E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

IMPLEMENTASI *SERVICE CHOREOGRAPHY PATTERN* ARSITEKTUR *MICROSERVICE CLASSROOM* AKADEMIK MENGGUNAKAN DOCKER

Dhymas J. Riyanto[1)], Pizaini[2)], Nazruddin Safaat H.[3)], Muhammad Affandes[4)]
[1, 2, 3, 4)] Program Studi Teknik Informatika, Universitas Islam Negeri Sultan Syarif Kasim
Pekanbaru, Riau, Indonesia
e-mail: dhymas.julyan.riyanto@students.uin-suska.ac.id[1)], pizaini@uin-suska.ac.id[2)], nazruddin.safaat@uin-suska.ac.id[3)], affandes@uin-suska.ac.id[4)]

ABSTRAK

Pengembangan aplikasi classroom akademik mewujudkan proses bisnis inti pada lembaga pendidikan secara umum masih menggunakan arsitektur monolitik. Arsitektur microservice hadir sebagai pola pengembangan aplikasi dimana keseluruhan fungsi perangkat lunak disediakan oleh komponen-komponen layanan atau service aplikasi yang lebih kecil. Service-service tersebut akan berkomunikasi melalui komunikasi berbasis event-driven menggunakan service choreography pattern dimana pertukaran data terjadi secara asynchronous melewati message broker. Namun dalam pengembangan arsitektur microservices, masing-masing service memiliki dependensi dan environment yang berbeda. Docker merupakan teknologi perangkat lunak yang berfungsi sebagai wadah untuk membungkus dan memasukkan aplikasi menggunakan teknik kontainerisasi yang mengisolasi masing-masing service dan juga environment-nya. Teknik kontainerisasi seperti ini mampu membantu mewujudkan penerapan arsitektur microservices menggunakan service choreography pattern terhadap aplikasi classroom akademik dinamis yang digunakan tidak hanya oleh satu lembaga pendidikan. Penelitian ini memiliki tujuan untuk mengimplementasikan Service Choreography Pattern Arsitektur Microservice Classroom Akademik Menggunakan Docker, dimana telah dilakukan pengujian benchmarking memakai tools Wrk Bench dengan membuat 400 HTTP connections menggunakan 6 threads CPU dan dijalankan selama 15 detik, yang kemudian divalidasi dan dievaluasi melalui beberapa parameter, seperti waktu tanggapan atau latency (ms), jumlah HTTP request/seconds (rps) dan kecepatan transfer/seconds (Mbps). Hasil penelitian menunjukkan jumlah request HTTP yang dapat diterima melebihi 40000 permintaan, menunjukkan betapa efektifnya penggunaan service choreography pattern yang diimplementasikan pada classroom akademik. Serta dengan adanya pembatasan request HTTP pada auth, menjadikan aplikasi menjadi lebih aman ketika menggunakan rate limit dari permintaan yang terlalu banyak.

***Kata Kunci**: Akademik, Choreography, Classroom, Docker, Microservice*

ABSTRACT

Development of academic class applications to realize the core business processes in educational institutions in general still uses a monolithic architecture. Microservices architecture exists as an application development pattern in which all software functions are provided by service components or smaller application services. These services will communicate via event-driven communication using a service choreography pattern in which data exchange occurs asynchronously via message broker. However, in microservices architecture development, each service has different dependencies and environments. Docker is a software technology that functions as a container to package and load applications using containerization techniques that isolate each service and its environment. This containerization technique is able to help realize the application of microservices architecture using service choreography patterns in dynamic academic classroom applications that are used not only by one educational institution. This research aims to implement the Service Choreography Pattern of Academic Classroom Microservice Architecture using Docker, where benchmarking tests have been carried out using the Wrk Bench tools by making 400 HTTP connections using 6 CPU threads and running for 15 seconds, which are then validated and evaluated through several parameters, such as response time or latency (ms), number of HTTP requests/second (rps) and transfer rate/second (Mbps). The results show that number of HTTP requests that can be received exceeds 40000 requests, indicating how effective use of service choreography patterns is in academic classroom. And with the limitations of HTTP requests on auth, making applications more secure when using rate limits of many requests.

***Keywords**: Academic, Choreography, Classroom, Docker, Microservice*

I. PENDAHULUAN

Proses bisnis inti dalam akademik terdiri dari beberapa proses yang saling ketergantungan satu sama lain, meliputi registrasi mahasiswa baru maupun mahasiswa lama, proses perkuliahan, perhitungan nilai sampai proses kelulusan mahasiswa [1]. Penggunaan teknologi seperti aplikasi *classroom* merupakan pilihan tepat untuk dapat mendukung lembaga di bidang akademik pada jenjang pendidikan apa pun dalam menjalankan proses bisnis inti tersebut [2]. Pengembangan aplikasi *classroom* akademik telah banyak dilakukan untuk membantu

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

lembaga akademik dalam menyelesaikan proses bisnis yang ada. Namun pengembangan tersebut secara umum masih menggunakan arsitektur yang bersifat monolitik. Arsitektur monolitik sendiri merupakan rancangan arsitektur sebuah aplikasi yang menjalankan semua logika dalam satu *server* aplikasi [3]. Hal tersebut mengakibatkan kemampuan adaptasi terhadap perubahan kebutuhan sistem (*requirement changes*) terutama dalam pengelolaan kompleksitas kode dan *maintainability* semakin sulit, sehingga ketika mengubah atau menambah teknologi yang dibutuhkan, maka harus mengubah isi aplikasi yang telah dibangun secara keseluruhan [4].

Arsitektur *microservices* hadir sebagai pola pengembangan aplikasi dimana keseluruhan fungsi perangkat lunak disediakan oleh komponen-komponen layanan atau *service* aplikasi yang lebih kecil [5]. *Service-service* tersebut akan berjalan sesuai dengan fungsinya masing-masing dan saling berkomunikasi satu sama lain sehingga menjadi kesatuan sistem yang besar. *Service-service* tersebut akan berkomunikasi menggunakan komunikasi berbasis *event-driven*, dimana pertukaran data akan terjadi secara *asynchronous.* Pada penerapannya, seluruh data yang akan dikirimkan ke *service* lain harus terlebih dahulu melewati *message broker* [6]. Peran *message broker* disini akan diisi oleh Redis Streams, sebuah tipe data baru dari platform Redis versi 5.0, dimana setiap *service* dapat membagikan data dengan melakukan *produce event* ke dalam Redis Streams, begitu pula dengan *service* lain akan melakukan *consume event* ketika membutuhkan data yang sudah di *produce* ke dalam Redis Streams. Pola seperti ini disebut dengan *service choreography pattern*, dimana proses komunikasi antar *service* berjalan secara paralel dan menggunakan logika bisnis yang *decentralized*, sehingga logika bisnis akan terdistribusi di masing-masing *service* tanpa harus membebani seluruh logika bisnis yang ada terhadap satu *service* saja [7]**.**

Dalam pengembangan arsitektur *microservices*, masing-masing *service* memiliki dependensi dan *environment* yang berbeda sehingga membuat proses *deployment* pada arsitektur *microservice* harus terisolasi antar satu service dengan yang lain, serta dibutuhkan kemudahan pada proses *deployment* aplikasi juga *software* pendukung seperti *web server*, *database server*, dependensi dan *environment* lain ke *server* [8]. Metode *deployment* yang umum digunakan saat ini ialah dengan melakukan instalasi aplikasi dan *environment* yang dibutuhkan ke dalam satu *server*. Metode ini tergolong mudah dan cepat dalam proses *deployment*, namun setiap aplikasi tidak terisolasi, sehingga apabila melakukan *deploy* beberapa aplikasi yang memiliki ketergantungan dengan *package* versi tertentu dapat menimbulkan konflik dependensi. Metode umum selanjutnya adalah dengan men-*deploy* aplikasi dan dependensi yang dibutuhkan ke dalam *virtual machine* (VM) yang berbeda. Metode ini akan meningkatkan *scalability* karena setiap aplikasi berjalan pada *resource* yang berbeda. Namun hal itu pula yang menyebabkan kebutuhan akan *resource* membengkak dikarenakan setiap VM menjalankan OS dan *kernel* masing-masing [9].

Teknologi yang mampu menjawab permasalahan tersebut pada saat ini adalah dengan menggunakan Docker. Docker merupakan teknologi perangkat lunak yang berfungsi sebagai wadah untuk membungkus dan memasukkan aplikasi ke dalam sebuah *environment* beserta dependensinya [10]. Docker di *deploy* dengan menggunakan teknik kontainerisasi (virtualisasi berbasis *container*) yang mengisolasi masing-masing *service* dan juga *environment*-nya, sehingga proses *deployment* yang awalnya harus menggunakan *resource* begitu besar, tergantikan dengan penggunaan *resource* yang kecil [11]. Teknik kontainerisasi seperti ini mampu membantu mewujudkan penerapan arsitektur *microservices* menggunakan *service choreography pattern* terhadap aplikasi *classroom* akademik dinamis yang dapat digunakan tidak hanya oleh satu lembaga pendidikan. Teknik ini juga dapat memudahkan proses pengembangan lanjutan, pengujian, perbaikan dan *deployment* aplikasi. Berdasarkan permasalahan tersebut, penelitian ini memiliki tujuan untuk mengimplementasikan *Service Choreography Pattern* Arsitektur *Microservice Classroom* Akademik Menggunakan Docker.

II. METODE PENELITIAN

Metode penelitian merupakan rangkaian tahapan yang disusun secara sistematis dan dijadikan pedoman pelaksanaan penelitian untuk mencapai tujuan dari penelitian. Penelitian yang dilakukan ini terdiri dari beberapa tahapan, yaitu: analisis dan skenario, perancangan, implementasi, pengujian serta kesimpulan dan saran.

A. Analisis dan Skenario

Analisis merupakan tahapan untuk mengetahui *service* apa yang diperlukan dalam membangun arsitektur *microservice* pada aplikasi *classroom* akademik. *Service-service* ini terisolasi sehingga fokus dengan tugas-tugas yang bersifat ringan dan saling berkomunikasi satu sama lain untuk bekerja sama dalam pengembangan aplikasi *classroom* akademik. Berikut merupakan gambaran umum *services* yang akan dibuat beserta dengan penjelasan dari masing-masing fungsinya, yaitu:

1) API Gateway

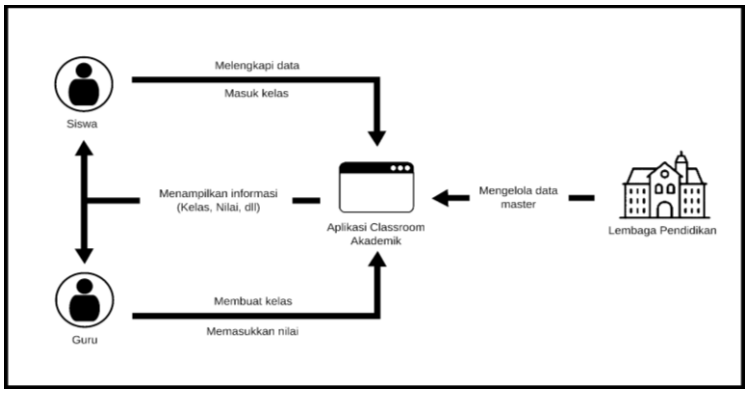
API *gateway* bertindak sebagai *single entry point* pada aplikasi *microservice* yang dibangun. API *gateway* juga bertindak sebagai *middleware* untuk melakukan enkapsulasi pada *services* di belakangnya.

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

2) Account Service
Account service merupakan *service* yang berfungsi untuk mengotentikasi dan mengelola data pengguna aplikasi.
3) Teacher Service
Teacher service merupakan *service* untuk mengelola data guru/dosen.
4) Student Service
Student service merupakan *service* untuk mengelola data siswa/mahasiswa.
5) Course Service
Course service merupakan *service* untuk mengelola data mata pelajaran/mata kuliah.
6) Classroom Service
Classroom service merupakan *service* untuk mengelola data kelas, tugas, dan nilai.
7) Database Service
Database service merupakan *service* penyedia layanan *database*, yakni sebagai wadah penyimpanan data.
8) Message Broker
Message broker merupakan aplikasi tambahan yang bertindak sebagai perantara dalam pengiriman pesan/data.

Skenario merupakan gambaran alur aplikasi yang akan dibangun, dimana aplikasi *classroom* akademik akan memerankan proses bisnis inti akademik itu sendiri sehingga mampu digunakan di tiap lembaga pendidikan. Berikut merupakan gambaran skenario umum dan juga merupakan proses bisnis inti pada aplikasi yang dibangun:



Gambar 1 Skenario Alur Aplikasi *Classroom* Akademik

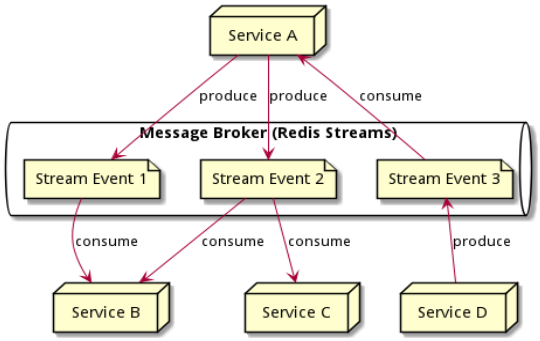
Dari gambar 1, dapat dilihat hasil yang ingin dicapai adalah:
1) *Civitas academica* dapat mendapat informasi yang dibutuhkan terkait dengan kegiatan belajar mengajar.
2) Proses bisnis inti pada seluruh lembaga pendidikan dapat terpenuhi.

B. Perancangan

Perancangan pada penelitian ini terbagi atas dua tahapan, yakni sebagai berikut:

1) Perancangan Arsitektur

Pada tahap ini dilakukan perancangan arsitektur sesuai dengan kebutuhan dari aplikasi *microservice classroom* akademik. Arsitektur *microservice* akan dibangun menggunakan *service choreography pattern*. Pada *pattern* ini, logika bisnis akan terdistribusi di tiap *service* tanpa harus membebani seluruh logika bisnis aplikasi terhadap satu *service* saja, sehingga masing-masing *service* akan memiliki logika bisnisnya sendiri [12]. Selanjutnya dalam *service choreography pattern*, metode komunikasi yang digunakan antar *service* menggunakan teknik *messaging*. Teknik ini memungkinkan proses pengiriman pesan akan berjalan secara *asynchronous* sehingga komunikasi yang dilakukan tidak memiliki waktu tunggu [13].



E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

Gambar 2 Metode Komunikasi Menggunakan Teknik *Messaging* Antar *Service* pada *Service Choreography Pattern*

Dari gambar 2, metode komunikasi dengan teknik *messaging* dalam penelitian ini akan menggunakan *streams* sebagai jembatan untuk mengirim dan menerima data. Setiap *service* dapat membagikan data dengan memproduksi *event* (*produce*) ke dalam *streams*, begitu pula dengan *service* lain akan mengonsumsi *event* (*consume*) ketika membutuhkan data yang sudah di *produce* ke dalam *streams* tersebut. *Streams* ini akan dikelompokkan menjadi beberapa *consumer group* sesuai dengan jenis kebutuhan datanya masing-masing. *Consumer group* ini dikelola di dalam sebuah aplikasi yang disebut sebagai *message broker* (perantara) [14].

2) Perancangan Services

Tahapan ini merupakan penentuan hasil dari analisis kebutuhan *service* yang telah ditentukan sebelumnya. Pada tahapan ini akan ditentukan teknologi apa saja yang akan digunakan, mulai dari bahasa pemrograman, *framework*, teknologi atau platform, *runtime* hingga OS pada masing-masing *service* tersebut. Berikut merupakan perancangan setiap *service* yang akan dibangun:

a) API *Gateway*

API Gateway

Bahasa Pemrograman	JavaScript
Framework	Express
Runtime	NodeJS
Sistem Operasi	Alpine Linux

Gambar 3 Perancangan API *Gateway*

Dapat dilihat dari gambar 3, API *Gateway* dibangun menggunakan bahasa pemrograman JavaScript menggunakan *framework* Express yang dijalankan pada *runtime* Node JS pada sistem operasi Alpine Linux.

b) Account Service

Account Service

Bahasa Pemrograman	PHP
Framework	Laravel
Runtime	Laravel Octane
Sistem Operasi	Alpine Linux

Gambar 4 Perancangan *Account Service*

Dapat dilihat dari gambar 4, *Account service* dibangun menggunakan bahasa pemrograman PHP menggunakan *framework* Laravel yang dijalankan pada *runtime* Laravel Octane pada sistem operasi Alpine Linux.

c) Teacher Service

Teacher Service

Bahasa Pemrograman	JavaScript
Framework	Express
Runtime	NodeJS
Sistem Operasi	Alpine Linux

Gambar 5 Perancangan *Teacher Service*

Dapat dilihat dari gambar 5, *Teacher service* dibangun menggunakan bahasa pemrograman JavaScript menggunakan *framework* Express yang dijalankan pada *runtime* Node JS pada sistem operasi Alpine Linux.

d) Student Service

Student Service

Bahasa Pemrograman	JavaScript
Framework	Express
Runtime	NodeJS
Sistem Operasi	Alpine Linux

Gambar 6 Perancangan *Account Service*

Dapat dilihat dari gambar 6, *Student service* dibangun menggunakan bahasa pemrograman JavaScript menggunakan *framework* Express yang dijalankan pada *runtime* Node JS pada sistem operasi Alpine Linux.

e) Course Service

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

Course Service	
Bahasa Pemrograman	JavaScript
Framework	Express
Runtime	NodeJS
Sistem Operasi	Alpine Linux

Gambar 7 Perancangan *Course Service*

Dapat dilihat dari gambar 7, *Course service* dibangun menggunakan bahasa pemrograman JavaScript menggunakan *framework* Express yang dijalankan pada *runtime* Node JS pada sistem operasi Alpine Linux.

f) Classroom Service

Classroom Service	
Bahasa Pemrograman	Java
Framework	Spring Boot
Runtime	JRE
Sistem Operasi	Alpine Linux

Gambar 8 Perancangan *Classroom Service*

Dapat dilihat dari gambar 8, *Classroom service* dibangun menggunakan bahasa pemrograman Java menggunakan framework Spring Boot yang dijalankan pada *runtime* JRE pada sistem operasi Alpine Linux.

g) Database Service

Database Service	
Database	PostgreSQL

Gambar 9 Perancangan *Database Service*

Dapat dilihat dari gambar 9, *Database service* menggunakan PostgreSQL sebagai tempat penyimpanan data.

h) Message Broker

Message Broker	
Platform	Redis
Fitur	Redis Streams

Gambar 10 Perancangan *Message Broker*

Dapat dilihat dari gambar 10, *Message broker* yang digunakan pada penelitian ini adalah Redis Streams yang merupakan tipe data baru dari platform Redis versi 5.0.

III. HASIL DAN PEMBAHASAN

A. Implementasi

Tahap implementasi merupakan tahap dimana seluruh *service* dari hasil analisis dan juga perancangan diimplementasikan dengan menggunakan teknologi Docker. Selain itu penggunaan *message broker* Redis Streams akan membantu mewujudkan penerapan *service choreography pattern* di dalam arsitektur aplikasi *microservice classroom* akademik.

1) Implementasi *Infrastructure as Code* Menggunakan *Dockerfile*

Seluruh *service* akan di-*build* menjadi *docker image* melalui *Dockerfile*. *Dockerfile* memungkinkan membangun seluruh *service* dengan *environment* dan dependensinya masing-masing hanya melalui baris kode. Teknik ini disebut dengan *Infrastructure as Code* (IaC) [15]. Dengan IaC memudahkan proses *deployment* tanpa harus melakukan instalasi secara manual dan tidak membutuhkan *resource* begitu besar, karena seluruh *docker image* akan berjalan di dalam lingkungan ter kontainerisasi dan terisolasi yang disebut dengan *docker container*. Dalam penerapan *service choreography pattern*, seluruh *docker container* nantinya akan saling bertukar data di dalam *docker network* dan berkomunikasi melalui *message broker* Redis Streams secara *asynchronous.*

Berikut merupakan hasil implementasi *Infrastructure as Code* menggunakan Docker yang ditulis melalui baris kode di dalam *Dockerfile* aplikasi *microservice classroom* akademik:

a) API *Gateway*

API *gateway service* akan dibangun menggunakan *framework* Express pada bahasa pemrograman JavaScript sesuai dengan tahap perancangan *service*. Gambar 11 menampilkan konfigurasi *Dockerfile* yang digunakan untuk mengimplementasikan *service* sebagai *docker image*:

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

```
FROM node:16-alpine

RUN apk update && apk add tzdata
ENV TZ=Asia/Jakarta

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 9000

CMD ["node", "server.mjs"]
```

Gambar 11 Konfigurasi *Dockerfile* API *Gateway Service*

Dengan menggunakan *Dockerfile* seperti yang tertera di gambar 11, maka API *gateway service* diimplementasikan sebagai IaC dengan keterangan sebagai berikut:

1. API *gateway service* berjalan diatas OS Alpine Linux.
2. API *gateway service* menggunakan *runtime* Node JS versi 16.
3. API *gateway service* dapat diakses melalui *port* 9000.

b) Account Service

Account service akan dibangun menggunakan *framework* Laravel pada bahasa pemrograman PHP sesuai dengan tahap perancangan *service*. Gambar 12 menampilkan konfigurasi *Dockerfile* yang digunakan untuk mengimplementasikan *service* sebagai *docker image*:

```
FROM php:8.1-fpm-alpine

RUN apk add tzdata
ENV TZ=Asia/Jakarta

RUN set -ex \
    && apk --no-cache add \
    postgresql-dev

RUN docker-php-ext-install pdo pdo_pgsql sockets pcntl

RUN apk --no-cache add pcre-dev ${PHPIZE_DEPS} \
  && pecl install redis \
  && docker-php-ext-enable redis \
  && apk del pcre-dev ${PHPIZE_DEPS} \
  && rm -rf /tmp/pear
# Use the default production configuration
RUN mv "$PHP_INI_DIR/php.ini-production" "$PHP_INI_DIR/php.ini"

COPY --from=composer /usr/bin/composer /usr/bin/composer

RUN set -x \
    addgroup -g 82 -S www-data \
    adduser -u 82 -D -S -G www-data www-data

COPY . /src
WORKDIR /src
RUN chmod -R 777 storage
RUN composer update
EXPOSE 9004
CMD php artisan octane:start --workers=6 --host=0.0.0.0 --port=9004
```

Gambar 12 Konfigurasi *Dockerfile Account Service*

Dengan menggunakan *Dockerfile* seperti yang tertera di gambar 12, maka *account service* diimplementasikan sebagai IaC dengan keterangan sebagai berikut:

1. *Account service* berjalan diatas OS Alpine Linux.
2. *Account service* menggunakan *runtime* PHP Laravel Octane.
3. *Account service* dapat diakses melalui *port* 9004.

c) Teacher Service

Teacher service akan dibangun menggunakan *framework* Express pada bahasa pemrograman JavaScript sesuai dengan tahap perancangan *service*. Gambar 13 menampilkan konfigurasi *Dockerfile* yang digunakan untuk mengimplementasikan *service* sebagai *docker image*:

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

```
FROM node:16-alpine

RUN apk update && apk add tzdata
ENV TZ=Asia/Jakarta

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 9002

CMD ["node", "server.mjs"]
```

Gambar 13 Konfigurasi *Dockerfile Teacher Service*

Dengan menggunakan *Dockerfile* seperti yang tertera di gambar 13, maka *teacher service* diimplementasikan sebagai IaC dengan keterangan sebagai berikut:

1. *Teacher service* berjalan diatas OS Alpine Linux.
2. *Teacher service* menggunakan *runtime* Node JS versi 16.
3. *Teacher service* dapat diakses melalui *port* 9002.

d) Student Service

Student service akan dibangun menggunakan *framework* Express pada bahasa pemrograman JavaScript sesuai dengan tahap perancangan *service*. Gambar 14 menampilkan konfigurasi *Dockerfile* yang digunakan untuk mengimplementasikan *service* sebagai *docker image*:

```
FROM node:16-alpine

RUN apk update && apk add tzdata
ENV TZ=Asia/Jakarta

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 9001

CMD ["node", "server.mjs"]
```

Gambar 14 Konfigurasi *Dockerfile Student Service*

Dengan menggunakan *Dockerfile* seperti yang tertera di gambar 14, maka *student service* diimplementasikan sebagai IaC dengan keterangan sebagai berikut:

1. *Student service* berjalan diatas OS Alpine Linux.
2. *Student service* menggunakan *runtime* Node JS versi 16.
3. *Student service* dapat diakses melalui *port* 9001.

e) Course Service

Course service akan dibangun menggunakan *framework* Express pada bahasa pemrograman JavaScript sesuai dengan tahap perancangan *service*. Gambar 15 menampilkan konfigurasi *Dockerfile* yang digunakan untuk mengimplementasikan *service* sebagai *docker image*:

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

```
FROM node:16-alpine

RUN apk update && apk add tzdata
ENV TZ=Asia/Jakarta

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 9003

CMD ["node", "server.mjs"]
```

Gambar 15 Konfigurasi *Dockerfile Course Service*

Dengan menggunakan *Dockerfile* seperti yang tertera di gambar 15, maka *course service* diimplementasikan sebagai IaC dengan keterangan sebagai berikut:

1. *Course service* berjalan diatas OS Alpine Linux.
2. *Course service* menggunakan runtime PHP Laravel Octane.
3. *Course service* dapat diakses melalui *port* 9003.

f) Classroom Service

Classroom service akan dibangun menggunakan *framework* Spring Boot pada bahasa pemrograman Java sesuai dengan tahap perancangan *service*. Gambar 8 menampilkan konfigurasi *Dockerfile* yang digunakan untuk mengimplementasikan *service* sebagai *docker image*:

```
FROM openjdk:12-alpine

RUN apk add tzdata
ENV TZ=Asia/Jakarta

ARG JAR_FILE=target/classroom-service-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar

RUN wget https://github.com/eficode/wait-for/releases/download/v2.2.3/wait-for

RUN chmod +x wait-for

EXPOSE 9005

ENTRYPOINT ["./wait-for", "postgres:5432", "--", "java", "-jar", "/app.jar", "--spring.profiles.active=docker"]
```

Gambar 16 Konfigurasi *Dockerfile Classroom Service*

Dengan menggunakan *Dockerfile* seperti yang tertera di gambar 16, maka *classroom service* diimplementasikan sebagai IaC dengan keterangan sebagai berikut:

1. *Classroom service* berjalan diatas OS Alpine Linux.
2. *Classroom service* menggunakan runtime JRE pada JDK versi 8.
3. *Classroom service* dapat diakses melalui *port* 9005.

2) Implementasi *Infrastructure as Code* Menggunakan *Docker Compose*

Seluruh *Dockerfile* akan di *build* melalui *docker compose*. Penggunaan *docker compose* akan memudahkan proses *build Dockerfile* menjadi *docker image*, juga memudahkan proses *pulling* dependensi *environment* dan *runtime* yang dibutuhkan, baik untuk tiap *service*, *database service* ataupun *message broker*.

Gambar 17 hasil implementasi *build Dockerfile* menjadi *docker image* melalui *docker compose*:

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

```
services:
  api:
    container_name: 'api-gateway-service'
    build: ./api-gateway-service
    ports:
      - '9000:9000'

  student:
    container_name: 'student-service'
    build: ./student-service
    ports:
      - '9001:9001'
    depends_on:
      - 'postgres'

  teacher:
    container_name: 'teacher-service'
    build: ./teacher-service
    ports:
      - '9002:9002'
    depends_on:
      - 'postgres'

  course:
    container_name: 'course-service'
    build: ./course-service
    ports:
      - '9003:9003'
    depends_on:
      - 'postgres'

  account:
    container_name: 'account-service'
    build: ./account-service
    ports:
      - '9004:9004'
    depends_on:
      - 'postgres'

  classroom:
    container_name: 'classroom-service'
    build: ./classroom-service
    ports:
      - '9005:9005'
    depends_on:
      - 'postgres'

  postgres:
    container_name: 'database-service'
    image: 'postgres:12.10-alpine'
    ports:
      - '5430:5432'
    environment:
      POSTGRES_USER: 'postgres'
      POSTGRES_PASSWORD: 'academic'
      POSTGRES_DB: 'academic'

  redis:
    container_name: 'message-broker'
    image: "redis:alpine"
    command: redis-server
    ports:
      - '6378:6379'
    volumes:
      - redis-data:/data
      - redis-conf:/usr/local/etc/redis/redis.conf
volumes:
  redis-data:
  redis-conf:
```

Gambar 17 Konfigurasi *Docker Compose*

Dapat dilihat pada gambar 17, seluruh *service* akan didefinisikan baik dari nama kontainer, lokasi folder untuk *Dockerfile* yang digunakan untuk melakukan *build, port* untuk *docker network* dan *port* yang dapat diakses *local*, sampai ketergantungan antar *service* seperti *database service.* Kemudian konfigurasi *database service* dan *message broker* memiliki keterangan sebagai berikut:

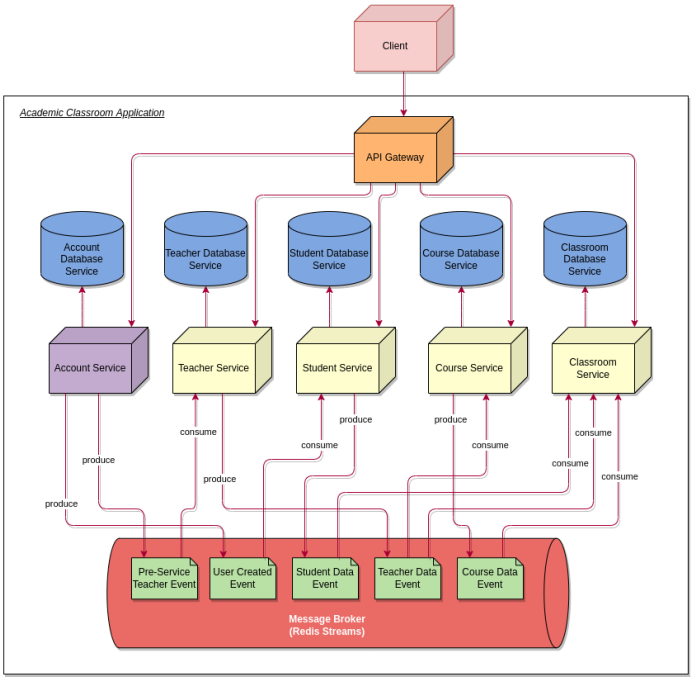
a) Database Service

Database service menggunakan PostgresQL, dari *docker image postgres:12.10-alpine* yang di *pull* dari Docker Hub. *Database service* dapat diakses melalui *port* 5432. Konfigurasi *environment* lainnya dapat dilihat pada gambar 17.

b) Message Broker

Message broker menggunakan Redis Streams, dari *docker image redis:alpine* yang di *pull* dari Docker Hub. Redis dapat diakses melalui *port* 6379. Redis menggunakan *volume* yang dapat diakses di *local* untuk melakukan *setting* tertentu.

Ketika *docker compose* dijalankan menggunakan perintah "*docker compose up*", maka seluruh *service* akan di *build* berdasarkan kode yang telah didefinisikan. *Service-service* yang telah di *build* akan menjadi *docker container* dan menjadi kesatuan aplikasi *microservice classroom* akademik yang mampu berjalan sesuai dengan proses bisnis inti akademik. Pada gambar 18 merupakan hasil implementasi arsitektur *microservice* pada aplikasi *classroom* akademik:



Gambar 18 Implementasi *Service Choreography Pattern* pada Arsitektur *Microservice* Aplikasi *Classroom* Akademik

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

3) Perangkat Implementasi

Perangkat yang digunakan untuk melakukan implementasi *microservices* terhadap aplikasi *classroom* akademik adalah:

a) Perangkat Keras
   1. *Processor* : Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
   2. *Memory* : 12 GB
   3. SSD : 512 GB
   4. *Hard disk* : 1 TB

b) Perangkat Lunak
   1. *Operating system* : Linux/Windows
   2. Docker : Docker 20.10.17
   3. *Runtime* : Apache, Node JS, JRE
   4. *Web browser* : Brave
   5. Bahasa Pemrograman : PHP, JavaScript, Java
   6. *Tools* : Wrk Bench, Postman, Neo Vim, Docker
   7. DBMS : PostgreSQL, Redis

B. Pengujian

Pengujian pada penelitian ini terbagi atas beberapa tahapan, yakni sebagai berikut:

1) *Tools* yang dipakai

Pada tahapan pengujian ini akan dilakukan serangkaian pengujian menggunakan *tools* Wrk Bench untuk mengetahui *benchmark* ketika menggunakan arsitektur *microservice*. Wrk Bench merupakan *tools benchmark* HTTP modern yang mampu menghasilkan beban signifikan saat dijalankan pada *single multi-core* CPU.

2) Cara analisis

Aplikasi yang dibuat kemudian divalidasi dan dievaluasi dengan beberapa parameter, seperti waktu tanggapan atau *latency* (ms), jumlah HTTP *request/seconds* (rps) dan kecepatan *transfer/seconds* (Mbps). Pengujian dilakukan dengan membuat sejumlah HTTP *connections* menggunakan sejumlah *threads* CPU dan dijalankan dalam waktu tertentu. Data tersebut nantinya akan ditampilkan ke dalam tabel yang digunakan untuk melihat karakteristik aplikasi yang dibuat dengan menggunakan *service choreography pattern* pada arsitektur *microservice classroom* akademik dalam menangani beban kerja yang telah ditentukan.

3) Hasil Evaluasi

Pada pengujian ini, dilakukan dengan membuat 400 HTTP *connections* yang kemudian dijalankan menggunakan 6 *threads* dari total 8 *threads* yang ada, dan akan dilakukan selama 15 detik. *Benchmarking* akan dilakukan terhadap *endpoint* API *gateway* yang mengarah ke *student service*, *endpoint user-profile* pada *account service*, serta *endpoint insert data* pada masing-masing *service*, yakni *student service*, *teacher service*, *course service* dan *classroom service*. Tujuan pengujian *benchmarking* ini adalah untuk mengukur jumlah waktu tanggapan atau *latency* (ms), jumlah HTTP *request/seconds* (rps) dan kecepatan *transfer/seconds* (Mbps) ketika menggunakan *service choreography pattern* pada arsitektur *microservice* guna mengetahui batas yang dapat dicapai oleh aplikasi *classroom* akademik itu sendiri.

Pada tabel I, ditampilkan secara berurutan hasil *benchmarking* terhadap *endpoint* API *gateway* yang mengarah ke *student service*, *endpoint user-profile* pada *account service*, serta *endpoint insert data* pada masing-masing *service*, yakni *student service*, *teacher service*, *course service* dan *classroom service*.

TABEL I
HASIL EVALUASI RATA-RATA

Endpoint	Rata-rata		Standar Deviasi		Maximum		+/- Standar Deviasi (%)	
	Latency (ms)	*Requests/ seconds* (rps)	*Latency* (ms)	*Requests/ seconds* (rps)	*Latency* (ms)	*Requests/ seconds* (rps)	*Latency* (%)	*Requests/ seconds* (%)
http://localhost:9000/students/	147.40	460.00	33.65	176.05	442.23	666.00	94.09	50.11
http://localhost:9004/api/user-profiles/	838.52	107.44	315.33	93.89	1.32s	520.00	61.13	79.45
http://localhost:9001/students/	128.07	528.89	30.03	140.77	470.40	666.00	94.91	85.42
http://localhost:9002/teachers/	117.34	566.02	13.60	109.30	250.72	666.00	88.99	86.67
http://localhost:9003/courses/	102.01	662.29	26.20	117.13	409.83	0.98k	93.26	82.41
http://localhost:9005/api/v1/classrooms/	151.98	552.54	167.01	185.32	1.67s	820.00	91.77	83.73

Tabel I menunjukkan bahwa *latency* rata-rata yang dihasilkan dari aplikasi yang menggunakan *service choreography pattern* ialah 102 ms sampai 838 ms. Dimana *latency* terendah pada *course service* dan tertinggi

E-ISSN : 2540 - 8984

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Volume 07, Nomor 03, September 2022 : 768–779

pada *endpoint user-profile*, hal ini dapat terjadi karena data yang dibutuhkan *course service* itu sendiri memiliki atribut yang cukup sedikit. Kemudian *endpoint user-profile* memiliki *latency* tertinggi dikarenakan proses *auth* yang memiliki beberapa *rules authentication* dan *authorization*. Pada saat *request* yang dilakukan oleh satu *user* yang sama, maka *account service* akan secara otomatis menutup permintaan dan membalikkan *response 429* (*too many request*).

TABEL II
HASIL EVALUASI TOTAL

Endpoint	Total *Request*	Total *MB* data terbaca *(MB)*	*Non-2xx/3xx Response*	Total *Requests/ seconds*	Total *Transfer/ seconds (MB)*
http://localhost:9000/students/	40391	13.64	40391	2678.90	0.90
http://localhost:9004/api/user-profiles/	6736	65.32	6598	446.50	4.33
http://localhost:9001/students/	46671	15.34	-	3105.14	1.02
http://localhost:9002/teachers/	50484	16.61	-	3355.19	1.10
http://localhost:9003/courses/	58627	19.28	-	3899.28	1.28
http://localhost:9005/api/v1/classrooms/	49094	16.15	-	3254.11	1.07

Pada tabel II menunjukkan performa yang dihasilkan setiap *service* mampu menampung 40391 sampai 58627 *request* dalam 15 detik pada 400 *connections*, namun tidak pada *endpoint user-profiles*. Hal ini disebabkan kembali oleh adanya sistem *auth* pada *account service* yang membatasi permintaan ketika terlalu banyak (*rate limit*). Di samping itu, hal ini menunjukkan betapa efektifnya penggunaan *service choreography pattern* yang diterapkan pada *classroom* akademik karena penggunaan *rate limit* untuk menangani permintaan yang terlalu banyak.

Kemudian pada tabel II terlihat adanya *response* selain kode 2xx dan 3xx hanya pada API *gateway* dan juga *endpoint user-profiles*. Pada API *gateway*, seluruh *response* merupakan bukan kode 2xx dan 3xx, hal ini disebabkan permintaan yang tidak memiliki otentikasi. Sedangkan pada *endpoint user-profiles*, terdapat 6598 *non-2xx/3xx response* dari total 6736 *request*. Hal ini menunjukkan sistem hanya mengijinkan 138 *request* yang masuk.

IV. KESIMPULAN

Kesimpulan dari penelitian dan pengujian ini adalah berhasilnya implementasi *service choreography pattern* pada arsitektur *microservice* untuk aplikasi *classroom* akademik yang terbentuk atas *service-service* yaitu API *gateway*, *account service*, *teacher service*, *student service*, *course service*, *classroom service*, *database service* dan *message broker*. Dalam pengujian *benchmarking* menggunakan Wrk Bench dengan membuat 400 HTTP *connections* yang kemudian dijalankan menggunakan 6 *threads* dari total 8 *threads* yang ada, dan dilakukan selama 15 detik, jika dilihat dari Tabel II, maka *request* HTTP yang dapat diterima melebihi 40000 permintaan, yang menunjukkan betapa efektifnya penggunaan *service choreography pattern* yang diterapkan pada *classroom* akademik. Serta dengan adanya pembatasan *request* HTTP pada *account service*, menjadikan sistem menjadi lebih aman ketika menggunakan *rate limit* untuk membatasi permintaan yang terlalu banyak.

DAFTAR PUSTAKA

[1] M. A. Ramdhani, "Pemodelan Proses Bisnis Sistem Akademik Menggunakan Pendekatan Business Process Modelling Notation (BPMN) (Studi Kasus Institusi Perguruan Tinggi Xyz)," *J. Inf.*, vol. 7, no. 2, pp. 83–93, 2015.

[2] M. Taufiq, "DAMPAK PERKEMBANGAN TEKNOLOGI INFORMASI DALAM PROFESI AKUNTAN DAN IMPLIKASINYA DALAM DUNIA PENDIDIKAN," *STMIK AMIKOM Yogyakarta*, 2008.

[3] S. D. N. Van Duy *et al.*, "Microservices from Theory to Practice Creating Applications in IBM Bluemix Using the Microservices Approach," *Ibm*, p. 170, 2015.

[4] P. S and Shilpa G.V, "Microservices architecture style," *Int. Res. J. Comput. Sci.*, vol. 4, no. 05, pp. 65–69, 2017, [Online]. Available: http://www.irjcs.com/volumes/vol4/iss05/15.MYCSSP10088.pdf.

[5] M. Sendiang, S. Kasenda, and J. Purnama, "Implementasi Teknologi Mikroservice pada Pengembangan Mobile Learning," *J. Appl. Informatics Comput.*, vol. 2, no. 2, pp. 63–66, 2018, doi: 10.30871/jaic.v2i2.1046.

[6] R. Petrasch, "Model-based engineering for microservice architectures using Enterprise Integration Patterns for inter-service communication," *Proc. 2017 14th Int. Jt. Conf. Comput. Sci. Softw. Eng. JCSSE 2017*, pp. 5–8, 2017, doi: 10.1109/JCSSE.2017.8025912.

[7] C. Richardson, *Microservices Patterns*. 2017.

[8] M. Romadlon Bik, "Implementasi Docker Untuk Pengelolaan Banyak Aplikasi Web (Studi Kasus : Jurusan Teknik Informatika UNESA)," *J. Manaj. Inform.*, vol. 7, no. 2, pp. 46–50, 2017.

[9] F. Adiputra, "Container dan Docker: Teknik Vertualisasi dalam Pengelolaan Banyak Aplikasi Web," *J. SimanteC*, vol. 4, no. 3, pp. 167–176, 2015.

[10] M. F. Razi, "Implementasi pengendali elastisitas sumber daya berbasis docker untuk aplikasi web," Institut Teknologi Sepuluh Nopember, 2017.

[11] S. Dwiyatno, E. Rakhmat, and O. Gustiawan, "Implementasi Virtualisasi Server Berbasis Docker Container," *Prosisko*, vol. 7, no. 2, pp. 165–175, 2020, [Online]. Available: https://e-jurnal.lppmunsera.org/index.php/PROSISKO/article/view/2520/.

[12] H. Kristianto and A. Zahra, "Performance Analysis of Choreography and Orchestration in Microservices Architecture," *J. Theor. Appl. Inf.*

Technol., vol. 99, no. 18, pp. 4220–4230, 2021.
[13] C. Richardson, *Microservices Patterns with Examples in Java*. Shelter Island: Manning, 2019.
[14] R. S. F. Jannatin, A. Suharsono, and A. Bhawiyuga, "Implementasi Publish-Subscribe Pada Delay Tolerant Network (DTN)," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 1, no. 2, pp. 118–124, 2017, [Online]. Available: http://j-ptiik.ub.ac.id/index.php/j-ptiik/article/download/51/27.
[15] I. P. Agus and E. Pratama, "Infrastructure as Code (IaC) Menggunakan OpenStack untuk Kemudahan Pengoperasian Jaringan Cloud Computing (Studi Kasus: Smart City di Provinsi Bali) Infrastructure as Code (IaC) Using OpenStack for Ease of Operation of Cloud Computing Network (Case Study of Smart City in Bali Province)," *J. Ilmu Pengetah. dan Teknol. Komun.*, vol. 23, no. 1, pp. 93–105, 2021, [Online]. Available: http://dx.doi.org/10.33169/iptekkom.23.1.2021.93-105.