E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756

PENERAPAN *EVENT-DRIVEN MICROSERVICES* PADA APLIKASI LAYANAN PENERIMAAN PESERTA DIDIK BARU

Umar Syarif[1)], Pizaini[2)]
[1, 2)] Program Studi Teknik Informatika, Universitas Islam Negeri Sultan Syarif Kasim Riau, Indonesia
e-mail: 11750115076@students.uin-suska.ac.id[1)], pizaini@uin-suska.ac.id [2)]

ABSTRAK

Banyaknya penelitian-penelitian yang dilakukan dan aplikasi penerimaan peserta didik baru yang dihasilkan telah membantu proses PPDB pada institusi pendidikan yang menjadi objek penelitian, namun aplikasi tersebut belum tentu mampu berjalan dengan baik apabila dihadapkan pada kasus yang berbeda dan aplikasi tersebut akan menjadi tidak relevan apabila dipakai pada tingkat lain yang tidak memerlukan isian jurusan. Adanya sebuah layanan yang bisa memfasilitasi proses PPDB di segala tingkat pendidikan akan memudahkan institusi pendidikan yang belum memiliki fasilitas pendaftaran online dalam melaksanakan program PPDB tanpa perlu memikirkan proses development aplikasi yang memakan waktu dan biaya, mereka cukup mendaftar pada Aplikasi Layanan PPDB dan dapat membuka pendaftaran online pada saat itu juga. Namun, dalam rangka memfasilitasi proses bisnis yang berbeda-beda tersebut diperlukan logika yang kompleks, sehingga akan menghasilkan codebase yang besar. Maka dibutuhkanlah teknologi Docker untuk menyederhanakan pemaketan software sesuai dengan kebutuhan dan meletakkannya dalam kontainer terisolasi yang disebut dengan docker container, sehingga cocok diterapkan pada arsitektur microservices. Microservice yang menerapkan event-driven akan memungkinkan terjadinya pertukaran data antar service melalui Redis Stream sebagai message broker, dimana sebuah service yang ingin menggunakan data dari service lain dapat men-subscribe sebuah event, dan service lain yang ingin membagikan data akan mempublish sebuah event dalam sebuah topik tertentu. Berdasarkan masalah tersebut, penulis bermaksud untuk melakukan penelitian yang berjudul "Penerapan Event-Driven Microservices pada Aplikasi Layanan Penerimaan Peserta Didik Baru".

***Kata Kunci**: Microservice, Docker, PPDB, Pendaftaran.*

ABSTRACT

The number of studies conducted and the application of new student admissions produced has helped the PPDB process at the educational institution that is the object of research, but the application is not necessarily able to run well when faced with different cases and the application will become irrelevant if used at other levels that do not require majors. The existence of a service that can facilitate the PPDB process at all levels of education will make it easier for educational institutions that do not have online registration facilities to implement the PPDB program without the need to think about the time-consuming and costly application development process, they simply register on the PPDB Service Application and can open online registration. at that very moment. However, in order to facilitate these different business processes, complex logic is needed, which will result in a large codebase. We need Docker technology to simplify software packaging according to needs and put it in an isolated container called a docker container, so it is suitable to be applied to microservices architecture. Microservices that implement event-driven will allow the exchange of data between services through Redis Stream as a message broker, where a service that wants to use data from another service can subscribe to an event, and another service that wants to share data will publish an event in an event. specific topic. Based on these problems, the author intends to conduct a study entitled "Implementation of Event-Driven Microservices in New Student Admission Service Applications".

***Keywords**: Microservice, Docker, PPDB, Registration.*

I. PENDAHULUAN

erdasarkan situs scholar.google.com hasil pencarian *keyword* "aplikasi penerimaan peserta didik baru" mendapatkan 33.800 hasil [1]. Penelitian-penelitian tersebut telah membantu institusi pendidikan setingkat SD, BSMP, SMA, bahkan tingkat Perguruan Tinggi dalam proses PPDB (Penerimaan Peserta Didik Baru). Aplikasi yang dibangun berhasil mengakomodir proses PPDB pada institusi pendidikan yang menjadi objek penelitian, namun aplikasi tersebut belum tentu mampu berjalan dengan baik apabila dihadapkan pada kasus yang berbeda. Contohnya aplikasi penerimaan peserta didik baru pada tingkat SMK mengharuskan calon peserta didik baru mengisi pilihan jurusan [2], aplikasi tersebut akan menjadi tidak relevan apabila dipakai pada tingkat lain yang tidak memerlukan isian jurusan [3].

Adanya sebuah layanan yang bisa memfasilitasi proses PPDB di segala tingkat pendidikan akan memudahkan institusi pendidikan yang belum memiliki fasilitas pendaftaran online dalam melaksanakan program PPDB.

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756

Institusi tersebut tidak perlu memikirkan proses *development* aplikasi yang memakan waktu dan biaya, ataupun mengurus hal-hal terkait operasional pembuatan aplikasi seperti *deployment* dan pemeliharaan *server*. Mereka cukup mendaftar pada Aplikasi Layanan PPDB dan dapat membuka pendaftaran online pada saat itu juga.

Namun, dalam rangka memfasilitasi proses bisnis yang berbeda-beda tersebut diperlukan logika yang kompleks, sehingga akan menghasilkan *codebase* yang besar. Hal tersebut akan mengakibatkan proses *maintenance* menjadi lebih lama karena *track* baris kode yang besar, sehingga untuk menambahkan fitur baru menjadi lebih sulit [4]. Pada penelitian ini, *microservices* hadir sebagai suatu arsitektur dalam pengembangan aplikasi dimana suatu sistem akan dipecah menjadi beberapa *service* kecil independen yang menjalankan tugas tertentu [5]. Hal ini menjadikan aplikasi *microservice* lebih *maintainable* dan *scalable* [6].

Selanjutnya, untuk dapat menerapkan *microservice* pada aplikasi ini, membutuhkan sebuah teknologi virtualisasi berbasis *hypervisor*, dimana setiap *container* akan diletakkan ke dalam *Virtual Machine* (VM) yang berbeda [7]. Peran VM tersebut dapat digantikan oleh teknologi Docker yang nantinya akan memberikan keuntungan dalam pengembangan aplikasi [8]. Docker menyederhanakan pemaketan *software* sesuai dengan kebutuhan dan meletakkannya dalam kontainer terisolasi yang disebut dengan *docker container*, sehingga cocok diterapkan pada arsitektur *microservices* [4]. Dengan adanya isolasi dan keamanan yang memadai memungkinkan pengguna Docker untuk mendefinisikan *service-service* menjadi *docker* image dan menjalankan banyak *container* di waktu yang bersamaan pada host tertentu [9]. Kemudian *container-container* tersebut dapat terhubung dengan *container* lain pada jaringan *docker network* yang sama, guna berkomunikasi sehingga menjadi satu kesatuan *microservice* [10].

Dalam pengimplementasiannya *microservice* yang telah dijalankan akan menerapkan *event-driven microservice* dengan *choreography pattern*. *Microservice* yang menerapkan *event-driven* akan memungkinkan terjadinya pertukaran data antar *service* melalui Redis Stream sebagai *message broker*, dimana sebuah *service* yang ingin menggunakan data dari *service* lain dapat men-subscribe sebuah event, dan *service* lain yang ingin membagikan data akan mempublish sebuah event dalam sebuah topik tertentu [11]. Pertukaran data yang berbasis *event* menjadikan komunikasi antar service dapat terjadi secara asynchronous. Pola ini dalam pengembangan aplikasi berbasis *microservice* disebut *choreography pattern*. Pattern ini menjadikan proses *microservice* berjalan secara paralel dan memiliki logika *service* yang decentralized, dimana tidak ada penumpukan logika proses bisnis pada *service* tertentu sehingga mengakibatkan waktu tunggu client menjadi lambat.

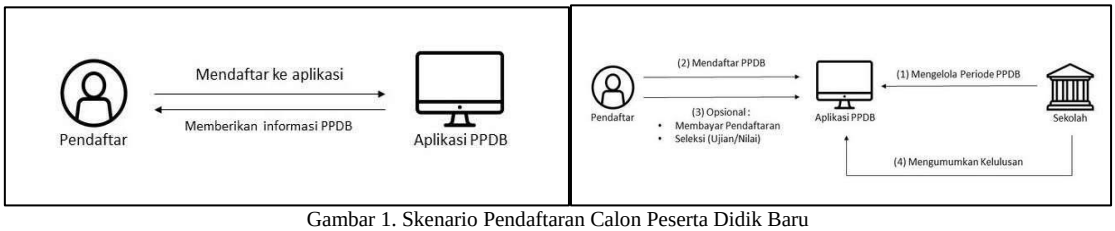
Berdasarkan keunggulan yang dimiliki oleh arsitektur *event-driven microservices* dan implementasinya menggunakan Docker, maka penulis bermaksud untuk melakukan penelitian mengenai penerapan *event-driven microservices* pada Aplikasi Layanan Penerimaan Peserta Didik Baru sebagai alternatif alat bantu bagi institusi pendidikan dalam proses melakukan PPDB secara online. Sehingga memudahkan institusi pendidikan untuk membuka dan mengelola pendaftaran online tanpa memikirkan hal-hal yang bersifat teknis pembuatan aplikasi.

II. METODE PENELITIAN

Metodologi penelitian dilakukan untuk mendapatkan hasil yang sesuai dengan tujuan. Tahapan yang dilakukan dalam penelitian ini adalah: Analisa, Perancangan, Implementasi, Pengujian, Kesimpulan dan Saran

A. Skenario

Rancangan *microservice* yang akan dibangun harus dapat menyesuaikan dengan alur penerimaan peserta didik baru pada masing-masing institusi pendidikan. Skenario yang terdapat pada rancangan, dapat dilihat pada gambar 1:



Gambar 1. Skenario Pendaftaran Calon Peserta Didik Baru

Dari skenario tersebut hasil yang ingin dicapai adalah:

1) Calon Peserta Didik (Pendaftar) dapat mendaftar pada program penerimaan peserta didik baru di institusi pendidikan yang diinginkan, mengikuti alur yang ditetapkan institusi, dan melihat kelulusan.

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756

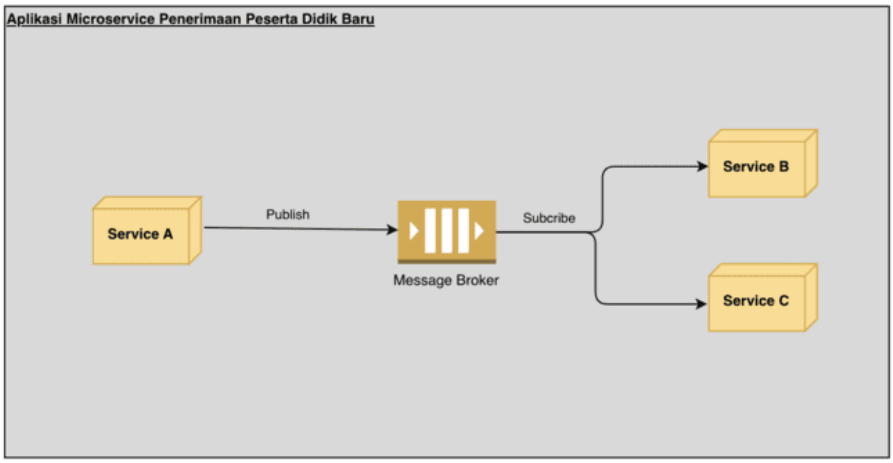
2) Institusi Pendidikan dapat mengadakan program penerimaan peserta didik baru secara online tanpa perlu memikirkan tahap *development* aplikasi.
3) Alur pendaftaran peserta didik baru dapat disesuaikan dengan kebutuhan institusi. Saat kebutuhan berubah alur dapat disesuaikan dengan kebutuhan yang baru.

B.Perancangan

Dalam penelitian ini, perancangan dibagi menjadi dua tahap, yaitu Perancangan Arsitektur dan Perancangan *Service*.

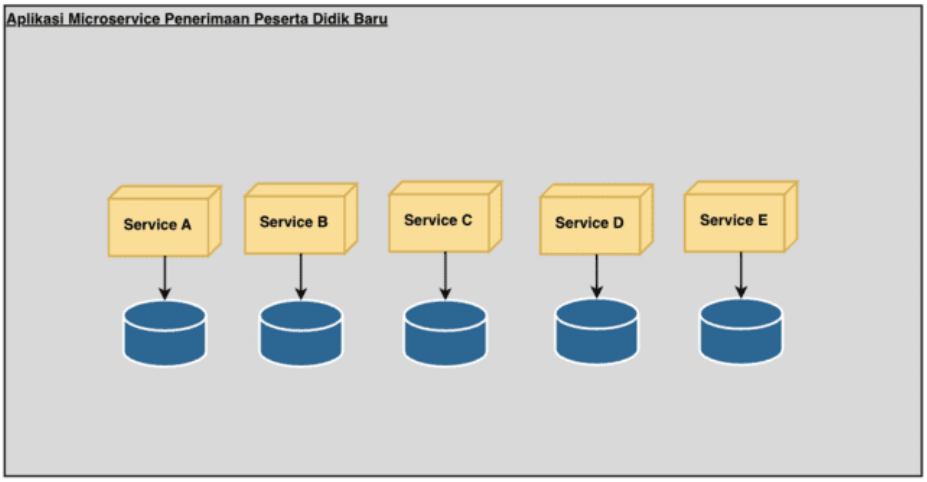
1) Perancangan Arsitektur

Pada tahap ini akan dirancang arsitektur keseluruhan dari aplikasi *microservice* yang akan dibangun. Termasuk didalamnya metode komunikasi antar *service*, penyimpanan data, metode komunikasi *client-server*, serta *authentication* dan *authorization.* Diagram alur komunikasi antar *service* dapat dilihat pada gambar 2 berikut:



Gambar 2. Diagram alur komunikasi antar *service*

Alur komunikasi antar *service* yang akan digunakan pada penelitian ini adalah menggunakan pola *choreography* yang mengandalkan *message broker* dalam pertukaran data dengan *event* sebagai *trigger* nya. *Service* yang membutuhkan data dari *service* lain akan men-*subscribe* pada sebuah *event*, dan ketika terjadi perubahan pada data yang dibutuhkan, *service* tersebut akan mengetahui perubahan data yang dipublish oleh *service* asal data [12].

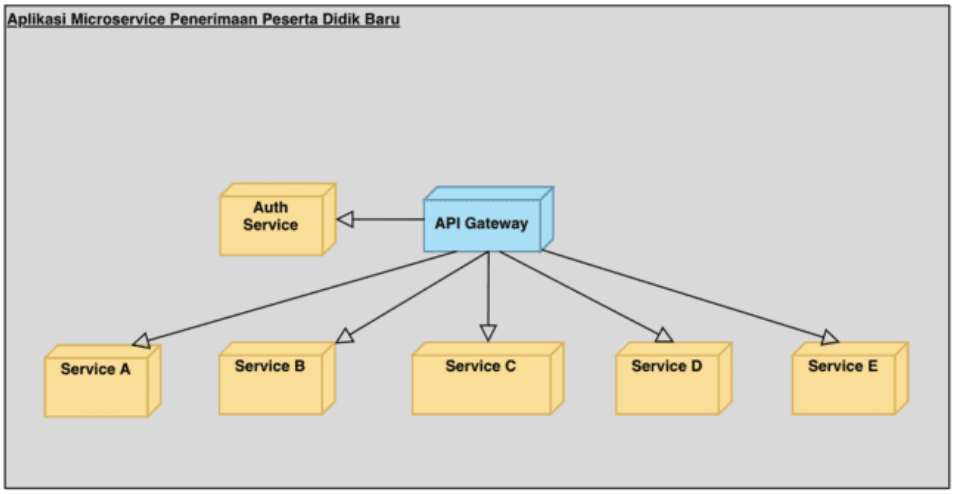


Gambar 3. Penyimpanan data menerapkan *database per service.*

Pada aplikasi *microservice* yang akan dibangun, diterapkan konsep penyimpanan data yang *decentralized*, dimana setiap *service* dapat mempunyai *database* masing-masing sesuai dengan kebutuhan *service* tersebut. Pola penyimpanan data seperti ini pada arsitektur *microservice* disebut dengan *database per service* [11]. Rancangan *database per service* dapat dilihat pada gambar 3.

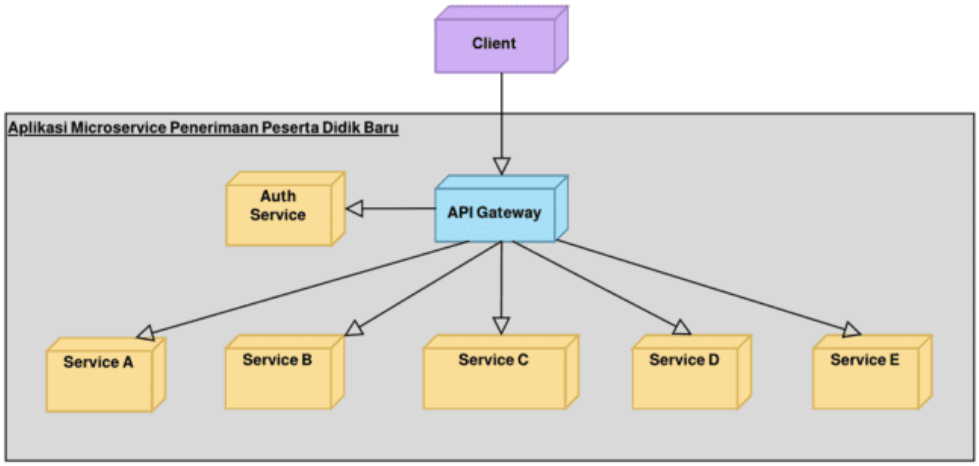
E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756



Gambar 4. Skema autentikasi pada *microservice.*

Authentication merupakan bagian yang sangat penting pada sebuah aplikasi. Pada aplikasi *microservice* yang akan dibangun, autentikasi diterapkan secara terpusat atau *centralized*, dimana setiap request yang masuk akan di autentikasi pada Auth Service. Sedangkan *authorization* akan dilakukan pada API Gateway, dengan cara mengecek *bearer authorization* pada request yang masuk. Skema dari autentikasi pada *microservice* dapat dilihat di gambar 4.



Gambar 4. Komunikasi *client-server* pada *microservice.*

Aplikasi *microservice* akan di bangun pada lingkungan yang *isolated* atau terisolasi. Maksudnya adalah semua *service* tidak dapat diakses langsung oleh *client*, akan tetapi setiap *request* dari *client* yang masuk harus melalui API Gateway, lalu API Gateway yang akan meneruskan *request* tersebut ke *service* yang ingin dituju [13]. Untuk tujuan tersebut, maka *service* yang diekspos dan dapat diakses oleh *client* hanyalah API Gateway. Komunikasi antar *client-server* dapat dilihat pada gambar 5.

2) Perancangan *Service*

Dalam tahap perancangan *service* akan dirumuskan gambaran umum dari setiap *service* yang dibutuhkan pada Aplikasi PPDB (Penerimaan Peserta Didik Baru). Di tahap ini akan dianalisa *service* apa saja yang diperlukan sehingga hasil finalnya adalah daftar *service* yang dibutuhkan pada aplikasi. Seluruh *service* tersebut berdiri sendiri sebagai sistem yang terpisah satu sama lain, namun memiliki fungsi yang saling mendukung, dengan setiap *service* dapat dibangun menggunakan bahasa pemrograman, *framework*, dan *runtime* aplikasi berbeda dengan *service* yang lainnya. Adapun perancangan setiap *service* sebagai berikut:

a) API Gateway adalah *service* yang berperan sebagai gerbang utama yang diakses oleh *client*, sehingga *client* tidak perlu terhubung ke setiap *service* yang ada. Cukup dengan mengakses API Gateway untuk dapat terhubung dengan *service* yang dibutuhkan. API Gateway akan dibangun menggunakan *framework* Krakend yaitu salah satu API Gateway *open source* yang dibangun dengan bahasa pemrograman Go *language* atau biasa disebut Golang.

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756

b) Auth Service adalah *service* yang berperan mengelola data akun, dan menangani *authorization*. *Service* ini akan dibagun menggunakan bahasa pemrograman PHP menggunakan *framework* Laravel yang dijalankan *runtime* PHP pada sistem operasi Ubuntu. *Service* ini juga memiliki database terpisah yang akan menggunakan MySQL sebagai DBMS.
c) Registration Service adalah *service* yang berperan untuk mengelola data pendaftaran. *Service* ini akan dibagun menggunakan bahasa pemrograman PHP menggunakan *framework* Laravel yang dijalankan *runtime* PHP pada sistem operasi Ubuntu. *Service* ini juga memiliki database terpisah yang akan menggunakan MySQL sebagai DBMS.
d) Registrant Service merupakan *service* yang berperan untuk mengelola data pendaftar. *Service* ini akan dibagun menggunakan bahasa pemrograman PHP menggunakan *framework* Laravel yang dijalankan *runtime* PHP pada sistem operasi Ubuntu. *Service* ini juga memiliki database terpisah yang akan menggunakan MySQL sebagai DBMS.
e) Announcement Service adalah *service* yang digunakan untuk mengelola data pengumuman. *Service* ini akan dibagun menggunakan bahasa pemrograman PHP menggunakan *framework* Laravel yang dijalankan *runtime* PHP pada sistem operasi Ubuntu. *Service* ini juga memiliki database terpisah yang akan menggunakan MySQL sebagai DBMS.
f) Institution Service adalah *service* yang berfungsi untuk mengelola data institusi pendidikan. *Service* ini akan dibagun menggunakan bahasa pemrograman PHP menggunakan *framework* Laravel yang dijalankan *runtime* PHP pada sistem operasi Ubuntu. *Service* ini juga memiliki database terpisah yang akan menggunakan MySQL sebagai DBMS.
g) Payment Service adalah *service* yang berfungsi untuk mengelola data pembayaran. *Service* ini akan dibagun menggunakan bahasa pemrograman Javascript menggunakan *framework* Fastify yang dijalankan *runtime* NodeJS pada sistem operasi Alpine Linux. *Service* ini juga memiliki database terpisah yang akan menggunakan PostgreSQL sebagai DBMS.
h) Mail Service merupakan *service* yang digunakan untuk kebutuhan pengiriman email. *Service* ini akan dibagun menggunakan bahasa pemrograman PHP menggunakan *framework* Laravel yang dijalankan *runtime* PHP pada sistem operasi Ubuntu. Service ini juga membutuhkan redis instance sebagai mail *queue driver*.
i) Redis *instance* yang digunakan sebagai *message broker* dan *queue driver*.

III. HASIL DAN PEMBAHASAN

A. Implementasi

Dalam tahap ini *service* sebagai komponen penyusun *microservice* diimplementasikan sebagai *docker images*. Penggunaan Docker sangat membantu dalam implementasi *microservice*, keseluruhan sistem yang sudah menjadi pecahan dari sistem utama dijadikan sebagai *docker images* yang kemudian dijalankan menjadi *docker container* dan memanfaatkan *docker network* untuk integrasi tiap *container* servicenya.

1) Implementasi *Microservice* Menggunakan Docker

Seluruh *service* yang telah dibuat yaitu antara lain API Gateway, Auth Service, Registration Service, Registrant Service, Announcement Service, Institution Service, Payment Service, Mail Service, dan Redis instance. Keseluruhan *service* tersebut kemudian di *build* sebagai *docker images*, dengan memenuhi poin-poin bahasa pemrograman, *framework*, *runtime*, dan sistem operasi sesuai dengan tahap perancangan sistem. Setiap *docker images* dapat saling bertukar data melalui *docker network* secara *asynchronous* menggunakan Redis Stream sebagai *message broker*. Sehingga terbentuk suatu kesatuan *service* berupa kumpulan *docker images* sebagai aplikasi *microservice* penerimaan peserta didik baru yang utuh yang menerapkan arsitektur *microservices*.

Berikut merupakan gambaran implementasi dari arsitektur *microservices* dalam aplikasi penerimaan peserta didik baru menggunakan Docker:

a) API Gateway

Sistem dibangun dengan menggunakan *framework* Krakend, yaitu sebuah *framework* API Gateway bersifat *open source* yang dibangun menggunakan bahasa Go. Docker *image* yang digunakan adalah

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756

devopsfaith/krakend:2.0.4. Konfigurasi *docker compose* API Gateway yang digunakan untuk mengimplementasikan sistem dapat dilihat pada gambar 5.

```yaml
api-gateway:
    image: devopsfaith/krakend:2.0.4
    volumes:
        - ./src/api-gateway:/etc/krakend
    networks:
        - sail
    ports:
        - "8080:8080"
    command: ["run", "-d", "-c", "/etc/krakend/krakend-config.json"]
    depends_on:
        - account-service
        - registration-service
        - registrant-service
        - announcement-service
        - mail-service
```

Gambar 5. Konfigurasi *Docker Compose* API Gateway.

b) Auth Service

Sistem dibangun menggunakan framework Laravel dengan bahasa pemrograman PHP. Konfigurasi *docker compose* Auth Service yang digunakan untuk mengimplementasikan sistem dapat dilihat pada gambar 6.

```yaml
account-service:
    build:
        context: ./
        dockerfile: Dockerfile
        args:
            WWWGROUP: '${WWWGROUP}'
    image: sail-8.1/app
    extra_hosts:
        - 'host.docker.internal:host-gateway'
    ports:
        - '${APP_PORT:-8006}:80'
    environment:
        WWWUSER: '${WWWUSER}'
        LARAVEL_SAIL: 1
        XDEBUG_MODE: '${SAIL_XDEBUG_MODE:-off}'
        XDEBUG_CONFIG: '${SAIL_XDEBUG_CONFIG:-client_host=host.docker.internal}'
    volumes:
        - './src/account-service:/var/www/html'
    networks:
        - sail
    depends_on:
        - account-db
        - redis
```

Gambar 6. Konfigurasi *Docker Compose* Auth Service.

c) Registration Service

Sistem dibangun menggunakan framework Laravel dengan bahasa pemrograman PHP. Konfigurasi *docker compose* Registration Service yang digunakan untuk mengimplementasikan sistem dapat dilihat pada gambar 7.

```yaml
registration-service:
    build:
        context: ./
        dockerfile: Dockerfile
        args:
            WWWGROUP: '${WWWGROUP}'
    image: sail-8.1/app
    extra_hosts:
        - 'host.docker.internal:host-gateway'
    ports:
        - '${APP_PORT:-8001}:80'
    environment:
        WWWUSER: '${WWWUSER}'
        LARAVEL_SAIL: 1
        XDEBUG_MODE: '${SAIL_XDEBUG_MODE:-off}'
        XDEBUG_CONFIG: '${SAIL_XDEBUG_CONFIG:-client_host=host.docker.internal}'
    volumes:
        - './src/registration-service:/var/www/html'
    networks:
        - sail
    depends_on:
        - registration-db
        - redis
```

Gambar 7. Konfigurasi *Docker Compose* Registration Service.

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756

d) Registrant Service

Sistem dibangun menggunakan framework Laravel dengan bahasa pemrograman PHP. Konfigurasi *docker compose* Registrant Service yang digunakan untuk mengimplementasikan sistem dapat dilihat pada gambar 8.

```
registrant-service:
    build:
        context: ./
        dockerfile: Dockerfile
        args:
            WWWGROUP: '${WWWGROUP}'
    image: sail-8.1/app
    extra_hosts:
        - 'host.docker.internal:host-gateway'
    ports:
        - '${APP_PORT:-8002}:80'
    environment:
        WWWUSER: '${WWWUSER}'
        LARAVEL_SAIL: 1
        XDEBUG_MODE: '${SAIL_XDEBUG_MODE:-off}'
        XDEBUG_CONFIG: '${SAIL_XDEBUG_CONFIG:-client_host=host.docker.internal}'
    volumes:
        - './src/registrant-service:/var/www/html'
    networks:
        - sail
    depends_on:
        - registrant-db
        - redis
```

Gambar 8. Konfigurasi *Docker Compose* Registrant Service.

e) Announcement Service

Sistem dibangun menggunakan framework Laravel dengan bahasa pemrograman PHP. Konfigurasi *docker compose* Announcement Service yang digunakan untuk mengimplementasikan sistem dapat dilihat pada gambar 9.

```
announcement-service:
    build:
        context: ./
        dockerfile: Dockerfile
        args:
            WWWGROUP: '${WWWGROUP}'
    image: sail-8.1/app
    extra_hosts:
        - 'host.docker.internal:host-gateway'
    ports:
        - '${APP_PORT:-8003}:80'
    environment:
        WWWUSER: '${WWWUSER}'
        LARAVEL_SAIL: 1
        XDEBUG_MODE: '${SAIL_XDEBUG_MODE:-off}'
        XDEBUG_CONFIG: '${SAIL_XDEBUG_CONFIG:-client_host=host.docker.internal}'
    volumes:
        - './src/announcement-service:/var/www/html'
    networks:
        - sail
    depends_on:
        - announcement-db
        - redis
```

Gambar 9. Konfigurasi *Docker Compose* Announcement Service.

f) Institution Service

Sistem dibangun menggunakan framework Laravel dengan bahasa pemrograman PHP. Konfigurasi *docker compose* Institution Service yang digunakan untuk mengimplementasikan sistem dapat dilihat pada gambar 10.

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756

```
institution-service:
    build:
        context: ./
        dockerfile: Dockerfile
        args:
            WWWGROUP: '${WWWGROUP}'
    image: sail-8.1/app
    extra_hosts:
        - 'host.docker.internal:host-gateway'
    ports:
        - '${APP_PORT:-8005}:80'
    environment:
        WWWUSER: '${WWWUSER}'
        LARAVEL_SAIL: 1
        XDEBUG_MODE: '${SAIL_XDEBUG_MODE:-off}'
        XDEBUG_CONFIG: '${SAIL_XDEBUG_CONFIG:-client_host=host.docker.internal}'
    volumes:
        - './src/institution-service:/var/www/html'
    networks:
        - sail
    depends_on:
        - institution-db
        - redis
```

Gambar 10. Konfigurasi *Docker Compose* Institution Service

g) Payment Service

Sistem dibangun menggunakan framework Fastify dengan bahasa pemrograman Javascript. Konfigurasi *docker compose* Payment Service yang digunakan untuk mengimplementasikan sistem dapat dilihat pada gambar 11.

```
payment-service:
    build:
        context: ./src/payment-service
        dockerfile: Dockerfile
    ports:
        - '${APP_PORT:-8007}:3353'
    networks:
        - sail
    depends_on:
        - payment-db
        - redis
```

Gambar 11. Konfigurasi *Docker Compose* Payment Service.

h) Mail Service

Sistem dibangun menggunakan framework Laravel dengan bahasa pemrograman PHP. Konfigurasi *docker compose* Mail Service yang digunakan untuk mengimplementasikan sistem dapat dilihat pada gambar 12.

```
mail-service:
    build:
        context: ./
        dockerfile: Dockerfile
        args:
            WWWGROUP: '${WWWGROUP}'
    image: sail-8.1/app
    extra_hosts:
        - 'host.docker.internal:host-gateway'
    ports:
        - '${APP_PORT:-8004}:80'
    environment:
        WWWUSER: '${WWWUSER}'
        LARAVEL_SAIL: 1
        XDEBUG_MODE: '${SAIL_XDEBUG_MODE:-off}'
        XDEBUG_CONFIG: '${SAIL_XDEBUG_CONFIG:-client_host=host.docker.internal}'
    volumes:
        - './src/mail-service:/var/www/html'
    networks:
        - sail
    depends_on:
        - redis
```

Gambar 12. Konfigurasi *Docker Compose* Mail Service.

i) Redis Instance

Redis instance adalah sebuah *service* pendukung dalam aplikasi *microservice* yang dibangun. Redis berperan sebagai *message broker*, yaitu sarana pertukaran data antar *service* dengan menerapkan Redis Stream. Selain itu Redis juga berfungsi sebagai *queue driver* yang dipakai dalam menyimpan *job* atau *process* yang membutuhkan antrian, contohnya *process* pengiriman email pada Mail Service. Konfigurasi *docker compose* Redis yang digunakan untuk mengimplementasikan sistem dapat dilihat pada gambar 13.

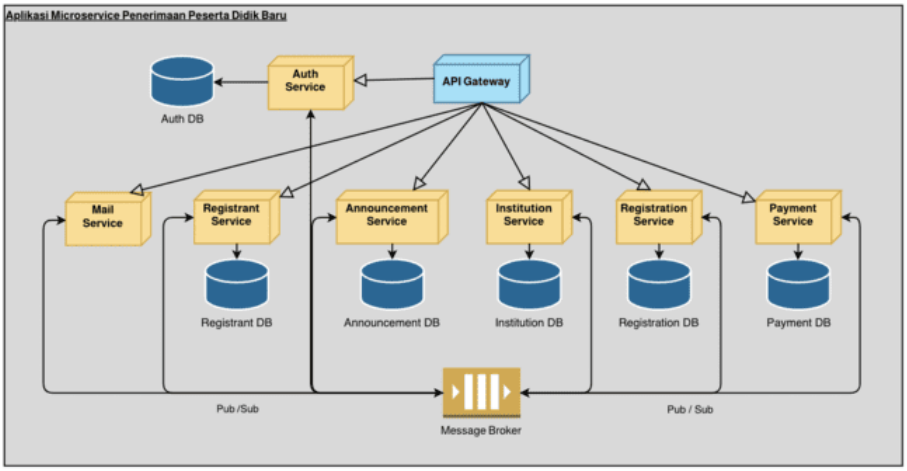
E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756

```
redis:
    image: 'redis:alpine'
    ports:
        - '${FORWARD_REDIS_PORT:-6379}:6379'
    volumes:
        - 'sail-redis:/data'
    networks:
        - sail
    healthcheck:
        test: ["CMD", "redis-cli", "ping"]
        retries: 3
        timeout: 5s
```

Gambar 13. Konfigurasi *Docker Compose* Redis.

Selanjutnya *service-service* yang telah didefinisikan pada *docker compose* akan dijalankan pada *docker container* masing-masing. Dengan demikian terbentuk suatu kesatuan aplikasi berupa *service* yang berjalan sebagai Aplikasi *Microservice* Layanan Penerimaan Peserta Didik Baru. Implementasi dari arsitektur *microservice* pada Aplikasi Layanan Peserta Didik Baru dapat dilihat pada gambar 14.



Gambar 14. Implementasi Arsitektur *Event-Driven Microservice* menggunakan *Docker*.

2) Perangkat Implementasi
   a) Perangkat Keras
   Dalam tahapan implementasi, perangkat keras yang digunakan adalah:
   Laptop
   Processor : Apple M1 chip 8-Core CPU
   Memory : 8GB Unified Memory
   Disk Storage : 512GB SSD
   b) Perangkat Lunak
   Perangkat lunak yang digunakan dalam tahap implementasi sistem terdiri dari:
   Sistem Operasi : *MacOS 12.2.1 Monterey*
   Docker : *Docker Engine 20.10.5*
   Web Server : *Apache, NodeJS*
   Web Browser : *Google Chrome*
   Bahasa Pemrograman: *PHP, Javascript, Go*
   Tools : *Docker Desktop, Postman, Visual Studio Code*
   DBMS : *MySQL, PostgreSQL, Redis*

B.Pengujian

Pengujian yang akan dilakukan pada penelitian ini menggunakan *Software Architecture Analysis Method* (SAAM). SAAM adalah sebuah metode yang dipakai dalam *software* arsitektur untuk mengevaluasi sebuah arsitektur sistem berdasarkan Quality Attribute yang telah ditentukan [14]. Metode ini berfokus pada penggunaan arsitektur yang signifikan, sehingga masalah yang ditemukan di awal siklus hidup pengembangan perangkat lunak lebih mudah untuk diperbaiki. Adapun langkah-langkah pengujian sebagai berikut [15]:

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756

1) Menentukan skenario (Quality Attribute)
Skenario adalah urutan langkah-langkah tertentu yang melibatkan modifikasi sistem. Dalam penelitian ini skenario dibagi menjadi empat sesuai dengan quality attribute yaitu *maintainability*, *reusability*, *scalability*, dan *availibility.* Dimana skenario tersebut diperoleh dari fokus utama penelitian yang tertuang pada pendahuluan, antara lain yaitu:
   a) Maintainability: Terjadi perubahan pada proses bisnis yang mengakibatkan perubahan pada alur PPDB.
   b) Maintainability: Terjadi penggantian payment gateway.
   c) Reusability: Sistem yang sama digunakan untuk beberapa institusi pendidikan.
   d) Scalability: Dibutuhkan ujian masuk berbasis komputer pada proses PPDB.
   e) Availibility: Terjadi kegagalan pada sistem akibat traffic yang terlalu tinggi pada proses pendaftaran sehingga mengakibatkan server down.
2) Menentukan kandidat arsitektur
Kandidat arsitektur yang akan dibandingkan pada penelitian ini adalah arsitektur *microservice* dan arsitektur *monolith*, seperti yang telah diuraikan pada bab pendahuluan.
3) Mengklasifikasi skenario
Selanjutnya akan diklasifikasikan skenario yang sesuai berdasarkan Quality Attribute yang telah ditentutkan tadi. Terdapat dua tipe skenario *direct* dan *indirect. Direct* adalah skenario yang dapat di eksekusi di dalam sistem tanpa modifikasi sistem. Sedangakan *indirect* adalah skenario yang membutuhkan modifikasi pada sistem itu sendiri. Pengklasifikasian skenario dan hasilnya dapat dilihat pada tabel 1:

TABEL I
MENGKLASIFIKASI SKENARIO

No.	Quality Attribute	Skenario	Type	
			Microservice	Monolith
1.	Maintainability	Terjadi perubahan pada proses bisnis yang mengakibatkan perubahan pada alur PPDB.	Direct	Indirect
2.	Maintainability	Terjadi penggantian payment gateway.	Indirect	Indirect
3.	Reusability	Sistem yang sama digunakan untuk beberapa institusi pendidikan.	Direct	Indirect
4.	Scalability	Dibutuhkan ujian masuk berbasis komputer pada proses PPDB.	Indirect	Indirect
5.	Availibility	Terjadi kegagalan pada sistem akibat traffic yang terlalu tinggi pada proses pendaftaran sehingga mengakibatkan server down.	Indirect	Indirect

4) Melakukan evaluasi skenario
Skenario yang telah diklasifikasi tadi kemudian dievaluasi dengan membuat hipotesis terhadap arsitektur yang dibandingkan. Pada setiap skenario *indirect* akan diidentifikasi komponen, koneksi data, koneksi kontrol, dan *interface* yang perlu ditambah, dihapus, ataupun diubah, kemudian akan diperkirakan tingkat kesulitan dari modifikasi tersebut. Tingkat kesulitan diperoleh dari *system architect* dan didasarkan pada jumlah komponen yang akan dimodifikasi and memiliki efek terhadap modifikasi. Evaluasi skenario dan hasilnya dapat dilihat pada tabel 2:

TABEL II
MELAKUKAN EVALUASI SKENARIO

No	**Quality Attribute**	**Skenario**	**Type**		**Modification**	
			Microservice	Monolith	Microservice	Monolith
1.	Maintainability	Terjadi perubahan pada proses bisnis yang mengakibatkan perubahan pada alur PPDB.	Direct	Indirect	-	Backend, Frontend, Database
2.	Maintainability	Terjadi penggantian *payment gateway*.	Indirect	Indirect	Payment Service	Backend
3.	Reusability	Sistem yang sama digunakan untuk beberapa institusi pendidikan.	Direct	Indirect	-	Backend, Frontend
4.	Scalability	Dibutuhkan ujian masuk berbasis komputer pada proses PPDB.	Indirect	Indirect	Registration Service, Registration Database, Client	Backend, Frontend, Database
5.	Availability	Terjadi kegagalan pada sistem akibat *traffic* yang terlalu tinggi pada proses pendaftaran sehingga mengakibatkan server down.	Indirect	Indirect	Registration Service	Backend

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 745–756

5) Mengamati interaksi skenario
Saat beberapa skenario *indirect* berefek pada komponen yang sama, ini bisa mengindikasikan sebuah masalah apabila skenario bukan merupakan variasi skenario-skenario lainnya. Contohnya pada sebuah komponen diperlukan modifikasi untuk beberapa skenario; ubah warna background menjadi putih, ubah port sistem ke platform yang berbeda. Pada dua modifikasi tersebut, skenario yang berefek pada komponen merupakan hal yang jauh berbeda, hal ini menandakan kemungkinan kurangnya *separation of concern.*
6) Hasil evaluasi keseluruhan
Skenario yang telah dievaluasi kemudian dibandingkan dan dan diberi skala prioritas dengan bobot tertentu berdasarkan kepentingan dalam proses bisnis utama aplikasi. Hasil dari evaluasi keseluruhan berupa *modification cost/effort*, yaitu estimasi biaya atau usaha yang dibutuhkan untuk menjalankan seluruh skenario. Semakin besar angka yang didapat, semakin besar pula usaha atau biaya yang dibutuhkan.

TABEL III
HASIL EVALUASI

No	Skenario		Modification *Cost / Effort*	
	Deskripsi	Bobot	Microservice	Monolith
1.	Terjadi perubahan pada proses bisnis yang mengakibatkan perubahan pada alur PPDB.	30%	-	3/3
2.	Terjadi penggantian payment gateway.	10%	1/16	1/3
3.	Sistem yang sama digunakan untuk beberapa institusi pendidikan.	25%	-	2/3
4.	Dibutuhkan ujian masuk berbasis komputer pada proses PPDB.	25%	3/16	3/3
5.	Terjadi kegagalan pada sistem akibat traffic yang terlalu tinggi pada proses pendaftaran sehingga mengakibatkan server down.	10%	1/16	1/3
	Overall		5,94	78,335

Tabel 3 merupakan hasil evaluasi keseluruhan dari skenario-skenario yang telah didefinisikan pada tahap awal. Arsitektur *microservice* mendapatkan poin keseluruhan 5,94 dan arsitektur *monolith* mendapatkan poin 78,335. Poin tersebut didapat berdasarkan total dari jumlah komponen yang perlu dimodifikasi pada tahap evaluasi skenario dikalikan dengan bobot, dimana bobot merupakan nilai kepentingan skenario pada proses bisnis utama yang ingin dicapai. Berdarkan skenario yang telah dievaluasi, arsitektur *microservice* bisa dikatakan lebih cocok diimplementasikan pada pengembangan Aplikasi Layanan Penerimaan Peserta Didik Baru.

IV. KESIMPULAN

Kesimpulan dari jurnal dan pembahasan kali ini ialah implementasi teknologi *event-driven microservice* dengan *choreography pattern* telah berhasil diterapkan pada aplikasi penerimaan peserta didik baru dengan *service* yang telah dibuat yaitu antara lain API Gateway, Auth Service, Registration Service, Registrant Service, Announcement Service, Institution Service, Payment Service, Mail Service, dan Redis instance. Selanjutnya pengujian menggunakan *Software Architecture Analysis Method (SAAM)* antara lain menentukan skenario, menentukan kandidat arsitektur, mengklasifikasi skenario, melakukan evaluasi skenario, mengamati interaksi skenario, hingga memiliki hasil evaluasi dengan nilai *modification cost* pada arsitektur *microservice* sebesar 5,94 sedangkan arsitektur *monolith* sebesar 78,335. Setelah membandingkan hasil evaluasi berdasarkan sifat *maintainability, reusability, scalability,* dan *availibility* diantara arsitektur *monolith* dengan *microservices*, maka dapat disimpulkan *microservices* lebih unggul dalam penerapannya dibandingkan dengan *monolith*.

DAFTAR PUSTAKA

[1] Google Scholar, “Hasil pencarian untuk ‘aplikasi penerimaan peserta didik baru’ - Google Scholar,” 2021. https://scholar.google.com/scholar?hl=id&as_sdt=0,5&q=aplikasi+penerimaan+peserta+didik+baru (accessed Apr. 15, 2021).
[2] M. Saufi, “Sistem Informasi Penerimaan Peserta Didik Baru (PPDB) Berbasis Web Menggunakan PHP dan MySQL di SMK Nasional Berbah,” vol. 2. pp. 227–249, 2018.
[3] U. Saly Tiara, “Implementasi Sistem Penerimaan Siswa Baru Online Pada SMP Negeri 53 Palembang,” 2013.
[4] L. Khoirunnisa, “Rancang BANGUN SISTEM E-LEARNING BERBASIS MICROSERVICES DAN DOMAIN DRIVEN DESIGN (STUDI KASUS PROBISTEK UIN MAULANA MALIK IBRAHIM MALANG),” 2019.
[5] D. Gonzalez, *Developing Microservices with Node.js*. Mumbai: Packt Publishing, 2016.
[6] M. Rezaldy, I. Asror, and I. L. Sardi, “Desain dan Analisis Arsitektur Microservices Pada Sistem Informasi Akademik Perguruan Tinggi Dengan Pendekatan Architecture Tradeoff Analysis Method (ATAM) (Studi Kasus : iGracias Universitas Telkom),” vol. 4, no. 2, 2017.
[7] H. N. A. Panjaitan, “Pengalokasian Resource Beberapa Conainer Pada Proxmox Virtual Environment Sebagai Server Cloud Computing,” *Skripsi*

Univ. Sumatera Utara, 2018.
[8] R. Jakaria, "Implementasi Cloud Computing Menggunakan Proxmox Virtual Environment Di Universitas Nurtanio Bandung," 2013.
[9] M. Fihri, R. M. Negara, and D. D. Sanjoyo, "Implementasi & Analisis Performansi Layanan Web Pada Platform Berbasis Docker Implementation & Analysis of Web Service Performance Based on Docker Platform," vol. 6, no. 2, pp. 3996–4001, 2019, [Online]. Available: https://libraryeproceeding.telkomuniversity.ac.id/index.php/engineering/article/viewFile/10367/10222
[10] M. F. Romadlon Bik and Asmunin, "Implementasi Docker Untuk Pengelolaan Banyak Aplikasi Web (Studi Kasus : Jurusan Teknik Informatika Unesa)," *J. Manaj. Inform.*, vol. 7, no. 2, pp. 46–50, 2017.
[11] C. Richards, *Microservices Patterns*, vol. 2018, no. March. 2018.
[12] S. Tallberg, "a Comparison of Data Real-Time Stream Processing Pipelines," 2020.
[13] R. S. Saputra, I. R. Munadi, and D. D. Sanjoyo, "Implementasi Dan Analisis Performansi Platform As a Service Untuk Api Gateway Menggunakan Kong," vol. 5, no. 3, pp. 4973–4979, 2018, [Online]. Available: https://libraryeproceeding.telkomuniversity.ac.id/index.php/engineering/article/viewFile/7883/7776
[14] L. Dobrica and E. Niemelá, "A survey on software architecture analysis methods," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 638–653, 2002, doi: 10.1109/TSE.2002.1019479.
[15] M. T. Ionita, D. K. Hammer, and H. Obbink, "Scenario-based software architecture evaluation methods: An overview," *Work. Methods Tech. Softw. Archit. Rev. Assess. Int. Conf. Softw. Eng.*, pp. 1–12, 2002, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.8382&rep=rep1&type=pdf