Jurnal Smart Teknologi
Vol. 5, No. 2, Januari 2024, Halaman 144 – 154
ISSN: 2774-1702, http://jurnal.unmuhjember.ac.id/index.php/JST

Implementasi Arsitektur *Microservice* pada Aplikasi MallDesa dengan Menggunakan Metode *Choreography Internal Comunication*

Implementation of Microservice Architecture in the MallDesa Application Using the Choreography Internal Communication Method

Zelda Ababil[1], Lutfi Ali Muharram[2], Triawan Adi Cahyanto[3]
[1]Mahasiswa Program Studi Teknik Informatika, Fakultas Teknik, Universitas Muhammadiyah Jember
Email: zeldaababil01@gmail.com
[2]Dosen Fakultas Teknik, Universitas Muhammadiyah Jember
Email: lutfi.muharom@unmuhjember.ac.id
[3]Dosen Fakultas Teknik, Universitas Muhammadiyah Jember
Email: triawanac@unmuhjember.ac.id

Abstrak

Tujuan dari penelitian ini yaitu untuk merancang dan mengetahui desain sistem arsitektur *microservice* yang terimplementasikan pada sistem aplikasi Malldesa menggunakan metode *Choreography Internal Communication* serta mengetahui kinerja aplikasi malldesa terhadap kemampuan dalam menangani beban traffic penggunaan. Tahapan penelitian dalam menerapkan arsitektur *microservice* menggunakan metode *Choreography Internal Communication*. Berdasarkan urutan dari proses – proses pada tahapan penelitian, metode pengembangan aplikasi yang penulis gunakan adalah metode pengembangan *Waterfall* yang bersifat linear. Berdasarkan penelitian serta hasil pengujian pada implementasi arsitektur *microservice* pada aplikasi Malldesa dengan menggunakan metode *Choreography Internal Communication* mendapati kesimpulan sebagai berikut: 1) Implementasi Arsitektur *Microservice* pada aplikasi Malldesa dengan menggunakan metode *Choreography Internal Communication* dapat dilakukan dan berjalan dengan cukup baik, serta metode yang digunakan dapat menjadi solusi dalam pertukaran data antar *service*. 2) Pengujian menunjukkan kinerja arsitektur *microservice* lebih baik pada skala pengguna besar (200 dan 400 *thread*). Namun, peningkatan beban pada Microservices juga mengakibatkan peningkatan tingkat kesalahan yang signifikan, mencapai 93.03% pada skala 400 *thread*.
***Keywords**:* Arsitektur *Microservice* , MallDesa, Metode *Choreography Internal Comunication*

Abstract

The aim of this research is to design and determine the design of the microservice architectural system implemented in the Malldesa application system using the Choreography Internal Communication method and to determine the performance of the Malldesa application regarding its ability to handle usage traffic loads. Research stages in implementing microservice architecture using the Choreography Internal Communication method. Based on the sequence of processes at the research stages, the application development method that the author uses is the linear Waterfall development method. Based on research and test results on the implementation of microservice architecture in the Malldesa application using the Choreography Internal Communication method, the following conclusions were found: 1) Implementation of Microservice Architecture in the Malldesa application using the Choreography Internal Communication method can be carried out and runs quite well, and the method used can be a solution for exchanging data between services. 2) Testing shows the microservice architecture performs better at large user scales (200 and 400 threads). However, the increased load on Microservices also resulted in a significant increase in error rates, reaching 93.03% at a scale of 400 threads.
***Keywords**: Microservice Architecture, MallDesa, Choreography Internal Communication Method*

Jurnal Smart Teknologi
Vol. 5, No. 2, Januari 2024, Halaman 144 – 154
ISSN: 2774-1702, http://jurnal.unmuhjember.ac.id/index.php/JST

1. PENDAHULUAN

Aplikasi Malldesa merupakan hasil dari pengembangan aplikasi yang telah ada sebelumnya pada desa Sidomulyo. Jauh sebelum aplikasi Malldesa tersebut ada, pemerintah desa Sidomulyo telah memanfaatkan media digital untuk mengupayakan masyarakatnya dalam melakukan permohonan surat tanpa harus mendatangi balai desa setempat yang disebut sebagai Sistem Pelayanan *Online*. Namun dalam sistem tersebut hanya berbasiskan form *online* saja, sehingga tercipta lah Aplikasi malldesa sebagai hasil pengembangan dari sistem tersebut (Nusantara et al., 2022).

Aplikasi Malldesa memiliki fitur sangat banyak dan kompleks. Secara garis besar aplikasi ini terdiri dari 3 fitur yang sangat menarik dan menjadi fitur utama dari aplikasi Malldesa. Aplikasi ini dapat digunakan untuk memudahkan masyarakat dalam hal pengajuan surat tanpa harus pergi ke balai desa. Selain fitur tersebut, pada Aplikasi Malldesa juga terdapat fitur jual beli *online* atau E-Commerce, sehingga dapat meningkatkan perekonomian warga. Kemudian fitur yang ketiga adalah aplikasi ini dilengkapi dengan fitur berita dan wisata yang dapat masyarakat ataupun wisatawan baca agar selalu mendapat informasi terbaru terkait berita dan wisata. Berdasarkan hal tersebut secara tidak langsung aplikasi malldesa diharuskan untuk selalu aktif sedia dalam menangani *request* atau permintaan dari pengguna. Terlebih Aplikasi Malldesa mempunyai tiga fitur yang sangat kompleks, sehingga menjadi tidak ayal apabila aplikasi akan menjadi lambat atau bahkan mati ketika aplikasi diakses oleh pengguna dengan *traffic* atau jumlah penggunaan yang banyak. Oleh sebab itu diperlukan sebuah cara agar aplikasi tersebut dapat menangani lalu lintas atau *traffic* yang dilakukan oleh pengguna ketika mengakses aplikasi.

Pembangunan aplikasi malldesa masih mengadopsi arsitektur atau pendekatan yang bersifat *monolith*. Arsitektur *monolith* merupakan sebuah metode dalam pembangunan sistem aplikasi yang mana dalam rancangan arsitektur ini akan menjalankan semua fungsi atau logika dalam satu kesatuan aplikasi (Daya et al., 2015). Hal tersebut tentu saja akan mengakibatkan kemampuan aplikasi dalam beradaptasi terhadap terjadinya perubahan kebutuhan sistem (*requirement changes*) semakin sulit. Pengelolaan kompleksitas kode dan *maintainability* akan mempengaruhi keseluruhan aplikasi. Begitu pula ketika akan mengubah atau menambahkan teknologi baru, harus melakukan *restarting* ulang aplikasi yang mana hal tersebut akan mengakibatkan aplikasi tidak akan bisa diakses untuk sesaat (Priyadarshini S & Shilpa G, 2017). Dalam hal ini penggunaan arsitektur *monolith* pada suatu saat tentu saja akan menjadi sebuah masalah ketika aplikasi tersebut mempunyai pengguna aktif yang sangat banyak, maka aplikasi tersebut tidak akan mampu dalam menangani *request* atau permintaan dari pengguna secara bersamaan dan akan mengakibatkan aplikasi tersebut mati atau *down* sehingga pengguna tidak dapat mengakses aplikasi.

Kekurangan serta kelemahan dalam arsitektur *monolith* dapat diatasi dengan menggunakan Arsitektur *microservice* . Arsitektur *microservice s* merupakan pola pendekatan untuk mengembangkan aplikasi berbasis web dengan memecah keseluruhan fungsi perangkat lunak menjadi beberapa layanan kecil yang saling terhubung dan dapat dikembangkan dan di-*deploy* secara terpisah (Sendiang et al., 2018). *Microservice* dapat dibangun dengan menggunakan bahasa pemrograman yang berbeda dan berjalan pada mesin yang berbeda, sehingga memudahkan untuk mengembangkan dan memelihara aplikasi. Pendekatan ini memungkinkan tim pengembang untuk bekerja secara paralel pada bagian-bagian tertentu dari aplikasi tanpa mengganggu bagian lainnya. Arsitektur *Microservice* juga memungkinkan untuk peningkatan atau skalabilitas yang lebih baik dan kemampuan untuk meng-*upgrade* aplikasi secara bertahap tanpa *downtime* (Richardson, 2018). Pada penerapan-nya aplikasi yang menggunakan arsitektur *microservice* akan dipecah menjadi bagian-bagian yang terpisah dan lebih kecil, sesuai dengan *logic* atau fungsi-nya masing-masing. Sehingga setiap *service* dapat dikembangkan, diuji, dan dioperasikan secara *independent*, yang mana hal ini dapat

Jurnal Smart Teknologi
Vol. 5, No. 2, Januari 2024, Halaman 144 – 154
ISSN: 2774-1702, http://jurnal.unmuhjember.ac.id/index.php/JST

memudahkan dalam pengembangan dan pemeliharaan aplikasi. Berdasarkan hal tersebut maka penggunaan arsitektur *microservice s* secara tidak langsung dapat menjawab masalah dari arsitektur *monolith*.

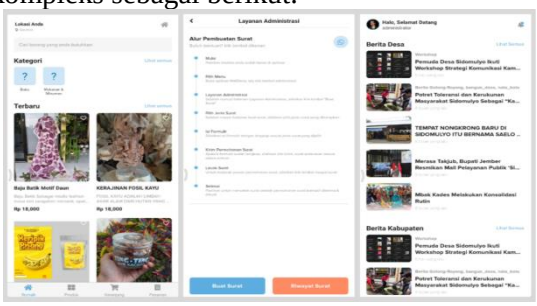
Pemecahan aplikasi menjadi lebih kecil sesuai dengan *service* dan fungsi-nya masing-masing akan menyebabkan komunikasi antar fungsi dalam aplikasi tidak dapat terjadi lagi. Oleh sebab itu, untuk melakukan pertukaran informasi dalam pendekatan *microservice* dibutuhkan suatu metode khusus. Terdapat beberapa cara yang dapat digunakan untuk melakukan interaksi pengambilan data. Pada penelitian ini komunikasi antar *service* menggunakan metode *Choreography Internal Communication*. Metode ini menekankan pada kesepakatan atau konsensus antar *microservice* terkait cara dan format yang digunakan dalam berkomunikasi, sehingga memudahkan dalam mengintegrasikan *microservice* satu dengan yang lain (Megargel et al., 2021). Metode komunikasi model ini melakukan pertukaran data antar *service* terjadi dengan cara *asynchronous* atau biasa disebut dengan *event-driven*. Sehingga pada penerapan-nya, seluruh data yang akan dikirimkan harus melewati *Message broker* terlebih dahulu sebelum di konsumsi oleh masing-masing *service* yang membutuhkan (Petrasch, 2017). Penulis akan menggunakan RabbitMQ sebagai *message broker*. *RabbitMQ* menggunakan konsep *Advanced Messaging Queuing Protocol* atau AMQP sebagai konsep transportasi data.

Dalam pengembangan aplikasi menggunakan arsitektur *microservice s*, mengharuskan tiap *service* tidak terikat satu sama lain atau *Loose Coupling* (Richardson, 2018). Sehingga pada implementasi *microservice* pada aplikasi Malldesa, penulis akan menggunakan teknologi *virtual machine* khusus, sebagai simulasi untuk memenuhi persyaratan dalam pengembangan aplikasi menggunakan arsitektur *microservice*. Penulis akan menggunakan teknologi *virtual machine* yang dimiliki oleh Docker. Docker adalah sebuah teknologi yang digunakan sebagai tempat untuk membungkus atau menempatkan aplikasi ke dalam sebuah kontainer (Jaramillo et al., 2016). Sehingga dapat dikatakan bahwa aplikasi akan terkontainerisasi atau terisolasi pada *environment* tersebut. Maka dengan menggunakan teknik ini penerapan arsitektur *microservice* dapat dilakukan tanpa harus mengeluarkan *cost* atau biaya yang sangat banyak karena harus menggunakan perangkat fisik. Oleh sebab itu penulis berharap implementasi arsitektur *microservice* pada aplikasi malldesa menggunakan metode *Choreography Internal Communication* dapat berjalan dengan lancar dan sesuai harapan.

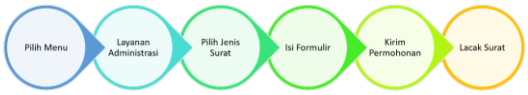
2. TINJAUAN PUSTAKA

A. Malldesa

Aplikasi malldesa memiliki fitur-fitur yang kompleks sebagai berikut:



Gambar 1. Fitur lapak, administrasi, dan berita
Sumber: Nusantara, 2022



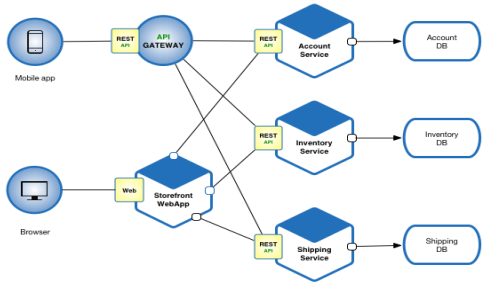
Gambar 2. Konsep alur pengajuan surat dalam fitur administrasi desa pada aplikasi malldesa
Sumber: Nusantara, 2022

Alur pengajuan surat dalam aplikasi adalah seperti yang tertuang pada gambar di atas, pengguna dapat memilih menu administrasi, kemudian memilih jenis surat yang diinginkan, setelah itu mengisi formulir kemudian mengirimkan permohonan surat tersebut. Langkah selanjutnya adalah menunggu pihak operator desa untuk menyetujui atau menolak permohonan surat tersebut.

Jurnal Smart Teknologi
Vol. 5, No. 2, Januari 2024, Halaman 144 – 154
ISSN: 2774-1702, http://jurnal.unmuhjember.ac.id/index.php/JST

B. Arsitektur *Microservice*

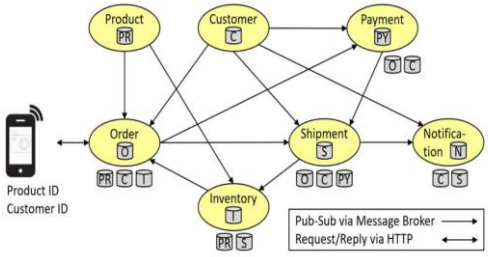
Seperti yang tertuang di dalam buku yang berjudul "*Microservice* Pattern" oleh Crish Richardson (Richardson, 2018), mengatakan bahwa Arsitektur *Microservice* adalah salah satu pendekatan dalam merancang bangun atau design sistem sebuah aplikasi. Setiap layanan tersebut memiliki fungsionalitas dan data yang terpisah, berfungsi spesifik (*high cohesion*) sesuai dengan *service* yang dijalankan, tidak saling bergantung pada layanan atau *service* lainnya (*loose coupling*), serta memiliki komunikasi yang ter definisi dan ter standarisasi melalui antarmuka API (*Application Programming Interface*) (Newman, 2015). Adapun secara umum konsep *microservice* terdiri dari beberapa layanan seperti pada gambar di bawah ini :



Gambar 3. Diagram arsitektur *microservice*
Sumber: microservice.io/patterns/microservice.html

C. *Choreography Internal Communication*

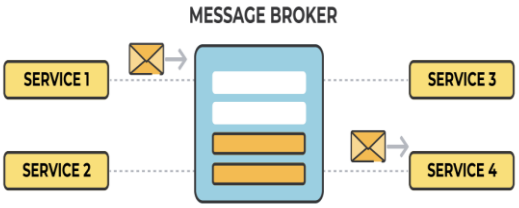
Pertukaran data antar *Microservice* adalah hal yang sangat krusial, pasalnya setiap *service* akan selalu membutuhkan data dari *service* yang lain meskipun hal tersebut sangat dihindari (Nadareishvili et al., 2016). Kemudian menurut (Rudrabhatla, 2018) pertukaran data antar *service* pada arsitektur *microservice* dapat dibagi menjadi dua metode yaitu menggunakan *Choreography* dan *Orchestration*. Perbedaan utama dari kedua pola tersebut adalah pada pola *Choreography* tidak memerlukan sebuah *service* khusus yang mengatur seluruh proses bisnis aplikasi, sehingga ketergantungan setiap *service* dapat dikurangi (Richardson, 2018). Metode komunikasi dengan gambar berikut:



Gambar 4. Ilustrasi *Choreography Internal Communication*
Sumber: Megargel, 2021

D. *Message broker*

Message broker adalah sebuah perangkat lunak (*software*) yang bertindak sebagai perantara (mediator) antara pengirim pesan (*message producer*) dan penerima pesan (*message consumer*) dalam sebuah sistem yang menggunakan teknik *messaging*. *Message broker* bertugas untuk menerima pesan dari *message producer*, menyimpan pesan dalam antrean atau topik (*queue* atau topic), dan meneruskan-nya *ke message consumer* yang berlangganan pada antrean atau topik yang sama (Kleppmann, 2017).



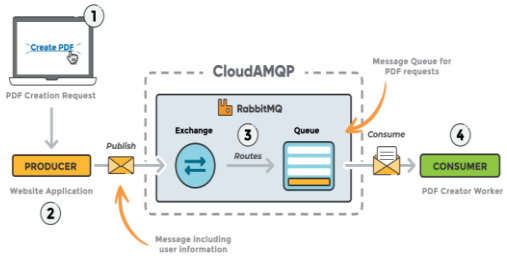
Gambar 5. Ilustrasi message broker
Sumber: Johansson, 2022

E. *RabbitMQ*

Berdasarkan dari situs resmi RabbitMQ, RabbitMQ adalah sebuah *Message broker* open-source yang dikembangkan dengan menggunakan bahasa Erlang. RabbitMQ dirancang untuk mendukung komunikasi antar aplikasi dalam sistem terdistribusi dengan menggunakan protokol AMQP (*Advanced Message Queuing Protocol*). AMQP dirancang untuk memastikan interoperabilitas dan probabilitas antar platform dan bahasa pemrograman yang berbeda, sehingga memudahkan pengembangan sistem terdistribusi yang kompleks. AMQP juga

Jurnal Smart Teknologi
Vol. 5, No. 2, Januari 2024, Halaman 144 – 154
ISSN: 2774-1702, http://jurnal.unmuhjember.ac.id/index.php/JST

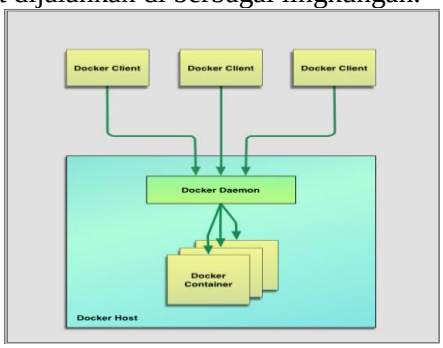
menawarkan keamanan, keandalan, dan performa yang tinggi dalam pengiriman pesan antar aplikasi (Johansson, 2022).



Gambar 6. RabbitMQ Workflow
Sumber: Johansson, 2022

F. *Docker*

Docker adalah platform open-source yang digunakan untuk membangun, mengemas, dan menjalankan aplikasi dalam container. Docker memungkinkan para pengembang untuk membuat aplikasi dalam lingkungan yang terisolasi, portabel, dan dapat berjalan pada berbagai platform dan infrastruktur (Turnbull, 2017). Menurut situs resmi Docker, cara kerja Docker adalah dengan mengemas aplikasi dan dependensi nya ke dalam sebuah container yang dapat dijalankan di berbagai lingkungan.



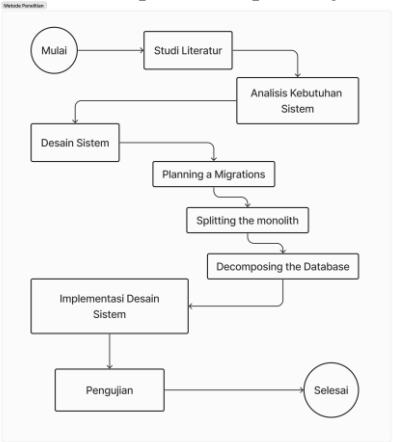
Gambar 7. Ilustrasi arsitektur docker
Sumber: Turnbull, 2017

3. METODE PENELITIAN

A. Tahapan Penelitian

Tahapan penelitian dalam menerapkan arsitektur *microservice* menggunakan metode *Choreography Internal Communication* adalah sebagai berikut. Langkah pertama dalam penelitian ini diawali dengan melakukan studi literatur, kemudian dilanjutkan dengan melakukan analisis kebutuhan sistem, kemudian pada tahap ke tiga adalah melakukan desain sistem. Setelah itu langkah selanjutnya adalah melakukan implementasi atau penerapan desain sistem yang telah dilakukan sebelumnya. Penerapan yang telah dilakukan akan dilakukan penelitian guna mendapati hasil dari implementasi yang telah dilakukan.

Berdasarkan urutan dari proses pada tahapan penelitian, metode pengembangan aplikasi yang penulis gunakan adalah metode pengembangan Waterfall. Metode pengembangan *Waterfall* identik dengan alur proses yang runtut dari atas ke bawah (Fitria et al., 2020). Metode pengembangan *Waterfall* bersifat linear, proses atau tahapan yang telah ditentukan harus terselesaikan terlebih dahulu kemudian dapat lanjut ke tahapan berikutnya, begitu seterusnya sampai seluruh proses terselesaikan dan aplikasi dapat berjalan.



Gambar 8. Diagram alur Tahapan Penelitian
Sumber: Alur Penelitian

B. Studi Literatur

Tahapan awal dalam melakukan penelitian pada penelitian ini diawali dengan melakukan studi literature. pada tahapan ini langkah-langkah yang dilakukan adalah dengan mencari sumber Pustaka atau studi literatur dari buku, jurnal, paper, dan penelitian yang telah dilakukan berkaitan dengan penelitian yang akan penulis lakukan. Guna dapat membantu dalam menyelesaikan persoalan yang dapat dipecahkan dalam penelitian ini.

C. Analisis Kebutuhan

Tahap Analisis Kebutuhan, pada tahap ini penulis membagi menjadi dua tahap analisis, tahap analisis yang pertama adalah analisis kebutuhan sistem berdasarkan studi literature

Jurnal Smart Teknologi
Vol. 5, No. 2, Januari 2024, Halaman 144 – 154
ISSN: 2774-1702, http://jurnal.unmuhjember.ac.id/index.php/JST

yang telah dilakukan pada tahap sebelumnya. Kemudian yang kedua adalah analisis kebutuhan perangkat keras dan perangkat lunak.

Tabel 1. Spesifikasi Kebutuhan Sistem

Perangkat Keras	**Nama**
Processor	AMD Ryzen 9 5900HX
RAM	DDR4 16GB
Storage	SSD NVME 512GB
Perangkat Keras Server	**Nama**
Processor	Intel I3 Gen 2
RAM	DDR3 12GB
Storage	SSD SATA 128GB
Perangkat Lunak	**Nama**
Operating System	Windows 11 dan Ubuntu server
Text Editor	VSCode
Tools Uji Coba	Apache Jmeter v5.5

Sumber: Hasil Penelitian

D. Desain Sistem

Tahap desain sistem, pada tahap ini penulis menerapkan langkah dalam melakukan implementasi arsitektur *microservice* atau migrasi dari arsitektur monolith ke arsitektur *microservice* (Newman, 2020). Proses migrasi dari arsitektur lama ke arsitektur *microservice* dibagi menjadi tiga tahapan, tahapan yang pertama adalah melakukan *planning a migration* atau rencana perpindahan, kemudian tahap yang kedua adalah *splitting the monolith* atau pemisahan aplikasi, lalu tahapan yang ketiga adalah *decomposing the database* atau penguraian *database*.

E. Implementasi Desain Sistem

Tahap Implementasi Desain Sistem, setelah semua tahapan selesai dilakukan, maka langkah selanjutnya adalah Implementasi desain sistem yang telah dirancang. Proses implementasi ini mencangkup proses penguraian *database*, proses penulisan ulang kode sesuai dengan desain sistem yang telah dilakukan pada tahap sebelumnya.

F. Pengujian

Tahap akhir pada penelitian ini adalah melakukan pengujian terhadap implementasi atau penelitian yang telah dilakukan. Pengujian pada penelitian ini dilakukan dengan mekanisme *Load Test*. *Load Test* atau *Load Testing* adalah merupakan jenis pengujian kinerja aplikasi yang bertujuan untuk mengevaluasi mengenai bagaimana sistem atau aplikasi bekerja, dengan memberikan beban atau sejumlah lalu lintas penggunaan aplikasi yang dilakukan secara *concurrent* atau bersamaan (Menascé, 2002). Arti kata "*Load*" dalam *load testing* merujuk pada aktivitas penggunaan aplikasi secara bersamaan yang dapat ditangani oleh sistem sebelum mengalami penurunan dalam hal waktu tanggapan, *throughput*, atau metrik kinerja lainnya (Zhen Ming Jiang et al., 2008). Tujuan penggunaan mekanisme *load testin*g ditujukan agar dapat membantu dalam mengidentifikasi batasan sistem, seperti maksimum pengguna yang dapat ditangani oleh sistem, tingkat transaksi maksimum serta waktu yang digunakan untuk mengembalikan respon dengan beban tertentu (Erinle, 2017).

Tabel 2. Skenario Pengujian

Arsitektur	**Jumlah Akses Virtual User**	**Ramp-up Periode**
Monolith	50 *user*	100 detik
	100 *user*	100 detik
	200 *user*	100 detik
	400 *user*	100 detik
Microservice	50 *user*	100 detik
	100 *user*	100 detik
	200 *user*	100 detik
	400 *user*	100 detik

Sumber: Rancangan Pengujian

4. HASIL DAN PEMBAHASAN

A. Implementasi Desain Sistem

Setelah melakukan perancangan desain sistem aplikasi yang akan dibangun, langkah selanjutnya adalah melakukan penerapan desain sistem yang telah dirancang sebelumnya. Pada bab ke – empat ini, proses dekomposisi yang dilakukan pada bab ke – tiga akan diimplementasikan ke dalam program.

Tahap implementasi desain sistem terbagi menjadi beberapa tahap. Tahap pertama adalah proses implementasi dari hasil *splitting the monolith* yang berupa *service-service* aplikasi yang telah dipisah, sesuai dengan proses bisnis dan kebutuhan untuk menjalankan aplikasi Malldesa, kemudian tahap kedua adalah proses *refactoring database* atau *decomposing the database* berupa meng-aplikasikan *database* yang telah dipecah menjadi *service* masing-

Jurnal Smart Teknologi
Vol. 5, No. 2, Januari 2024, Halaman 144 – 154
ISSN: 2774-1702, http://jurnal.unmuhjember.ac.id/index.php/JST

masing, sesuai dengan *service* aplikasi Malldesa.

1. Implementasi *Splitting the Monolith*

Berdasarkan pada tahap desain sistem yang ada pada bab ke-tiga, proses *Splitting the Monolith* dapat dibagi menjadi empat hal utama, diantaranya adalah:

1) Memahami arsitektur pada aplikasi lama
2) memetakan domain
3) memahami bahasa pemrograman dan teknologi yang akan diadopsi atau digunakan
4) menyiapkan infrastruktur atau alat yang dibutuhkan

Setelah mengadopsi ke-empat *point* tersebut, didapatkan *service-service* yang akan diterapkan pada aplikasi Malldesa dengan arsitektur *Microservice* seperti berikut:

Tabel 3. Kebutuhan Sistem *Service*

Nama *Service*	**Sistem yang dibutuhkan**	**Sistem yang diterapkan**
API Gateway *Service*	Sistem Operasi	Alpine Linux
	Engine / Web server	Nginx Web Server
	Bahasa Pemrograman	PHP
	Runtime	PHP-FPM 8.1
	Database	MySQL
Layanan *Service*	Sistem Operasi	Alpine Linux
	Engine / Web server	Nginx Web Server
	Bahasa Pemrograman	PHP
	Runtime	PHP-FPM 8.1
	Database	MySQL
Parekraf *Service*	Sistem Operasi	Alpine Linux
	Engine / Web server	Nginx Web Server
	Bahasa Pemrograman	PHP
	Runtime	PHP-FPM 8.1
	Database	MySQL
Lapak *Service*	Sistem Operasi	Alpine Linux
	Engine / Web server	Nginx Web Server
	Bahasa Pemrograman	PHP
	Runtime	PHP-FPM 8.1
	Database	MySQL
Sistem *Service*	Sistem Operasi	Alpine Linux
	Engine / Web server	Nginx Web Server
	Bahasa Pemrograman	PHP
	Runtime	PHP-FPM 8.1
	Database	MySQL
Surat *Service*	Sistem Operasi	Alpine Linux
	Engine / Web server	Nginx Web Server
	Bahasa Pemrograman	PHP
	Runtime	PHP-FPM 8.1
	Database	MySQL
	Sistem Operasi	Alpine Linux
Message broker Service	Platform	RabbitMQ
Bucket *Service*	Webserver	Nginx Web Server

Sumber: Hasil Penelitian

2. Penerapan *Service* ke dalam Docker

Guna memenuhi pola penerapan arsitektur *microservice*, *service* yang telah dihasilkan akan diimplementasikan dengan menggunakan *virtual machine* atau mesin virtual dengan menggunakan *docker* sebagai virtualisasi sistem, guna mendapati *service* yang terisolasi selayaknya berada pada server yang berbeda. Penggunaan docker harus menuliskan kode program sistem yang dibutuhkan pada file yaitu *Dockerfile* dan *Docker Compose*. *Dockerfile* dan *Docker Compose* yang dijalankan akan membentuk *Docker Container* yang saling terisolasi. *Docker Container* dapat saling berkomunikasi antar *Container* dengan menggunakan *Docker Network*.

3. Implementasi *Load Balancer*

Setiap *service* yang terdapat lebih dari satu *node* atau aplikasi penulis implementasi kan menggunakan *load balancer*. *Load balancer* adalah sebuah mekanisme untuk mendistribusikan lalu lintas jaringan atau membagi beban kerja dalam beberapa server. *Load balancer* yang digunakan pada penelitian ini menggunakan konfigurasi *NGINX load balancer* dengan konfigurasi seperti berikut:

```
1  upstream site-containers {
2      server nginx-gateway-service2;
3      server nginx-gateway-service;
4  }
5
6  server {
7      listen 80;
8      server_name _ ;
9      index index.php index.html;
10     error_log  /var/log/nginx/error.log;
11     access_log /var/log/nginx/access.log;
12     root /var/www/public;
13     location / {
14         proxy_pass http://site-containers;
15         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
16         proxy_set_header Host $http_host;
17         proxy_set_header X-Forwarded-Proto $scheme;
18         proxy_buffering off;
19     }
20 }
```

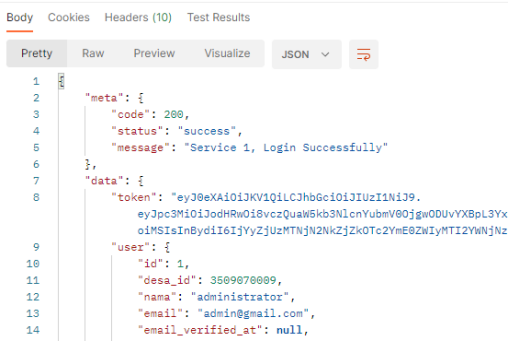
Gambar 9. Konfigurasi *Load Balancer* pada NGINX

Sumber: Hasil Penelitian

Pada baris 1 sampai 4 adalah menjelaskan mengenai *upstream site-containers* atau konfigurasi yang mendefinisikan dari group

Jurnal Smart Teknologi
Vol. 5, No. 2, Januari 2024, Halaman 144 – 154
ISSN: 2774-1702, http://jurnal.unmuhjember.ac.id/index.php/JST

server yang tersedia sebagai pembagi beban pada *load balancer*. Pada konfigurasi tersebut, blok *upstream site-containers* mendefinisikan sebuah kelompok (group) dari server-server yang disebut "site-containers". Dua server yang terdaftar di dalamnya, "nginx-gateway-*service*2" dan "nginx-gateway-*service*", akan menjadi target bagi distribusi lalu lintas oleh *Nginx*. Hasil implementasi *load balancer* dapat dilihat pada hasil keluaran yang di berikan oleh masing-masing server, seperti yang terlihat pada gambar berikut:

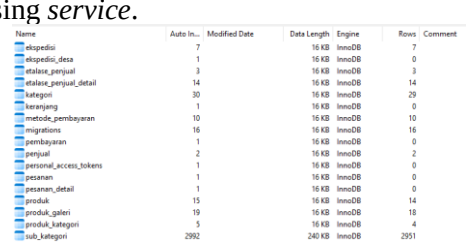


Gambar 10. Hasil Tanggapan dari *Load Balancer*
Sumber: Hasil Penelitian

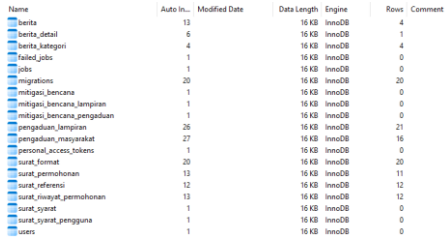
Hasil tanggapan dari *load balancer* dapat diidentifikasi pada baris nomor 5. Pada baris tersebut pesan dengan kalimat "*Service* 1, login successfully", yang mana pesan tersebut adalah indikasi bahwasanya hasil tanggapan tersebut dikirim melalui *service* 1.

4. Implementasi *Decompositing the Database*

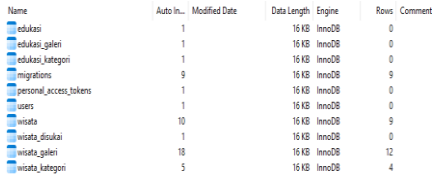
Implementasi *Decompositing the Database* berdasarkan pada setiap *service* yang sesuai. Berikut adalah hasil dari implementasi *database* yang telah dipisah, sesuai dengan masing – masing *service*.



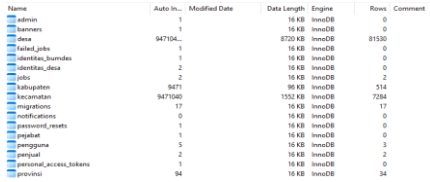
Gambar 11. Tabel *Database Lapak Service*
Sumber: Hasil Penelitian



Gambar 12. Tabel *Database* Layanan *Service*
Sumber: Hasil Penelitian



Gambar 13. Tabel *Database* Parekraf *Service*
Sumber: Hasil Penelitian



Gambar 14. Tabel *Database* Sistem *Service*
Sumber: Hasil Penelitian

B. Hasil Implementasi Sistem

Hasil dari implementasi desain sistem yang telah dikembangkan sesuai dengan rancangan yang telah dilakukan pada bab tiga. Hasil implementasi arsitektur *microservice* menggunakan metode *Choreography Internal Communication* menghasilkan sebanyak 9 *stack*, 33 *container docker*, dengan jumlah *image* yang digunakan adalah 10 image dengan total ukuran keseluruhan image adalah 5.78GB. dengan penggunaan 1 *volume* serta total 16 *Network*. Hasil implementasi arsitektur *microservice* tidak mengubah *parameter endpoint* yang telah ada. Sehingga untuk *parameter endpoint* aplikasi Malldesa adalah sebagai berikut.

C. Pengujian Hasil Implementasi Desain Sistem

Setelah tahap implementasi desain sistem, tahap selanjutnya dilanjutkan proses pengujian dari hasil implementasi desain sistem yang telah dirancang pada tahap sebelumnya. Proses pengujian dilakukan dengan menggunakan

Jurnal Smart Teknologi
Vol. 5, No. 2, Januari 2024, Halaman 144 – 154
ISSN: 2774-1702, http://jurnal.unmuhjember.ac.id/index.php/JST

mekanisme *load test* dan di lakukan dengan menggunakan tools *Apache Jmeter*. Mekanisme pengujian *load testing* dilakukan dengan membuat skenario pengujian dengan mengirimkan permintaan terhadap beberapa *endpoint API*, yang sesuai dengan kondisi asli ketika aplikasi digunakan. Sehingga skenario pengujian yang akan dijalankan adalah sebagai berikut:

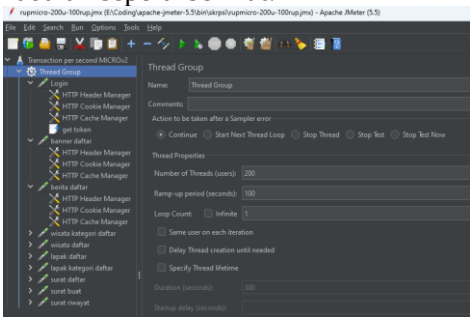
Tabel 4. Skenario *Endpoint* pengujian

Endpoint API	Keterangan
Login	Melakukan login untuk mendapatkan token
Banner Daftar	Mendapatkan data daftar banner
Berita Daftar	Mendapatkan daftar berita
Wisata Kategori Daftar	Mendapatkan data daftar kategori wisata
Wisata Daftar	Mendapatkan data daftar wisata
Lapak Daftar	Mendapatkan data daftar lapak
Lapak Kategori Daftar	Mendapatkan data daftar kategori lapak
Surat Daftar	Mendapatkan data daftar surat
Surat Buat	Mengirimkan data untuk pembuatan surat
Surat Riwayat	Mendapatkan data riwayat surat

Sumber: Hasil Penelitian

1. Implementasi Pengujian

Proses implementasi pengujian dilakukan dengan pengiriman *request* atau permintaan ke server dengan skema seperti pada bab ke 3. setiap transaksi yang dijalankan, akan melalui ke 10 *endpoint* skenario yang tertera pada tabel 4. Implementasi pengujian dilakukan dengan menggunakan *tools Apache Jmeter*. Mekanisme implementasi pengujian dengan mengirimkan sejumlah permintaan transaksi per detik yang sesuai dengan skema pada bab 3. Sehingga parameter yang ditetapkan pada aplikasi *Apache Jmeter* adalah seperti berikut.



Gambar 15. Salah Satu Contoh Parameter Pengujian pada *Apache Jmeter*
Sumber: Hasil Penelitian

Salah satu contoh parameter pengujian dengan aplikasi *Apache Jmeter* seperti pada gambar 45 diatas. Pengaturan yang perlu dilakukan adalah terdapat pada kolom "Number of Threads" sebagai jumlah pengguna *virtual* yang akan mengeksekusi pengujian, kemudian "Ramp-up periode" adalah waktu yang digunakan untuk mengeksekusi sejumlah pengguna yang telah ditetapkan pada parameter "Number of Threads", sehingga jika pada gambar diatas parameter pada "Number of threads" adalah sejumlah 200 *user*, kemudian pada "Ramp-up periode" adalah 100, yang mana konfigurasi ini berarti selama 100 detik jumlah pengguna akan bertambah secara bertahap dari 0 sampai 200 pada detik ke 100 setelah pengujian dilakukan.

2. Analisias Hasil Pengujian

Tahap analisis hasil pengujian dilakukan setelah tahap implementasi pengujian yang dilakukan sesuai dengan skenario yang terdapat pada bab 3 di tabel 2, Serta berdasarkan tabel 4 skenario *endpoint* pengujian. Tahap analisis hasil pengujian dilakukan pada kedua arsitektur baik itu *microservice* maupun *monolith*.

Tabel 5. Percobaan 1, 50 Number of Thread, 100 Ramp up periode, 500 Sample Request

Arsitektur	Response Time			Error
	Average	Min	Max	
Monolith	7110.53	191	13179	0%
Microservice	10887.41	386	21910	0%

Sumber: Hasil Pengujian

Tabel 6. Percobaan 2, 100 Number of Thread, 100 Ramp up periode, 1000 Sample Request

Arsitektur	Response Time			Error
	Average	Min	Max	
Monolith	22992.41	192	32211	0%
Microservice	29713.33	431	48788	0%

Sumber: Hasil Pengujian

Tabel 7. Percobaan 3, 200 Number of Thread, 100 Ramp up periode, 2000 Sample Request

Arsitektur	Response Time			Error
	Average	Min	Max	
Monolith	52829.30	556	60150	52.20%
Microservice	13254.27	2	120034	74.15%

Sumber: Hasil Pengujian

Tabel 8. Percobaan 4, 400 Number of Thread, 100 Ramp up periode, 4000 Sample Request

Arsitektur	Response Time			Error
	Average	Min	Max	
Monolith	57805.07	1162	60118	92.50%
Microservice	8595.72	2	120045	93.03%

Sumber: Hasil Pengujian

Pengujian yang telah dilakukan pada kedua arsitektur menunjukkan kinerja arsitektur *monolith* lebih baik daripada arsitektur *microservice* pada pengujian dengan pengguna dibawah 200 pengguna. Meski demikian, pada percobaan ketiga, arsitektur *microservice* berhasil melampaui arsitektur *monolith* dalam waktu respons rata-rata, dengan arsitektur *microservice* memerlukan waktu rata-rata 13 detik, sementara arsitektur *monolith* membutuhkan waktu rata-rata 52 detik. Namun, perbedaan signifikan terlihat dalam waktu maksimum yang dibutuhkan, di mana arsitektur *microservice* mencapai 120 detik, sementara arsitektur *monolith* hanya 60 detik. Hal ini menandakan bahwa arsitektur *microservice* tidak selalu lebih baik dalam semua skenario.
Pada pengujian dengan hasil balikan *error*, arsitektur *microservice* menunjukkan tingkat kesalahan yang lebih tinggi, mencapai 74.15%, dibandingkan dengan 52.20% pada arsitektur *monolith* sedangkan pada pengujian skenario keempat arsitektur *microservice* mengembalikan hasil *error* sebesar 93.03% sedangkan pada arsitektur monolith adalah sebesar 92.50%. Hasil ini menegaskan bahwa arsitektur *microservice* dengan metode *Choreography Internal Communication* belum secara konsisten lebih unggul daripada arsitektur *monolith* dalam memproses permintaan pengguna pada rentang 50-100 pengguna. Namun, ketika jumlah pengguna mendekati 200 atau lebih, implementasi arsitektur *microservice* performa lebih baik.

5. KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan penelitian serta hasil pengujian pada implementasi arsitektur *microservice* pada aplikasi Malldesa dengan menggunakan metode *Choreography Internal Communication* mendapati kesimpulan sebagai berikut: 1) Implementasi Arsitektur *Microservice* pada aplikasi Malldesa dengan menggunakan metode *Choreography Internal Communication* dapat dilakukan dan berjalan dengan cukup baik, serta metode yang digunakan dapat menjadi solusi dalam pertukaran data antar *service*. 2) Pengujian menunjukkan kinerja arsitektur *microservice* lebih baik pada skala pengguna besar (200 dan 400 *thread*). Namun, peningkatan beban pada Microservices juga mengakibatkan peningkatan tingkat kesalahan yang signifikan, mencapai 93.03% pada skala 400 *thread*.

B. Saran

Berdasarkan kesimpulan yang didapatkan. Berikut adalah beberapa saran yang dapat dilakukan terkait penelitian implementasi arsitektur *microservice* pada aplikasi Malldesa dengan menggunakan metode *Choreography Internal Communication* adalah sebagai berikut: 1) Api Gateway *service* yang digunakan pada penelitian ini menggunakan Nginx dan *framework laravel* sebagai *Authentikasi*, sehingga disarankan untuk menerapkan teknologi Api *Gateway* yang lebih sesuai guna mendapatkan hasil yang lebih baik, teknologi Api *Gateway* yang dapat penulis sarankan adalah seperti berikut: *kong Api Gateway, AWS Api Gateway*, *Spring Cloud*, *Microsoft Azure*, dll; 2) Alat pengujian yang digunakan hanya sebatas menggunakan *Apache Jm*eter. Sehingga disarankan menggunakan alat pengujian yang lainnya, seperti: *load runner, appium, locust, dan gatling*; 3) Mekanisme pengujian yang dilakukan pada penelitian ini hanya sebatas pengujian *Load Testing*. Sehingga disarankan untuk melakukan pengujian performa dengan mekanisme yang lain, seperti: *Performance Testing*, *Stress Testing, Spike Testing dan Soak Testing*. 4) Uji coba serta implementasi dilakukan dengan menggunakan komputer server dengan spesifikasi yang berbeda dan lebih memadai, sehingga memungkinkan untuk mendapatkan hasil pengujian yang lebih akurat dan optimal.

6. DAFTAR PUSTAKA

Daya, S., Van, N., Kameswara, D., Carlos, E., Ferreira, M., Glozic, D., Gucer, V., Gupta, M., Joshi, S., Lampkin, V., Martins, M., Narain, S., & Vennam, R. (2015). *Redbooks Microservice s from Theory to Practice Creating Applications in IBM Bluemix Using the Microservice s Approach*.

Erinle, B. (2017). *Performance testing with JMeter 3 : enhance the performance of your web application*.

Fitria, O., Hasanah, N., Pd, M., & Untari, R. S. (2020). *BUKU AJAR REKAYASA PERANGKAT LUNAK Diterbitkan oleh UMSIDA PRESS UNIVERSITAS MUHAMMADIYAH SIDOARJO 2020*.

Jaramillo, D., Nguyen, D. V., & Smart, R. (2016). *Leveraging microservice s architecture by using Docker technology*.

Johansson, lovisa. (2022). *THE OPTIMAL RABBITMQ GUIDE From Beginner to Advanced*.

Kleppmann, M. (2017). *Designing Data-Intensive Applications THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS*.

Megargel, A., Poskitt, C. M., & Shankararaman, V. (2021). *Microservice s Orchestration vs. Choreography: A Decision Framework*.

Menascé, D. A. (2002). Load testing of Web sites. *IEEE Internet Computing*, *6*(4), 70–74. https://doi.org/10.1109/MIC.2002.1020328

Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). *microservice - architecture-aligning-principles-practices-and-culture (1)*.

Newman, S. (2015). *Building Microservice s*. http://safaribooks*online*.com

Nusantara, A. F. P., Ali Muharom, L., & Oktavianto, H. (2022). Pendampingan Penggunaan Aplikasi Malldesa Untuk Administrasi Persuratan Desa. *I-Com: Indonesian Community Journal*, *2*(3), 754–764. https://doi.org/10.33379/icom.v2i3.2002

Petrasch, R. (2017). *Model-based Engineering for Microservice Architectures using Enterprise Integration Patterns for inter-service Communication*.

Priyadarshini S, & Shilpa G. (2017). *Microservice* s Architecture. *International Research Journal of Computer Science (IRJCS) Issue 05, 4*. www.irjcs.com

Rudrabhatla, C. K. (2018). Comparison of Event Choreography and Orchestration Techniques in *Microservice* Architecture. In *IJACSA) International Journal of Advanced Computer Science and Applications* (Vol. 9, Issue 8). www.ijacsa.thesai.org

Sendiang, M., Kasenda, S., & Purnama, J. (2018). Implementasi Teknologi Mikro*service* pada Pengembangan Mobile Learning. In *Journal of Applied Informatics and Computing (JAIC)* (Vol. 2, Issue 2). http://jurnal.polibatam.ac.id/index.php/JAIC

Turnbull, J. (2017). *The Docker Book*.

Zhen Ming Jiang, & Ahmed E. Hassan. (2008). *ICSM 2008 : Proceedings of the 2008 IEEE International Conference on Software Maintenance : September 28 - October 4, 2007, Beijing, China, [and] ; Proceedings of the 2008 Frontiers of Software Maintenance : September 30 - October 2, 2007, Beijing, China*. IEEE.