struktur data, dan skalabilitas secara bersamaan. Oleh karena itu, pendekatan database terdistribusi menjadi solusi yang umum diterapkan, dengan memanfaatkan kekuatan dari berbagai jenis database untuk fungsi yang berbeda.

Dalam proyek ini, penulis mengembangkan sebuah sistem distribusi data produk untuk simulasi e-commerce sederhana menggunakan tiga jenis database:

- PostgreSQL digunakan untuk menyimpan data produk utama seperti nama dan harga,
- Redis digunakan untuk menyimpan informasi stok secara cepat,
- MongoDB digunakan untuk menyimpan metadata produk seperti deskripsi dan review.

Semua layanan ini dijalankan secara terintegrasi menggunakan Docker dan Flask API, sehingga pengguna dapat mengakses data lengkap melalui satu endpoint API. Pendekatan ini mencerminkan praktik nyata dalam pengembangan sistem terdistribusi modern.

### **1.2 Rumusan Masalah**

1. Bagaimana membangun sistem database terdistribusi dengan PostgreSQL, Redis, dan MongoDB?
2. Bagaimana merancang integrasi antar database yang berbeda jenis dalam satu API?
3. Bagaimana menyajikan data produk secara lengkap melalui satu endpoint terpadu?

### **1.3 Tujuan**

1. Merancang dan mengimplementasikan sistem database terdistribusi.

2. Mengintegrasikan PostgreSQL, Redis, dan MongoDB dalam satu layanan backend.
3. Menghasilkan endpoint API yang menyajikan data lengkap dari ketiga sumber database.

#### **1.4 Manfaat**

1. Memberikan pemahaman praktis tentang pengelolaan database terdistribusi.
2. Menjadi referensi implementasi nyata penggunaan multibase dalam pengembangan sistem.
3. Mengasah kemampuan teknis dalam penggunaan Docker, Flask, dan berbagai jenis database.

## **BAB II**

### **LANDASAN TEORI**

#### **2.1 Sistem Terdistribusi**

Sistem terdistribusi adalah sekumpulan komputer atau komponen perangkat lunak yang saling berinteraksi dan berkomunikasi untuk mencapai tujuan bersama, namun tersebar secara fisik di berbagai lokasi. Dalam konteks pengolahan data, sistem terdistribusi digunakan untuk meningkatkan performa, skalabilitas, dan keandalan sistem. Sistem ini memungkinkan pemrosesan data dari berbagai sumber dan tipe database yang berbeda secara terpadu.

#### **2.2 Database Terdistribusi**

Database terdistribusi adalah sistem penyimpanan data yang tersebar di beberapa lokasi fisik namun terintegrasi secara logis. Artinya, meskipun data tersimpan di banyak tempat atau dalam jenis database yang berbeda, pengguna dapat mengaksesnya seolah-olah dari satu sistem tunggal. Pendekatan ini cocok digunakan dalam sistem modern yang membutuhkan efisiensi, kecepatan akses data, dan fleksibilitas struktur data.

#### **2.3 PostgreSQL**

PostgreSQL adalah sistem manajemen basis data relasional open-source yang mendukung SQL standar dan berbagai fitur lanjutan. Dalam proyek ini, PostgreSQL digunakan untuk menyimpan data produk seperti id, name, price, dan relasi dengan kategori.

#### **2.4 Redis**

Redis (Remote Dictionary Server) adalah database key-value yang sangat cepat dan sering digunakan untuk caching, pub-sub, dan penyimpanan data ringan sementara. Pada proyek ini, Redis digunakan untuk menyimpan informasi stok produk karena sifatnya yang cepat dan efisien untuk pembacaan berulang.

#### **2.5 MongoDB**

MongoDB adalah database NoSQL yang menyimpan data dalam format dokumen BSON (mirip JSON). MongoDB cocok digunakan untuk data

semi-terstruktur atau data yang bentuknya fleksibel. Dalam proyek ini, MongoDB digunakan untuk menyimpan metadata produk seperti description dan reviews.

## **2.6 Flask**

Flask adalah framework web mikro berbasis Python yang digunakan untuk membangun API REST secara ringan dan efisien. Pada sistem ini, Flask berperan sebagai pusat integrasi yang menyatukan data dari PostgreSQL, Redis, dan MongoDB ke dalam satu endpoint API /products.

## **2.7 Docker dan Docker Compose**

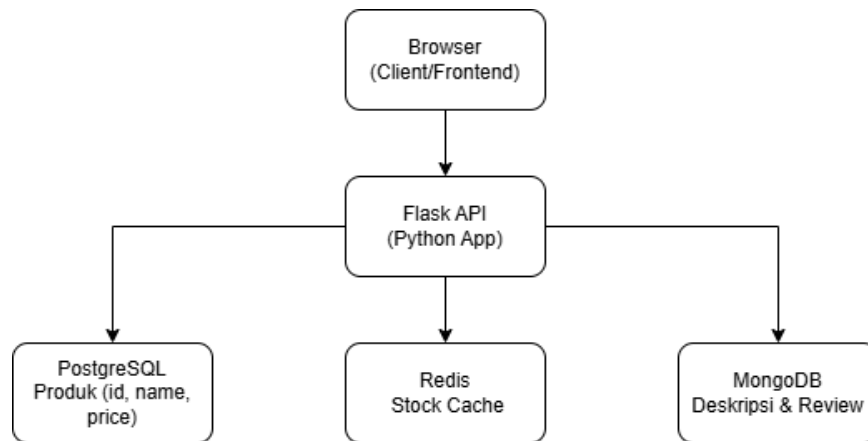
Docker adalah platform yang memungkinkan aplikasi dijalankan dalam wadah (container) terisolasi. Docker Compose adalah alat bantu untuk menjalankan banyak container secara bersamaan menggunakan satu file konfigurasi. Dalam proyek ini, Docker Compose digunakan untuk menjalankan container PostgreSQL, Redis, MongoDB, dan Flask secara otomatis dan terintegrasi.

## BAB III

### PERANCANGAN SISTEM

#### 3.1 Arsitektur Sistem

Sistem ini terdiri dari beberapa komponen utama yang berjalan secara terdistribusi dan terintegrasi melalui sebuah aplikasi backend berbasis Flask. Berikut adalah arsitektur sistem yang digunakan:



Setiap jenis database dalam sistem ini memiliki fungsi yang berbeda dan saling melengkapi:

##### 1. PostgreSQL

Digunakan untuk menyimpan data produk yang bersifat terstruktur, seperti:

- a. id produk
- b. nama produk
- c. harga
- d. kategori

2. PostgreSQL dipilih karena mendukung relasi antar tabel dan cocok untuk data transaksional.

##### 3. Redis

Digunakan untuk menyimpan informasi stok produk secara real-time.

Redis menyimpan data dalam format key-value yang sangat cepat diakses, misalnya:

- a. stock:product:1 → 15
- b. stock:product:2 → 40

Redis dipilih karena sangat efisien untuk proses pembacaan data yang cepat dan berulang (caching).

#### 4. MongoDB

Menyimpan metadata produk yang bersifat semi-terstruktur dan fleksibel, seperti:

- a. deskripsi produk
- b. ulasan/review dari pengguna

MongoDB dipilih karena mampu menyimpan data dalam bentuk dokumen (JSON-like) dan mendukung nested object seperti array reviews.

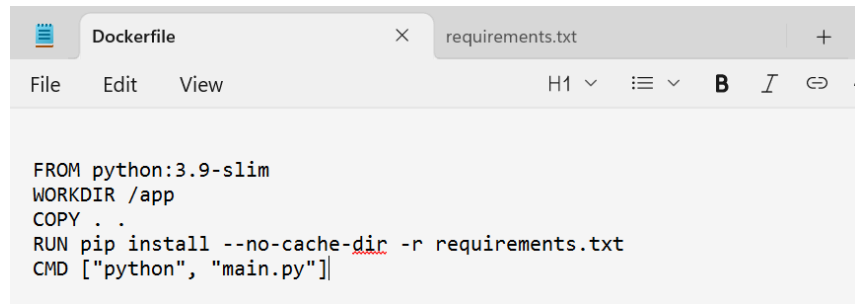
### 3.2 Alur Kerja Sistem (Workflow)

Proses pengembangan dan eksekusi sistem ini dimulai dari tahap pembuatan file, konfigurasi layanan, hingga integrasi data dan penyajian output API. Berikut adalah langkah-langkah alur kerja sistem secara menyeluruh:

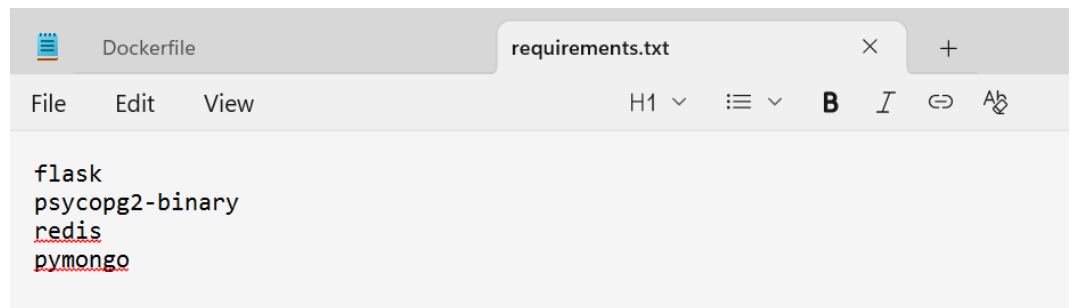
#### 1. Pembuatan File Aplikasi

- Membuat file Python utama main.py yang berisi endpoint /products.
- Membuat tiga file bantu:
  - db\_postgres.py → mengakses PostgreSQL
  - db\_redis.py → mengakses Redis
  - db\_mongo.py → mengakses MongoDB
- Menyusun requirements.txt dan Dockerfile untuk keperluan containerisasi aplikasi Flask.

Name	Date modified	Type	Size
__pycache__	29/06/2025 14:19	File folder	
db_mongo.py	29/06/2025 14:19	Python Source File	1 KB
db_postgres.py	29/06/2025 13:48	Python Source File	1 KB
db_redis.py	29/06/2025 13:48	Python Source File	1 KB
Dockerfile	29/06/2025 13:48	File	1 KB
main.py	29/06/2025 14:21	Python Source File	1 KB
requirements.txt	29/06/2025 14:22	Text Document	1 KB

A screenshot of a code editor with two tabs: 'Dockerfile' and 'requirements.txt'. The 'Dockerfile' tab is active, showing the following content:

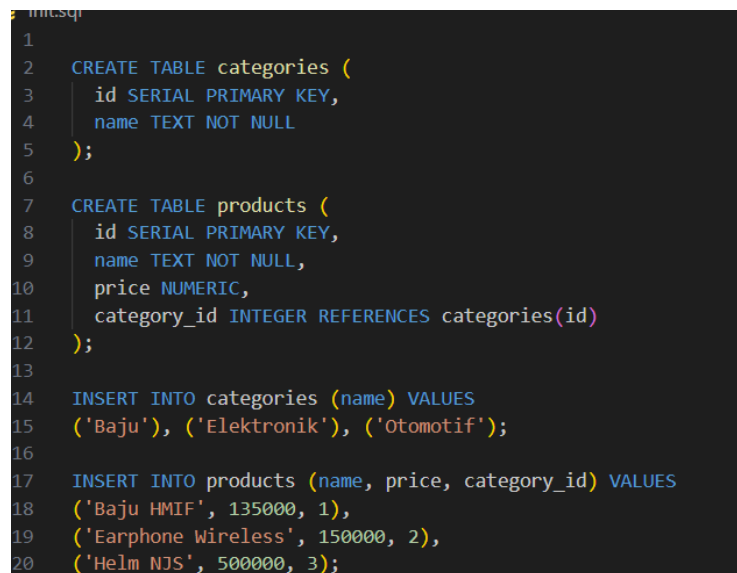
```
FROM python:3.9-slim
WORKDIR /app
COPY . .
RUN pip install --no-cache-dir -r requirements.txt
CMD ["python", "main.py"]
```

A screenshot of a code editor with two tabs: 'Dockerfile' and 'requirements.txt'. The 'requirements.txt' tab is active, showing the following content:

```
flask
psycpg2-binary
redis
pymongo
```

## 2. Setup Database

- Menyusun skrip init.sql untuk membuat tabel dan mengisi data awal di PostgreSQL.

A screenshot of a code editor showing a SQL script. The script is as follows:

```
1
2 CREATE TABLE categories (
3     id SERIAL PRIMARY KEY,
4     name TEXT NOT NULL
5 );
6
7 CREATE TABLE products (
8     id SERIAL PRIMARY KEY,
9     name TEXT NOT NULL,
10    price NUMERIC,
11    category_id INTEGER REFERENCES categories(id)
12 );
13
14 INSERT INTO categories (name) VALUES
15 ('Baju'), ('Elektronik'), ('Otomotif');
16
17 INSERT INTO products (name, price, category_id) VALUES
18 ('Baju HMIF', 135000, 1),
19 ('Earphone Wireless', 150000, 2),
20 ('Helm NJS', 500000, 3);
```

- Menyiapkan key stok produk di Redis secara manual menggunakan redis-cli.



- Memasukkan metadata produk dan review ke MongoDB melalui perintah insertMany() menggunakan mongosh.

```
ecommerce> db.product_metadata.insertMany([
... {
...   product_id: 1,
...   description: "Kaos eksklusif acara HMIF. Bahan adem dan premium.",
...   reviews: [
...     { user: "hasbi", rating: 5, comment: "Keren banget!" }
...   ]
... },
... {
...   product_id: 2,
...   description: "Earphone nirkabel dengan bass mantap.",
...   reviews: [
...     { user: "dinda", rating: 4, comment: "Enak suaranya" }
...   ]
... },
... {
...   product_id: 3,
...   description: "Helm full face, cocok buat touring jauh.",
...   reviews: []
... }
... ])
```

### 3. Konfigurasi Docker Compose

- Menulis docker-compose.yml untuk menjalankan 4 container:
  - postgres untuk PostgreSQL
  - redis untuk Redis
  - mongodb untuk MongoDB
  - app untuk aplikasi Flask

```
docker-compose.yml
services:
  postgres:
    ports:
  volumes:
    - pgdata:/var/lib/postgresql/data

  mongodb:
    image: mongo
    container_name: mongodb
    ports:
    - "27017:27017"
    volumes:
    - mongoddata:/data/db

  redis:
    image: redis
    container_name: redis
    ports:
    - "6379:6379"

  app:
    build: ./app
    container_name: app
    ports:
    - "5000:5000"
    depends_on:
    - postgres
    - redis
    volumes:
    - ./app:/app
    working_dir: /app
    command: sh -c "pip install -r requirements.txt && python main.py"

volumes:
  pgdata:
  mongoddata:
```

### b. Redis:

Key: stock:product:{id}

Value: jumlah stok produk

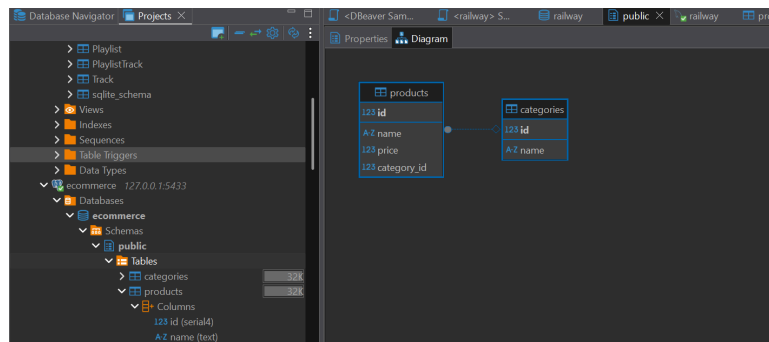
Contoh: stock:product:1 → 15

### c. MongoDB:

Koleksi: product\_metadata

Dokumen contoh:

```
{
  "product_id": 1,
  "description": "Kaos eksklusif acara HMIF...",
  "reviews": [
    { "user": "hasbi", "rating": 5, "comment": "Keren banget!" }
  ]
}
```



```
localhost:5000/products
Pretty print
[
  {
    "description": "Kaos eksklusif acara HMIF. Bahan adem dan premium.",
    "id": 1,
    "name": "Baju HMIF",
    "price": 135000.0,
    "reviews": [
      {
        "comment": "Keren banget!",
        "rating": 5,
        "user": "hasbi"
      }
    ],
    "stock": 15
  },
  {
    "description": "Earphone nirkabel dengan bass mantap.",
    "id": 2,
    "name": "Earphone Wireless",
    "price": 150000.0,
    "reviews": [
      {
        "comment": "Enak suaranya",
        "rating": 4,
        "user": "dinda"
      }
    ],
    "stock": 40
  },
  {
    "description": "Helm full face, cocok buat touring jauh.",
    "id": 3,
    "name": "Helm NDS",
    "price": 500000.0,
    "reviews": [],
    "stock": 20
  }
]
```

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Lingkungan Pengembangan

Implementasi sistem dilakukan pada perangkat dengan spesifikasi dan perangkat lunak sebagai berikut:

Komponen	Detail
OS	Windows 11
Bahasa Pemrograman	Python 3.9
Framework Backend	Flask
Manajemen DB	PostgreSQL, Redis, MongoDB
Tool Admin DB	DBeaver (untuk PostgreSQL)
Platform Kontainer	Docker & Docker Compose
API Endpoint	<a href="http://localhost:5000/products">http://localhost:5000/products</a>
Editor Kode	Visual Studio Code (VS Code)

#### 4.2 Pengisian Data ke Database

Data produk dimasukkan ke masing-masing database sebagai berikut:

##### 1. PostgreSQL

Data products dan categories diinput menggunakan file init.sql melalui DBeaver. Contoh data:

id	name	price	category_id
1	Baju HMIF	135000.0	1
2	Earphone Wireless	150000.0	2

3	Helm NJS	500000.0	3
---	----------	----------	---

## 2. Redis

Stok produk disimpan dalam format key-value melalui Redis CLI:

SET stock:product:1 15

SET stock:product:2 40

SET stock:product:3 20

```
C:\ecommerce-distributed>docker exec -it redis redis-cli
127.0.0.1:6379> SET stock:product:1 15
stock:product:3 20
OK
127.0.0.1:6379> SET stock:product:2 40
OK
127.0.0.1:6379> SET stock:product:3 20
OK
127.0.0.1:6379> █
```

## 3. MongoDB

Metadata dan review dimasukkan menggunakan perintah insertMany() di mongosh:

```
db.product_metadata.insertMany([
  {
    product_id: 1,
    description: "Kaos eksklusif acara HMIF...",
    reviews: [
      { user: "hasbi", rating: 5, comment: "Keren banget!" }
    ]
  },
  ...
])
```

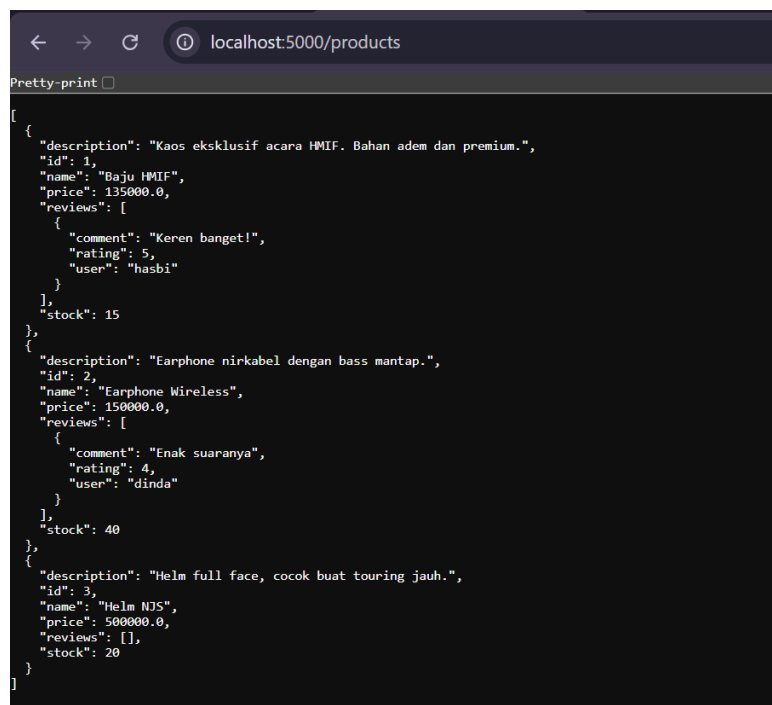
```

test> use ecommerce
...
switched to db ecommerce
ecommerce> db.product_metadata.insertMany([
... {
...   product_id: 1,
...   description: "Kaos eksklusif acara HMIF. Bahan adem dan premium.",
...   reviews: [
...     { user: "hasbi", rating: 5, comment: "Keren banget!" }
...   ]
... },
... {
...   product_id: 2,
...   description: "Earphone nirkabel dengan bass mantap.",
...   reviews: [
...     { user: "dinda", rating: 4, comment: "Enak suaranya" }
...   ]
... },
... {
...   product_id: 3,
...   description: "Helm full face, cocok buat touring jauh.",
...   reviews: []
... }
... ])
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('686103751a7254cc7169e328'),
    '1': ObjectId('686103751a7254cc7169e329'),
    '2': ObjectId('686103751a7254cc7169e32a')
  }
}

```

### 4.3 Pengujian API Endpoint /products

Pengujian dilakukan dengan mengakses endpoint <http://localhost:5000/products>. Sistem berhasil menggabungkan data dari tiga database ke dalam satu JSON output.



```

localhost:5000/products
Pretty-print ☐
[
  {
    "description": "Kaos eksklusif acara HMIF. Bahan adem dan premium.",
    "id": 1,
    "name": "Baju HMIF",
    "price": 135000.0,
    "reviews": [
      {
        "comment": "Keren banget!",
        "rating": 5,
        "user": "hasbi"
      }
    ],
    "stock": 15
  },
  {
    "description": "Earphone nirkabel dengan bass mantap.",
    "id": 2,
    "name": "Earphone Wireless",
    "price": 150000.0,
    "reviews": [
      {
        "comment": "Enak suaranya",
        "rating": 4,
        "user": "dinda"
      }
    ],
    "stock": 40
  },
  {
    "description": "Helm full face, cocok buat touring jauh.",
    "id": 3,
    "name": "Helm NJS",
    "price": 500000.0,
    "reviews": [],
    "stock": 20
  }
]

```

## **BAB IV**

### **IMPLEMENTASI DAN PENGUJIAN**

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, dapat disimpulkan bahwa:

1. Sistem distribusi data berbasis multibase berhasil dibangun

Sistem ini berhasil menggabungkan tiga jenis database berbeda—PostgreSQL, Redis, dan MongoDB—dalam satu layanan API yang terintegrasi melalui framework Flask.
2. Setiap jenis database memiliki fungsi tersendiri sesuai karakteristik datanya
  - PostgreSQL menyimpan data produk utama yang bersifat terstruktur dan relasional.
  - Redis menyimpan informasi stok produk secara cepat menggunakan format key-value.
  - MongoDB menyimpan metadata produk seperti deskripsi dan ulasan pelanggan dalam format semi-struktur.
3. Pengguna dapat mengakses data produk lengkap dari satu endpoint

Melalui endpoint `/products`, pengguna dapat memperoleh informasi lengkap yang sudah digabungkan dari ketiga sumber data tersebut dalam format JSON. Hal ini membuktikan keberhasilan integrasi sistem terdistribusi.
4. Penggunaan Docker dan Docker Compose memudahkan pengelolaan seluruh service

Seluruh komponen sistem dapat dijalankan bersamaan secara otomatis dalam container, tanpa perlu instalasi manual dan konfigurasi yang rumit.

Saran (Opsional)

Untuk pengembangan lebih lanjut, sistem ini dapat ditambahkan fitur:

- Penambahan user interface (frontend)
- Endpoint untuk input dan update data
- Monitoring performa tiap database

- Implementasi FDW (Foreign Data Wrapper) di PostgreSQL sebagai eksperimen lanjutan