Studi Perbandingan Performa Aplikasi *Web* Monolitik Dan *Microservice* Berbasis Apache Kafka

Yusuf Christian[1], Rahadian Bisma[2]

[1,2] Jurusan Teknik Informatika/Teknik Informatika, Universitas Negeri Surabaya

[1]yusufchristian@mhs.unesa.ac.id

[2]rahadianbisma@unesa.ac.id

***Abstrak*— Perkembangan sistem teknologi informasi menjadi salah satu alasan dan upaya untuk mengantisipasi adanya tantangan yang dapat berdampak langsung kepada kualitas layanan sebuah sistem. Pembuatan aplikasi tidak terlepas dari sebuah perancangan dan desain aplikasi itu sendiri. Terdapat dua cara perancangan sebuah infrastruktur aplikasi, yaitu monolitik dan *microservice*. Pada sistem monolitik pengujian dan perbaikan *error* sangat sulit, karena sistem tersebut tidak dapat dipisahkan dan dilokalisasikan. Sedangkan dalam sistem *microservice* sistemnya terpisah dan sangat independen, fokus untuk menyelesaikan tugas-tugas yang ringan, cara kerjanya yang modular, serta sangat cocok untuk sistem yang dinamis dan konstan dalam perkembangannya. Setiap layanan yang dibuat pada *microservice* juga bisa menggunakan teknologi yang berbeda.**

Tujuan dari penelitian ini adalah untuk melakukan perbandingan performa aplikasi *web* monolitik dan *microservice*. Pengujian dilakukan dengan cara memberikan beban *request* yang dikirimkan dari aplikasi *stress tool* kepada *server*. Pengujian dilakukan dengan menggunakan pola *ramp*, yaitu pola yang memberikan beban yang meningkat seiring berjalannya waktu hingga batas yang telah ditentukan.

Dari hasil pengujian yang dilakukan, performa aplikasi *web microservice* lebih baik dibanding dengan aplikasi *web* monolitik karena dapat menerima load yang lebih besar. Meskipun memiliki performa yang lebih baik, aplikasi *web microservice* menggunakan *resource* CPU yang lebih tinggi dibandingkan dengan aplikasi *web* monolitik.

***Kata Kunci*— sistem, perbandingan, performa, *Microservice*, Monolitik.**

I. PENDAHULUAN

Perkembangan sistem dalam dunia teknologi informasi menjadi salah satu pertimbangan dan upaya untuk mengantisipasi adanya tantangan yang dapat berdampak langsung kepada kualitas layanan sebuah sistem. Dalam pembuatan aplikasi pastinya tidak terlepas dari sebuah perancangan dan desain sebuah aplikasi itu sendiri. Terdapat dua cara perancangan sebuah infrastruktur aplikasi, yaitu monolitik dan *microservice*. Aplikasi monolitik merupakan sebuah aplikasi atau arsitektur yang dalam pembuatannya semua komponen menjadi satu kesatuan [1]. Pada sistem monolitik pengujian dan perbaikan galat sangatlah sulit, dikarenakan sistem tersebut tidak dapat dipisahkan dan dilokalisasikan. Sistem tersebut juga sulit untuk menyediakan fasilitas pengamanan. Pada perancangan sistem monolitik apabila setiap komputer harus menjalankan *kernel* monolitik, hal tersebut bisa dikatakan pemborosan yang sangat besar. Apabila ada kesalahan pemrograman pada suatu bagian dari *kernel*, maka menyebabkan matinya seluruh sistem. Sedangkan pada *Microservice* mempunyai *node* yang berfungsi membagi proses menjadi beberapa proses yang lebih kecil yang mana proses itu akan dipecah untuk memperoleh hasil pemrosesan yang lebih ringan. Dalam *microservice* sendiri sistemnya terpisah dan sangat independen, fokus untuk menyelesaikan tugas-tugas yang ringan, cara kerjanya yang modular serta sangat cocok untuk sistem yang dinamis dan konstan dalam perkembangannya.

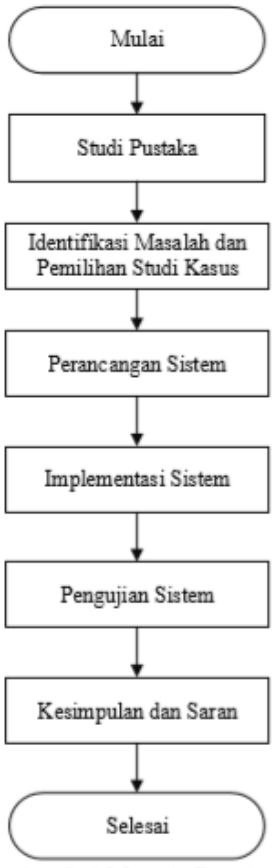
Berdasarkan hasil pengamatan beberapa penelitian menemukan bahwa sangat sedikit yang menggunakan aplikasi *microservice* untuk dikembangkan. Pada penelitian "*Implementing Secure Applications in Smart City Cloud using Microservices*" [2], enkripsi lebih cepat membantu melindungi data yang sensitif dari orang-orang yang tidak berhak mengaksesnya pada aplikasi *smart city* dan membuka kemungkinan untuk mengintegrasikan sistem ini ke infrastruktur yang sudah ada karena *microservice* lebih fleksibel. Sedangkan dalam penelitian "*Application Deployment using Microservice and Docker containers: Framework and Optimization*" [3], hasil pembagian *resource utilization* yang efisien dapat menekan biaya pada aplikasi berbasis *microservice* dengan wadah Docker. Pada penelitian "*Modern Messaging for Distributed Sytems*" [4], *messaging system* memberikan manfaat untuk aplikasi terdistribusi dalam hal skalabilitas dan fleksibiltas. *Apache Kafka* dikatakan lebih efektif untuk menangani data dengan jumlah yang besar. Di penelitian "*Kafka: a Distributed Messaging System for Log Processing*" [5], Kafka memiliki hasil yang lebih baik dibandingkan dengan dua *messaging system* yang lain yaitu ActiveMQ dan RabbitMQ. ActiveMQ dan RabbitMQ adalah *message broker*. Yang membedakan antara Apache Kafka, ActiveMQ dan RabbitMQ adalah bahasa pemrograman yang digunakan dalam pengembangan dan *developer* yang mengembangkan masing-masing *message broker*. ActiveMQ dikembangkan oleh Apache Software Foundation dengan menggunakan bahasa pemrograman Java. RabbitMQ dikembangkan oleh Pivotal Software yang merupakan anak perusahaan dari VMware, dengan menggunakan bahasa pemrograman Erlang. Dan Apache Kafka dikembangkan oleh Apache Software Foundation dengan menggunakan bahasa pemrograman Scala dan Java.

Karena setiap layanan memiliki infrastruktur sendiri, maka aplikasi dapat dengan mudah membuat tanpa memikirkan

hubungan atau ketergantungan *module* dengan layanan yang lain. Seperti halnya bisa membuat aplikasi dengan beberapa bahasa pemrograman yang berbeda. Apabila sebelumnya pada arsitektur sistem monolitik hanya menggunakan satu proses yang dirasa belum efisien maka dengan *microservice* dapat dipecah menjadi bagian yang lebih kecil dan diharapkan menghasilkan sistem yang lebih baik dan efisien. Misalkan terdapat sebuah *server* sendiri yang khusus untuk mengatasi layanan *queue* atau antrean, terdapat hosting yang khusus untuk mengatasi layanan untuk *user interface*, terdapat *server* khusus untuk mengatasi atau menangani *database*, dan lain-lain. Dengan uraian permasalahan yang telah dijelaskan maka salah satu upaya untuk membuat sistem yang lebih efisien dapat dilakukan dengan menerapkan suatu sistem yang menggunakan teknologi terbaru yaitu *microservice* dengan Kafka pada aplikasi.

II. METODOLOGI PENELITIAN

Berikut ini merupakan alur jalannya penelitian untuk melakukan penelitian "Studi Perbandingan Performa Aplikasi *Web* Monolitik Dan *Microservice* Berbasis Apache Kafka" secara umum dapat dilihat pada Gambar. 1.

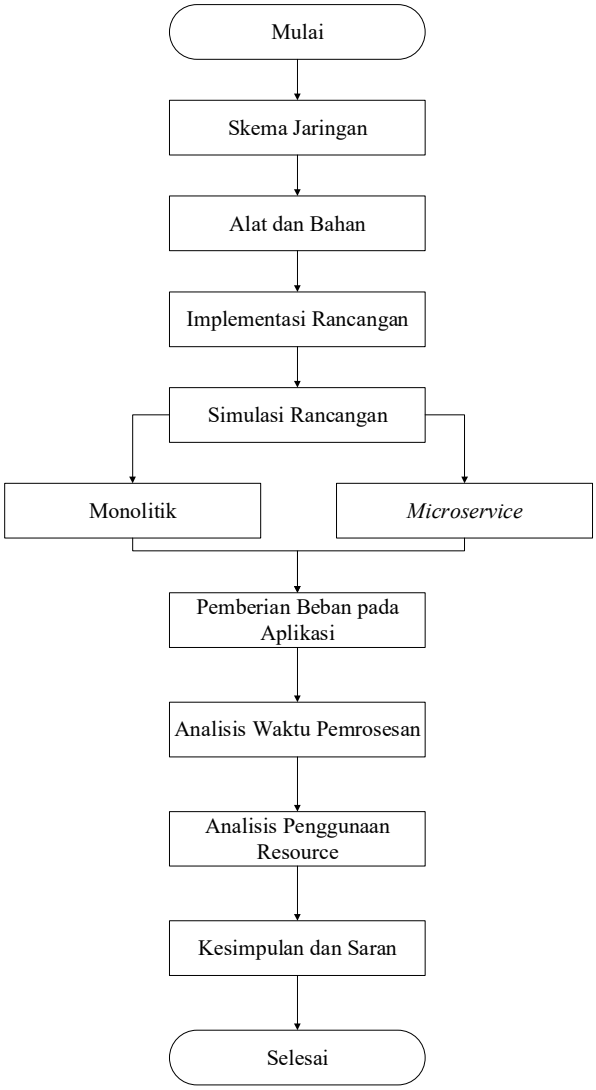


Gambar. 1 Alur Penelitian

Pada Gambar. 1 membahas tentang alur penelitian tentang studi perbandingan performa aplikasi *web* monolitik dan *microservice* berbasis Apache Kafka. Adapun tahapan yang dilakukan pada penelitian ini yaitu melakukan studi pustaka untuk mencari literatur yang relevan yang dapat menjadi referensi pendukung dalam penelitian. Literatur yang digunakan berhubungan dengan *web* monolitik dan *microservice*. Selanjutnya melakukan identifikasi masalah dan pemilihan studi kasus agar dapat mengetahui perbandingan performa dari *web* monolitik dan *microservice* dengan menggunakan Apache Kafka. Pada tahap perancangan sistem penelitian ini menggunakan 2 topologi jaringan yang akan digunakan pada sistem serta alur sistem. Selanjutnya yaitu implementasi sistem akan diimplementasikan menggunakan Apache Kafka. Pada tahap pengujian sistem dilakukan pada uji perbandingan performa dari *web* monolitik dan *microservice*. Tahap terakhir adalah menarik kesimpulan atas pengujian yang telah dilakukan dan saran yang dibutuhkan dalam penelitian yang akan dilakukan selanjutnya.

A. Rancangan Penelitian

Alur sistem yang dikembangkan pada penelitian ini dijelaskan dalam rancangan penelitian. Proses ini digunakan untuk mengetahui hasil uji sistem yang telah dibuat, yaitu waktu pemrosesan serta penggunaan CPU(*Central Processing Unit*) dan RAM(*Random Access Memory*) dari sistem. Pada penelitian ini terdapat beberapa tahapan dan untuk tahapan yang lebih rinci dapat dilihat pada Gambar.2.

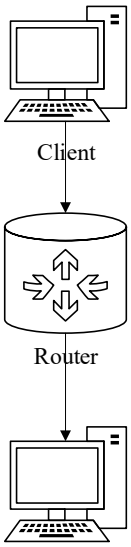


Gambar. 2 Rincian Alur Sistem

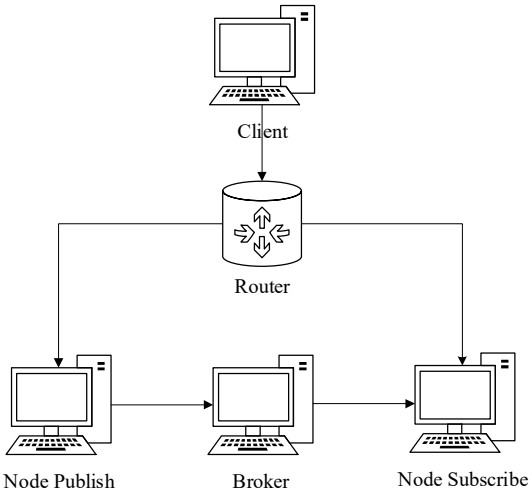
Pada Gambar. 2 menampilkan tentang rincian alur sistem pada penelitian studi perbandingan performa *web* monolitik dan *microservice* dengan Apache Kafka secara umum. Adapun rincian yang dilakukan pada penelitian ini yaitu dibagi menjadi beberapa langkah, yang pertama adalah skema jaringan yang bertujuan untuk melihat detail alur topologi jaringan yang dibutuhkan. Kemudian mempersiapkan segala kebutuhan alat dan bahan penelitian untuk diimplementasikan. Pada langkah selanjutnya, simulasi dilakukan dengan aplikasi dan topologi yang berbeda, dalam hal ini parameter yang akan diuji yaitu *response time* dari aplikasi monolitik dan *microservice* serta *resource utilization* yang dibutuhkan kedua aplikasi *web* tersebut. Kemudian langkah selanjutnya melakukan perbandingan pada monolitik dan *microsevice* untuk mengetahui performa yang terbaik. Langkah terakhir adalah menarik kesimpulan atas pengujian yang telah dilakukan dan saran yang dibutuhkan dalam penelitian yang akan dilakukan selanjutnya.

B. Skema Jaringan

Pada penelitian ini skema topologi jaringan yang digunakan dalam penelitian ini terdapat dua topologi yang akan dibandingkan yaitu topologi dengan arsitektur monolitik dan topologi dengan arsitektur *microservice*. Topologi jaringan yang dirancang dapat dilihat pada Gambar.3 dan Gambar.4 :



Gambar. 3 Topologi Arsitektur Monolitik



Gambar. 4 Topologi Arsitektur *Microservice*

Dalam penelitian ini terdapat beberapa peralatan yang akan dibutuhkan. Istilah-istilah yang akan digunakan diantaranya yaitu *client*, Kafka *broker*, serta *microservice node*. *Client* mengirimkan *request*, kemudian *request* atau permintaan tersebut akan diproses oleh masing-masing *microservice*. Setiap *node microservice* memproses permintaan yang berbeda sesuai dengan fungsi dari *node* tersebut. Data yang masuk ke *node publish* akan disimpan dan dikirimkan ke *node subscribe*, yang kemudian akan ditampilkan ketika ada *client* yang melakukan *request* kepada *node subscribe*. Penelitian ini menggunakan Apache Kafka sebagai *message broker* yang akan dipasang pada *node publish*.

C. Sistem Monolitik

Sistem monolitik merupakan salah satu struktur dari berbagai macam struktur dalam sistem operasi. Karakteristik dari sistem monolitik, yaitu prosedur dapat saling dipanggil oleh prosedur lain pada sistem apabila diperlukan, *kernel* berisi semua layanan yang disediakan. Sistem monolitik mampu beradaptasi terhadap perubahan kebutuhan sistem terutama dalam pengelolaan kompleksitas kode, dan *maintainability*-nya, dimana hal ini akan menyebabkan *bottleneck* pada proses pendistribusiannya karena tingkat dependensi yang tinggi [6].

D. Sistem *Microservice*

Microservice adalah pembagian *service* ke dalam bagian yang lebih kecil dimana layanan-layanan tersebut saling berhubungan satu sama lain. Selain itu, dalam setiap *service* yang dibuat bisa menggunakan teknologi yang berbeda [7]. Suatu *microservice* merupakan komponen yang dapat digunakan secara independen dari lingkup terbatas yang mendukung *port* interoperabilitas melalui komunikasi berbasis pesan. Arsitektur *microservice* adalah gaya rekayasa yang sangat otomatis, sistem perangkat lunak yang dapat dikembangkan, terdiri dari *microservice* dengan kemampuan selaras [8]. Pola arsitektur yang dimiliki oleh *microservice* secara signifikan dapat mempengaruhi hubungan antara aplikasi dengan *database*.

E. Apache Kafka

Apache Kafka merupakan platform terdistribusi yang berhubungan dengan *data streaming*. Apache Kafka adalah sistem *publish/subscribe messaging* yang memiliki istilah *producer* yang didalamnya terdapat satu atau lebih sistem yang men-*generate* data untuk suatu topik tertentu secara *real-time*. Sedangkan *consumer* merupakan istilah dimana topik tersebut dapat dibaca oleh satu atau lebih sistem yang membutuhkan data-data dari topik tersebut secara *real-time* [9].

F. Spesifikasi Aplikasi

Spesifikasi aplikasi yang akan digunakan dalam penelitian studi perbandingan performa aplikasi *web* monolitik dan *microservice* ini adalah sebagai berikut :

Spesifikasi aplikasi untuk pengujian *web* monolitik :
1. *Web Server* : Nginx
2. Php : Php7
3. Mysql : Mysql 5

Spesifikasi aplikasi untuk pengujian *web microservice* :
1. *Web Server* : Nginx
2. Php : Php7
3. Mysql : Mysql 5
4. Apache Zookeeper
5. Apache Kafka

Untuk pengujian aplikasi *web* monolitik dan *microservice* menggunakan *web* aplikasi yang di-*deploy* menggunakan Docker. Kemudian *hardware* yang digunakan dalam pengujian *web* aplikasi menggunakan komputer *server* yang memiliki spesifikasi sebagai berikut :

1. *Processor* : Intel Core i7-4790 CPU @ 3.60GHz
2. RAM : 8 GB
3. *Operating system* : Windows 10 Pro 64 bit

Sedangkan untuk komputer *client* menggunakan spesifikasi:
1. *Processor* : Intel Core i5-4210U CPU @ 1.70GHz ~ 2.40 GHz
2. RAM : 12 GB
3. *Operating system* : Windows 10 Pro 64 bit

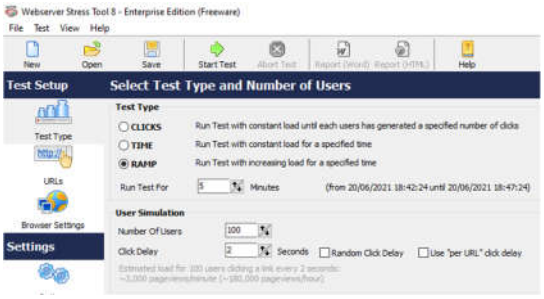
Komputer *client* dan *server* dihubungkan dengan jaringan lokal yang memiliki IP komputer *server* 192.168.10.1, dan IP komputer *client* 192.168.10.2

G. Pengujian Sistem

Pada tahap ini digunakan untuk pengujian terhadap sistem yang telah dibuat. Pengujian dilakukan dengan membandingkan situs *web* yang menggunakan arsitektur monolitik dan situs *web* yang menggunakan arsitektur *microservice*. Pada pengujian ini bertujuan untuk mengetahui performa yang terbagi atas rata-rata waktu *response*, jumlah *error* yang terjadi, dan *resource* (CPU dan RAM). Pengujian dilakukan pada kinerja sistem dalam menangani permintaan atau *request* yang dikirim oleh *stress tool* Webserver Stress Tool 8. Aplikasi ini mensimulasikan beban kepada *server* dengan berbagai pola.

Pola yang akan digunakan dalam pengujian ini adalah pola *ramp test*, yaitu pola pemberian beban yang terus meningkat seiring waktu sampai batas yang telah ditentukan. Langkah-langkah pengujian yang akan dilakukan adalah sebagai berikut :

1. Mempersiapkan semua kebutuhan yang digunakan dalam pengujian seperti PC yang telah terpasang Docker dan PC yang terpasang Webserver Stress Tool 8.
2. Kemudian Docker tersebut diisi dengan *container-container* yang dibutuhkan dalam pemasangan sistem *microservice* dan monolitik.
3. Mengatur *container* pada Docker untuk mendukung pemasangan *microservice*.
4. Mengatur aplikasi Webserver Stress Tool 8 sesuai batasan yang telah ditentukan, contohnya seperti pada Gambar. 5.



Gambar. 5 Pengaturan Aplikasi Webserver Stress Tool 8

5. Melakukan percobaan menggunakan Webserver Stress Tool 8.
6. Memonitor penggunaan *resource* pada sistem menggunakan aplikasi Performance Monitor bawaan Windows.
7. Melakukan pengamatan pada hasil pengujian kecepatan waktu yang digunakan untuk memproses permintaan atau *request*.

III. HASIL DAN PEMBAHASAN

Pada bagian ini menjelaskan hasil pengujian terhadap implementasi yang telah dilakukan beserta analisisnya. Pengujian dilakukan untuk menilai apakah kebutuhan dan analisis yang telah dispesifikasikan sebelumnya telah terpenuhi secara keseluruhan. Dalam pengujian ini didapatkan hasil perbandingan performa dari monolitik dan *microservice* sebagai berikut:

A. Hasil Pengujian Aplikasi Monolitik

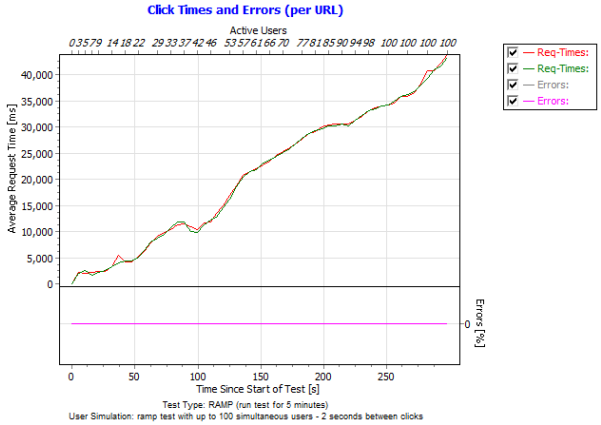
Pengaturan yang dipakai untuk pengujian aplikasi *web* monolitik dengan menggunakan Webserver Stress Tool 8 adalah sebagai berikut :

1. Pola yang digunakan adalah *ramp*, yaitu pola pemberian beban yang terus bertambah seiring waktu sampai dengan batas yang telah ditentukan.
2. Pengujian dilakukan selama 5 menit.
3. *Delay user* untuk melakukan *request* adalah 2 detik.
4. Batasan *user* aktif untuk tiap pengujian adalah 100, 200, 300, 400 *user*.
5. URL (*Uniform Resource Locator*) yang digunakan adalah :
   a. 192.168.10.1/inventories?merk=Percobaan&produk=CobaCoba
   b. 192.168.10.1/stats

   Kedua URL diatas ditangani oleh aplikasi monolitik.

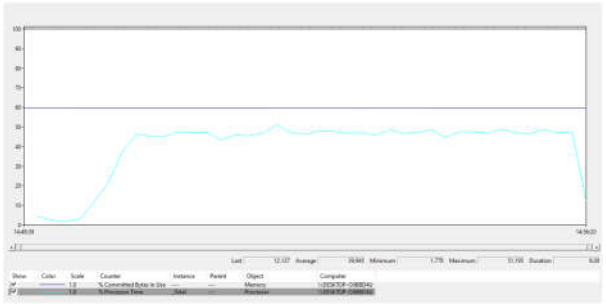
Hasil dari pengujian yang dilakukan adalah sebagai berikut :

1. Dengan Batasan 100 *User* :



Gambar. 6 Grafik Hasil Pengujian 100 *User*

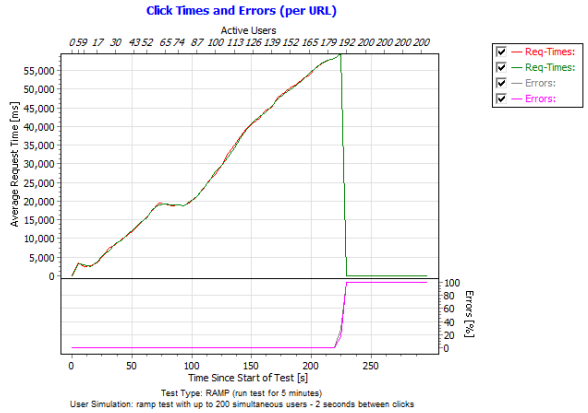
Dengan batasan 100 *user*, *server* masih bisa memproses *request* yang datang hingga pengujian selesai. Berdasarkan Gambar. 6, rata-rata waktu pemrosesan terus naik seiring dengan bertambahnya *user* yang aktif. Rata-rata waktu pemrosesan di akhir pengujian adalah sekitar 40.000 *milisecond* (ms) dan persentase *error* sebesar 0%.



Gambar. 7 Grafik Penggunaan *Resource* Dengan Batasan 100 *User*

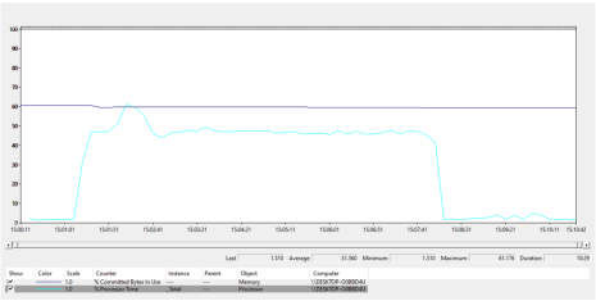
Penggunaan *Resource* aplikasi monolitik dapat dilihat pada Gambar. 7. Dapat dilihat bahwa penggunaan RAM sekitar 60% dan CPU sekitar 50%.

2. Dengan Batasan 200 *User*



Gambar. 8 Grafik Hasil Pengujian 200 *User*

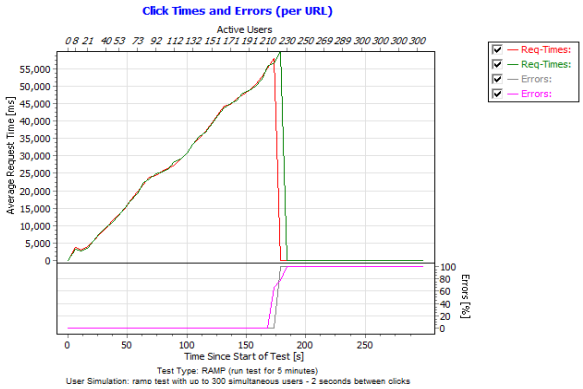
Dengan batasan 200 *user*, *server* mulai *down* ketika jumlah *user* mendekati 192. Berdasarkan Gambar. 8, rata-rata waktu pemrosesan terus naik seiring dengan bertambahnya *user* yang aktif, hingga pada akhirnya *server down*. Rata-rata waktu pemrosesan paling tinggi sebelum *server down* adalah sekitar 60.000 ms dan persentase *error* sebesar 100% ketika jumlah *user* aktif mendekati 192.



Gambar. 9 Grafik Penggunaan *Resource* Dengan Batasan 100 *User*

Penggunaan *Resource* aplikasi monolitik dapat dilihat pada Gambar. 9. Dapat dilihat bahwa penggunaan RAM sekitar 60% dan CPU sekitar 50%.

3. Dengan Batasan 300 *User*



Gambar. 10 Grafik Hasil Pengujian 300 *User*

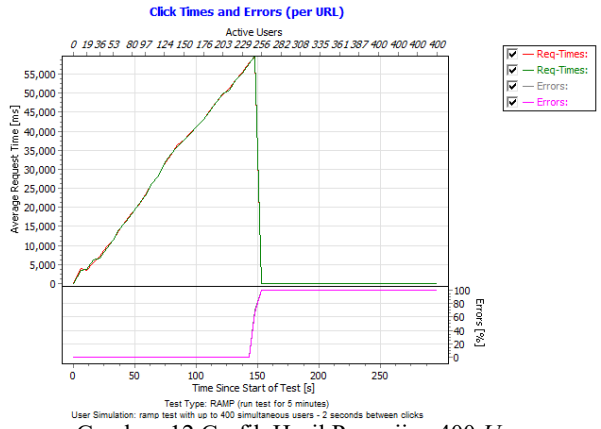
Dengan batasan 300 *user*, *server* mulai *down* ketika jumlah *user* mendekati 230. Berdasarkan Gambar. 10, rata-rata waktu pemrosesan terus naik seiring dengan bertambahnya *user* yang aktif, hingga pada akhirnya *server down*. Rata-rata waktu pemrosesan paling tinggi sebelum *server down* adalah sekitar 60.000 ms dan persentase *error* sebesar 100% ketika jumlah *user* aktif mendekati 230.

Gambar. 11 Grafik Penggunaan *Resource* Dengan Batasan 300 *User*

Penggunaan *Resource* aplikasi monolitik dapat dilihat pada Gambar. 11. Dapat dilihat bahwa penggunaan RAM sekitar 60% dan CPU sekitar 50%.

4. Dengan Batasan 400 *User*

Dengan batasan 400 *user*, *server* mulai *down* ketika jumlah *user* mendekati 256. Berdasarkan Gambar. 12, rata-rata waktu pemrosesan terus naik seiring dengan bertambahnya *user* yang aktif, hingga pada akhirnya *server down*. Rata-rata waktu pemrosesan paling tinggi sebelum *server down* adalah sekitar 60.000 ms dan persentase *error* sebesar 100% ketika jumlah *user* aktif mendekati 256.



Gambar. 12 Grafik Hasil Pengujian 400 *User*

Gambar. 13 Grafik Penggunaan *Resource* Dengan Batasan 400 *User*

Penggunaan *Resource* aplikasi monolitik dapat dilihat pada Gambar. 13. Dapat dilihat bahwa penggunaan RAM sekitar 60% dan CPU sekitar 50%.

Tabel 1 merupakan ringkasan hasil pengujian aplikasi monolitik dengan beberapa batasan *user*.

Tabel 1 : Rangkuman Hasil Pengujian Aplikasi Monolitik

Batasan *User*	**Puncak Rata-Rata Waktu Pemrosesan**	**Jumlah *User* Sebelum *Server Down***	**Persen RAM**	**Persen CPU**
100	± 40.000 ms	-	±60%	±50%
200	± 60.000 ms	< 192	±60%	±50%
300	± 60.000 ms	< 230	±60%	±50%
400	± 60.000 ms	< 256	±60%	±50%

B. Hasil Pengujian Aplikasi *Microservice*

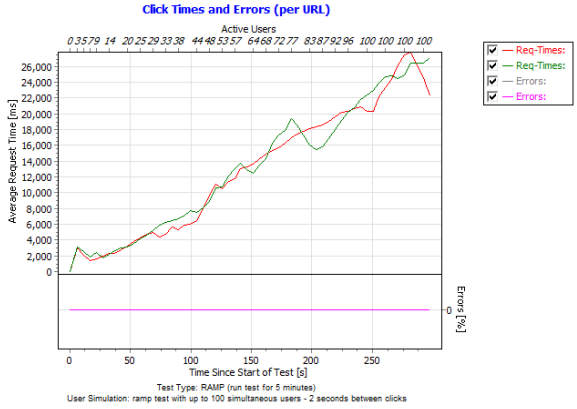
Pengaturan yang dipakai untuk pengujian aplikasi *web microservice* dengan menggunakan Webserver stress tool 8 adalah sebagai berikut :

1. Pola yang digunakan adalah *ramp*, yaitu pola pemberian beban yang terus bertambah seiring waktu sampai dengan batas yang telah ditentukan.
2. Pengujian dilakukan selama 5 menit.
3. *Delay user* untuk melakukan *request* adalah 2 detik.
4. Batasan *user* aktif untuk tiap pengujian adalah 100, 200, 300, 400 *user*.
5. URL yang digunakan adalah :
   a. 192.168.10.1:8787/inventories?merk=Percobaan&produk=CobaCoba
   b. 192.168.10.1/stats

   Kedua URL diatas ditangani oleh *node* yang berbeda, URL pada poin a ditangani oleh *node publish*, sedangkan URL pada poin b ditangani oleh *node subscribe*

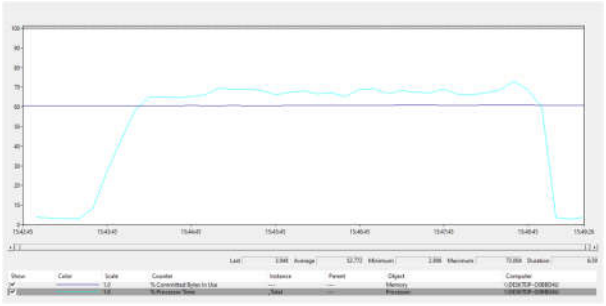
Hasil dari pengujian yang dilakukan adalah sebagai berikut :

1. Dengan Batasan 100 *User* :



Gambar. 14 Grafik Hasil Pengujian *Microservice* 100 *User*

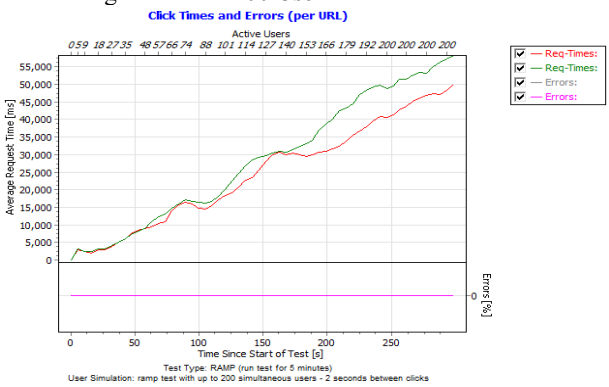
Dengan batasan 100 *user*, *server* masih bisa memproses *request* yang datang hingga pengujian selesai. Berdasarkan Gambar. 14, rata-rata waktu pemrosesan terus naik seiring dengan bertambahnya *user* yang aktif. Rata-rata waktu pemrosesan di akhir pengujian adalah sekitar 26.000 ms dan persentase *error* sebesar 0%.



Gambar. 15 Grafik Penggunaan *Resource* Dengan Batasan 300 *User*

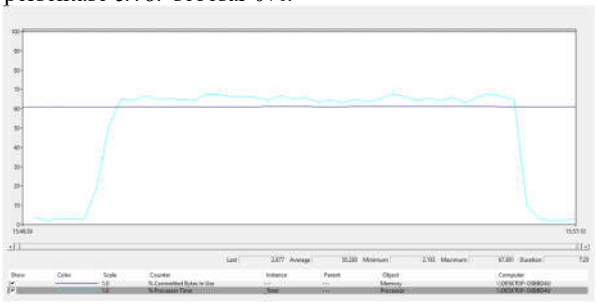
Penggunaan *Resource* aplikasi *microservice* dapat dilihat pada Gambar. 15. Dapat dilihat bahwa penggunaan RAM sekitar 60% dan CPU sekitar 70%.

2. Dengan Batasan 200 *User*



Gambar. 16 Grafik Hasil Pengujian *Microservice* 200 *User*

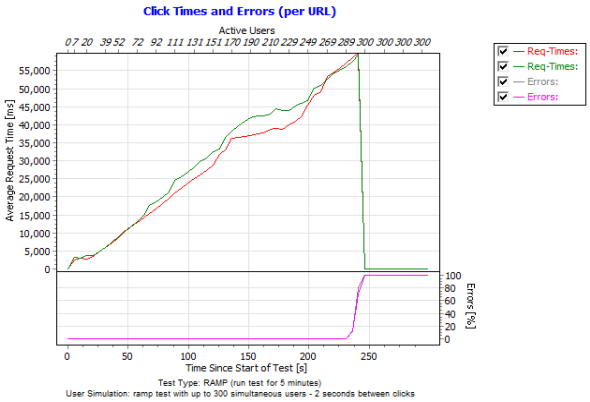
Dengan batasan 200 *user*, *server* masih bisa memproses *request* yang datang hingga pengujian selesai. Berdasarkan Gambar. 16, rata-rata waktu pemrosesan terus naik seiring dengan bertambahnya *user* yang aktif. Rata-rata waktu pemrosesan di akhir pengujian adalah sekitar 55.000 ms dan persentase *error* sebesar 0%.



Gambar. 17 Grafik Penggunaan *Resource* Dengan Batasan 200 *User*

Penggunaan *Resource* aplikasi *microservice* dapat dilihat pada Gambar. 17. Dapat dilihat bahwa penggunaan RAM sekitar 60% dan CPU sekitar 65%.

3. Dengan Batasan 300 *User*



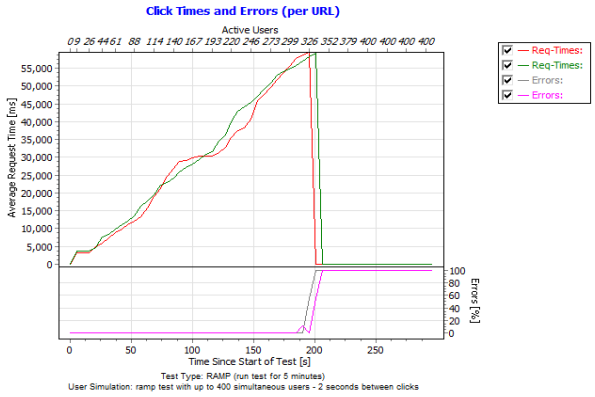
Gambar. 18 Grafik Hasil Pengujian *Microservice* 300 *User*

Dengan batasan 300 *user*, *server* mulai *down* ketika jumlah *user* mendekati 300. Berdasarkan Gambar. 17, rata-rata waktu pemrosesan terus naik seiring dengan bertambahnya *user* yang aktif hingga pada akhirnya *server down*. Rata-rata waktu pemrosesan paling tinggi sebelum *server down* adalah sekitar 60.000 ms dan persentase *error* sebesar 100% ketika jumlah *user* aktif mendekati 300.

Gambar. 19 Grafik Penggunaan *Resource* Dengan Batasan 300 *User*

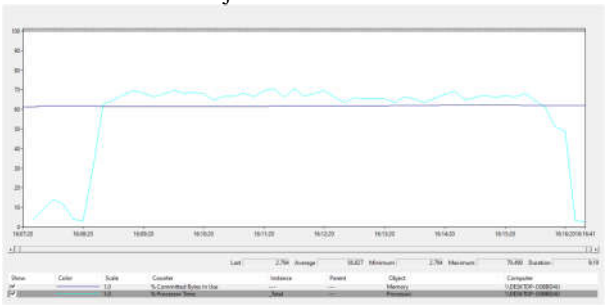
Penggunaan *Resource* aplikasi *microservice* dapat dilihat pada Gambar. 19. Dapat dilihat bahwa penggunaan RAM sekitar 60% dan CPU sekitar 65%.

4. Dengan Batasan 400 *User*



Gambar. 20 Grafik Hasil Pengujian *Microservice* 400 *User*

Dengan batasan 400 *user*, *server* mulai *down* ketika jumlah *user* mendekati 330. Berdasarkan Gambar. 20, rata-rata waktu pemrosesan terus naik seiring dengan bertambahnya *user* yang aktif hingga pada akhirnya *server down*. Rata-rata waktu pemrosesan paling tinggi sebelum *server down* adalah sekitar 60.000 ms dan persentase *error* sebesar 100% ketika jumlah *user* aktif mendekati 330.



Gambar. 21 Grafik Penggunaan *Resource* Dengan Batasan 400 *User*

Penggunaan *Resource* aplikasi *microservice* dapat dilihat pada Gambar. 21. Dapat dilihat bahwa penggunaan RAM sekitar 60% dan CPU sekitar 70%.

Tabel 2 adalah rangkuman hasil pengujian aplikasi *microservice* dengan berbagai batasan *user*.

Tabel 2 : Rangkuman Hasil Pengujian Aplikasi *Microservice*

Batasan *User*	**Puncak Rata-Rata Waktu Pemrosesan**	**Jumlah *User* Sebelum *Server Down***	**Persen RAM**	**Persen CPU**
100	± 26.000 ms	-	±60%	±70%
200	± 55.000 ms	-	±60%	±65%
300	± 60.000 ms	< 300	±60%	±65%
400	± 60.000 ms	< 330	±60%	±70%

C. Hasil perbandingan performa

Pada pengujian dengan batasan 100 *user* aplikasi *web* monolitik memiliki rata-rata waktu pemrosesan tertinggi sekitar 40.000 ms, sedangkan aplikasi *web microservice* memiliki rata-rata waktu pemrosesan tertinggi sekitar 26.000 ms. Pada pengujian dengan batasan 200 *user*, aplikasi *web* monolitik *down* setelah pengujian berjalan sekitar 4 menit dengan sekitar 180 *user* aktif, sedangkan aplikasi *web microservice* masih bisa menerima *request* hingga akhir pengujian. Pada pengujian dengan batasan 300 *user*, kedua aplikasi mengalami *down*, tetapi memiliki perbedaan pada waktu dan jumlah *user* ketika aplikasi *down*. Aplikasi *web* monolitik *down* ketika pengujian telah berjalan sekitar 3 menit dan dengan jumlah *user* aktif sekitar 220 *user*. Sedangkan aplikasi *web microservice* baru mengalami *down* ketika pengujian telah berjalan sekitar 4 menit dan jumlah *user* aktif sekitar 290 *user*. Begitu juga pada pengujian dengan 400 *user*, kedua aplikasi mengalami *down*. Aplikasi *web* monolitik *down* ketika pengujian telah berjalan sekitar 2 menit dengan jumlah *user* aktif sekitar 240 *user*. Sedangkan aplikasi *web microservice* baru mengalami *down* ketika pengujian telah berjalan selama 3 menit dengan jumlah *user* aktif sekitar 330 *user*.

Tabel 3 menunjukkan perbandingan rata-rata waktu pemrosesan dari masing-masing aplikasi, monolitik dan *microservice*.

Tabel 3 : Perbandingan Puncak Rata-Rata Waktu Pemrosesan Aplikasi Monolitik dan Microservice

Batasan *User*	**Puncak Rata-Rata Waktu Pemrosesan**	
	Monolitik	***Microservice***
100	± 40.000 ms	± 26.000 ms
200	± 60.000 ms	± 55.000 ms
300	± 60.000 ms	± 60.000 ms
400	± 60.000 ms	± 60.000 ms

Tabel 4 menunjukkan perbandingan jumlah *user* sebelum aplikasi monolitik dan *microservice* mengalami *server down*.

Tabel 4 : Perbandingan Jumlah *User* Sebelum *Server Down* Aplikasi Monolitik dan *Microservice*

Batasan *User*	Jumlah User Sebelum *Server Down*	
	Monolitik	***Microservice***
100	-	-
200	< 192	-
300	< 230	< 300
400	< 256	< 330

Aplikasi *web microservice* menggunakan lebih banyak *resource* CPU dibandingkan dengan aplikasi *web* monolitik. Penggunaan CPU pada aplikasi *web microservice* adalah sekitar 60% sampai 70%, sedangkan pada aplikasi *web* monolitik menggunakan sekitar 40% sampai 50%. Hal ini dikarenakan aplikasi *web microservice* yang digunakan dalam pengujian memiliki dua aplikasi yang menangani proses yang berbeda. Sedangkan aplikasi *web* monolitik hanya memiliki satu aplikasi yang menangani semua *request* yang datang. Tetapi dalam hal persentase RAM yang digunakan, kedua aplikasi menggunakan jumlah yang kurang lebih sama, yaitu 60%. Meskipun jumlah batasan *user* pada tiap pengujian diubah, penggunaan *resource* CPU dan RAM oleh aplikasi *web* monolitik dan aplikasi *web microservice* cenderung stabil.

Tabel 5 menunjukkan perbandingan rata-rata penggunaan *resource* oleh aplikasi monolitik dan *microservice*.

Tabel 5 : Perbandingan Rata-Rata Penggunaan *Resource* oleh Aplikasi Monolitik dan *Microservice*

Batasan *User*	**Monolitik**		***Microservice***	
	RAM	**CPU**	**RAM**	**CPU**
100	±60%	±50%	±60%	±70%
200	±60%	±50%	±60%	±65%
300	±60%	±50%	±60%	±65%
400	±60%	±50%	±60%	±70%

IV. KESIMPULAN

Aplikasi *web microservice* memiliki performa yang lebih baik dibandingkan aplikasi *web* monolitik. Hal tersebut bisa dilihat dari hasil pengujian yang menunjukkan bahwa aplikasi *web microservice* dapat memproses *request* dengan waktu yang lebih singkat dibanding dengan aplikasi *web* monolitik. Selain waktu pemrosesan, aplikasi *web microservice* juga bisa menerima *load* atau beban yang lebih banyak dibanding dengan aplikasi monolitik.

Meskipun memiliki performa yang lebih baik, aplikasi *web microservice* menggunakan lebih banyak *resource* CPU dibandingkan dengan aplikasi *web* monolitik. Penggunaan CPU pada aplikasi *web microservice* sekitar 10 sampai 20 % lebih tinggi dibandingkan dengan penggunaan CPU pada aplikasi *web* monolitik. Tetapi persentase RAM yang digunakan oleh kedua aplikasi kurang lebih sama, yaitu 60%.

REFERENSI

[1] Z. R. Mair, Teori dan Praktek Sistem Operasi, 1 red., Yogyakarta: Deepublish, 2018.

[2] M. Krämer , S. Frese och A. Kuijper, ”*Implementing Secure Applications in Smart City Cloud using Microservice*s,” *Fiture Generation Computer System,* vol. 99, nr 99, pp. 308-320, 2019.

[3] X. Wan, X. Guan, T. Wang, G. Bai och B.-Y. Choi, ”*Application deployment using Microservice and Docker containers: Framework and optimization*,” *Journal of Network and Computer Applications,* pp. 97-109, 2018.

[4] L. Magnoni, ”*Modern Messaging for Distributed Sytems*,” Switzerland, 2015.

[5] K. Jay, N. Narkhede och J. Rao, ”*Kafka: a Distributed Messaging System for Log Processing*,” pp. 1-7, 2011.

[6] G. Munawar och A. Hodijah, ”Analisis Model Arsitektur *Microservice* Pada Sistem Informasi DPLK,” *Publikasi Jurnal & Penelitian Teknik Informatika ,* pp. 232-239, 2018.

[7] S. Newman, *Building Microservice*s, 1 red., Sebastopol: O'Reilly, 2015.

[8] I. Nadareishvili, R. Mitra, M. McLarty och M. Amundsen, *Microservice* Architecture, 1 red., Sebastopol: O’Reilly Media, 2016.

[9] M. Kumar och C. Singh, *Building Data Streaming Applications with Apache Kafka*, 1 red., Birmingham: Packt Publishing Ltd., 2017.