TESIS - KI142502

PENINGKATAN KETERSEDIAAN APLIKASI WEB MENGGUNAKAN ARSITEKTUR LAYANAN MIKRO BERDASARKAN IDENTIFIKASI LOG AKSES

ARDYANTO HERMAWAN
5114201025

Dosen Pembimbing
Dr. Ir. Siti Rochimah, MT.
Rizky Januar Akbar, S.Kom., M.Eng.

PROGRAM MAGISTER
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:
ARDYANTO HERMAWAN
Nrp. 5114201025

Dengan judul :
Peningkatan Ketersediaan Aplikasi Web Menggunakan Arsitektur Layanan Mikro Berdasarkan Identifikasi Log Akses

Tanggal Ujian : 12-7-2017
Periode Wisuda : 2016 Genap

Disetujui oleh:

Dr. Ir. Siti Rochimah, M.T
NIP. 196810021994032001 (Pembimbing 1)

Rizky Januar Akbar, S.Kom, M.Eng
NIP. 198701032014041001 (Pembimbing 2)

Daniel Oranova Siahaan, S.Kom, M.Sc, PD.Eng.
NIP. 197411232006041001 (Penguji 1)

Sarwosri, S.Kom, M.T
NIP. 197608092001122001 (Penguji 2)

Nurul Fajrin Ariyani, S.Kom, M.Sc
NIP. 198607222015042003 (Penguji 3)

Dekan Fakultas Teknologi Informasi,

Dr. Agus Zainal Arifin, S. Kom., M. Kom.
NIP. 197208091995121001

Peningkatan Ketersediaan Aplikasi Web Menggunakan Arsitektur Layanan Mikro Berdasarkan Identifikasi Log Akses

Nama mahasiswa	: Ardyanto Hermawan
NRP	: 5114201025
Pembimbing I	: Dr. Ir. Siti Rochimah, MT.
Pembimbing II	: Rizky Januar Akbar, S.Kom.,M.Eng.

ABSTRAK

Pola arsitektur *microservices* (layanan mikro) yang menjadi tren dalam beberapa tahun terakhir menawarkan modernisasi pada aplikasi dengan pola arsitektur monolitik. Jika terdapat *bug* pada aplikasi yang menerapkan arsitektur monolitik, dapat mematikan seluruh proses karena setiap modul dijalankan dalam proses yang sama sehingga mempengaruhi ketersediaan aplikasi. Arsitektur layanan mikro bertujuan untuk mengembangkan sistem yang terdiri dari kumpulan layanan yang kohesif yang dapat dievolusi berulang-ulang dan dapat dikembangkan atau dipasang secara mandiri. Untuk menerapkan arsitektur layanan mikro tidak harus mengubah aplikasi yang ada secara keseluruhan. Kandidat yang dapat dijadikan layanan mikro yaitu komponen yang terus berubah, komponen yang membutuhkan sumber daya yang besar, atau pada tingkat antarmuka pengguna. Aplikasi web yang sering dikunjungi akan meningkatkan kebutuhan sumber daya pada komponen yang di akses. Pemrosesan log akses dan kode sumber dari sistem dilakukan untuk mendapatkan kandidat layanan mikro. Kode sumber pada kandidat yang terpilih akan ditulis ulang berdasarkan karakteristik dari layanan mikro. Setiap layanan mikro akan dipasang pada kontainer. Dari hasil pengujian *Performance*, *Average Response Time*, dan ketersediaan, dapat disimpulkan bahwa layanan mikro dapat meningkatkan persentase *Average Response Time* dan juga meningkatkan ketersediaan.

Kata kunci: Arsitektur perangkat lunak, layanan mikro, log akses, kontainer

Increasing Web Application Availability Using Microservice Architecture Based on Access Log Identification

Name	: Ardyanto Hermawan
Student Identity Number	: 5114201025
Supervisor	: Dr. Ir. Siti Rochimah, MT.
Co-Supervisor	: Rizky Januar Akbar, S.Kom.,M.Eng.

ABSTRACT

The microservices architecture pattern that has become a trend in recent years offers modernization in applications to monolithic architectural patterns. If a bug exists in an application that implements a monolithic architecture, it can shut down the entire process because each module is run in the same process affecting the availability of the application. The microservice architecture aims to develop systems that consist of a cohesive set of services that can be evolved over and over and can be developed or deployed independently. To implement the microservice architecture does not have to change the whole existing application. Candidates that can be used as microservices are components that are constantly changing, components that require large resources, or at the user interface level. The frequently visited web app will increase the resource requirements of the accessed component. Access log processing and source code from the system is done to obtain microservice candidates. The base code of the selected candidate will be rewritten based on the characteristics of the microservices. Each microservice will be deployed on the container. Based on the performance, Average Response Time, and availability testing results, it can be concluded that microservices architecture pattern can increase the percentage of Average Response Time and also increase availability than monolithic architecture

Keywords: Software architecture, microservice, access log, container

KATA PENGANTAR

Puji syukur ke hadirat Allah SWT yang telah memberikan Taufiq, Hidayah, dan Inayah-Nya sehingga Tesis yang berjudul "PENINGKATAN KETERSEDIAAN APLIKASI WEB MENGGUNAKAN ARSITEKTUR LAYANAN MIKRO BERDASARKAN IDENTIFIKASI LOG AKSES" dapat diselesaikan dengan baik. Tesis ini diharapkan dapat bermanfaat bagi perkembangan ilmu pengetahuan terutama bidang rekayasa perangkat lunak dan dapat memberikan kontribusi untuk penelitian selanjutnya.

Penulis mengucapkan terima kasih atas doa, bantuan dan dukungan yang diberikan selama proses penyusunan tesis ini kepada :

1. Kedua orang tua dan Kakak Adik yang selalu memberikan dukungan, doa, semangat dan motivasi kepada penulis selama masa kuliah hingga penyelesaian tesis ini.
2. Ibu Dr. Ir. Siti Rochimah, M.T., selaku dosen pembimbing pertama atas bimbingan dan dukungannya hingga terselesaikannya tesis ini.
3. Bapak Rizky Januar Akbar, S.Kom., M.Eng., selaku dosen pembimbing kedua atas bimbingan dan dukungannya hingga terselesaikannya tesis ini.
4. Bapak Daniel Oranova, S.Kom, M.Sc, PhD.Eng, Ibu Sarwosri, S.Kom, M.T, dan Ibu Nurul Fajrin Ariyani, S.Kom, M.Sc selaku dosen penguji tesis yang telah memberikan banyak saran demi kemajuan penulis dan tesis ini.
5. Bapak dan Ibu dosen pascasarjana Teknik Informatika ITS Surabaya yang telah memberikan ilmu dan bimbingan selama masa kuliah.
6. Bapak Tiyo Avianto, selaku direktur utama PT. Eyro Digital Teknologi yang telah memberikan kesempatan dan dukungan untuk studi.
7. Rekan – rekan di PT. Eyro Digital Teknologi yang telah memberikan dukungan, doa dan ilmu.
8. Teman – teman Pascasarjana Teknik Informatika ITS Angkatan 2014 yang memberi dukungan dan bantuan ilmu.
9. Semua pihak yang telah mendukung, membantu dan mendoakan.

Penulis menyadari masih terdapat banyak kekurangan. Untuk itu saran dan kritik yang membangun akan sangat membantu agar tesis ini dapat menjadi lebih baik. Akhir kata penulis ucapkan terima kasih.

Surabaya, Juli 2017

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Aplikasi dengan pola arsitektur monolitik akan bertambah besar dari waktu ke waktu sehingga menjadikannya sulit, berisiko, dan membutuhkan banyak biaya untuk dievolusi (Levcovitz, et al., 2015). Banyak sistem yang menggunakan arsitektur ini meskipun memiliki permasalahan tersebut. Secara logika mungkin memiliki arsitektur modular, tetapi aplikasi dibungkus dan dipasang secara monolitik. Arsitektur mudah untuk diuji, dipasang dan dapat diskala di belakang penyeimbang beban dan akan bekerja dengan baik di awal. Ada beberapa tantangan besar untuk evolusi sistem monolitik seperti jadwal yang ketat, dana yang terbatas, tetapi harus tetap berkualitas, selalu tersedia, dan handal (Richardson & Smith, 2016).

Dengan pola arsitektur monolitik, aplikasi akan sulit dipahami dan dimodifikasi. Sebuah kesalahan dalam modul, dapat mematikan seluruh proses karena setiap modul dijalankan dalam proses yang sama. Sebuah kesalahan dapat mempengaruhi ketersediaan layanan. Layanan harus dapat disampaikan dalam cara yang dapat diandalkan dan tepat waktu, dan informasi harus dapat disimpan dan diambil sesuai kebutuhan. Ketersediaan layanan penting untuk semua aplikasi, oleh karena itu langkah-langkah tertentu harus diambil untuk mencegah gangguan layanan. Ketersediaan mendeskripsikan sebaik mana sistem dalam menyediakan layanan dalam periode waktu tertentu (Nabi, et al., 2016).

Saat ini pola arsitektur *microservices* (layanan mikro) muncul sebagai alternatif untuk evolusi aplikasi monolitik. Banyak perusahaan besar telah sukses dalam menggunakan layanan mikro, di antaranya yaitu Amazon dan Netflix. Arsitektur ini bertujuan untuk mengembangkan sistem yang terdiri dari kumpulan layanan yang kohesif yang dapat dievolusi berulang-ulang dan dapat dikembangkan atau dipasang secara mandiri. Layanan mikro terdiri dari bagian-bagian kecil yang menangani aktivitas secara spesifik dan berkomunikasi melalui mekanisme yang ringan (Fowler & Lewis, 2014).

Penelitian sebelumnya yang dilakukan oleh (Levcovitz, et al., 2015), melakukan migrasi sistem secara keseluruhan dari pola arsitektur monolitik ke layanan mikro berdasarkan titik masuk, fungsi bisnis dan tabel pada basis data. Tetapi untuk menerapkan layanan mikro tidak harus mengubah secara keseluruhan (Gupta, 2015). Kandidat layanan mikro yang dapat dilakukan ekstraksi yaitu komponen yang terus berubah, komponen yang membutuhkan sumber daya yang besar, atau pada tingkat antarmuka pengguna (Richardson & Smith, 2016).

Layanan yang sering dikunjungi maka akan meningkatkan kebutuhan sumber daya pada layanan yang di akses. Log akses dari sistem dapat dianalisa untuk mengetahui tingkat penggunaan dari suatu layanan. Oleh karena itu, pada proposal ini mengajukan peningkatan ketersediaan pada aplikasi web dengan memanfaatkan kelebihan dari pola arsitektur layanan mikro dengan mengekstraksi kandidat layanan mikro menggunakan log akses dari sistem.

1.2 Perumusan Masalah

Beberapa permasalahan dalam penelitian ini dirumuskan sebagai berikut:

1. Bagaimana mengidentifikasi kandidat layanan mikro dari sistem yang menggunakan pola arsitektur monolitik.
2. Bagaimana meningkatkan ketersediaan dengan menggunakan pola arsitektur layanan mikro.
3. Bagaimana mengevaluasi ketersediaan dari arsitektur layanan mikro.

1.3 Tujuan dan Manfaat Penelitian

Tujuan yang ingin dicapai dari penelitian ini adalah meningkatkan ketersediaan dari aplikasi dengan menggunakan arsitektur layanan mikro. Dengan mengetahui tingkat kemanfaatan diharapkan dapat memberi dampak pada aplikasi untuk terus-menerus hidup, mengurangi kompleksitas dari pola arsitektur monolitik dan juga dapat memudahkan proses perawatan..

1.4 Batasan Masalah

Dalam penelitian ini, batasan masalah yang dapat diuraikan sebagai berikut:

1. Sistem yang diuji menggunakan kerangka kerja PHP Laravel versi 4.2.
2. Log akses yang digunakan merupakan hasil dari web server.

1.5 Kontribusi Penelitian

Penelitian ini diharapkan dapat memberikan kontribusi yaitu meningkatkan ketersediaan layanan pada aplikasi web dengan memanfaatkan kelebihan dari arsitektur layanan mikro dan mengidentifikasi kandidat layanan mikro dari sistem berdasarkan analisa log akses dari sistem yang sedang berjalan.

[Halaman ini sengaja dikosongkan]

BAB II
DASAR TEORI DAN KAJIAN PUSTAKA

2.1 Penelitian Terkait

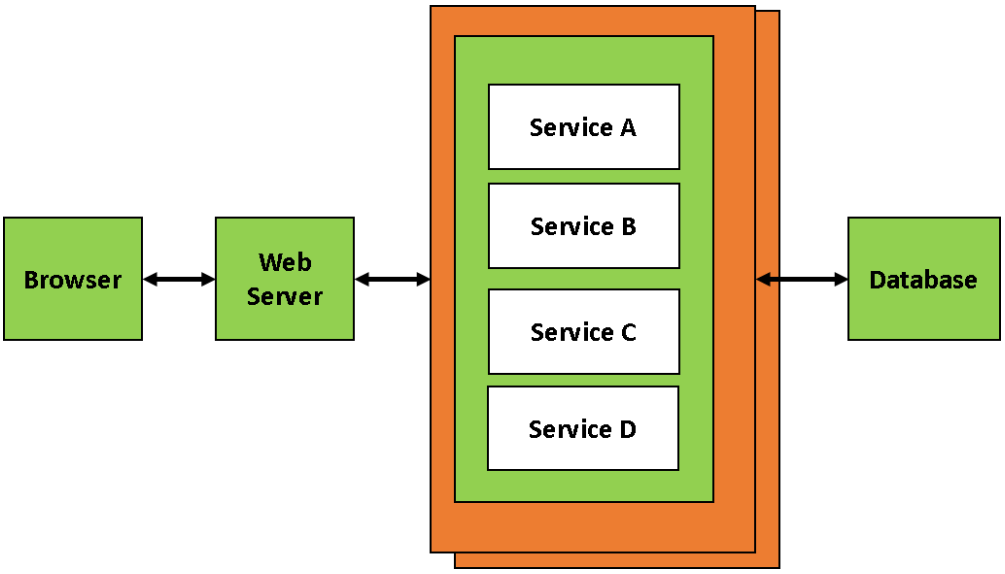
Penelitian sebelumnya mengenai layanan mikro dibahas oleh (Levcovitz, et al., 2015) mengenai pendekatan dalam melakukan ekstraksi sistem monolitik dari sistem perbankan yang telah berjalan. Setiap logika bisnis memiliki titik awal, pemrosesan dan juga tabel pada basis data sehingga membentuk graf ketergantungan. Setiap satu titik awal, logika bisnis dan tabel akan menjadi satu layanan mikro. Layanan mikro yang terbentuk akan semakin banyak. Tetapi dalam menerapkan pola arsitektur layanan mikro tidak harus mengubah sistem secara keseluruhan (Gupta, 2015). Layanan mikro yang terbentuk di-*deploy* tanpa menggunakan kontainer.

Penelitian yang dilakukan oleh (Villamizar, et al., 2015), menguji performa dari layanan mikro dengan menggunakan server pada infrastruktur awan. Peforma pola arsitektur layanan mikro dibandingkan dengan pola arsitektur monolitik dengan menggunakan data uji sebuah aplikasi yang dibangun dengan menggunakan kerangka kerja dan pola *deployment* menggunakan *API Gateway*. Tetapi pada pola arsitektur layanan mikro memiliki *Average Response Time* lebih tinggi dibandingkan pola arsitektur monolitik. Hal ini dikarenakan layanan diberikan permintaan dalam jumlah besar dan secara simultan tanpa ada penyeimbang beban.

2.2 Arsitektur Monolitik

Aplikasi dengan pola *deployment* monolitik berarti sebuah aplikasi dengan satu basis kode/repositori besar, menawarkan puluhan atau ratusan layanan menggunakan berbagai antarmuka seperti halaman HTML, layanan web atau / dan layanan REST (Villamizar, et al., 2015). Memungkinkan memiliki arsitektur modular secara logika, tetapi tetap dalam satu paket dan di-*deploy* secara monolitik. sebagai contoh, banyak aplikasi dengan menggunakan PHP dibungkus dalam satu direktori dan di-*deploy* pada web server seperti Apache atau Nginx seperti pada Gambar 2.1.

Dengan pola arsitektur ini, mudah untuk dibangun, diuji secara *black-box* testing ataupun *white-box* testing, mudah untuk di-*deploy*, dan juga dapat di skala di belakang penyeimbang beban. Tetapi ketika dilakukan pengubahan dan juga pembaharuan yang berkelanjutan, maka basis kode akan semakin besar dan kompleks. Untuk perbaikan *bug* dan juga penambahan fitur baru akan membutuhkan waktu lebih banyak dan lebih sulit. Jika basis kode sulit untuk dipahami, maka perubahan tidak dapat dilakukan dengan benar.



Gambar 2.1 Contoh arsitektur monolitik

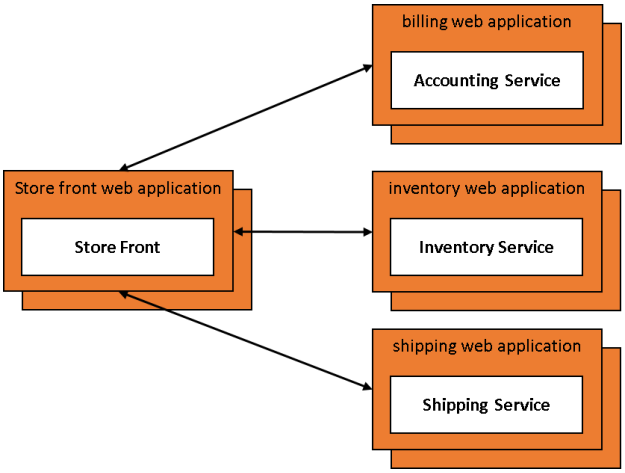
2.3 Arsitektur Layanan Mikro

Arsitektur layanan mikro muncul pada beberapa tahun terakhir, menjadi populer pada kalangan *Developments and Operations* (DevOps) karena menawarkan beberapa solusi yang menjadi kekurangan pada aplikasi monolitik. Kode program akan bertambah besar sesuai dengan penambahan fitur baru, tentu akan menambah kerumitan untuk mengetahui di mana perubahan yang harus dilakukan karena kode program yang besar (Fowler & Lewis, 2014). Sebagai contoh, sebuah sistem memiliki tiga layanan dengan antar muka yang terpisah. Setiap layanan akan berkomunikasi dengan layanan lainnya. Setiap layanan di-*deploy* secara terpisah seperti pada Gambar 2.2.

Netflix merupakan salah satu cerita sukses dalam menggunakan layanan mikro. Perusahaan tersebut melakukan transisi dari model pembangunan tradisional

dengan 100 pengembang aplikasi untuk memproduksi aplikasi monolitik persewaan DVD ke arsitektur layanan mikro dengan banyak tim kecil yang bertanggung jawab untuk pengembangan *end-to-end* dari ratusan layanan mikro yang bekerja sama untuk *streaming* hiburan digital yang dapat diakses oleh jutaan pelanggan Netflix setiap hari.

Awal tahun 2012, Netflix mulai mengadopsi layanan mikro. Ketika Netflix mengalami kerusakan basis data dalam skala besar karena titik koma yang hilang, mereka menyadari bahwa aplikasi monolitik rentan terhadap satu titik kegagalan.



Gambar 2.2 Contoh arsitektur layanan mikro

Sumber: Newman, 2015

Jika salah satu titik koma yang hilang memiliki potensi untuk mematikan seluruh sistem, maka perlu dilakukan perubahan. Mereka membuat keputusan untuk memecah aplikasi monolitik menjadi lebih kecil, tiap layanan mikro menurunkan kemungkinan kesalahan sistem dan memastikan stabilitas sistem jangka panjang yang lebih baik. Netflix mengikuti ini blok pembangunan layanan mikro sebagai dasar untuk mengembangkan arsitektur.

Ada lima karakteristik dari layanan mikro sebagai berikut (Gupta, 2015):

a. Melakukan dekomposisi fungsional dengan menggunakan metode dari Domain-Driven Design (DDD);
b. Prinsip tanggung jawab tunggal yaitu setiap layanan memiliki tanggung jawab terhadap satu fungsional;

c. Antarmuka didefinisikan dengan jelas yaitu sebuah layanan produser menerbitkan sebuah antarmuka yang digunakan oleh layanan konsumen;
d. Setiap layanan dapat digunakan, diperbarui, diganti, dan diskala secara mandiri;
e. Komunikasi ringan yaitu komunikasi sederhana dengan menggunakan *Representational State Transfer* (REST) melalui *HyperText Transfer Protocol* (HTTP), STOMP melalui WebSocket, dan protokol ringan lainnya. Jika dibandingkan dengan *Enterprise Service Bus* (ESB) dan pendekatan yang sejenis, di mana mekanisme komunikasi menyediakan fungsionalitas yang canggih untuk transformasi pesan dan koreografi, layanan mikro menggunakan media komunikasi untuk pertukaran data dengan menggunakan HTTP *Request-Response* atau dengan menggunakan *message bus* untuk komunikasi *asynchronous* dengan *routing*;

Dari karakteristik layanan mikro dapat diperoleh beberapa kelebihan antara lain (Gupta, 2015):

a. Skala secara mandiri, setiap layanan mikro dapat diperbanyak secara mandiri melalui skala sumbu X (menambah CPU atau memori) dan skala sumbu *Z* (partisi), tergantung pada kebutuhan.
b. *Upgrade* secara mandiri, setiap layanan dapat di-*deploy* secara mandiri. Pengembang dapat melalukan perubahan pada layanan tanpa harus berkoordinasi dengan tim lain.
c. Perawatan mudah, kode pada layanan mikro dibatasi hanya satu fungsional sehingga mudah dipahami.
d. Heterogen dan poligotisme, pengembang memiliki kebebasan untuk memilih bahasa pemrograman dan perangkat lainnya yang cocok dengan karakteristik layanan yang dibuat.
e. Isolasi kesalahan dan sumber daya, layanan yang berjalan tidak semestinya seperti kekurangan memori atau yang lainnya hanya mempengaruhi layanan itu sendiri. Hal ini meningkatkan isolasi kesalahan dan membatasi berapa banyak aplikasi yang mengalami kegagalan.

f. Memperkuat komunikasi antar kelompok, layanan mikro dibangun oleh tim *full-stack*. Seluruh anggota yang berkaitan dengan domain dapat bekerja pada satu tim.

2.4 Komunikasi antar layanan

Pada tahun 2000, Roy Fielding, satu dari kontributor kunci HTTP dan URI, mengajukan sebuah artistektur dengan nama *REpresentational State Transfer* (REST) pada penelitiannya. Gaya arsitektur REST terinspirasi dari teknologi dasar *World Wide Web* (WWW). Teknologi yang digunakan meliputi *HyperText Transfer Protocol* (HTTP), *Uniform Resource Identifier* (URI), bahasa markup seperti HTML dan XML, atau format *JavaScript Object Notation* (JSON) yang paling banyak digunakan.

REST merupakan salah satu jenis *web service* yang menerapkan konsep perpindahan antar *state*. Konsep perpindahan *state* dalam hal ini dapat dianalogikan seperti klien meminta suatu halaman web, maka server akan mengirimkan *state* halaman web ke klien. Dengan kata lain jika melakukan navigasi melalui tautan-tautan yang disediakan pada halaman web sama dengan mengganti *state* dari halaman web.

REST bekerja dengan bernavigasi melalui tautan-tautan HTTP untuk melakukan aktivitas tertentu, seakan-akan terjadi perpindahan *state* satu sama lain. Perintah HTTP yang biasa digunakan adalah fungsi GET, POST, PUT, PATCH, atau DELETE (Tabel 2.1).

Tabel 2.1 Contoh metode HTTP pada REST

No.	Verb	Arti
1	*GET*	Digunakan untuk mendapatkan resource.
2	*POST*	Digunakan untuk membuat resoirce baru.
3	*PUT*	Digunakan untuk melakukan perubahan resource secara keseluruhan.
4	*DELETE*	Digunakan untuk menghapus resource.
5	*PATCH*	Digunakan untuk melakukan perubahan pada resource secara parsial

Balasan yang dikirim oleh server dapat berupa XML ataupun JSON, sehingga informasi yang diterima lebih mudah dibaca dan diproses pada sisi klien. Dalam pengaplikasiannya, REST lebih banyak digunakan untuk *web service* yang berorientasi pada sumber daya. Maksud orientasi pada sumber daya adalah orientasi

yang menyediakan sumber daya sebagai layanannya dan bukan kumpulan-kumpulan dari aktivitas yang mengolah sumber daya itu. REST menggunakan standar HTTP, membuatnya lebih mudah dimengerti.

2.5 Deployment

Dalam proses *deployment*, layanan mikro dapat dipasang dengan menggunakan server fisik, mesin virtual ataupun kontainer. *Deployment* menggunakan kontainer paling banyak digunakan karena kemudahannya. Kontainer menyediakan paket teknologi terintegrasi yang memungkinkan tim operasi dan pengembangan IT untuk membangun, menyalurkan, dan menjalankan aplikasi terdistribusi di mana saja. Kontainer memungkinkan pengembang perangkat lunak untuk membungkus aplikasi beserta semua dependensinya ke dalam sebuah unit standar untuk pengembangan perangkat lunak. Kontainer dari membungkus sebuah perangkat lunak dalam *filesystem* yang lengkap, berisi segala sesuatu yang dibutuhkan untuk menjalankan: kode, *runtime*, *system tools*, *system libraries* dan apa pun yang dapat diinstal pada server. Hal ini menjamin bahwa kontainer yang berjalan akan selalu sama, terlepas dari lingkungan sedang berjalan.

Beberapa karakteristik dari kontainer sebagai berikut:

a. Ringan

   Semua kontainer berjalan pada satu mesin dengan berbagi kernel sistem operasi yang sama sehingga lebih efisien penggunaan RAM. *Image* dibangun dari *filesystem* berlapis sehingga mereka dapat berbagi file umum, membuat penggunaan disk dan pengunduhan *image* jauh lebih efisien.

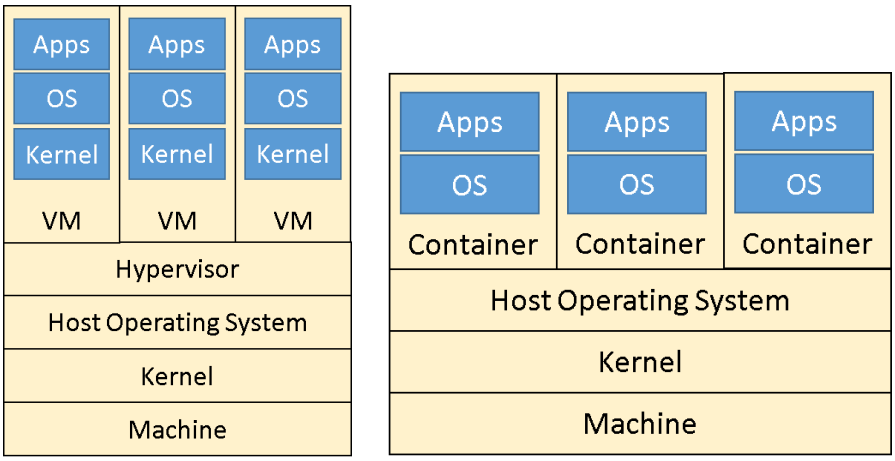
b. Terbuka

   Kontainer pada didasarkan pada standar *open source* yang memungkinkan kontainer berjalan di semua distro utama Linux dan sistem operasi Microsoft dengan dukungan setiap infrastruktur.

c. Aman

   Kontainer mengisolasi aplikasi dari satu sama lain dan infrastruktur dasar memberikan lapisan tambahan perlindungan untuk aplikasi.

Kontainer memiliki perbedaan jika dibandingkan dengan mesin virtual (gambar 2.3). Setiap mesin virtual meliputi aplikasi, binari yang diperlukan dan *libraries* dan seluruh *guest* sistem operasi - yang semuanya mungkin memiliki ukuran puluhan GB.



Gambar 2.3 Perbedaan mesin virtual dengan *container*

Sumber: Newman, 2015

Sedangkan kontainer termasuk aplikasi dan semua yang tergantung padanya, tapi berbagi kernel dengan kontainer lainnya. Kontainer dijalankan sebagai proses yang terisolasi di *userspace* pada host sistem operasi. Kontainer juga tidak terikat pada infrastruktur tertentu - kontainer berjalan di komputer manapun, infrastruktur apapun dan di awan apapun. Kontainer memiliki isolasi sumber daya dan alokasi manfaat seperti mesin virtual tetapi pendekatan arsitektur yang berbeda memungkinkan untuk menjadi lebih portabel dan efisien. Pada Tabel 2.2, menunjukkan cara untuk membuat suatu kontainer yang memiliki sistem operasi Ubuntu dengan menginstal beberapa *library* PHP dan juga web server Nginx, kemudian kontainer dapat diakses melalui port 80.

Tabel 2.2 Contoh Dockerfile untuk membuat docker image

```
FROM ubuntu:trusty

RUN apt-get -y update
RUN apt-get -y install php5-fpm php5 php5-mcrypt php5-gd php5-cli php5-json
php5-curl curl php-pear php5-mysql nginx
RUN apt-get clean
RUN rm -rf /var/lib/apt/lists/*

EXPOSE 80
```

Tabel 2.3 Penulisan perintah pada *docker engine*

No	Perintah	Keterangan
1	docker pull IMAGE	Mengambil docker image dari repositori docker.
2	docker build [OPTIONS] PATH | URL | -	Membuat docker image berdasarkan Dockerfile
3	docker run [OPTIONS] IMAGE [COMMAND] [ARG...]	Menjalankan docker container.

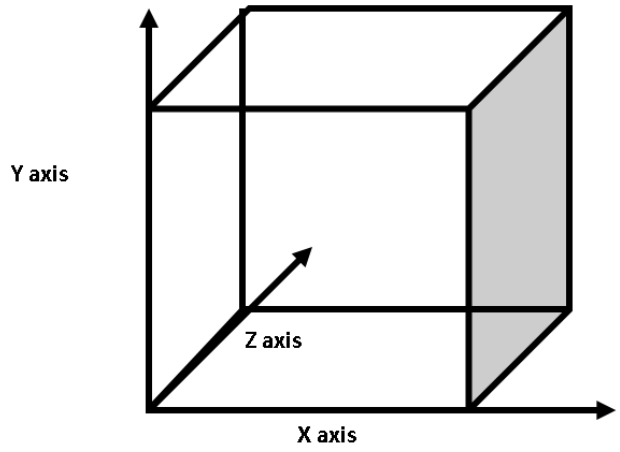
Beberapa perintah lain yang sering digunakan pada *docker engine* seperti pada Tabel 2.3. Perintah *docker pull* digunakan untuk mengambil *docker image* dari repositori resmi docker. *Docker build* digunakan untuk membuat *docker image* dari Dockerfile yang telah dibuat sebelumnya, dan *docker run* digunakan untuk menjalankan kontainer yang berisi *docker image*. Untuk memudahkan pengoperasian, manajemen serta monitoring kontainer, membutuhkan kerangka kerja orkestrasi dengan menggunakan beberapa perangkat lunak pendukung seperti:

- Apache Zookeeper sebagai *service discovery*, yaitu berfungsi sebagai penyimpanan informasi mengenai layanan yang telah terdaftar. Setiap kontainer akan didaftarkan agar dapat dilacak oleh Apache Mesos.
- Apache Mesos berfungsi sebagai pengatur distribusi dan menjalankan kontainer. Dengan menggunakan Apache Mesos, memungkinkan distribusi kontainer dalam bentuk *cluster*.
- Marathon sebagai manajemen kontainer seperti *deploy*, skala, dan monitoring kesehatan dari kontainer.

2.6 Ketersediaan

Tujuan dari *availability* (ketersediaan) yaitu meningkatkan ketersediaan suatu layanan sehingga tetap berjalan meskipun mendapatkan banyak permintaan dari klien (Nabi, et al., 2016). Mekanisme yang dapat dilakukan untuk meningkatkan ketersediaan yaitu *load balancing* (penyeimbang beban) atau *auto-scaling* (skala otomatis). Penyeimbang beban merupakan teknik untuk mendistribusikan beban lalu lintas pada dua atau lebih jalur koneksi secara seimbang, agar lalu lintas dapat berjalan optimal, memaksimalkan *throughput*, memperkecil waktu tanggap dan menghindari beban berlebih pada salah satu jalur koneksi. Pembagian beban digunakan pada saat sebuah server telah memiliki jumlah pengguna yang telah melebihi maksimal kapasitasnya.

Skala otomatis dapat membantu menjaga ketersediaan aplikasi dan memungkinkan aplikasi untuk di skala ke atas atau ke bawah secara otomatis sesuai dengan kondisi yang ditentukan. Kubus skala (Gambar 2.4) merupakan model tiga dimensi dari skalabilitas (Richardson & Smith, 2016). Pola arsitektur layanan mikro berhubungan dengan sumbu Y pada kubus skala.

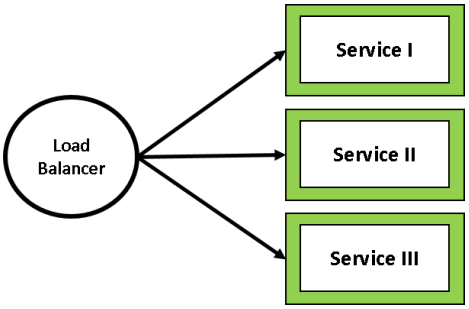


Gambar 2.4 Kubus Skala

Sumber: Richardson & Smith, 2016

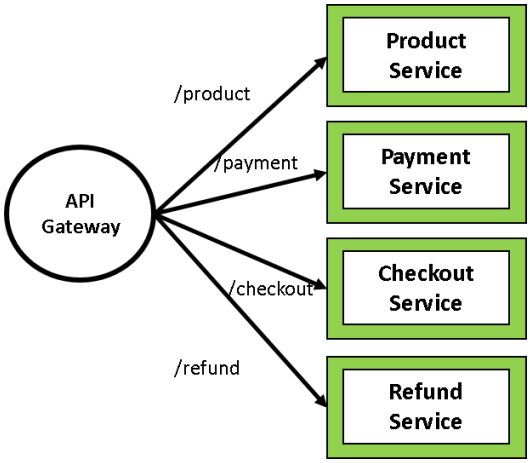
Pada skala sumbu X, menjalankan beberapa salinan identik dari aplikasi di belakang penyeimbang beban. Pendekatan yang umum digunakan untuk skala

aplikasi. Jika terdapat *n* salinan maka setiap salinan menangani 1 / *n* dari beban seperti pada gambar 2.5.



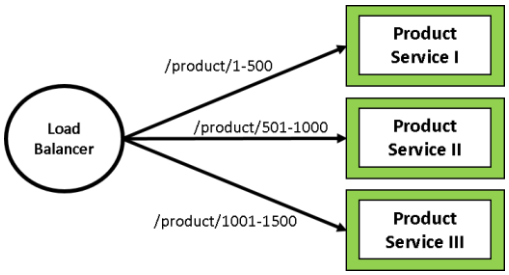
Gambar 2.5 Skala sumbu X

Skala sumbu Y membagi aplikasi menjadi beberapa aplikasi dengan layanan yang berbeda (Gambar 2.6). Setiap layanan bertanggung jawab menjalankan satu atau lebih fungsi yang terkait. Ada beberapa cara yang berbeda dari memecah aplikasi ke dalam layanan. Salah satu pendekatannya adalah dekomposisi dengan menggunakan kata kerja, menentukan layanan yang menerapkan kasus penggunaan tunggal seperti pembelian. Pendekatan lainnya yaitu dengan menggunakan kata benda, memiliki tanggung jawab untuk semua operasi yang berhubungan dengan entitas tertentu seperti manajemen pelanggan. Aplikasi mungkin menggunakan kombinasi dari dekomposisi berbasis kata benda dan kata kerja.



Gambar 2.6 Skala sumbu Y

Skala sumbu Z (Gambar 2.7), yang mana atribut dari permintaan digunakan sebagai rute permintaan ke server tertentu. Umumnya digunakan untuk skala pada basis data. Data dipartisi melalui beberapa server berdasarkan atribut dari permintaan.



Gambar 2.7 Skala sumbu Z

2.7 Pengolahan log akses

Pada umumnya, web server secara standar akan memiliki log berupa log akses ada log eror. Log akses berisi rekaman akses navigasi pada halaman web yang dikunjungi. Log akses menyimpan informasi mengenai permintaan dari klien seperti alamat IP, waktu, URL, metode HTTP, kode status, URL pengarah, dan agen pengguna (Gambar 2.8).

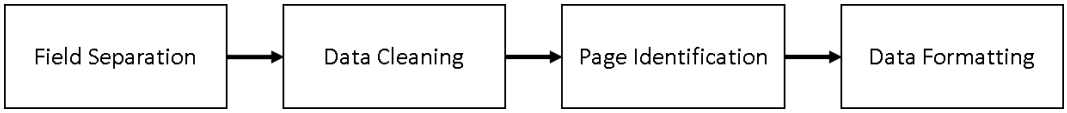
```
115.178.195.239 - - [13/Feb/2016:09:34:01 -0500] "GET
/apps/785/beacon HTTP/1.1" 200 6771
"https://example.com/apps/785/home" "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/48.0.2564.109 Safari/537.36"

115.178.195.239 - - [13/Feb/2016:09:34:11 -0500] "GET
/apps/785/beacon/add HTTP/1.1" 200 5563
"https://example.com/apps/785/beacon" "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/48.0.2564.109 Safari/537.36"
```

Gambar 2.8 Contoh Log Akses Standar

Setiap informasi pada satu baris log akses dipisahkan dengan menggunakan spasi. Pengolahan log akses dilakukan untuk mendapatkan informasi seperti ketersediaan layanan, identifikasi perilaku pengguna, ataupun deteksi serangan.

Setiap baris log akses terkadang menyimpan informasi mengenai file seperti gif, jpeg, png, dll. Oleh karena itu, pemilihan log akses perlu dilakukan untuk memperoleh data yang bersih. Pengolahan log akses menggunakan 4 tahap seperti pada Gambar 2.9 (Lopes & Roy, 2015):

1. *Field Separation*, untuk membagi data menjadi beberapa informasi yang dibutuhkan.
2. *Data Cleaning*, untuk menghapus data yang tidak diperlukan seperti gambar, *stylesheet*, video, dll.
3. *Page Identification*, untuk mengidentifikasi URL yang diakses dan kode status.
4. *Data Formatting*, untuk menghasilkan file keluaran dari hasil tiga tahap sebelumnya.



Gambar 2.9 Langkah-langkah pemrosesan log akses

Dalam setiap baris log akses, akan memiliki bermacam-macam respons kode status dari server. Ada lima klasifikasi kode status, yang dimulai dari angka 1 - 5. Kode dimulai dengan angka satu (1XX) bersifat informatif. Kode dimulai dengan angka dua (2XX) mengindikasikan bahwa permintaan oleh klien sudah diterima, dimengerti dan di proses dengan sukses. Kode dimulai dengan angka tiga (3XX) menangani semua masalah *Redirect.* Kode dimulai dengan angka 4 (4XX) menangani status error di klien. Dan kode dimulai dengan angka 5 (5XX) menangani status error di server.

a. 1XX

- **100**, *Continue*, server telah menerima request header yang dikirim, sehingga client dapat mengirimkan request body (contoh request POST atau GET).

- **101**, *Switching Protocols*, client meminta server untuk mengganti protokol, dan server menyetujui untuk mengganti protokol.
- **102**, *Processing*, server telah menerima request tetapi masih belum ada response yang tersedia karena request masih di proses.

b. 2XX

- **200**, *OK*, Response standard yang menyatakan bahwa request HTTP berhasil (OK).
- **201**, Created, Request telah selesai diproses dan suatu bentuk baru juga telah berhasil diciptakan.
- **202**, *Accepted*, Request telah diterima untuk di proses, tetapi proses tersebut belum selesai.
- **203**, *Non-Authorative Information*, Response code 203 mengindikasikan bahwa request telah diterima dan dipahami.
- **204**, *No Content*, Request telah diterima dan dimengerti oleh server, tetapi tidak ada informasi yang dapat di tampilkan ke client.
- **205**, *Reset Content*, Server telah selesai melakukan proses dari request dan user agent harus melakukan reset terhadap semua field pada form.
- **206**, *Partial Content*, Server telah selesai memproses sebagian request GET.

c. 3XX

- **300**, *Multiple Choices*, Kode status ini mengindikasikan bahwa halaman yang direquest tidak dapat lagi di akses dan sudah di pindahkan ketempat yang baru.
- **301**, *Moved Permanently*, Response dengan kode ini menunjukan bahwa informasi yang dipanggil sudah dipindahkan secara permanen ketempat yang baru oleh karena itu response server harus menyertakan link yang baru.

- **302**, *Found*, Sebuah status kode yang memberitahukan kepada client bahwa halaman website yang mereka kunjungi untuk sementara pindah ke lokasi yang baru tetapi dalam waktu yang tidak berapa lama, halaman website lama dapat di akses kembali.
- **303**, *See Other*, Kode status ini mengindikasikan bahwa respon dari request client dapat ditemukan pada URL yang sudah di tentukan. Ini tidak berarti halaman yang diakses telah dipindahkan, tetapi halaman tersebut memang hanya menyediakan alamat URL yang dapat di tampilkan ke client.
- **304**, *Not Modified*, Status ini di kirimkan oleh server jika informasi yang disimpan pada sisi client sama dengan informasi yang ada pada sisi server "Not Modified".
- **305**, *Use Proxy*, Kode ini menyatakan bahwa client yang akan mengakses halaman tersebut harus menggunakan proxy yang akan disertakan dalam respon header.
- **306**, *(Unused)*, Kode ini untuk semantara di reserve untuk penggunaan selanjutnya.
- **307**, *Temporary Redirect*, Kode ini identik dengan kode 302. Perbedaannya 307 tidak kompatibel dengan HTTP/1.0.

d. 4XX

- **400**, *Bad Request*, Karena syntax yang tidak dipahami oleh server, maka server mengembalikan kode status 400. Client tidak diperkenankan untuk mengirimkan request ulang ke server tanpa memperbaiki syntax.
- **401**, *Unauthorized*, Request dari user memerlukan otentifikasi. Ketika user mengisikan pasword yang tidak sesuai dengan yang dikenal server, maka server akan mengembalikan staus code tersebut.
- **402**, *Payment Required*, Kode status ini di reserve dan masih belum di gunakan sekarang.

- **403**, *Forbidden*, Kode status ini mengindikasikan bahwa halaman yang direquest oleh user merupakan halaman yang dilarang.
- **404**, *Not Found*, Client mengakses alamat yang tidak disediakan oleh server.
- **405**, *Method Not Allowed*, Kode 405 mengindikasikan bahwa client mencoba mengirimkan method yang tidak dikenali oleh server.
- **406**, *Not Acceptable*, Error Kode ini mengindikasikan bahwa response yang dikirim oleh server tidak dapat dimengerti oleh user agen/client.
- **407**, *Proxy Authentication Required*, Hampir sama dengan kode error 401, kode error ini berarti bahwa client harus melakukan verifikasi terlebih dahulu menggunakan proxy.
- **408**, *Request Timeout*, Server mencapai timeout limit menunggu request client yang terlalu lama.
- **409**, *Conflict*, Mengindikasikan bahwa request tidak dapat di proses. Sering terjadi data yang akan diproses bentrok dengan data yang sudah ada pada server.
- **410**, *Gone*, Hampir sama dengan kode error 404, kode error ini mengindikasikan bahwa halaman website yang diakses tidak tersedia secara permanent.
- **411**, *Length Required*, Server mengharuskan user menyertakan besar size dari content. Apabila client tidak menyertakannya, server akan mengembalikan error ini.
- **412**, *Precondition Failed*, Server tidak menemukan syarat-syarat yang disertakan request client.
- **413**, *Request Entity Too Large*, Request lebih besar dari batas limit size yang diperbolehkan oleh server.
- **414**, *Request-URI Too Long*, URI yang di panggil oleh client terlalu panjang.
- **415**, *Unsupported Media Type*, Tipe media yang di request oleh client tidak dikenali oleh server

- **416**, *Requested Range Not Satisfiable*, Client merequest sebagian bagian dari file, tetapi server tidak dapat menyediakan bagian tersebut.
- **417**, *Expectation Failed*, Server mengindikasikan bahwa request header dari client tidak memenuhi syarat minimal untuk di baca oleh server (Expect request-header).

e. 5XX

- **500**, *Internal Server Error*, Kode ini mengindikasikan bahwa server mengalami masalah dan tidak dapat memproses request yang diminta client.
- **501**, *Not Implemented*, Kode error ini muncul karena server tidak dapat memenuhi syarat minimum diperlukan untuk sebuar request diproses
- **502**, *Bad Gateway*, Error ini mengindikasikan bahwa server, biasanya merupakan proxy server, tidak menerima response yang valid dari upstream server. Upstream server adalah server yang memberikan service tertentu kepada server lain.
- **503**, *Service Unavailable*, Server untuk sementara waktu tidak tersedia (untuk sementara), biasanya ketika server sedang dalam keadaan maintenance atau sedang down.
- **504**, *Gateway Timeout*, Server, biasanya merupakan proxy server, tidak menerima response dari upstream server dalam kurum waktu yang sudah di tentukan. Upstream server adalah server yang memberikan service tertentu kepada server lain.
- **505**, *HTTP Version Not Supported*, Server tidak dapat membaca versi dari protokol HTTP yang di request oleh client.

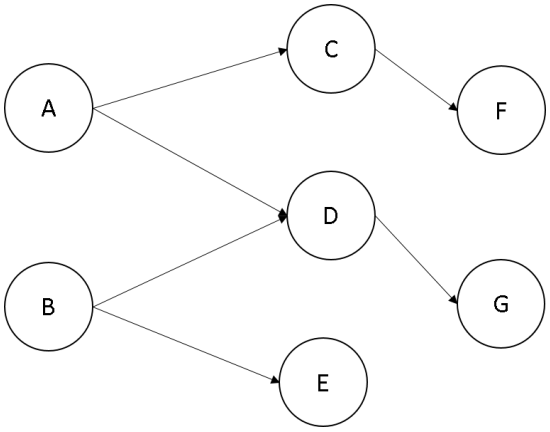
2.8 Graf Ketergantungan

Graf Ketergantungan (*Dependency Graph*) berbentuk *Directed Acyclic Graph* (DAG) yaitu graf yang memiliki arah tanpa siklus. DAG terdiri dari simpul dan

busur. Simpul merepresentasikan variabel, dan busur merepresentasikan adanya hubungan kebergantungan antara variabel yang dihubungkannya. Graf ketergantungan digunakan untuk mengetahui ketergantungan variabel yang satu dengan variabel lainnya. Pada Graf, terdapat beberapa terminologi dasar yang penting. Terminologi tersebut akan mendukung pembahasan pada makalah ini. Terminologi-terminologi itu antara lain :

a. Ketetanggaan
   Dua simpul dikatakan bertetangga apabila keduanya terhubung langsung.
b. Bersisian
   Sembarang sisi e(v j ,v k) dikatakan bersisian dengan simpul v j dan v k
c. Simpul Terpencil
   Simpul yang tidak memiliki sisi yang bersisian dengannya.
d. Graf Kosong
   Graf yang sisinya merupakan himpunan kosong
e. Derajat
   Jumlah sisi yang bersisian dengan suatu simpul
f. Lintasan
   Banyak barisan sisi dari suatu simpul ke simpul lainnya.
g. Siklus atau Sirkuit
   Lintasan yang berawal dan berakhir pada simpul yang sama
h. Terhubung
   Dua buah simpul terhubung jika terdapat lintasan dari satu simpul ke simpul yang lainnya

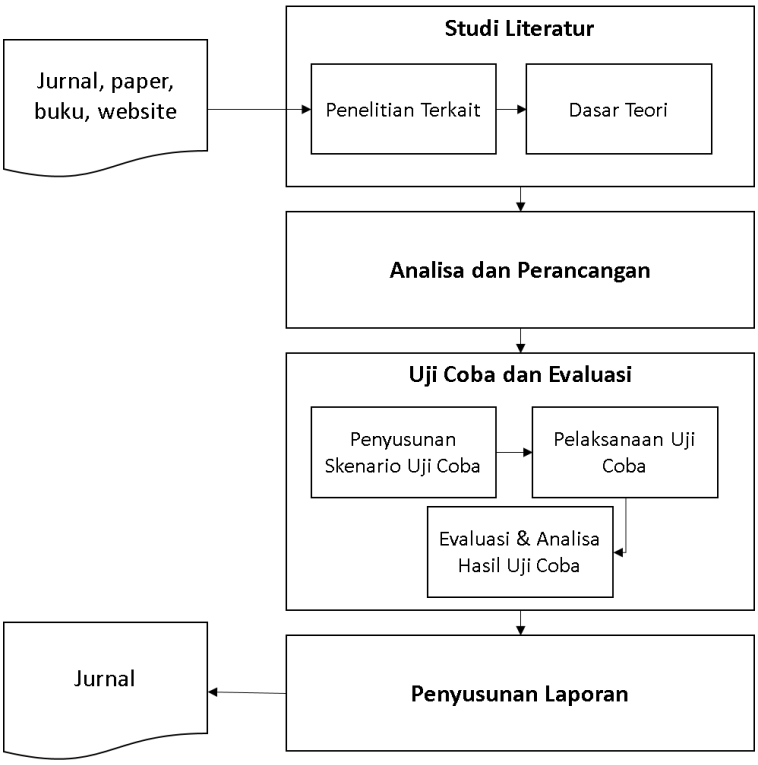
Sebagai contoh suatu aplikasi web dibangun berdasarkan kelas *A*, *B*, *C*, *D*, *E*, *F*, dan *G*. Selanjutnya dilakukan analisa ketergantungan pada tingkat kelas berdasarkan pemanggilan kelas pada masing-masing kelas. Sebagai contoh, kelas A memiliki ketergantungan terhadap kelas *C* dan *D*, kelas *B* memiliki ketergantungan dengan kelas *D* dan *E*, kelas *C* bergantung pada kelas *F* dan kelas *D* bergantung pada kelas *G* seperti pada Gambar 2.10. Setiap kelas merepresentasikan simpul dan panah merepresentasikan tepian pada grafik.



Gambar 2.10 Contoh grafik berdasarkan pemanggilan kelas

BAB III
METODE PENELITIAN

Secara umum, penelitian ini diawali dengan studi literatur meliputi penelitian terkait dan dasar teori, perumusan masalah beserta metode penyelesaian, uji coba evaluasi serta diakhiri dengan analisa. Sedangkan penulisan laporan tesis dimulai dari awal sampai akhir penelitian. Secara rinci, penelitian ini dirancang dengan urutan seperti pada Gambar 3.1.



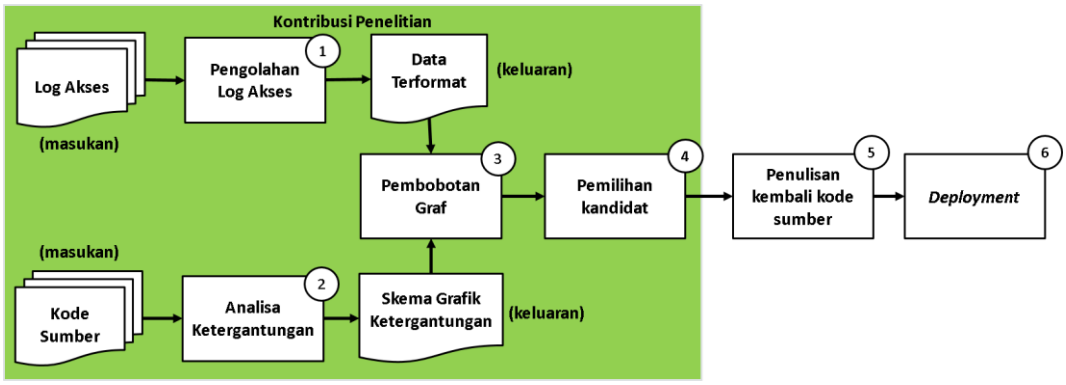
Gambar 3.1 Metodologi Penelitian

3.1 Studi literatur

Studi literatur merupakan tahap awal pada penelitian. Pada tahap ini, dilakukan pengumpulan informasi dari penelitian-penelitian sebelumnya yang diperoleh dari beberapa jurnal dan literatur lainnya. Pada proposal ini menggunakan beberapa literatur yang berkaitan dengan layanan mikro, pengolahan log akses, grafik ketergantungan, dan ketersediaan.

3.2 Analisa dan Perancangan Metode

Pada tahap ini dilakukan analisa mengenai perancangan solusi metode yang dibangun untuk menyelesaikan permasalahan dalam penelitian meliputi: rancangan detil langkah-langkah yang akan ditempuh dan kriteria-kriteria yang akan digunakan. Metode yang diusulkan meliputi enam proses seperti pada Gambar 3.2.



Gambar 3.2 Metode yang diajukan

3.2.1 Pengolahan Log Akses

Seluruh navigasi pada halaman web yang telah dikunjungi akan disimpan dalam format file log yang umumnya digunakan. Mencakup atribut seperti alamat IP, waktu, kode status, URL, metode HTTP, agen pengguna dan URL pengarah. Data yang diperoleh tidak terstruktur seperti pada Gambar 3.3, oleh karena itu perlu dilakukan pemrosesan terlebih dahulu. Kualitas data masukan yang baik perlu didapatkan untuk analisis yang lebih baik. Data yang berlebihan dihilangkan dengan menggunakan langkah-langkah yang digunakan oleh (Lopes & Roy, 2015). Hasil akhir tampak seperti pada Tabel 3.1.

a. *Field Separation*, setiap informasi pada baris log akses mentah dipisahkan dengan spasi (Gambar 3.3). Bertujuan untuk membedakan satu atribut dari atribut lain.
b. *Data Cleaning*, untuk menghapus data yang tidak diperlukan seperti gambar, *stylesheet*, video, dll.

c. *Page Identification*, untuk mengidentifikasi URL yang diakses dan kode status. Untuk kode status yang digunakan pada pengolahan log akses yaitu kode status 2XX dan 5XX. Dalam hal ini, kode status 2XX dimaksudkan memberi tahu bahwa permintaan telah diterima oleh klien. Sedangkan status kode 5XX dimaksudkan untuk mengetahui bahwa server tidak dapat melayani permintaan klien karena kesalahan pada sisi server.

d. *Data Formatting*, untuk menghasilkan file keluaran dari hasil tiga tahap sebelumnya. File keluaran dengan format Comma Separated (CSV). Hasil keluaran ini yang akan digunakan sebagai masukan pada proses pembobotan grafik.

```
125.160.197.53 - - [05/Apr/2016:01:27:46 -0400] "GET /users
HTTP/1.1" 200 39 "https://example.com/" "Mozilla/5.0 (Windows
NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/39.0.2171.95 Safari/537.36"

125.160.197.53 - - [05/Apr/2016:01:27:49 -0400] "PUT /products
HTTP/1.1" 500 238 "https://example.com/" "Mozilla/5.0 (Windows
NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/39.0.2171.95 Safari/537.36"
```

Gambar 3.3 Contoh log akses aplikasi

Tabel 3.1 Bagian-bagian dari log akses

No	Bagian	Keterangan
1	`125.160.197.53 - -`	Alamat IP
2	`[05/Apr/2016:01:27:46 -0400]`	Waktu
3	`"GET /users HTTP/1.1"`	Permintaan
4	`200`	Kode Status
5	`39`	Jumlah byte terkirim
6	`"https://example.com/"`	URL referer
7	`"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36"`	User Agent

Tabel 3.2 Contoh hasil pengolahan log akses

No.	HTTP Method	URL Permintaan	Kode Status
1.	GET	/users	200
2.	POST	/users	200
3.	GET	/users/1002	200
4.	GET	/users/1000	500
5.	GET	/users/1002	200
6.	GET	/Products	500
7.	POST	/Products	200
8.	GET	/checkout/230	200
9.	PUT	/shipping/200	500
10.	DELETE	/products/201	500

Setiap URL yang diakses memiliki pola yang berbeda tergantung dengan pola *routing* di dalam aplikasi. Halaman yang tidak sesuai dengan pola *routing* di dalam aplikasi tidak akan ditemukan dan menghasilkan kode status 404. Untuk pencocokan URL di dalam log akses dengan routing di dalam aplikasi, diperlukan pencocokan string dengan menggunaan *regular expression*. *Regular expression* atau ekspresi reguler adalah deretan karakter yang menentukan pola pencarian. Biasanya pola ini digunakan oleh algoritma pencarian string untuk menemukan karakter pada string. Karakter khusus pada Tabel 3.2 memiliki fungsi yang berbeda. Sebagai contoh *regular expression* untuk pencocokan string di dalam log akses pada Tabel 3.3.

Tabel 3.3 Karakter khusus dalam *regular expression*

Karakter	Arti
.	Cocok dengan sembarang satu karakter
*	Cocok dengan sembarang lebih dari karakter
^	Cocok dengan awal baris
$	Cocok dengan akhir baris
\<	Cocok dengan awal kata
\>	Cocok dengan akhir kata
[]	Cocok dengan salah satu karakter dalam kurung siku
[^]	Cocok dengan salah satu karakter yang tidak terdapat pada kurung siku
\	Karakter selanjutnya dianggap literal

Tabel 3.4 Contoh pencocokan string menggunakan *regular expression*

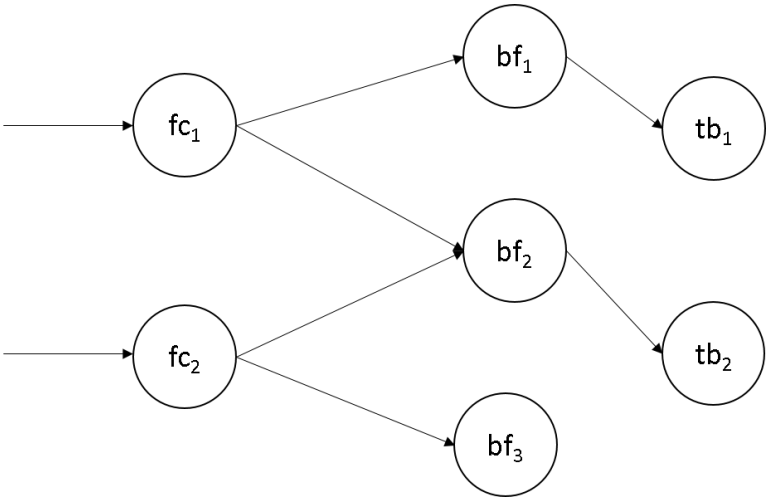
URL	Regular Expression
/login	^(/login)$
/confirmation/**bg8123kjasd**	^(/confirmation/**([a-zA-Z0-9])**\\w+)$
/password/reset/confirmation/**912kas23**	^(/password/reset/confirmation/**([a-zA-Z0-9])**\\w+)$
/apps/demo	^(/apps/demo)$
/apps/activate/11	^(/apps/activate/([0-9])\\w+)$

3.2.2 Analisa Ketergantungan

Dependency Graph (graf ketergantungan) digunakan untuk mengetahui ketergantungan antara simpul yang satu dengan simpul lainnya. Penelitian sebelumnya yang dilakukan oleh (Levcovitz, et al., 2015), membangun graf ketergantungan berdasarkan titik masuk, fungsi bisnis, dan tabel pada basis data. Grafik ketergantungan dari sistem direpresentasikan dengan (*F, B, D*), dimana $F = \{ fc_1, fc_2, ..., fc_n \}$ adalah set dari *facade*, $B = \{ bf_1, bf_2, ..., bf_n \}$ adalah set dari fungsi bisnis dan $D = \{ tb_1, tb_2, ..., tb_n \}$ adalah set tabel pada basis data.

Facade merupakan *entry point* (titik masuk) dari sistem yang memanggil fungsi bisnis. Fungsi bisnis merupakan fungsi yang mengolah aturan bisnis dan bergantung pada tabel basis data. Grafik ketergantungan (*V, E*) pada Gambar 3.4, di mana simpul merepresentasikan *facade* ($fc_i \in F$), fungsi bisnis ($bf_i \in B$), atau tabel basis data ($tb_i \in D$), dan tepian dapat merepresentasikan (i) pemanggilan fungsi dari *facade* ke fungsi bisnis, (ii) pemanggilan antar fungsi bisnis, dan (iii) akses dari fungsi bisnis ke tabel basis data.

Pada tahap ini grafik ketergantungan dibangun dengan menggunakan kakas bantu berdasarkan analisa kode sumber aplikasi monolitik. Alat bantu *PHPDependencyAnalysis* digunakan untuk membangun grafik secara otomatis pada tingkat pemanggilan kelas. Setiap kelas yang merepresentasikan *node* akan dikunjungi satu per satu untuk mendapatkan informasi mengenai ketergantungan



Gambar 3.4 Contoh grafik ketergantungan

dengan kelas lain. Perbedaan dengan penelitian sebelumnya yaitu grafik dibangun berdasarkan ketergantungan antar kelas.

Sebagai contoh suatu aplikasi sederhana dibangun menggunakan pemrograman berbasis obyek dengan bahasa pemrograman PHP yang menggunakan konsep *Model-View-Controller* (MVC). Selanjutnya dilakukan analisa ketergantungan pada tingkat kelas berdasarkan pemanggilan kelas seperti pada Gambar 3.5. Keluaran dari analisa dapat berupa file JSON yang akan digunakan untuk proses selanjutnya. Di dalamnya terdapat informasi mengenai grafik ketergantungan seperti tepian (*edges)* dan simpul (*vertices).* Dari kode sumber aplikasi, dilakukan analisa mengenai (*endpoints*) titik akhir yang dapat diakses. Contoh daftar titik akhir yang dapat diakses seperti pada Tabel 3.2.

Tabel 3.5 Contoh hasil pemetaan titik akhir pada kode sumber

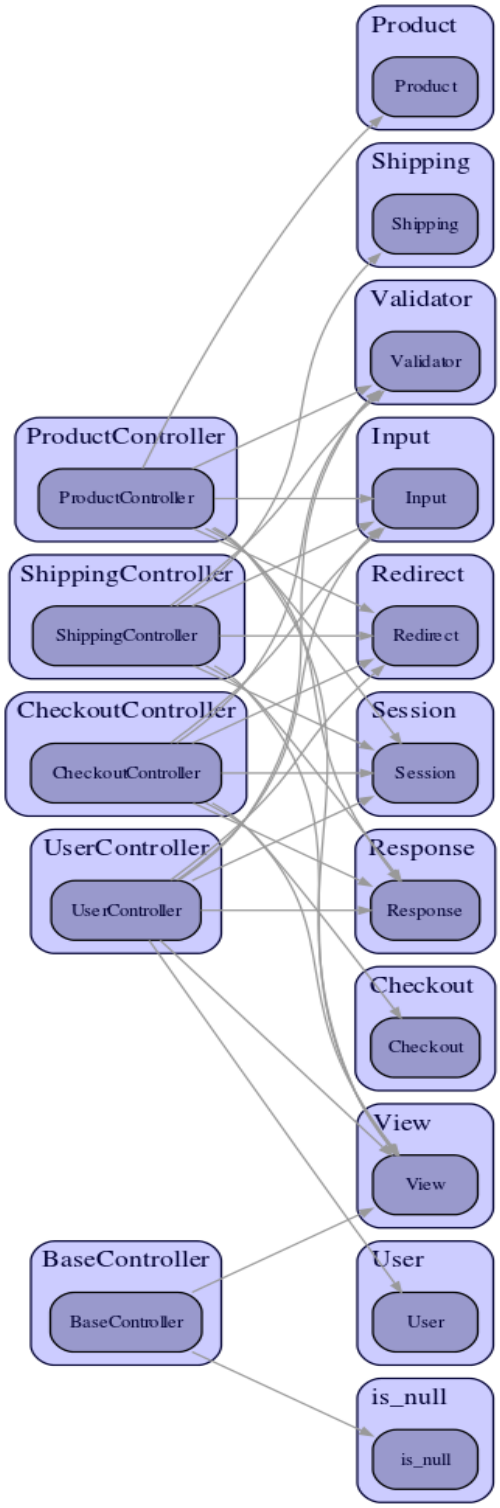
No.	HTTP Method	URL	Regular Expression	Kelas
1.	GET	/users	^(/user)$	UserController
2.	POST	/users	^(/user)$	UserController
3.	GET	/users/:id	^(/user/([0-9])\\w+)$	UserController
4.	PUT	/users/:id	^(/user/([0-9])\\w+)$	UserController
5.	DELETE	/users/:id	^(/user/([0-9])\\w+)$	UserController
6.	GET	/products	^(/products)$	ProductController

7.	POST	/products	^(/products)$	ProductController
8.	GET	/products/:id	^(/products/([0-9])\\w+)$	ProductController
9.	PUT	/products/:id	^(/products/([0-9])\\w+)$	ProductController
10.	DELETE	/products/:id	^(/products/([0-9])\\w+)$	ProductController
11.	GET	/checkouts	^(/checkouts)$	CheckoutController
12.	POST	/checkouts	^(/checkouts)$	CheckoutController
13.	GET	/checkouts/:id	^(/checkouts/([0-9])\\w+)$	CheckoutController
14.	PUT	/checkouts/:id	^(/checkouts/([0-9])\\w+)$	CheckoutController
15.	DELETE	/checkouts/:id	^(/checkouts/([0-9])\\w+)$	CheckoutController
16.	GET	/shipping	^(/shipping)$	ShippingController
17	POST	/shipping	^(/shipping)$	ShippingController

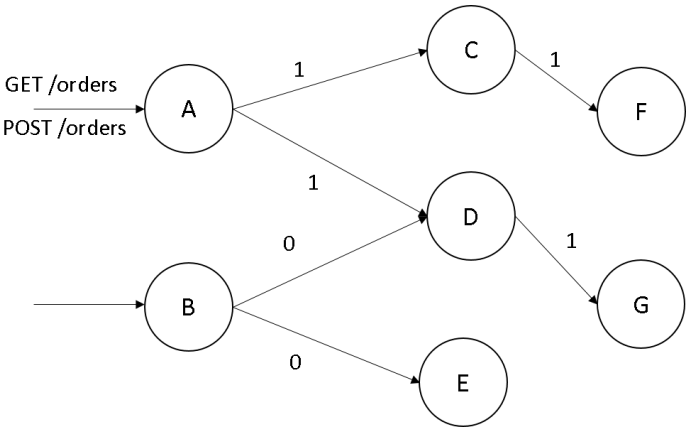
3.2.3 Pembobotan graf

Pada tahap ini, menggunakan dua masukan dari tahap pengolahan log akses dan analisa ketergantungan. Keluaran dari proses analisa ketergantungan akan diolah dengan menggunakan kakas bantu untuk memudahkan pembobotan grafik. Grafik yang terbentuk akan diberi bobot pada tepian grafik. Pemberian bobot berdasarkan pembacaan setiap data pada keluaran dari tahap pemrosesan awal data.

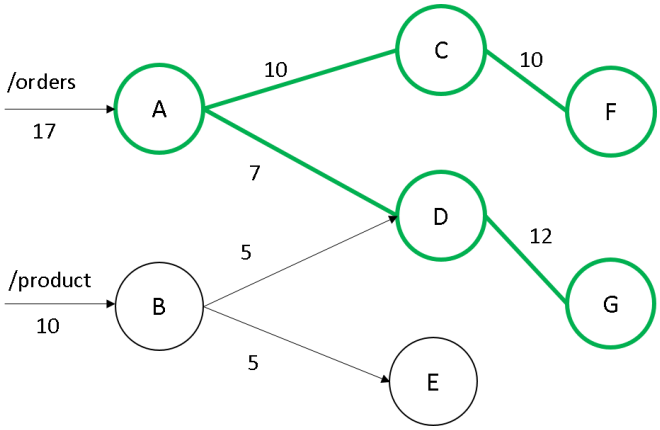
Sebagai contoh, terdapat satu baris log akses yang mengunjungi URL /*orders* dengan *method* GET pada titik awal *A*. Titik awal *A* dengan *method* GET pada grafik memiliki ketergantungan terhadap *C* dan C bergantung pada F. Maka nilai masing-masing simpul antara *A* dengan *C* dan C dengan *F* akan bertambah satu. Begitu pula dengan URL */orders* dengan method POST bergantung pada simpul D, dan D bergantung pada simpul G, maka nilai masing-masing tepi akan bertambah satu (Gambar 3.6). Proses pembobotan ini akan berulang sampai seluruh log akses terbaca seperti pada Gambar 3.7.



Gambar 3.5 Contoh grafik ketergantungan yang dihasilkan oleh kakas bantu



Gambar 3.6 Pembobotan grafik berdasarkan titik awal



Gambar 3.7 Contoh hasil akhir pembobotan grafik ketergantungan

Setelah proses pembobotan selesai, maka akan didapatkan graf seperti pada gambar 3.7. Titik awal A memiliki bobot lebih besar dibandingkan dengan titik awal B, dan simpul yang berhubungan dengan titik awal A juga akan mendapatkan bobot lebih besar. Dengan begitu dapat dilakukan pemilihan kandidat layanan mikro pada tahap selanjutnya. Jumlah URL yang diakses melalui titik awal sama dengan jumlah bobot keseluruhan setiap tepi yang bergantung pada titial awal tersebut.

3.2.4 Pemilihan kandidat

Pada tahap ini, pemilihan kandidat berdasarkan hasil akhir dari proses pembobotan. Simpul yang terpilih akan diidentifikasi terhadap titik masuk, sebagai contoh simpul yang memiliki bobot paling besar adalah *G*, maka titik masuk yang memiliki ketergantungan pada simpul *G* akan menjadi kandidat layanan mikro. Pada gambar 3.7, terdapat dua titik masuk yang memiliki ketergantungan terhadap simpul *G*, yaitu simpul *A* dan *B*. Maka simpul yang berada pada jalur dari simpul *A* ke *G* dan A ke F pada graf ketergantungan menjadi kandidat layanan mikro. Menurut (Gupta, 2015) dan (Richardson & Smith, 2016), kandidat layanan mikro dipilih berdasarkan kata kerja, kata benda atau kasus penggunaan. Setiap kandidat akan didefinisikan sebagai berikut:

- Nama: Nama layanan mikro, sebagai contoh *OrderService*
- Tujuan: Berisi tujuan utama dari layanan tersebut, sebagai contoh untuk memanipulasi data mengenai pemesanan.
- Masukan/Keluaran: Data masukan yang diperlukan dan data keluaran yang dihasilkan dari layanan mikro, sebagai contoh data masukan berupa ID produk dan hasil keluaran berupa informasi lengkap mengenai produk tersebut.

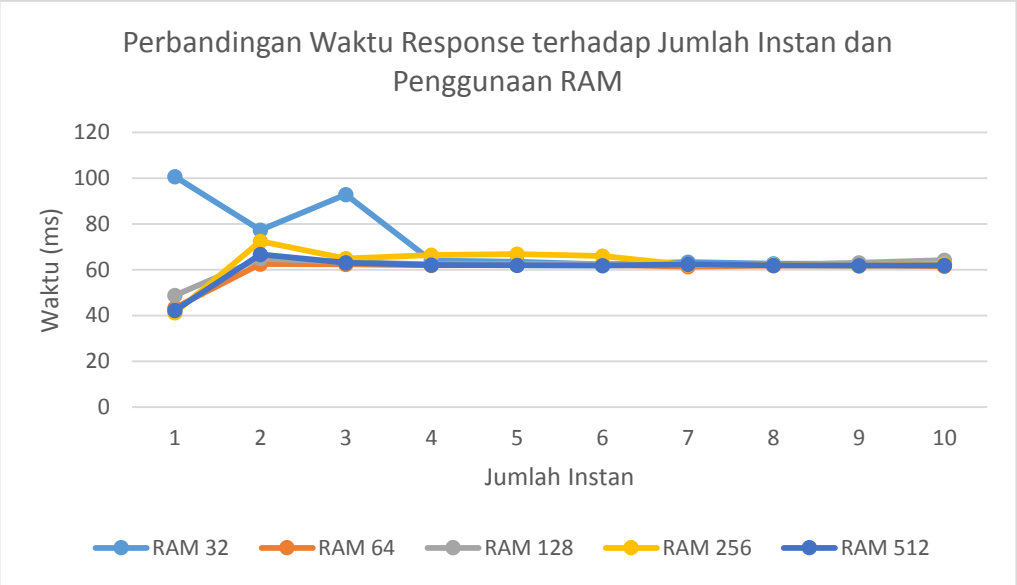
Setiap layanan mikro akan diberi batasan dalam penggunaan sumber daya seperti CPU dan RAM. Pembatasan ini akan mempengaruhi tingkat performa dalam memproses permintaan dari klien. Jumlah kandidat didapat dari pembagian sumber daya dengan jumlah instan pada setiap kandidat. Untuk mengetahui jumlah sumber daya yang dibutuhkan, dapat ditentukan dengan menggunakan persamaan (1).

$$N_{ms} = \frac{Rs}{Nx * Ra} \qquad (1)$$

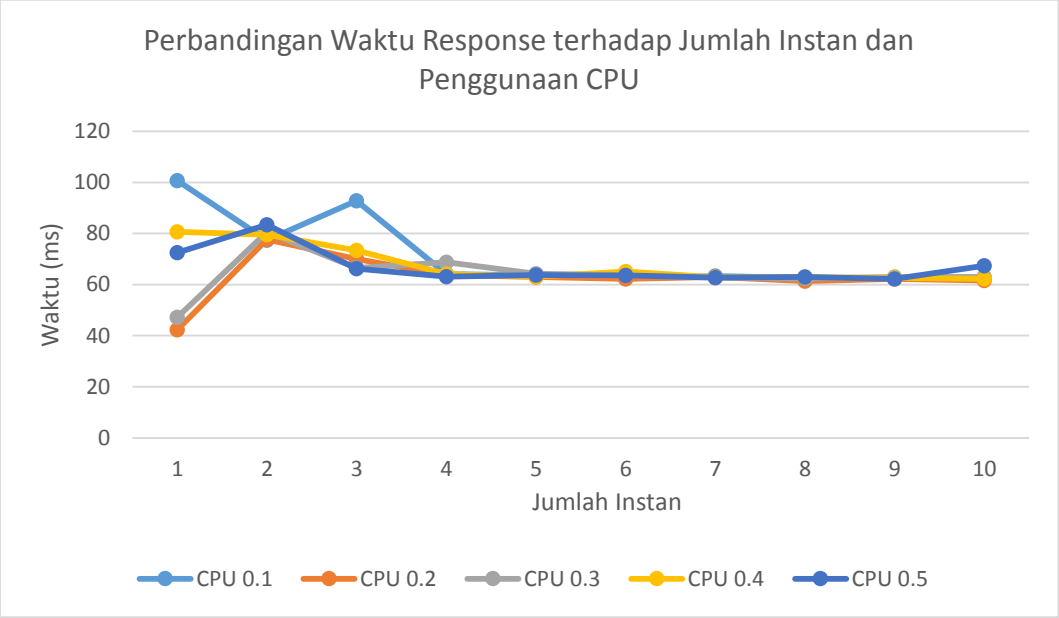
Di mana N_m merupakan jumlah layanan mikro yang akan terbentuk, R_s merupakan jumlah kuota sumber daya, N_x merupakan jumlah instans maksimal dari layanan, dan R_a merupakan jumlah alokasi sumber daya untuk setiap instan. Jika

server memiliki kuota sumber daya sebesar 4GB RAM dengan 4 CPU, dengan instans maksimal yang terbentuk adalah 4 instans dan penggunaan RAM sebesar 256 MB, maka layanan mikro yang akan terbentuk adalah 4 layanan.

Untuk mengetahui berapa jumlah sumber daya optimal untuk setiap instans dapat diketahui dengan melakukan percobaan melalui beberapa skenario. Skenario pertama yaitu membandingkan penggunaan RAM dan CPU dengan jumlah instan. Jumlah RAM dimulai dari 32, 64, 128, 256, dan 512 MB, jumlah CPU mulai 0.1 - 0.5 dan jumlah instan mulai 1-10. Pada Gambar 3.8 menunjukkan bahwa dengan jumlah RAM yang berbeda, memiliki dampak terhadap hasil rata-rata waktu respons. Jika dibandingkan antara 32 MB dengan 512 MB, RAM dengan jumlah paling besar memiliki konsistensi terhadap jumlah RAM yang lebih kecil. Pada Gambar 3.9 menunjukkan bahwa dengan jumlah CPU yang berbeda, memiliki dampak terhadap hasil rata-rata waktu respons. Jika dibandingkan antara 0.1 dengan 0.5, CPU dengan jumlah paling besar memiliki konsistensi terhadap jumlah CPU yang lebih kecil. Pada saat *n=1*, memiliki rata-rata waktu respons lebih rendah, hal ini disebabkan karena banyak permintaan yang tidak dapat dilayani. Rata-rata waktu respons tidak mengalami penurunan atau kenaikan yang signifikan dimulai dari *n=4*, maka jumlah instan sudah dalam keadaan optimal.



Gambar 3.8 Perbandingan waktu respons terhadap jumlah instan dan penggunaan RAM pada layanan mikro.



Gambar 3.9 Perbandingan waktu respons terhadap jumlah instan dan penggunaan CPU pada layanan mikro.

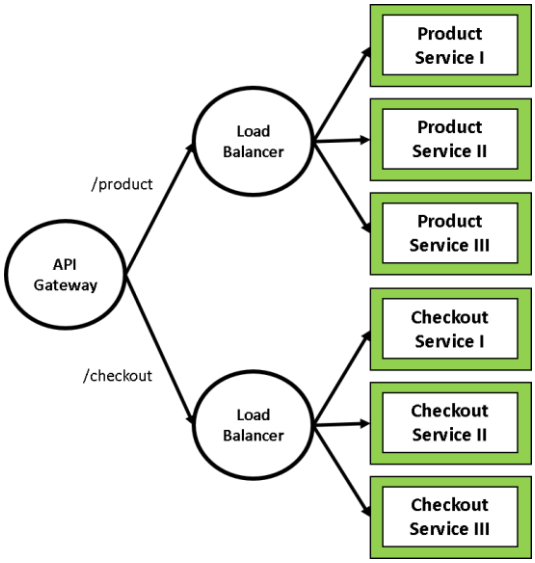
3.2.5 Penulisan kembali kode sumber aplikasi

Pada tahap ini, kandidat yang telah teridentifikasi pada proses sebelumnya akan dilakukan penulisan ulang kode. Setiap kandidat akan menjadi sebuah layanan yang mandiri, sehingga dapat diskala menurut kebutuhan permintaan. Pada tesis ini menggabungkan skala sumbu X dan Y untuk menerapkan kelebihan layanan mikro seperti pada Gambar 3.8. Dalam penulisan kembali kode sumber, menerapkan karakteristik dari layanan mikro seperti prinsip tanggung jawab tunggal, memiliki *interface* untuk diakses oleh layanan lain, dapat di skala secara mandiri.

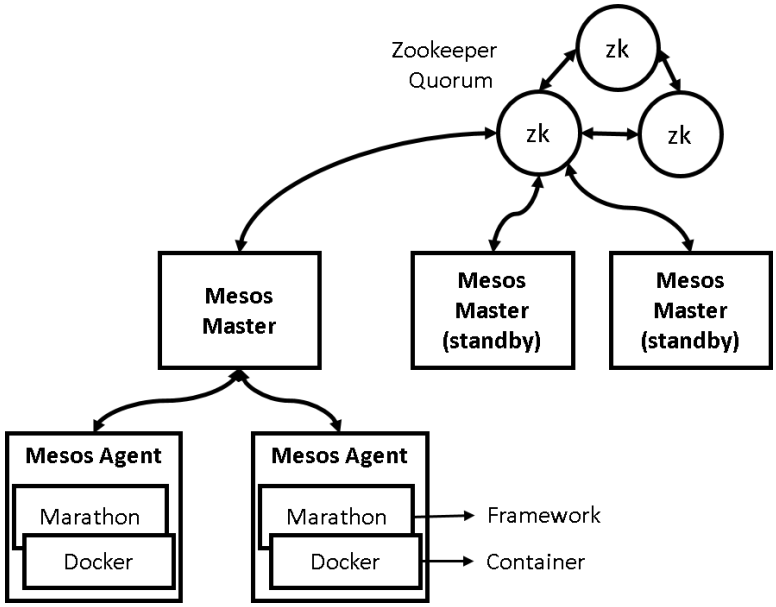
3.2.6 *Deployment*

Pada tahap ini, kode sumber yang telah ditulis ulang akan dipasang menggunakan kontainer. Untuk menjalankan orkestrasi kontainer pada proposal ini menggunakan sistem operasi Linux dengan beberapa perangkat lunak pendukung seperti Apache Zookeeper, Apache Mesos, dan Marathon. Apache Zookeeper berfungsi sebagai pencari layanan, Apache Mesos berfungsi sebagai pengatur

distribusi dan menjalankan kontainer, dan Marathon sebagai antarmuka dari kontainer yang telah dipasang Gambar 3.9.



Gambar 3.10 Contoh arsitektur layanan mikro dengan skala sumbu X dan Y



Gambar 3.11 Arsitektur *deployment* berbasis kontainer

3.3 Pengujian dan Analisa

Pembuatan kakas bantu uji coba dilakukan untuk mempercepat dan mempermudah pengujian dan analisis terhadap arsitektur yang diusulkan. Evaluasi dilakukan untuk mengetahui proses dan kinerja arsitektur dalam aspek *Performance Test*, *Average Response Time*, dan *Availability*. Pada pengujian performa juga didapatkan nilai 90% *line response* yaitu dengan cara mengurutkan seluruh data dari nilai terbesar hingga terkecil, menghapus 10% dari data, dan nilai 90% *response time* yaitu nilai yang terbesar dari sisa data. Grafik hasil pengujian setiap arsitektur akan dibandingkan untuk mendapatkan analisa. log akses dari hasil pengujian dianalisa kembali untuk mendapatkan perbandingan antara respons server yang sukses dan respons server yang gagal.

BAB IV
UJI COBA DAN PEMBAHASAN

Dalam bab ini diuraikan tahapan-tahapan yang dilakukan dalam implementasi arsitektur layanan mikro untuk menentukan tingkat ketersediaan dari suatu layanan. Dari hasil rekomendasi layanan mikro yang diekstraksi akan didapatkan beberapa layanan yang berdiri sendiri ataupun bergantung pada layanan lain. Dalam bab ini juga akan diuraikan mengenai penerapan skenario pengujian dan evaluasi metode dengan menganalisis data hasil pengujian yang telah dilakukan. Hasil analisis disajikan pada bagian akhir bab ini.

4.1 Tahapan Pengujian

Pengujian ini dilakukan dengan menggunakan lingkungan nyata. Tahapan implementasi metode penelitian yang digunakan secara umum pada bab ini terdiri dari tiga tahapan:

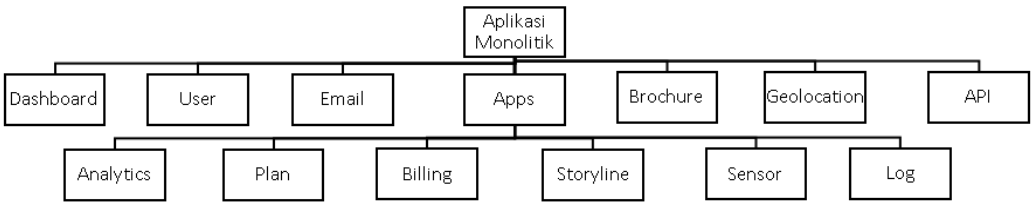
1. Tahap pertama adalah tahap perancangan dan implementasi. Perancangan dan implementasi ini menghasilkan beberapa layanan mikro yang di-*deploy* menggunakan kontainer, telah terintegrasi dengan layanan mikro lain dan layanan lain.
2. Tahap kedua adalah perancangan skenario pengujian yang digunakan untuk menguji performa, rata-rata waktu respons dan ketersediaan.
3. Tahap ketiga adalah melakukan pengujian keseluruhan sistem melalui uji coba pada lingkungan real.

4.2 Perancangan dan Implementasi

Tahapan perancangan dan implementasi ini meliputi informasi mengenai sistem yang diuji, variabel pengujian, lingkungan pengujian. Tahap pertama menentukan kasus uji coba yang dilakukan. Tahap kedua adalah menentukan variabel pengujian. Tahap terakhir adalah menentukan lingkungan pengujian yang digambarkan dengan topologi jaringan.

4.2.1 Sistem yang akan diuji coba

Sistem yang digunakan sebagai data uji merupakan sebuah sistem dari *Small Medium Enterprise* berupa aplikasi *Software As a Service* (SaaS) yang bertujuan untuk pengelolaan data sensor. Sistem telah *live production* dan dikembangkan dengan menggunakan kerangka kerja PHP Laravel 4.2. Setiap data akan dipanggil oleh klien melalui REST API yang telah disediakan. Pada Gambar 4.1 dan Tabel 4.1 menunjukkan modul-modul dan penjelasan yang terdapat di dalam sistem. Sistem memiliki 13 modul yang terdiri dari 51 kelas. Setiap modul yang terdapat dalam aplikasi menerapkan RESTful URL untuk mengaksesnya.



Gambar 4.1 Sitemap modul dari aplikasi yang akan diuji

Tabel 4.1 Daftar informasi modul pada data yang diuji

No.	Modul	Keterangan
1	Dashboard	Menampilkan statistik dan informasi dalam bentuk grafik
2	Analytics	Menampilkan jumlah interaksi pada sensor
3	User	Mengelola informasi mengenai pengguna
4	Email	Mengirimkan email notifikasi kepada pengguna
5	Plan	Memberikan pilihan paket pembayaran
6	Billing	Mengelola data pembayaran dari pengguna
7	Apps	Mengelola data aplikasi dari web/perangkat
8	Storyline	Mengelola data iklan berdasarkan jadwal
9	Brochure	Mengelola tayangan pada iklan
10	Sensor	Mengelola data sensor
11	Geolocation	Mengelola data lokasi pada sensor
12	Log	Menyimpan log dari sensor sebagai analisa
13	API	Menyediakan REST API untuk integrasi dengan sistem lain

4.2.2 Variabel Pengujian

Hasil evaluasi akan didapatkan dengan melakukan pengukuran *Performance Test, Average Response Time* dan *Availability*.

a. *Performance Test*

   Performance Test dilakukan untuk mengetahui kemampuan arsitektur yang diuji jika menerima beban berlebih, sehingga dapat diketahui pada keadaan yang seperti apa server dapat bekerja dengan normal. Server akan menerima banyak permintaan dalam waktu yang bersamaan. Dalam *Performance test* ini menggunakan Apache Bench sebagai alat bantu (Suzumura, et al., 2008).

b. *Average Response Time*

   Pengujian *response time* dilakukan untuk mengetahui seberapa lama server dalam melayani satu permintaan. Pengujian dilakukan dengan mengakses layanan yang telah menjadi layanan mikro.

c. *Availability*

   Ketersediaan merupakan probabilitas dari perangkat akan melakukan fungsi yang diperlukan tanpa kegagalan dalam kondisi persyaratan untuk jangka waktu tertentu. Sebelum ketersediaan sistem dapat ditentukan, ketersediaan perangkat lunak yang harus dipahami. Setiap perangkat lunak akan memiliki probabilitas kegagalan. Pengujian dilakukan dengan mematikan salah satu layanan ketika sedang dilakukan pengujian *Average Respons Time.*

4.2.3 Lingkungan Pengujian

Pengujian dilakukan pada lingkungan yang nyata dengan menggunakan beberapa perangkat yang terdiri dari empat buah komputer dan sebuah router untuk menghubungkan keempat komputer dalam satu jaringan.

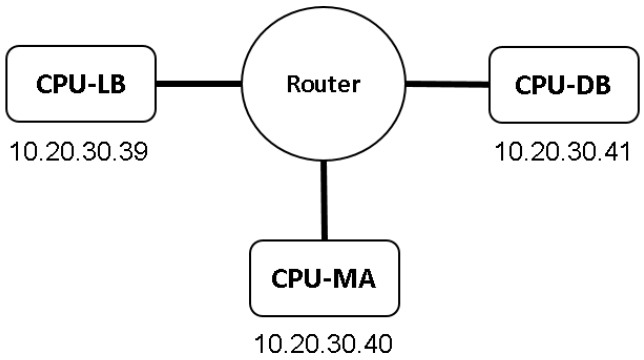
1. Tiga buah komputer (CPU-MA, CPU-DB, dan CPU-LB) yang memiliki spesifikasi berbeda untuk mewakili *node* master-agent, *node* basis data, *node* penyeimbang beban. Spesifikasi ketiga komputer dapat dilihat pada Tabel 4.2.

2. Sebuah *router* untuk menghubungkan ketiga komputer yang memiliki akses internet dan LAN.

Tabel 4.2 Spesifikasi Komputer Pengujian

No	Kode	CPU	RAM
1	CPU-MA	Intel Core™ i7	8GB
2	CPU-DB	Intel Core™ i3	4GB
3	CPU-LB	Intel Core™ i3	4GB

3. Semua komputer tersebut memiliki sistem operasi yang sama yaitu Ubuntu 14.04 LTS yang telah terinstal kerangka kerja Apache Zookeeper, Mesos, Marathon dan Docker.
4. Topologi jaringan pada lingkungan pengujian yang digunakan dapat dilihat pada Gambar 4.2.



Gambar 4.2 Topologi jaringan lingkungan pengujian

4.3 Skenario Pengujian

Pengujian dilakukan untuk membandingkan dua arsitektur, monolitik dan layanan mikro. Arsitktur monolitik di-*deploy* pada server tanpa menggunakan kontainer (*full resource*), sedangkan arsitektur layanan mikro menggunakan kontainer (*containerization*). Skenario yang diuji coba yaitu pengujian performa dengan melakukan *load test* pada arsitektur berdasarkan titik akhir yang tersedia. Pada pengujian performa juga akan didapatkan *Average Response Time* untuk mengetahui seberapa lama server dapat melayani permintaan. Pengujian juga

dilakukan pada beberapa server yang saling terkoneksi untuk mengetahui tingkat kemampuan layanan mikro ketika diskala.

4.4 Hasil uji coba dan analisis

Dari hasil pemilihan kandidat layanan mikro yang telah dilakukan, maka setiap layanan mikro akan di-*deploy* secara mandiri dengan menggunakan kontainer. Setiap layanan dapat diskala secara mandiri di belakang penyeimbang beban. Informasi mengenai kandidat layanan mikro yang terpilih dapat dilihat Tabel 4.3, Tabel 4.4, dan Tabel 4.5.

Tabel 4.3 Informasi Layanan Dashboard

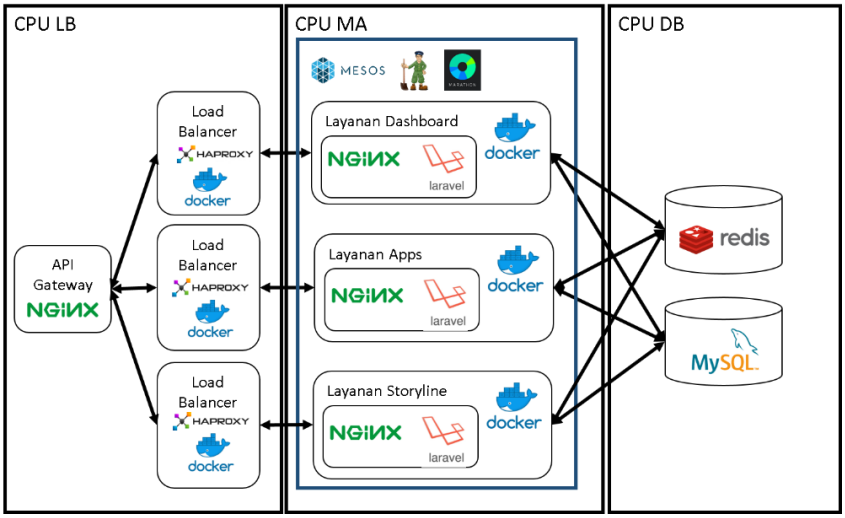
Nama	Layanan Dashboard
Tujuan	Memberikan informasi mengenai statistik berdasarkan data pada basis data
Masukan/Keluaran	Data masukan berupa ID dari aplikasi dan hasil keluaran berupa grafik.

Tabel 4.4 Informasi Layanan Apps

Nama	Layanan Apps
Tujuan	Manajemen data aplikasi untuk pengguna.
Masukan/Keluaran	Data masukan berupa nama aplikasi, jenis platform Mobile yang digunakan dan ID unik dari aplikasi.

Tabel 4.5 Informasi Layanan Storyline

Nama	Layanan Storyline
Tujuan	Manajemen skenario untuk menampilkan iklan pada aplikasi Mobile.
Masukan/Keluaran	Data masukan berupa nama skenario, jenis platform Mobile yang digunakan, dan ID unik dari aplikasi.

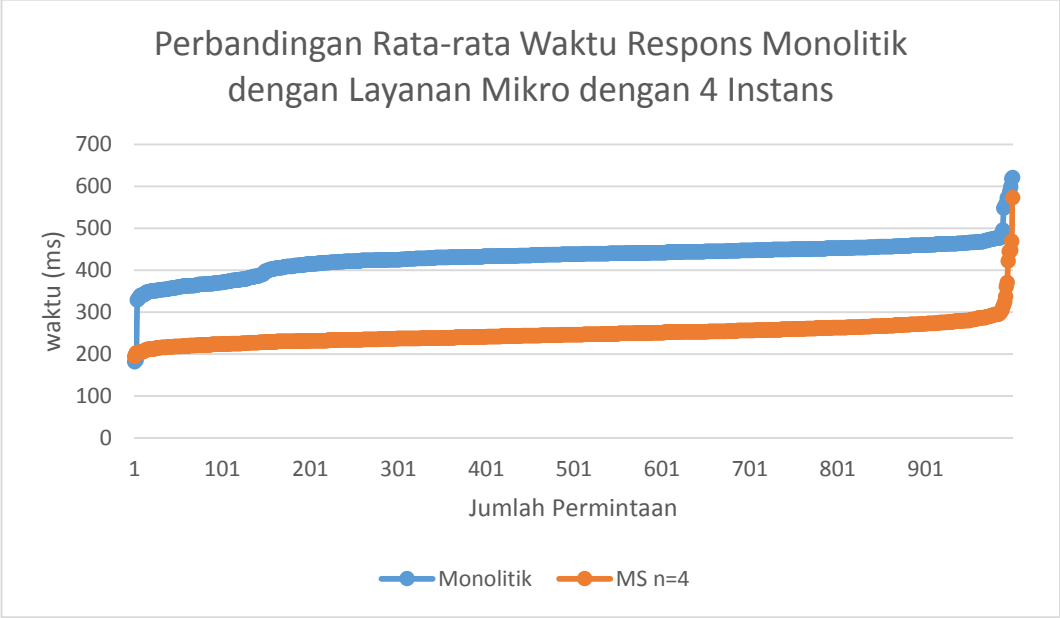


Gambar 4.3 *Deployment* arsitektur layanan mikro

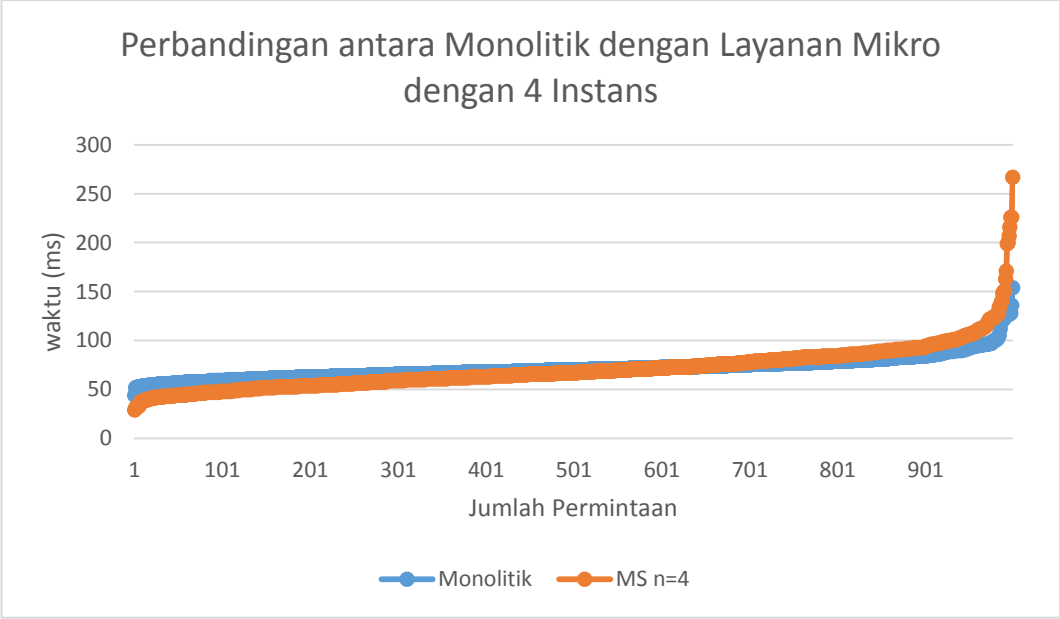
4.4.1 Hasil Uji Performa (*Performance Test*)

Hasil pengujian performa dilakukan untuk mengetahui kemampuan arsitektur ketika mendapatkan beban berlebih (Gambar 4.3). Setiap layanan mikro akan dibandingkan dengan arsitektur monolitik. Pengujian dilakukan dengan memberi beban sebanyak 1000 permintaan dengan tingkat konkuren 10. Pada Gambar 4.4, layanan mikro *dashboard* (*n*=4), persentase waktu respons meningkat 43,48% daripada arsitektur monolitik.

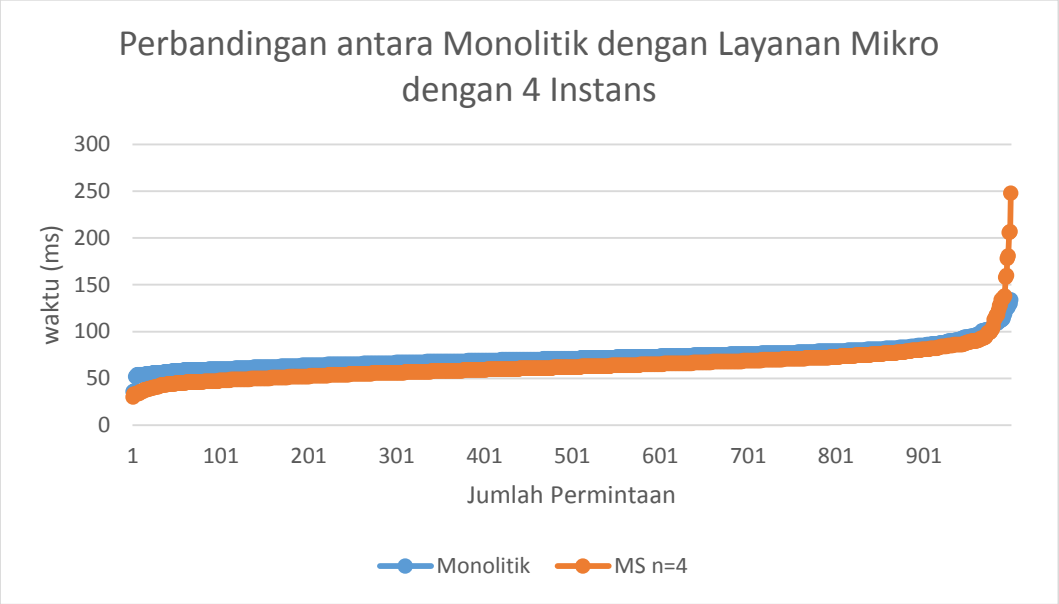
Pada layanan mikro *apps* (*n*=4) Gambar 4.4, arsitektur layanan mikro memiliki waktu respons 5,90% lebih cepat. Hal ini dikarenakan pada layanan apps tidak terdapat proses yang membutuhkan waktu lama. Sedangkan pada layanan mikro *storyline* (*n*=4), waktu respons meningkat 13,12% jika dibandingkan dengan arsitektur monolitik. Semakin besar tingkat konkurensi maka setiap instan akan mendapatkan beban yang merata.



Gambar 4.4 Grafik pada layanan mikro *dashboard*



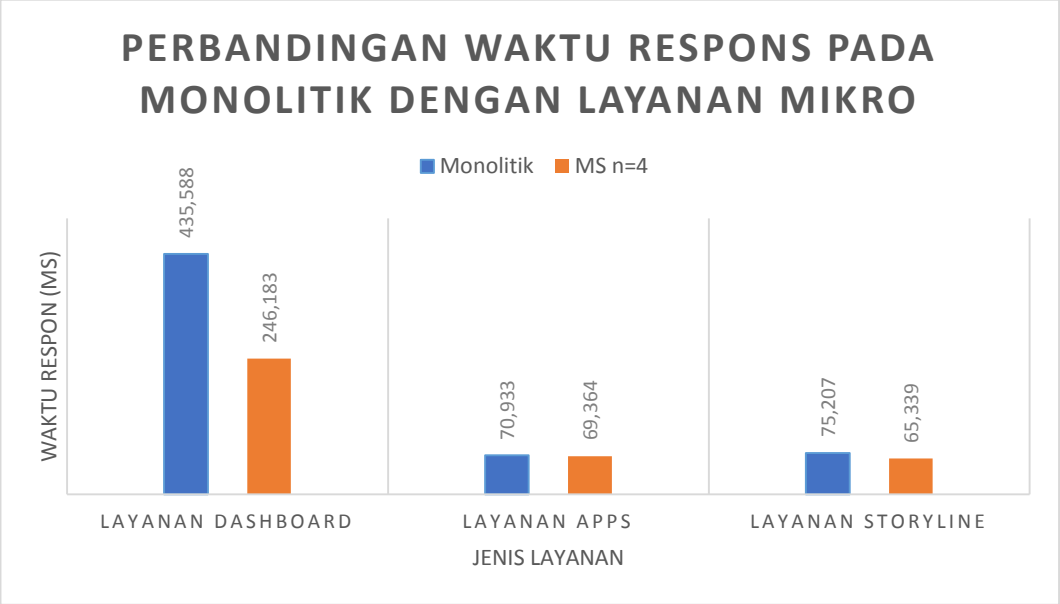
Gambar 4.5 Grafik pada layanan mikro *apps*



Gambar 4.6 Grafik pada layanan mikro *storyline*

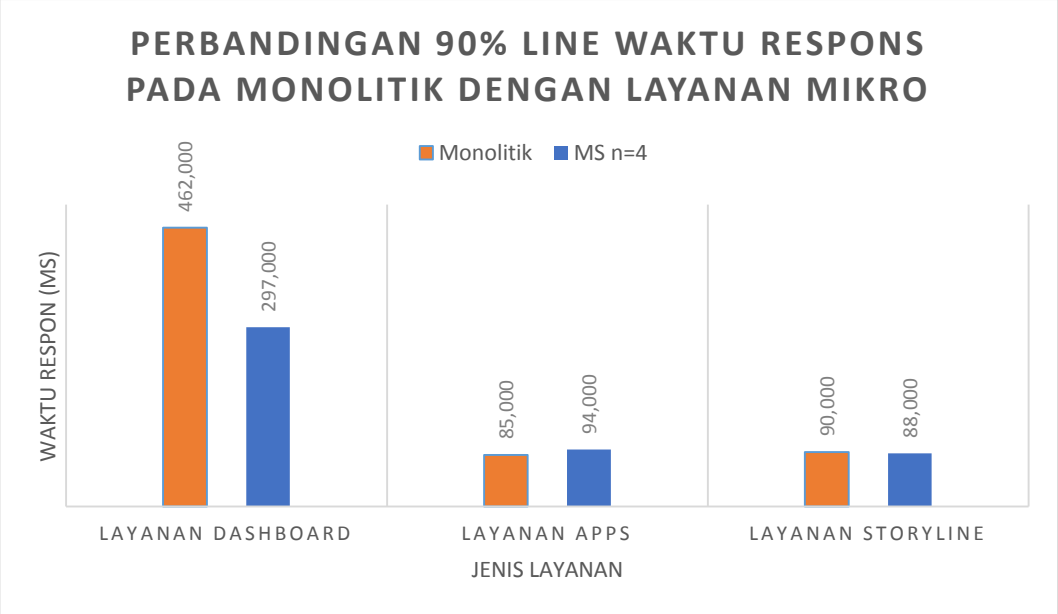
4.4.2 Hasil Uji Rata-Rata Waktu Respons (Average Response Time)

Hasil pengujian rata-rata waktu respons dilakukan untuk mengetahui berapa lama kemampuan arsitektur untuk memberikan layanan. Dengan memberikan beban sebanyak 1000 permintaan pada tingkat konkurensi 10 seperti pada pengujian performa. Nilai rata-rata yang paling sedikit menunjukkan bahwa layanan memiliki waktu respons lebih cepat. Pada Gambar 4.7, layanan mikro dashboard (n=4), rata-rata waktu respons lebih cepat daripada arsitektur monolitik. Pada layanan mikro apps (n=4) memiliki waktu respons lebih cepat daripada arsitektur monolitik. Dan pada layanan mikro storyline (n=4), rata-rata waktu respons lebih cepat jika dibandingkan dengan arsitektur monolitik.



Gambar 4.7 Grafik perbandingan waktu respons pada monolitik dengan layanan mikro

Pada Gambar 4.8 menunjukkan perbandingan 90% line waktu respons pada monolitik dengan layanan mikro. Pengukuran ini dilakukan untuk mengetahui nilai terbesar dalam 90% data. Pada layanan *Dashboard* dan layanan *Storyline*, memiliki nilai yang lebih rendah, menunjukkan bahwa proses *request-response* tidak mengalami anomali seperti overhead.



Gambar 4.8 Grafik perbandingan 90% Line waktu respons pada monolitik dengan layanan mikro

4.4.3 Hasil Uji Ketersediaan (Availability)

Hasil pengujian ketersediaan dilakukan untuk mengetahui tingkat data dapat dilayani oleh arsitektur monolitik dan arsitektur layanan mikro. Pada pengujian ini dilakukan analisa terhadap seluruh permintaan dapat dilayani dengan kode status 200 (OK) dan kode status selain 200 atau web server tidak dapat melayanai sama sekali. Tingkat konkurensi dinaikan secara bertahap dan memulai ulang web server ketika permintaan sedang berjalan. Untuk layanan mikro akan dilakukan pengurangan instan. Pada Tabel 4.6 menunjukkan bahwa antara arsitektur monolitik dan arsitektur layanan mikro dapat menangani permintaan tanpa mengalami kegagalan dengan 1000 permintaan ketika web server tersedia. Tetapi ketika web server tidak tersedia atau dalam keadaan *restart*, beberapa permintaan mengalami kegagalan (Tabel 4.7).

Tabel 4.6 Hasil pengujian ketersediaan ketika web server tersedia

No.	**Req**	**Con**	**Monolitik**		**Layanan Mikro (*n*=4)**	
			Sukses	**Gagal**	**Sukses**	**Gagal**
1	1000	10	1000	0	1000	0
2	1000	20	1000	0	1000	0
3	1000	30	1000	0	1000	0
4	1000	40	1000	0	1000	0
5	1000	50	1000	0	1000	0

Tabel 4.7 Hasil pengujian ketersediaan ketika web server tidak tersedia

No.	**Req**	**Con**	**Monolitik**		**Layanan Mikro (*n*=4)**	
			Sukses	**Gagal**	**Sukses**	**Gagal**
1	1000	5	454	546	987	12
2	1000	10	462	538	989	11
3	1000	20	455	545	983	17
4	1000	30	467	533	998	2
5	1000	40	464	536	977	23
6	1000	50	489	511	999	1

Pada arsitektur layanan mikro jumlah kegagalan lebih sedikit dibandingkan dengan arsitektur monolitik. Hal ini dikarenakan penyeimbang beban pada layanan mikro yang diuji mengalami *restart* untuk mendapatkan konfigurasi yang terbaru. Berbeda halnya dengan arsitektur monolitik, ketika web server di matikan, akan mematikan seluruh proses pada aplikasi.

[Halaman ini sengaja dikosongkan]

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil pengujian dan analisis hasil yang telah dilakukan, maka dapat diambil kesimpulan sebagai berikut:

1. Untuk mengidentifikasi kandidat layanan mikro dari sistem yang menggunakan pola arsitektur monolitik dapat dilakukan dengan mengekstraksi aplikasi web dengan pola arsitektur monolitik menjadi layanan mikro yaitu dengan menggunakan identifikasi log akses. Pengolahan akses log memiliki peranan untuk menentukan layanan mana saja yang akan menjadi layanan mikro. Layanan yang mendapatkan permintaan paling banyak dari klien akan menjadi kandidat layanan mikro. Layanan yang terpilih akan dilakukan pengujian performa dan pengujian ketersediaan.
2. Untuk meningkatkan ketersediaan dengan menggunakan pola arsitektur layanan mikro dapat dilakukan dengan menerapkan mekanisme penyeimbang beban. Untuk membuktikan mekanisme tersebut, dilakukan pengujian performa dan didapatkan nilai rata-rata waktu respons. Nilai persentase rata-rata waktu respons pada arsitektur layanan mikro *dashboard* dengan n=4 memiliki persentase lebih besar yaitu 43,48% jika dibandingkan dengan arsitektur monolitik. Begitu pula pada layanan mikro *apps* dan *storyline,* masing-masing 5,90% dan 13,12%. Dengan nilai persentase waktu respons yang besar menunjukkan bahwa semakin banyak permintaan dari klien yang dapat dilayani. Persentase waktu respons yang besar dipengaruhi oleh penggunaan penyeimbang beban. Sehingga beban setiap layanan merata dan ketersediaan dari layanan terus ada walaupun dengan jumlah permintaan yang besar. Dapat disimpulkan bahwa metode yang diusulkan dalam penelitian ini mampu meningkatkan ketersediaan layanan terhadap klien

3. Untuk mengevaluasi ketersediaan dari arsitektur layanan mikro, dilakukan dengan cara menguji ketersediaan dengan simulasi mematikan layanan ketika dalam proses pengujian performa. Dari hasil pengujian dapat diambil kesimpulan bahwa arsitektur layanan mikro memiliki tingkat ketersediaan lebih tinggi daripada arsitektur monolitik. Sehingga arsitektur yang diajukan pada penelitian ini mampu melayani permintaan dari klien jika dibandingkan dengan arsitektur monolitik dalam satu perangkat keras yang sama.

5.2 Saran

Perkembangan pola arsitektur layanan mikro semakin berkembang dari tahun ke tahun. Hal ini dikarenakan arsitektur ini memiliki kemampuan yang lebih baik dibandingkan dengan pola arsitektur monolitik. Salah satu topik yang menarik dalam penerapan pola arsitektur ini adalah dekomposisi dari arsitektur monolitik ke arsitektur layanan mikro. Pemilihan layanan yang akan di dekomposisi menjadi layanan mikro. Selain itu proses *deployment* dari kode sumber ke dalam kontainer membutuhkan waktu lebih banyak untuk memastikan layanan dapat berjalan dengan baik dan juga terdistribusi ke dalam beberapa server.

DAFTAR PUSTAKA

Fowler, M. & Lewis, J., 2014. Microservices. [Online] Available at: http://www.martinfowler.com/articles/microservices.html [Diakses 18 February 2016].

Gupta, A., 2015. Getting Started with Microservices. Moving Towards Microservices? Don't Forget DevOps, pp. 215-219.

Levcovitz, A., Terra, R. & Valente, M. T., 2015. Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems. s.l., s.n.

Lopes, P. & Roy, B., 2015. Dynamic Recommendation System Using Web Usage Mining for E-commerce Users. Precedia Computer Science 45, Volume 45, pp. 60-69.

Nabi, M., Toeroe, M. & Khendek, F., 2016. Availability in the cloud: State of the art. Journal Network and Computer Applications, Volume 60, pp. 54-67.

Newman, S., 2015. Building Microservices. California: O'Reilly Media Inc..

Richardson, C. & Smith, F., 2016. Microservices From Design to Deployment. s.l.:NGINX.Inc.

Suzumura, T. et al., 2008. Performance Comparison of Web Service Engines in PHP, Java, and C. s.l., IEEE International Conference on Web Services.

Villamizar, M. et al., 2015. Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud. Bogota, IEEE, pp. 583-590.

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Ardyanto Hermawan, lahir di Magetan, 3 Agustus 1990. Anak ketiga dari 4 bersaudara. Penulis menempuh pendidikan formal di SD Negeri Blimbing 3 Malang, SMP Negeri 3 Malang, SMA Negeri 8 Malang, Diploma III Program Studi Manajemen Informatika Politeknik Negeri Malang, melanjutkan pendidikan Diploma IV Teknik Informatika di Politeknik Elektronika Negeri Surabaya. Penulis bekerja sebagai *Developments and Operations Engineer* di PT. Eyro Digital Teknologi. Pada tahun 2014 penulis melanjutkan pendidikan S2 di Teknik Informatika, Institut Teknologi Sepuluh Nopember Surabaya. Penulis dapat dihubungi melalui email: ardyanto.hermawan@gmail.com.

[Halaman ini sengaja dikosongkan]

LAMPIRAN

1. Potongan kode untuk melakukan pembersihan, pengolahan, dan perhitungan log akses dengan keluaran berupa file CSV.

```
#!/bin/bash

# Penggunaan: /bin/bash accesslog.sh {$nama_domain}
{$folder_input} {$folder_output}

FILTER="$1"
INPUT_DIR="$2"
OUTPUT_DIR="$3"

for OUTPUT in $(ls $INPUT_DIR/)
do
      cat $INPUT_DIR/$OUTPUT | grep $FILTER | grep -vwE
"(css|jpg|jpeg|js|eot|png|ttf|woff|ico|robot|null|svg)" | awk
'{ print $6, $7 }' | sort | uniq -c | sort -rn | awk '{
printf("%d,%s\",\"%s\"\n", $1, $2, $3) }' >
$OUTPUT_DIR/$OUTPUT.csv
done
```

2. File konfigurasi untuk membuat graf ketergantungan secara otomatis.

```
mode: 'call'
source: './source-code/app'
filePattern: '*.php'
ignore: 'tests'
formatter: 'PhpDA\Writer\Strategy\Json'
target: 'source-code/app'case-study.json'
groupLength: 3
visitor:
  - PhpDA\Parser\Visitor\TagCollector
  - PhpDA\Parser\Visitor\SuperglobalCollector
```

3. Potongan kode untuk membuat graf ketergantungan secara otomatis.

```
#!/bin/bash

# Penggunaan: /bin/bash dependency-graph.sh
{$folder_aplikasi} {$file_konfigurasi}

SOURCE="$1"
CONFIG="$2"

/usr/bin/php $SOURCE/vendor/mamuz/php-dependency-
analysis/bin/phpda analyze $CONFIG
```

4. Kode untuk membangun *docker image* dari hasil pembobotan.

```
#!/bin/bash

# Penggunaan: /bin/bash deployment.sh {$nama_service}
{$folder_dockerfile}

SERVICE=$1
DIR=$2

docker build -t tesis/$SERVICE:latest $DIR
```

5. Kode untuk melakukan pengujian setiap layanan mikro dan monolitik.

```
#!/bin/bash

#penggunaan: bash request.sh container service-apps 1000 10
http://localhost:8200/apps/demo 2 0.1 64 8200

ACTION="$1"
SERVICE="$2"
NUM="$3"
CON="$4"
URL="$5"
INSTANCE="$6"
CPU="$7"
MEMORY="$8"
SERVICEPORT="$9"
COOKIE="cookie_string"

if [[ "$ACTION" = "host" ]]; then
      printf 'starting to test monolithic\n'
      mkdir -p logs/$NUM-$CON-$CPU-$MEMORY/$SERVICE;
      mkdir -p data/$NUM-$CON-$CPU-$MEMORY/$SERVICE;
      ab -n $NUM -c $CON -C $COOKIE -g data/$NUM-$CON-$CPU-
$MEMORY/$SERVICE/monolitik.data $URL > logs/$NUM-$CON-$CPU-
$MEMORY/$SERVICE/monolitik.txt
      printf 'cooling down in 5 seconds\n'
      sleep 5;
fi

if [[ "$ACTION" = "container" ]]; then
      mkdir -p logs/$NUM-$CON-$CPU-$MEMORY/$SERVICE;
      mkdir -p data/$NUM-$CON-$CPU-$MEMORY/$SERVICE;

      printf 'scaling instances '$SERVICE' to '$INSTANCE'\n'
      curl -X PUT -H "content-type: application/json" -d
```

```
'{"id":"'$SERVICE'","cpus":'$CPU',"mem":'$MEMORY',"instances"
:'$INSTANCE',"container":{"type":"DOCKER","docker":{"image":"
tesis/'$SERVICE':latest","network":"BRIDGE","portMappings":[{
"containerPort":80,"hostPort":0,"servicePort":'$SERVICEPORT',
"protocol":"tcp","labels":{}}],"privileged":false,"parameters
":[{"key":"add-host","value":"nuc:10.20.30.41"},{"key":"add-
host","value":"ardyanto-
m6:10.20.30.40"}],"forcePullImage":false}},"labels":{"HAPROXY
_GROUP":"'$SERVICE'"}}'
http://localhost:8080/v2/apps/$SERVICE
      printf '\n'
      printf 'wait 5 seconds to make sure load balancer is
reload\n'
      sleep 5;
      printf 'starting to test microservices container\n'
      ab -n $NUM -c $CON -C $COOKIE -g data/$NUM-$CON-$CPU-
$MEMORY/$SERVICE/layanan-mikro-$INSTANCE.data $URL >
logs/$NUM-$CON-$CPU-$MEMORY/$SERVICE/layanan-mikro-
$INSTANCE.txt
      printf 'cooling down in 5 seconds\n'
      sleep 5;
fi
```

6. Potongan kode untuk API pemrosesan log akses.

```
<?php
$router->post('/accesslog/process', function() use ($request,
$response){
      try {
            $dir = $request->request-
>get('accessLogDirectory');
            $filter = $request->request->get('filter');
            exec("/bin/bash command/access-log-processing.sh
{$filter} {$dir}/raw {$dir}/processed");
            exec("ls {$dir}/processed", $data);
            $response->setStatusCode(Response::HTTP_OK);
      } catch (Exception $e) {
            $data = array(
                              'error' => true,
                              'message' => $e->getMessage()
                        );
            $response-
>setStatusCode(Response::HTTP_BAD_REQUEST);
      }
      $response->setContent(json_encode($data));
      $response->headers->set('Content-Type',
'application/json');
      return $response->sendHeaders()->getContent();
});
```

7. Potongan kode untuk API pemrosesan source code.

```
<?php
$router->post('/sourcecode/process', function() use
($connection, $request, $response){
	try {
		$dir = $request->request-
>get('sourceCodeDirectory');
		$contents = new Analyze($connection, "case-
study.json", "storage.json", "routes.json");
		$contents->storeGroups();
		$contents->storeEdges();
		$contents->storeVertices();
		$contents->storeParser();
		$contents->storeMethods();
		$data = $contents->getGroups();
		$response->setStatusCode(Response::HTTP_OK);
	} catch (Exception $e) {
		$data = array(
				'error' => true,
				'message' => $e->getMessage()
			);
		$response-
>setStatusCode(Response::HTTP_BAD_REQUEST);
	}
	$response->setContent(json_encode($data));
	$response->headers->set('Content-Type',
'application/json');
	return $response->sendHeaders()->getContent();
});
```

8. Potongan kode untuk API perankingan layanan mikro.

```
<?php
$router->get('/sourcecode/ranking', function() use ($connection,
$request, $response){
	try {
		$limit = (int) $request->query->get('limit');
		$data = array();
		if ($limit > 0) {
			$candidate = new Candidate($connection);
			$output = $candidate-
>getLimitedCandidate($limit);
			$data = $output['result'];
		}
		$response->setStatusCode(Response::HTTP_OK);
	} catch (Exception $e) {
		$data = array(
				'error' => true,
				'message' => $e->getMessage()
```

```
                        );
            $response-
>setStatusCode(Response::HTTP_BAD_REQUEST);
      }
      $response->setContent(json_encode($data));
      $response->headers->set('Content-Type',
'application/json');
      return $response->sendHeaders()->getContent();
});
```

9. Potongan kode untuk API penulisan kembali kode sumber layanan mikro.

```
<?php
$router->get('/sourcecode/ranking/rewrite', function() use
($connection, $request, $response){
      try {
            $servicename = $request->query->get('servicename');
            $nodeId = $request->query->get('nodeId');
            $dir = "source-code/{$servicename}";
            if (!is_dir($dir)) {
            exec("cp -R source-code/case-study-boilerplate {$dir}");
            }
            $candidate = new Candidate($connection);
            $dagOutput = $candidate->getOutboundDAGFile($nodeId);
            if (isset($dagOutput['result'])) {
            foreach ($dagOutput['result'] as $row) {
                  $path_parts = pathinfo($row);
                  if (isset($path_parts['dirname'])) {
                        $newDirName = str_replace('case-study',
$servicename, $path_parts['dirname']);
                        if (!file_exists($newDirName)) {
                              mkdir($newDirName, 0777, true);
                        }
                  }
      $code = new CodeWriter($row, 'case-study', $servicename);
            $code->write();
                  }
            }
            exec("/bin/bash command/deployment.sh {$servicename}
source-code/{$servicename}");
            $data = array('message' => "success");
            $response->setStatusCode(Response::HTTP_OK);
      } catch (Exception $e) {
            $data = array(
                        'error' => true,
                        'message' => $e->getMessage()
                  );
            $response->setStatusCode(Response::HTTP_BAD_REQUEST);
      }
      $response->setContent(json_encode($data));
      $response->headers->set('Content-Type', 'application/json');
      return $response->sendHeaders()->getContent();
});
```