Jurnal Elektronik Ilmu Komputer Udayana
Volume 11, No 4. Mei 2023

p-ISSN: 2301-5373
e-ISSN: 2654-5101

Comparison Between Microservices and Monolith Software Architecture

Gede Gery Sastrawan[a1], I Putu Gede Hendra Suputra[a2]

[a]Informatics Department, Udayana University
South Kuta, Badung, Bali, Indonesia
[1]gerysastrawan123@gmail.com
[2] hendra.suputra@unud.ac.id

Abstract

A comparative analysis is the process of comparing items to one another and distinguishing their similarities and differences. Big Data is a set of massive data that has 3 characteristics in general. The characteristic is volume, variety, and velocity. In order to process such a big amount of data, we need some kind of software that can efficiently process the data with an available resource. Two of much architectural style on software development known are microservices and monolithic architecture. There are many differences between microservices and monolithic architecture that need to be considered when choosing the right architecture to use. The study will compare between microservices and monolithic architecture to decide which architecture would be preferred to handle and process many data such as big data. The result showed that microservices has less and distributed resource usage compared with monolithic architecture.

Keywords: *Big Data, Microservices, Monolithic*

1. Pendahuluan

Big data merupakan sebuah istilah yang mendeskripsikan data dengan volume yang besar, baik data terstruktur maupun tidak terstruktur yang saat ini sudah tidak asing lagi. Namun yang terpenting bukan jumlah dari data tersebut, melainkan bagaimana sebuah organisasi atau perusahaan memanfaatkan data tersebut agar dapat berguna bagi mereka. Big data dapat dianalisa untuk menambah wawasan yang digunakan untuk mengambil keputusan maupun strategi yang lebih baik kedepannya demi kemajuan organisasi. Big data memiliki beberapa karakteristik yang biasanya dikenal sebagai “3V’s” yaitu Volume, Variety, dan Velocity. Volume memiliki maksud yaitu big data memiliki jumlah data yang sangat besar. Menurut artikel yang ditulis oleh Daniel Price pada cloudtweaks.com[1], setidaknya ada 2,5 triliun byte data yang dihasilkan tiap harinya melalui aktifitas aktifitas digital seperti media sosial. Data tersebut meliputi: 3,5 milyar request diproses oleh Google tiap harinya, Facebook mendapatkan 500 terrabyte data tiap harinya, Amazon mengambil data dari 152 juta pembelian dari customer untuk membantu pengguna menemukan barang untuk dibeli, dan masih banyak lagi. Variety memiliki maksud yaitu big data dapat memiliki tipe data yang beragam baik itu data terstruktur, tidak terstruktur, maupun semi terstruktur. Velocity memiliki maksud yaitu kecepatan data yang diterima ataupun diproses. Hal hal ini akan memunculkan berbagai permasalahan seperti masalah pada *availability* dan *reliability*. Dalam pengembangan perangkat lunak untuk mengolah data, terdapat beberapa gaya arsitektural untuk menyusun sebuah aplikasi atau *service* salah satunya yaitu arsitektur microservices.

Dalam pengembangan perangkat lunak, microservices merupakan sebuah gaya arsitektural yang menyusun aplikasi sebagai sebuah koleksi layanan yang memiliki koneksi yang renggang, mudah dipelihara dan diuji, dapat dideploy secara independen yang mempermudah para developer untuk membangun aplikasi skala besar dan melakukan scaling pada aplikasi tersebut[2]. Alasan penggunaan microservices sebagai gaya arsitektural pada sebuah aplikasi yaitu: *scalability*, pengembangan yang lebih cepat, keamanan data yang ditingkatkan, tata kelola data yang lebih baik, dapat menggunakan tech stack yang berbeda tiap service pada microservice. Microservices lebih mudah untuk di*scale* dibandingkan dengan arsitektur monolith yang dimana developer dapat melakukan *scaling* pada suatu service secara spesifik daripada melakukan *scaling* pada keseluruhan aplikasi. Microservices juga lebih mudah untuk dikembangkan karena developer hanya perlu fokus pada suatu service yang

membutuhkan *deployment* atau *debugging*. Masing masing service pada microservice saling berkomunikasi melalui API yang aman yang memberikan keamanan data lebih baik daripada arsitektur monolith. Banyak perusahaan yang mengubah aplikasinya dari arsitektur monolith menjadi microservices. Hal ini untuk memungkinkan optimasi pemakaian sumber daya komputasi sehingga pemakaian resource menjadi lebih optimal. Microservices menggunakan *containerization* untuk memisahkan antar service. Selain microservice, gaya arsitektur lain yang biasanya digunakan dalam pembangunan aplikasi ialah arsitektur monolith. Arsitektur monolith merupakan sebuah model tradisional dari sebuah program perangkat lunak, yang dimana dibangun sebagai satu unit[3]. Studi sebelumnya[4] mengatakan bahwa microservice lebih efisien dibanding monolith.

Pada penelitian ini, penulis akan melakukan perbandingan beban CPU pada aplikasi dengan arsitektur microservices dan arsitektur monolith. Beban CPU akan dilihat melalui jumlah CPU yang terpakai ketika menerima request. Disini penulis akan menggunakan K6 untuk melakukan *load testing* pada masing masing arsitektur.

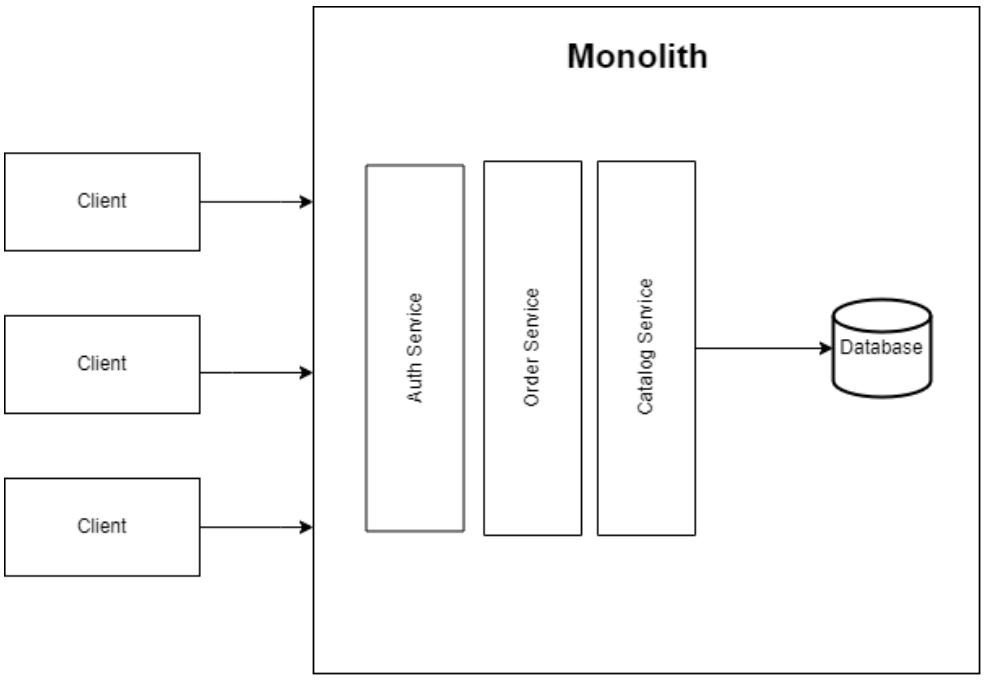
2. Metode Penelitian

2.1. Metode Pengumpulan Data

Pada penelitian ini, data yang digunakan sebagai perbandingan dari dua arsitektur ini ialah menggunakan metode observasi dan juga studi literatur melalui beberapa referensi. Observasi dilakukan dengan cara mengecek status dari masing masing *service* ketika dikirim request dari K6. K6 dapat mensimulasikan banyak *virtual user* untuk mengirim request ke suatu service. Baik service dengan arsitektur monolith maupun arsitektur microservice, akan dikontainerisasi menggunakan Docker sedangkan bahasa pemrograman yang digunakan untuk membangun aplikasi ialah Golang yang merupakan bahasa yang terkompilasi.

2.2. Perancangan Arsitektur Monolith

Arsitektur monolith merupakan salah satu gaya arsitektur yang digunakan untuk mengembangkan suatu layanan atau aplikasi. Aplikasi monolith dibuat dengan cara meletakkan semua layanan kedalam satu aplikasi. Misalnya pada aplikasi yang saya buat, secara garis besar terdapat 3 buah layanan yaitu *authentication*, *order*, dan *catalog*. Layanan *catalog* berfungsi untuk mengurus segala hal yang berhubungan dengan data barang. Layanan *order* berfungsi untuk mengurus segala hal yang berhubungan pemesanan barang. Sedangkan layanan *authentication* berfungsi untuk mengurus segala hal yang berhubungan dengan autentikasi pengguna layanan. Biasanya aplikasi yang dirancang menggunakan arsitektur monolith hanya menggunakan 1 bahasa pemrograman. Apabila digambarkan, arsitektur monolith akan terlihat seperti berikut:

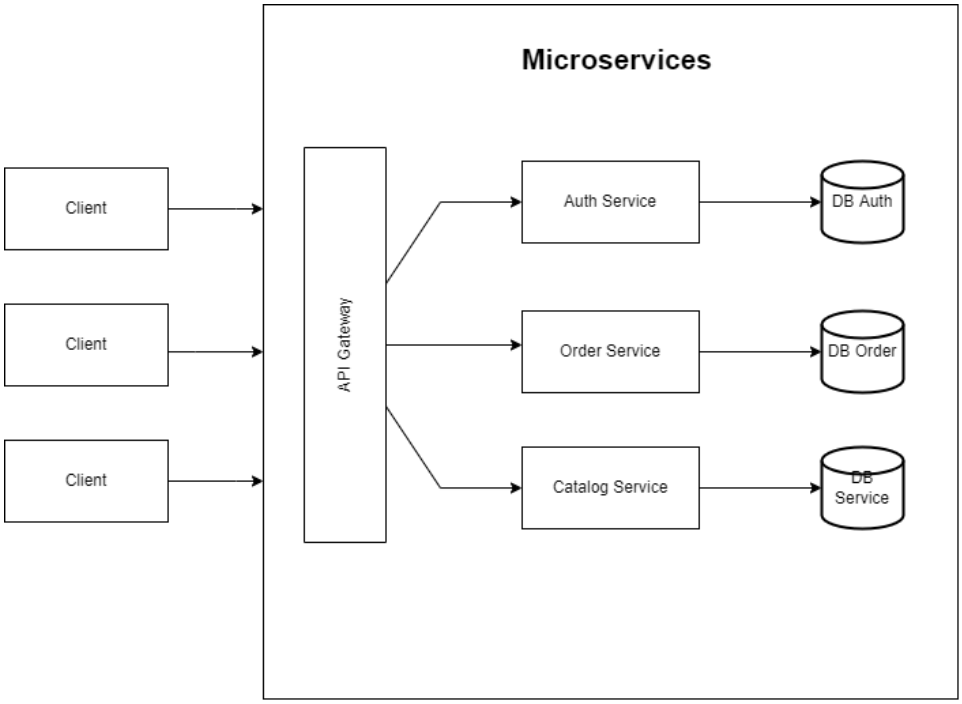


Gambar 1. Arsitektur Monolith

p-ISSN: 2301-5373
e-ISSN: 2654-5101

2.3. Perancangan Arsitektur Microservices

Microservices merupakan salah satu gaya arsitektur untuk membangun suatu aplikasi dengan cara memecah suatu aplikasi berdasarkan layanan yang dimilikinya. Misalnya pada aplikasi yang saya buat, aplikasi tersebut secara umum memiliki 3 layanan yaitu layanan *authentication*, *order*, dan *catalog*. Dalam microservices, layanan layanan tersebut dibuat dan dijalankan menjadi aplikasi atau layanan berbeda tiap layanan. Layanan layanan inilah yang saling berkomunikasi sehingga membentuk suatu microservices. Sesuai dengan namanya, *micro* artinya kecil dan *service* artinya layanan yang apabila diartikan menjadi layanan layanan kecil yang membentuk suatu aplikasi. Salah satu cara layanan layanan tersebut berkomunikasi ialah dengan menggunakan protokol HTTP. Apabila digambarkan, arsitektur microservices akan terlihat seperti berikut:



Gambar 2. Arsitektur Microservices

2.4. Mempersiapkan File Untuk Load Test

Pengujian dilakukan pada laptop MacBook dengan processor Intel Core i5. Teknologi yang saya gunakan untuk melakukan pengujian atau testing ialah K6. Grafana K6 merupakan alat *load testing* yang *opensource* yang digunakan untuk mempermudah pengujian performa suatu aplikasi. Saya melakukan testing dengan cara melakukan request terhadap ketiga layanan baik pada arsitektur monolith maupun microservices. Testing digunakan dengan mensimulasikan 100 virtual user melalui K6 ke masing masing service selama 30 detik melakukan request terus menerus tanpa henti. Berikut merupakan gambaran file testing yang akan dipakai:

```
import http from "k6/http";

import { sleep } from "k6";

export default function () {
  http.get("http://localhost:8080/ping");
  http.get("http://localhost:8081/items");
  http.get("http://localhost:8082/orders");
}
```

Gambar 3. File Testing K6

3. Hasil dan Pembahasan

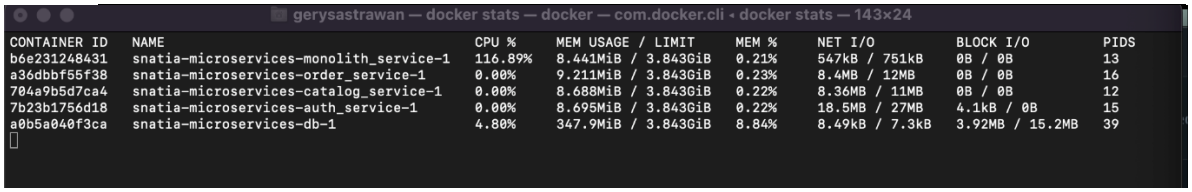
3.1. Hasil Perbandingan Pemakaian CPU Tertinggi

Berdasarkan tiga kali pengujian yang dilakukan dengan menggunakan Grafana K6 yang mensimulasikan 100 *virtual user* dengan jenis request yang sama yaitu ke layanan orders, catalog, dan authentication, didapatkan bahwa jumlah CPU yang terpakai ialah sebagai berikut:

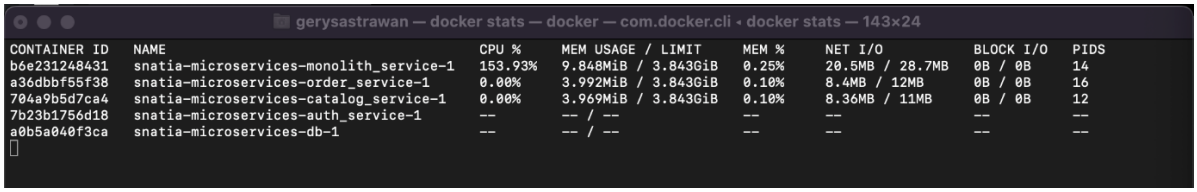
- Arsitektur Monolith

Tabel 1. Pengujian Arsitektur Monolith

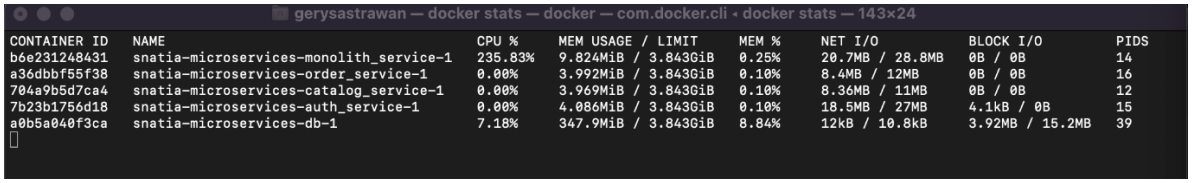
Pengujian ke-i	Pemakaian CPU tertinggi
1	116.89%
2	153.93%
3	235.83%
Rata rata	168.88%



Gambar 4. Pengujian ke – 1 Arsitektur Monolith



Gambar 5. Pengujian ke – 2 Arsitektur Monolith



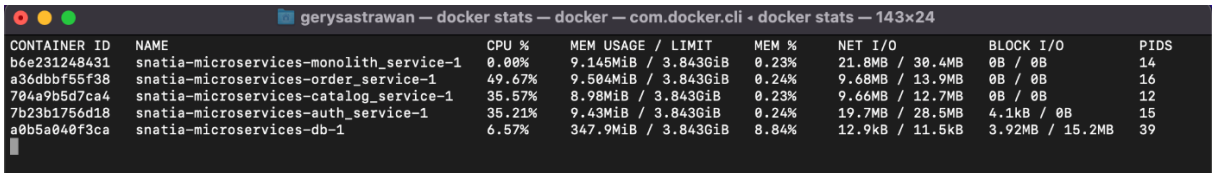
Gambar 6. Pengujian ke – 3 Arsitektur Monolith

- Arsitektur Microservice

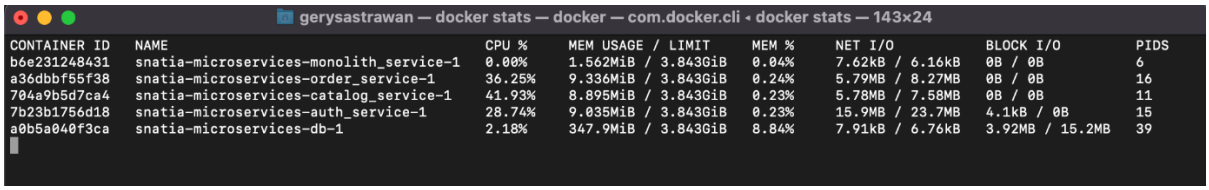
Tabel 2. Pengujian Arsitektur Microservice

Pengujian ke-i	Pemakaian CPU Tertinggi		
	Order Service	Catalog Service	Auth Service
1	49.67%	35.57%	35.21%
2	36.25%	41.93%	28.74%
3	44.32%	42.06%	50.45%
Rata rata	43,41%	39,85	38,13

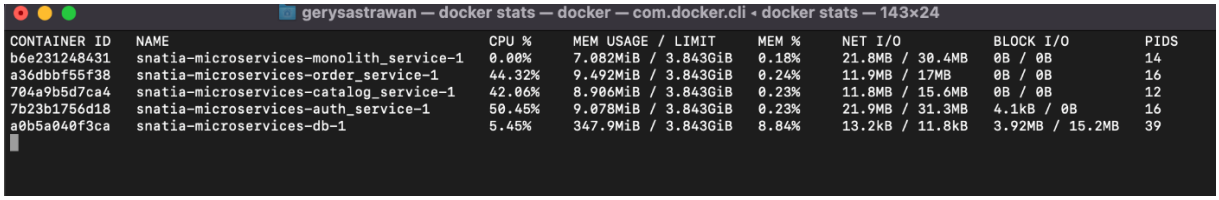
p-ISSN: 2301-5373
e-ISSN: 2654-5101



Gambar 7. Pengujian ke - 1 Arsitektur Microservice



Gambar 8. Pengujian ke – 2 Arsitektur Microservice

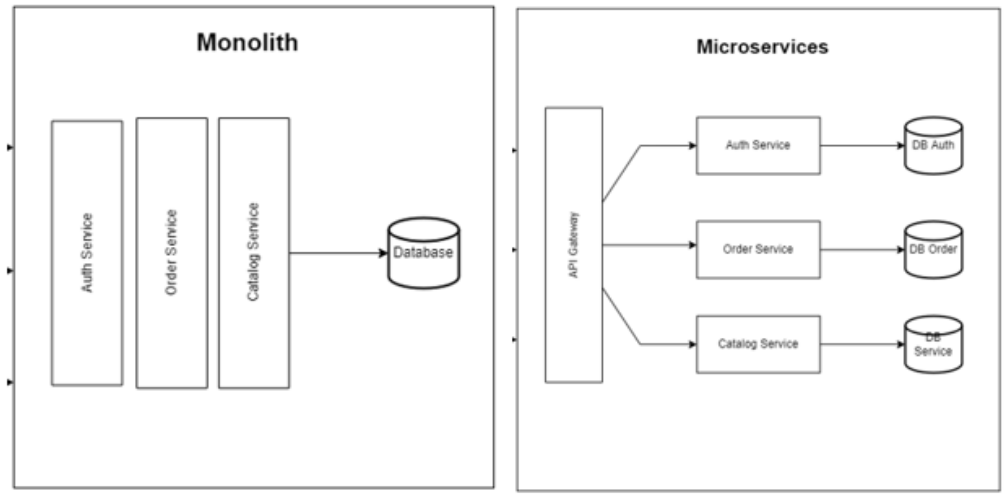


Gambar 9. Pengujian ke – 3 Arsitektur Microservice

Pemakaian CPU pada microservices lebih terdistribusi ke masing masing layanan. Sedangkan pada arsitektur monolith, terlihat bahwa pemakaian CPU naik diatas 100% yang dikarenakan 1 aplikasi monolith menerima banyak request ke masing masing layanan yang berada pada satu aplikasi.

3.2. Analisis Arsitektur

Pada microservices, apabila terdapat salah satu layanan yang mati, maka layanan lain tidak akan ikut mati[5]. Itu dikarenakan masing masing layanan pada microservice dijalankan secara independen. Kita masih bisa mengirim request ke layanan yang masih hidup namun fungsionalitasnya akan sedikit bermasalah ketika layanan tersebut membutuhkan layanan yang tadinya mati. Berbeda dengan aplikasi monolith yang semua layanan dibuat menjadi satu aplikasi. Sehingga apabila salah satu layanan pada aplikasi monolith tersebut bermasalah atau *crash* maka akan mempengaruhi seluruh aplikasi. Namun, pembuatan aplikasi monolith cenderung lebih mudah pada aplikasi dengan skala yang kecil.



Gambar 10. Perbandingan skema layanan pada microservice dan monolith

4. Kesimpulan

Berdasarkan hasil pengujian dan analisis pada arsitektur microservice dan monolith, maka penulis dapat menyimpulkan bahwa:

1) Pemakaian resource yaitu salah satunya CPU pada aplikasi dengan arsitektur microservice cenderung lebih terdistribusi ke layanan yang dimintai request oleh client dibanding dengan arsitektur microservice karena masing masing layanan berjalan pada aplikasi yang berbeda.
2) Apabila salah satu service pada microservice tersebut mati/ crash, maka aplikasi lain tidak akan ikut mati karena berjalan pada aplikasi yang berbeda. Sedangkan pada arsitektur monolith, apabila terdapat satu layanan yang mati maka seluruh aplikasi akan terdampak karena berjalan pada aplikasi yang sama.
3) Microservice akan lebih berguna pada aplikasi dengan skala yang besar karena dapat dengan mudah melakukan pengembangan dan fokus pada layanan tertentu dibandingkan dengan aplikasi monolith yang memungkinkan adanya ketergantungan tiap layanan.
4) Microservice lebih cocok digunakan dalam pemrosesan Big Data dibandingkan dengan Monolith dilihat dari pengujian yang dimana penggunaan sumber daya pada microservice lebih terdistribusi dibanding arsitektur monolith.

Diharapkan pembaca setelah ini mengetahui bagaimana performa dari arsitektur microservice dan arsitektur monolith. Sehingga pembaca dapat memilih penggunaan arsitektur perangkat lunak sesuai kebutuhan.

Referensi

[1] D. Price, “INFOGRAPHIC: HOW MUCH DATA IS PRODUCED EVERY DAY?”, 2015. [Online]. Available: https://cloudtweaks.com/2015/03/how-much-data-is-produced-every-day/#:~:text=A.,a%20staggering%2018%20zeros. [01 October 2022]

[2] C. Richardson, “What are microservices?”, 2021. [Online]. Available: https://microservices.io/. [01 October 2022]

[3] C. Harris, “Microservices vs. monolithic architecture.”. [Online]. Available: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith#:~:text=A%20monolithic%20application%20is%20built,of%20smaller%2C%20independently%20deployable%20services. [02 October 2022].

[4] F. Tapia, M. ángel Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, “From monolithic systems to microservices: A comparative study of performance,” *Applied Sciences (Switzerland)*, vol. 10, no. 17, Sep. 2020, doi: 10.3390/app10175797.

[5] S. Diguer, “Microservices Advantages and Disadvantages: Everything You Need to Know”, June 2020. [Online]. Available: https://solace.com/blog/microservices-advantages-and-disadvantages/. [02 October 2022]