

SKRIPSI

**ANALISIS KINERJA GRPC DAN REST API PADA
PERTUKARAN DATA ANTAR *MICROSERVICES***

Disusun dan diajukan oleh:

SALAHUDDIN

D121181327



DEPARTEMEN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS HASANUDDIN

2022

HALAMAN JUDUL

**ANALISIS KINERJA GRPC DAN REST API PADA
PERTUKARAN DATA ANTAR *MICROSERVICES***

Disusun dan diajukan oleh:

SALAHUDDIN

D121181327



DEPARTEMEN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS HASANUDDIN

2022

LEMBAR PENGESAHAN SKRIPSI
ANALISIS KINERJA GRPC DAN REST API PADA PERTUKARAN
DATA ANTAR MICROSERVICES

Disusun dan diajukan oleh

SALAHUDDIN


D121181327


Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Program Sarjana Program Studi Teknik Informatika Fakultas Teknik Universitas Hasanuddin pada tanggal 14 Juli 2022 dan dinyatakan telah memenuhi syarat kelulusan.

Menyetujui,

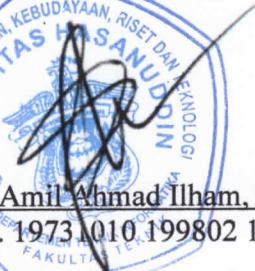
Pembimbing Utama,

Pembimbing Pendamping,


Dr. Eng. Ir. Muhammad Niswar, S.T.,
M.InfoTech.
Nip. 197309221999031001


Iqra' Aswad, S.T., M.T
Nip. 199011282019043001

Ketua Program Studi,


Dr. Amil Ahmad Ilham, S.T., M.IT.
Nip. 197310101998021002

PERNYATAAN KEASLIAN

Yang bertanda tangan dibawah ini :

Nama : Salahuddin
Nim : D1211181327
Program Studi : Teknik Informatika
Jenjang : S1

Menyatakan dengan ini karya tulisan saya berjudul:

ANALISIS KINERJA GRPC DAN REST API PADA PERTUKARAN DATA ANTAR MICROSERVICES

Adalah karya tulisan saya sendiri dan bukan merupakan pengambilan alihan tulisan orang lain bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri.

Apabila di kemudian hari terbukti atau dapat dibuktikan bahwa sebagian atau keseluruhan skripsi ini hasil karya orang lain, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Gowa, 04 Juli 2022

Yang Menyatakan,



Salahuddin

KATA PENGANTAR

Segala puji dan syukur penulis panjatkan kepada Tuhan yang Maha Esa karena hanya atas limpahan rahmat dan karunia-Nya sehingga tugas akhir ini dapat diselesaikan dengan baik. Tugas akhir ini berjudul ”**ANALISIS KINERJA GRPC DAN REST API PADA PERTUKARAN DATA ANTAR *MICROSERVICES***”. Penyusunan tugas akhir ini adalah sebagai salah satu syarat dalam menyelesaikan jenjang Strata-1 di Departemen Teknik Informatika Universitas Hasanuddin.

Penulis menyadari sepenuhnya bahwa tanpa bantuan, bimbingan, serta dukungan dari berbagai pihak, sangatlah sulit untuk menyelesaikan Tugas Akhir ini. Baik di masa perkuliahan sampai dengan masa penyusunan Tugas Akhir ini. Oleh karena itu, penulis menyampaikan ucapan terima kasih kepada:

1. Kedua Orang tua penulis, Bapak Lahapi dan Ibu Sanaria yang atas kasih sayang, didikan dan semangat dari mereka yang telah menuntun perjalanan hidup penulis;
2. Bapak Dr. Eng. Muhammad Niswar, S.T., M.IT., selaku pembimbing utama sekaligus pembimbing akademik yang telah menyempatkan waktu, tenaga, dan pikiran untuk mengarahkan penulis selama masa perkuliahan hingga penyusunan tugas akhir;
3. Bapak Iqra Aswad S.T., M.T., selaku pembimbing pendamping yang telah menyempatkan waktu, tenaga, dan pikiran untuk mengarahkan penulis dalam penyusunan tugas akhir;

4. Bapak Dr. Amil Ahmad Ilham, ST., M.IT., selaku Ketua Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin atas bimbingannya selama masa perkuliahan penulis;
5. Bapak dan ibu Dosen di Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin atas didikan dan arahnya selama masa perkuliahan;
6. Segenap Staf Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah banyak membantu semasa perkuliahan dan dalam penyelesaian tugas akhir;
7. Semua orang yang telah membantu dan menginspirasi penulis namun tidak sempat disebutkan.

Akhir kata, penulis menyadari bahwa dalam tugas akhir ini mungkin masih terdapat kekurangan, oleh karenanya penulis mengharapkan saran serta masukan yang membangun dari semua pihak. Semoga tugas akhir ini dapat memberi manfaat bagi penulis sendiri maupun kepentingan bersama.

Gowa, 04 Juli 2021

Penulis

ABSTRAK

Perkembangan aplikasi modern dengan segala kompleksitasnya menjadikan *microservices* sebagai salah satu arsitektur yang banyak digunakan. *Microservices* mendukung pengembangan dan pemeliharaan sistem yang lebih mudah karena membagi setiap layanan ke mesin yang berbeda. Layanan yang terdistribusi tersebut juga memberi tantangan dalam mengimplementasikan arsitektur ini. Yaitu bagaimana pertukaran data antar layanan tersebut. REST dan gRPC adalah dua metode pertukaran data yang dapat digunakan pada *microservices*. Untuk itu, pada penelitian ini dilakukan analisis kinerja *microservices* yang menggunakan dua metode pertukaran data tersebut. Pada penelitian ini *microservices* yang dikembangkan adalah sistem manajemen akademik yang memiliki tiga *service*. *Microservices* dibangun dengan menggunakan bahasa Go dan *runtime* Node.js serta basis data Redis. Setiap *service* menggunakan virtual mesin berbeda yang merupakan *instance* dari AWS EC2. Parameter kinerja yang diukur adalah *response time*, *throughput* dan CPU *utilization*. Hasil pengukuran menunjukkan bahwa gRPC memiliki kinerja lebih baik dibandingkan REST pada semua parameter pengujian. Hal tersebut karena gRPC menggunakan *protobuf* serta mampu menggunakan koneksi yang sama untuk beberapa *request* sementara REST menggunakan JSON sebagai format pengiriman datanya dan harus membangkitkan koneksi baru pada setiap request.

Kata kunci: Analisis kinerja, *Microservices*, REST, gRPC, Go, Node.js

DAFTAR ISI

HALAMAN JUDUL	i
LEMBAR PENGESAHAN SKRIPSI	ii
PERNYATAAN KEASLIAN	iii
KATA PENGANTAR	iii
ABSTRAK	iv
DAFTAR ISI	vii
DAFTAR GAMBAR	ix
DAFTAR TABEL	xi
DAFTAR LAMPIRAN	xii
BAB 1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	3
1.3 Tujuan Penelitian	3
1.4 Manfaat Penelitian	4
1.5 Batasan Masalah	4
1.6 Sistematika Penulisan	5
BAB II TINJAUAN PUSTAKA	6
2.1 Microservices	6
2.2 REST	7
2.3 GRPC	9
2.4 Go	12
2.5 JavaScript	13
2.6 NodeJs	14
2.7 Redis	15
2.8 AWS EC2	16
2.9 Apache Jmeter	17
BAB III METODE PENELITIAN	19
3.1 Tahapan Penelitian	19
3.2 Waktu dan Lokasi Penelitian	21

3.3	Instrumen Penelitian	21
3.4	Perancangan Sistem	21
3.4.1	Service KRS	25
3.4.2	Service Auth	26
3.4.3	Service Payment	26
3.5	Struktur Database	27
3.6	Implementasi Sistem	28
3.6.1	Menggunakan gRPC	28
3.6.2	Menggunakan REST	33
3.6.3	Men-deploy sistem	36
3.7	Pengukuran Kinerja Sistem	37
3.7.1	Response Time	40
3.7.2	Throughput	40
3.7.3	CPU Utilization	41
BAB IV HASIL DAN PEMBAHASAN		44
4.1	Hasil Pengujian	44
4.1.1	Response Time	45
4.1.2	Throughput	49
4.1.3	CPU Utilization	52
4.1.4	Perbandingan Node.Js dan Go	56
4.1.5	Perbandingan Arsitektur Sistem	63
4.2	Pembahasan	65
BAB V PENUTUP		68
5.1	Kesimpulan	68
5.2	Saran	69
DAFTAR PUSTAKA		70
LAMPIRAN		73

DAFTAR GAMBAR

Gambar 2.1 Perbedaan Microservies dan monolitik	6
Gambar 2.2 Empat metode pengiriman data gRPC.....	11
Gambar 3.1 Tahapan Penelitian.....	19
Gambar 3.2 Desain arsitektur sistem	23
Gambar 3.3 Sequence diagram sistem	24
Gambar 3.4 Potongan kode file proto service KRS	29
Gambar 3.5 Potongan kode file proto service Auth	29
Gambar 3.6 Potongan kode file proto service Payment	30
Gambar 3.7 Kontrak server KRS dengan interface	31
Gambar 3.8 Potongan kode fungsi VerifyToken pada gRPC	32
Gambar 3.9 Potongan kode fungsi VerifyPayment pada gRPC	32
Gambar 3.10 Kontrak server KRS dengan interface	35
Gambar 3.11 Instance AWS EC2	36
Gambar 3.5 Flowchart program pengukur CPU Utilization	47
Gambar 4.1 Tampilan hasil pengukuran di peramban	44
Gambar 4.2 Hasil pengukuran CPU utilization	45
Gambar 4.3 Bagan <i>response time</i> pada fungsi <i>Create service</i> KRS	46
Gambar 4.4 Bagan <i>response time</i> pada fungsi <i>Read service</i> KRS	46
Gambar 4.5 Bagan <i>response time</i> pada fungsi <i>Update service</i> KRS	46
Gambar 4.6 Bagan <i>response time</i> pada fungsi <i>Delete service</i> KRS	47
Gambar 4.7 Bagan <i>response time</i> pada fungsi <i>Login service</i> Auth	47
Gambar 4.8 Bagan <i>response time</i> pada fungsi <i>Verify service</i> Auth	47
Gambar 4.9 Bagan <i>response time</i> pada fungsi <i>Pay service</i> Payment	48
Gambar 4.10 Bagan <i>response time</i> pada fungsi <i>Verify service</i> Payment	48
Gambar 4.11 Bagan <i>throughput</i> pada fungsi <i>Create service</i> KRS	49
Gambar 4.12 Bagan <i>throughput</i> pada fungsi <i>Read service</i> KRS	50
Gambar 4.13 Bagan <i>throughput</i> pada fungsi <i>Update service</i> KRS	50
Gambar 4.14 Bagan <i>throughput</i> pada fungsi <i>Delete service</i> KRS	50
Gambar 4.15 Bagan <i>throughput</i> pada fungsi <i>Login service</i> Auth	51

Gambar 4.16 Bagan <i>throughput</i> pada fungsi <i>Verify service Auth</i>	51
Gambar 4.17 Bagan <i>throughput</i> pada fungsi <i>Pay service Payment</i>	51
Gambar 4.18 Bagan <i>throughput</i> pada fungsi <i>Verify service Payment</i>	52
Gambar 4.19 Bagan <i>CPU utilization</i> pada fungsi <i>Create service KRS</i>	53
Gambar 4.20 Bagan <i>CPU utilization</i> pada fungsi <i>Read service KRS</i>	53
Gambar 4.21 Bagan <i>CPU utilization</i> pada fungsi <i>Update service KRS</i>	54
Gambar 4.22 Bagan <i>CPU utilization</i> pada fungsi <i>Delete service KRS</i>	54
Gambar 4.23 Bagan <i>CPU utilization</i> pada fungsi <i>Login service Auth</i>	54
Gambar 4.24 Bagan <i>CPU utilization</i> pada fungsi <i>Verify service Auth</i>	55
Gambar 4.25 Bagan <i>CPU utilization</i> pada fungsi <i>Pay service Payment</i>	55
Gambar 4.26 Bagan <i>CPU utilization</i> pada fungsi <i>Verify service Payment</i>	55
Gambar 4.27 Bagan <i>response time</i> fungsi Login menggunakan REST ...;.....	57
Gambar 4.28 Bagan <i>response time</i> fungsi Login menggunakan gRPC	57
Gambar 4.29 Bagan <i>response time</i> fungsi Verify menggunakan REST	58
Gambar 4.30 Bagan <i>response time</i> fungsi Verify menggunakan gRPC	58
Gambar 4.31 Bagan <i>throughput</i> fungsi Login menggunakan REST	59
Gambar 4.32 Bagan <i>throughput</i> fungsi Login menggunakan gRPC	59
Gambar 4.33 Bagan <i>throughput</i> fungsi Verify menggunakan REST	60
Gambar 4.34 Bagan <i>throughput</i> fungsi Verify menggunakan gRPC	60
Gambar 4.35 Bagan <i>CPU utilization</i> fungsi Login menggunakan REST	61
Gambar 4.36 Bagan <i>CPU utilization</i> fungsi Login menggunakan gRPC	61
Gambar 4.37 Bagan <i>CPU utilization</i> fungsi Verify menggunakan REST	62
Gambar 4.38 Bagan <i>CPU utilization</i> fungsi Verify menggunakan gRPC	62
Gambar 4.39 Bagan <i>response time</i> desain satu dan desain kedua	64
Gambar 4.40 Bagan <i>throughput</i> desain satu dan desain kedua	64
Gambar 4.41 Bagan <i>CPU utilization</i> desain satu dan desain kedua	64
Gambar 4.42 Koneksi TCP pada HTTP/2.0 (a) dan HTTP/1.1 (b)	66
Gambar 4.43 Profiling process Node.j	67

DAFTAR TABEL

Tabel 3.1 Fungsi <i>service</i> KRS	25
Tabel 3.2 Fungsi <i>service</i> Auth	26
Tabel 3.3 Fungsi <i>service</i> Payment	27
Tabel 3.4 Model database key-value redis	27
Tabel 3.5 Fungsi yang diuji	38

DAFTAR LAMPIRAN

Lampiran 1 Hasil pengukuran response time	73
Lampiran 2 Hasil pengukuran throughput	78
Lampiran 3 Hasil pengukuran CPU utilization	84
Lampiran 4 Hasil pengukuran response time antara Node.js dan Go	89
Lampiran 5 Hasil pengukuran throughput antara Node.js dan Go,,,.....	92
Lampiran 6 Hasil pengukuran CPU utilization antara Node.js dan Go	95
Lampiran 7 Source code pendefinisian file proto	98
Lampiran 8 Source code service KRS.....	100
Lampiran 9 Source code service AUTH menggunakan Node.js	118
Lampiran 10 Source code service AUTH menggunakan Go	124
Lampiran 11 Source code service Payment	135
Lampiran 12 File jmx sebagai definisi pengujian	146
Lampiran 13 Program pengukur CPU Utilization	150

BAB I

PENDAHULUAN

1.1 Latar Belakang

Kompleksitas aplikasi modern yang terus meningkat menjadikan arsitektur monolitik menjadi tidak praktis. Hal ini karena arsitektur monolitik menjalankan semua logika aplikasi dalam satu *server* (Shafabakhsh, 2020). Sehingga untuk menskalakan aplikasi tersebut harus dilakukan pada semua komponen. Pada sistem yang berbasis monolitik setiap komunikasi antar komponen terjadi pada tingkat “bahasa” dalam satu proses di sebuah mesin (Shafabakhsh, Lagerström dan Hacks, 2020). Hal ini berimplikasi pada sulitnya untuk dapat menggunakan teknologi sesuai kebutuhan komponen tersebut. Pemeliharaan pada sistem monolitik juga sangat sulit dilakukan karena semua anggota tim harus terus saling berkoordinasi untuk dapat mengubah *codebase* sistem tersebut (Barakat, 2017).

Permasalahan sebagaimana yang dijelaskan diatas mendorong hadirnya solusi alternatif yaitu arsitektur *microservices*. Arsitektur *microservices* membagi logika aplikasi dalam beberapa layanan independen yang berjalan diatas proses yang berbeda (Akbulut & Perros, 2019). Arsitektur *microservices* banyak diimplementasikan pada saat ini karena ketahanannya serta mampu menyediakan perangkat lunak dengan cepat karena layanan independen yang lebih kecil (Chamas, Cordeiro dan Eler, 2017). Selain itu, *microservices* dapat mendukung pengembangan yang *agile*. yaitu arsitektur *microservices* memungkinkan bisnis berkembang dan menawarkan fitur baru dengan iterasi kecil dan waktu yang

singkat dengan tetap memastikan skalabilitas, keandalan, dan keamanan sistem yang tinggi (Stefanic, 2021).

Pada saat arsitektur *microservices* semakin populer, standar bagaimana layanan tersebut berkomunikasi juga berkembang. Pendekatan yang umum adalah dengan menggunakan REST atau *Representational State Transfer* melalui JSON sebagai IDL atau *Interface Definition Language*-nya (Gonzalez, 2019). REST menggunakan protokol HTTP/1.1 dengan JSON yang menggunakan format serialisasi data berbasis teks (Stefanic, 2021). Setiap permintaan dari klien, HTTP/1.1 akan membuka koneksi TCP baru ke server dan hanya dapat bekerja dengan mekanisme *request-response* (Grigorik & Surma). Keterbatasan inilah yang berdampak pada terbatasnya performa *microservices* yang menggunakan pendekatan REST sebagai metode komunikasi antar layanannya.

Selain REST, gRPC adalah alternatif lain yang dapat digunakan sebagai metode komunikasi dalam sistem *microservices*. GRPC adalah kerangka kerja RPC (*Remote Procedure call*) yang bersifat *open source* yang dikembangkan oleh insinyur perangkat lunak di Google dan resmi dirilis pada Agustus 2016 (Rebrošová, 2021). GRPC mendukung berbagai jenis data streaming dengan performa tinggi dan pengiriman data yang cepat karena didukung oleh penggunaan protokol buffer (protobuf) dan HTTP/2 untuk transfer data (Stefanic, 2021). Protokol buffer adalah sebuah teknologi *open source* untuk melakukan serialisasi data. Berbeda dengan JSON, protokol buffer memiliki performa lebih baik karena melakukan serialisasi data menjadi format binary (Berge, Hagaseth &

Bø, 2019). Protokol buffer juga mempertegas kepada pengembang bagaimana komunikasi antar gRPC akan dibangun (Araújo, Maia, Rego, & Souza, 2020).

Berdasarkan hal tersebut, penulis mengajukan penelitian untuk melakukan analisis perbandingan kinerja antara gRPC dengan REST pada pertukaran data dalam sebuah sistem *microservices*. Penelitian tersebut berjudul “Analisis Kinerja gRPC dan REST API pada Pertukaran Data Antar *Microservices*”. Dalam penelitian ini, penulis akan mengemangkan sistem *microservices* yang menggunakan gRPC dan REST untuk melakukan pertukaran data. Selanjutnya dilakukan pengukuran dan analisis kinerja pada kedua sistem tersebut. Adapun parameter kinerja yang diukur adalah *respon time*, *throughput* dan CPU *utilization*.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka rumusan masalah pada tugas akhir ini adalah:

- a. Bagaimana merancang dan mengimplementasikan gRPC dan REST pada sebuah sistem *microservices*.
- b. Bagaimana mengukur dan menganalisis kinerja *microservices* yang dibangun menggunakan gRPC dan REST.

1.3 Tujuan Penelitian

Tujuan yang ingin dicapai dari penelitian ini sebagai berikut :

- a. Mengetahui kinerja *microservices* yang dibangun menggunakan gRPC dan REST.

- b. mengembangkan sistem *microservices* yang dapat dijadikan referensi untuk melakukan migrasi REST ke gRPC, sebaliknya atau menggabungkan keduanya.

1.4 Manfaat Penelitian

Dengan dilakukannya penelitian ini, diharapkan manfaat yang didapatkan antara lain:

- a. Memberikan sumbangan ilmiah berupa data hasil analisis kinerja gRPC dan REST pada sebuah *microservices*.
- b. Hasil analisis dapat dijadikan sebagai pertimbangan dalam pengembangan sistem yang menggunakan arsitektur *microservices*.
- c. Kode sumber penelitian dapat dijadikan sebagai acuan dalam migrasi *microservices* baik dari REST ke gRPC, sebaliknya maupun menggabungkan keduanya.

1.5 Batasan Masalah

Adapun batasan masalah dalam penelitian ini adalah :

- a. Kinerja yang akan diukur adalah *response time* dan *throughput* dan CPU *utilization*.
- b. *Microservices* yang dikembangkan menggunakan tiga *service*.
- c. *Service* dibangun dengan menggunakan bahasa Go atau NodeJS
- d. Masing-masing *service* menggunakan satu mesin virtual yang saling terhubung dalam satu jaringan lokal.

- e. Pengukuran *request time* dan *throughput* menggunakan Apache Jmeter sedangkan pengukuran CPU *utilization* menggunakan program yang dikembangkan secara mandiri.

1.6 Sistematika Penulisan

Adapun sistematika penulisan penelitian adalah sebagai berikut :

BAB 1 PENDAHULUAN

Bab ini menguraikan latar belakang penelitian, rumusan masalah, tujuan dan manfaat penelitian, batasan masalah, serta sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Bab ini membahas tentang landasan teori yang menunjang setiap proses pada penelitian yang dilakukan.

BAB III METODOLOGI PENELITIAN

Bab ini menjelaskan tentang tahap penelitian, alat dan bahan penelitian, gambaran sistem, implementasi sistem hingga skenario pengujian kinerja sistem yang dikembangkan.

BAB IV HASIL DAN PEMBAHASAN

Bab ini membahas secara menyeluruh hasil pengukuran dan analisis kinerja *microservice* yang dikembangkan menggunakan REST dan gRPC.

BAB V PENUTUP

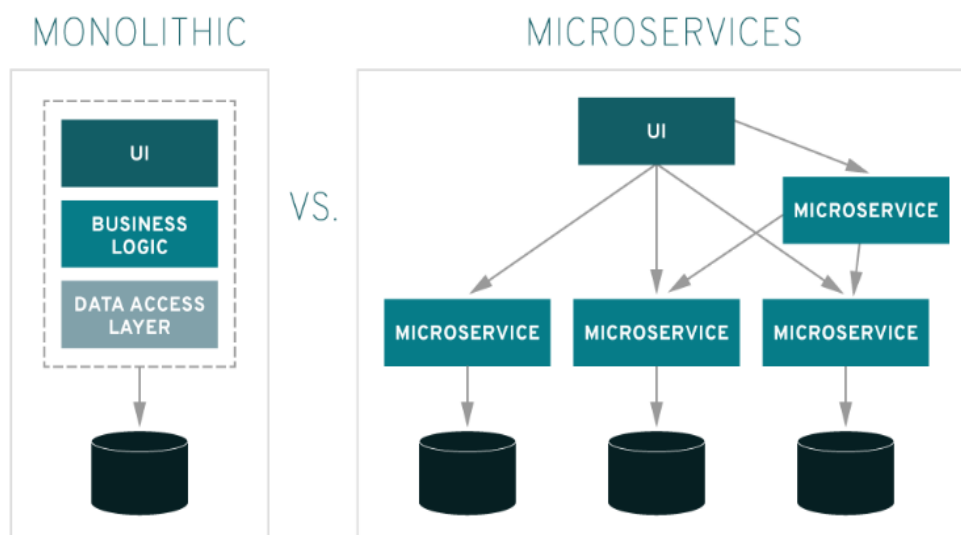
Bab ini berisi kesimpulan dari hasil penelitian dan saran-saran untuk pengembangan lebih lanjut.

BAB II

TINJAUAN PUSTAKA

2.1 *Microservices*

Microservices merupakan salah satu pendekatan untuk mengembangkan sebuah aplikasi. Gaya arsitektur yang digunakan pada *microservices* adalah dengan mengembangkan aplikasi perangkat lunak yang kompleks dari kumpulan layanan-layanan kecil. Setiap layanan kecil tersebut bertanggung jawab untuk menyelesaikan tugas tertentu. Layanan tersebut dapat dikembangkan dengan berbagai bahasa pemrograman oleh tim terpisah pada periode yang berbeda (Aksakalli, Turgay, Can & Bedir, 2021). Sehingga, arsitektur *microservices* memiliki kelebihan dibandingkan arsitektur monolitik karena keseluruhan sistem yang dipecah menjadi bagian-bagian kecil yang berjalan independen (shafabakhsh, 2020).



Gambar 2.1 Perbedaan *microservices* dan monolitik (shafabakhsh, 2020)

Pengembangan aplikasi menggunakan arsitektur *microservices* tidak memiliki patokan yang baku. Namun, menurut Benjamin Shafaback ada beberapa karakteristik aplikasi yang harus dimiliki agar dapat dikatakan menggunakan arsitektur *microservices*. Karakter tersebut antara lain tetapi tidak terbatas pada ukuran setiap *service* yang relatif kecil, terikat dalam konteks, terdistribusi dan terakhir dapat digunakan secara mandiri. Sehingga Shafaback beranggapan bahwa arsitektur *microservices* dapat memberi manfaat seperti kemudahan menggunakan berbagai teknologi seperti bahasa pemrograman atau *framework*, kemudahan untuk melakukan penskalaan, ketersediaan yang tinggi dan meningkatkan keselarasan dalam organisasi (shafabakhsh, 2020).

Bersama kelebihan yang diberikan *microservices*, arsitektur ini juga memiliki beberapa tantangan pada pengaplikasiannya. Sebagaimana yang dijelaskan Richardson (2019) dalam buku "*Microservices patterns: with examples in Java*" seperti bagaimana menentukan susunan layanan yang tepat, bagaimana mengembangkan, menguji, menyebarkan hingga bagaimana membangun komunikasi antar *service*. Pada arsitektur *microservices* sendiri terdapat dua metode komunikasi *synchronous* yang paling umum digunakan yaitu menggunakan REST atau menggunakan gRPC.

2.2 REST

REST (*Representational state transfer*) adalah sebuah arsitektur pengembangan API (*Application Programming Interface*) yang menyediakan komunikasi berbasis *client-server* melalui protokol HTTP (Prayogi, Niswar, Indrabayu & Rija, 2019). REST pertama kali diperkenalkan oleh Roy Fielding

pada tahun 2000 sebagai disertasi doktoralnya di University of California. Disertasi tersebut berjudul “*Architectural Styles and the Design of Network-based Software Architecture*”. Dalam disertasi tersebut Fielding memperkenalkan REST yaitu sebuah gaya arsitektur untuk sistem hypermedia yang terdistribusi (Doglio, 2018).

REST menggunakan protokol HTTP/1.1 untuk mengirimkan data dari klien ke peladen (Masse, 2012). Dalam sistem yang menggunakan REST, setiap *service* biasanya memiliki *endpoint* tertentu agar bisa melakukan interaksi antar *service* dan bertukar data (Shafabakhsh, 2020). Pada REST terdapat beberapa method yang dapat digunakan. Namun yang populer digunakan adalah :

- POST untuk menambahkan data baru
- GET untuk menerima data yang tersedia
- PUT untuk memperbaharui data yang telah ada
- DELETE untuk menghapus data

Sedangkan beberapa method lain yang agak jarang digunakan pada REST seperti HEAD, OPTION dan PATCH. REST mendukung beberapa format untuk mempresentasikan data seperti JSON dan XML. Namun, JSON lebih sering digunakan karena lebih mudah dipahami. Hal ini karena JSON adalah format data yang berbasis teks (Stefanic, 2021).

Sebagai sebuah gaya arsitektur yang terdistribusi, menurut Doglio (2018) dalam buku “Rest api development with node.js” REST mampu meningkatkan beberapa hal berikut:

- *Performance*: Gaya komunikasi yang digunakan pada REST ditujukan untuk efisiensi dan sederhana. Sehingga dapat mengoptimalkan kinerja dari sistem yang menggunakannya.
- *Scalability of component interaction*: Gaya komunikasi REST adalah bersifat terdistribusi dan saling terpisah. Sehingga memudahkan untuk melakukan *scalability* dari interaksi antar komponen.
- *Simplicity of interface*: Sebuah antarmuka yang sederhana akan memudahkan sistem menggunakan arsitektur ini.
- *Modifiability of components*: Sistem terdistribusi yang memecah konsentrasi memungkinkan setiap komponen dapat dikembangkan dengan resiko dan biaya yang kecil.
- *Portability*: REST adalah teknologi yang bersifat *language-agnostic*, sehingga teknologi ini dapat digunakan pada bahasa apa saja.
- *Reliability*: REST yang berbasis *stateless* memungkinkan proses pemulihan dari sebuah sistem yang gagal menjadi lebih mudah.
- *Visibility*: Dengan sistem *stateless*, REST juga memudahkan untuk melakukan proses *monitoring* karena sistem tidak perlu melihat lebih jauh untuk mengetahui keadaan dari sebuah request.

2.3 GRPC

GRPC adalah salah satu jenis RPC yang sering digunakan dan terus dikembangkan. RPC (*Remote Procedure Call*) adalah suatu bentuk komunikasi *synchronous* tingkat bahasa untuk melakukan transfer pengendalian antar program dengan cara memanfaatkan kanal komunikasi terbatas pada sistem terdistribusi

(Nelson, 1981). Jadi gRPC adalah teknologi komunikasi antar proses untuk menggabungkan atau memanggil sistem yang terdistribusi semudah memanggil fungsi lokal (Indrasiri & Kuruppu, 2020). GRPC bersifat *open source* yang pada awalnya dikembangkan oleh insinyur perangkat lunak di Google dan resmi dirilis pada Agustus 2016 (Rebrošová, 2021).

Pada gRPC *transport protocol* yang digunakan adalah HTTP/2.0. Sedangkan format data yang dikirimkan menggunakan protokol buffer (protobuf) (Stefanic, 2021). Protokol buffer adalah sebuah teknologi *open source* untuk melakukan serialisasi data. Berbeda dengan JSON, protokol buffer memiliki performa lebih baik karena melakukan serialisasi data menjadi format *binary* (Berge dkk, 2019). Protokol buffer juga mempertegas kepada pengembang bagaimana komunikasi antar gRPC akan dibangun (Araújo dkk, 2020). Kasun Indrasiri dan Danesh Kuruppu (2020) menjelaskan bahwa terdapat empat metode pengiriman data pada gRPC. Metode tersebut adalah sebagai berikut :

a. Unary RPC

Unary adalah metode dimana klien yang mengirimkan sebuah permintaan tunggal ke peladen kemudian juga mendapatkan respon yang tunggal. Penggunaan Unary seperti pada pemanggilan fungsi pada umumnya.

b. Client Streaming RPC

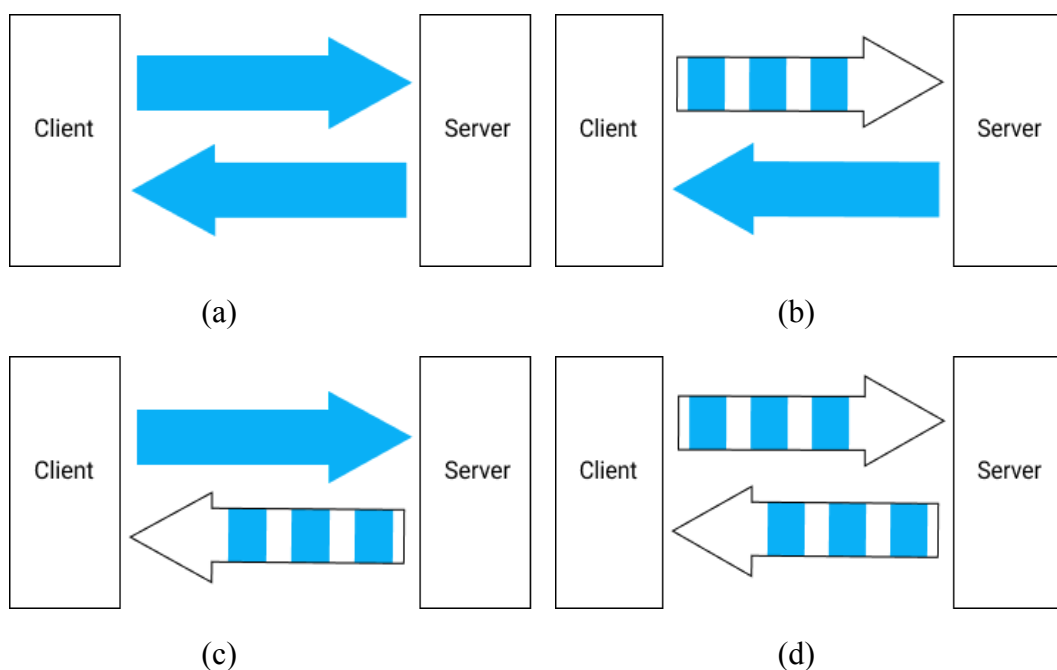
Client Streaming adalah metode dimana klien mengirimkan rangkaian permintaan ke peladen. Selanjutnya peladen akan menunggu semua permintaan sebelum meresponya secara tunggal.

c. Server Streaming RPC

Pada metode Server Streaming, klien hanya mengirimkan sebuah permintaan tunggal ke peladen. Namun, peladen akan merespon dengan rangkaian balasan yang berurutan ke klien. Klien kemudian membaca respon tersebut secara berurutan sampai tidak ada lagi respon yang masuk.

d. Bidirectional Streaming RPC

Metode Bidirectional Streaming memungkinkan klien dan peladen sama-sama mengirimkan pesan. Namun, keduanya tetap dapat beroperasi secara terpisah dan independen sehingga keduanya dapat menerima dan mengirimkan pesan sesuai dengan kebutuhan mereka.



Gambar 2.2 Empat metode pengiriman data gRPC, (a) Unary RPC, (b) Client Streaming RPC, (c) Server Streaming RPC, (d) Bidirectional Streaming RPC

Langkah pertama untuk mengembangkan aplikasi gRPC adalah mendefinisikan *service interface* yang berisi informasi bagaimana service tersebut

dapat digunakan seperti fungsi yang dapat digunakan secara remote. Adapun *service interface* yang digunakan pada gRPC adalah Protocol buffers. Selanjutnya *service interface* tersebut digunakan untuk menghasilkan *server skeleton* dan *client stub*. Baik *server* maupun *client* pada gRPC dapat menggunakan bahasa pemrograman yang berbeda diantara bahasa pemrograman yang populer seperti Go, Java, C, C++, Node, Python dll (Indrasiri & Kuruppu, 2020).

2.4 Go

Go adalah sebuah bahasa pemrograman yang digagas oleh Robert Griesemer, Rob Pike, dan Ken Thompson di Google pada tahun 2007. Baru setelah dua tahun, Go kemudian diperkenalkan yaitu pada tahun 2009. Go memiliki banyak kemiripan dengan bahasa C seperti aturan penulisan, tipe data dasar, pointer dan kemampuan untuk di-*compile* menjadi bahasa mesin (Donovan & Kernighan, 2016). Saat ini Go atau yang sering disebut dengan Golang (Go Language) terus dikembangkan oleh Google bersama dengan kontributor dari komunitas *open source*.

Go mulai banyak digunakan pada berbagai macam perusahaan baik untuk perusahaan berskala besar maupun startup yang bergerak di bidang teknologi. Menurut Meyerson (2014) beberapa karakteristik bahasa Go secara garis besar dapat dilihat berdasarkan fundamental, Run-time, dan model package-nya:

a. Fundamental

Go memiliki aturan penulisan yang ringkas dan bersih. Selain itu, bahasa ini juga desain yang ringan sehingga tidak meningkatkan beban kinerja komputer. Beberapa karakteristik Go seperti interface dapat dipisah dari

implementasinya, tidak ada konversi implisit dan aturan lainnya sehingga dapat mempersingkat serta mempermudah penulisan program.

b. Run-Time

Go memiliki *garbage collection* yang akan menghapus penggunaan memori yang tidak digunakan pada saat program tersebut dijalankan. Selain itu Go juga mendukung konkurensi untuk meningkatkan kinerja program.

c. Package Model

Go menggunakan konsep dependensi eksplisit untuk mengelolah dependensi yang digunakan sebuah program. Sehingga waktu yang dibutuhkan ketika melakukan *build* pada program menjadi lebih singkat.

2.5 JavaScript

JavaScript diperkenalkan pada tahun 1995 sebagai sebuah metode untuk menjalankan program pada sebuah halaman web di peramban Netscape Navigator. Setelah itu, bahasa ini kemudian banyak diadopsi oleh peramban lainnya. Setelah JavaScript banyak diadopsi oleh peramban lainnya, dibuatlah standarisasi JavaScript pada tahun 1997. Standar tersebut disebut ECMAScript karena distandarisasi oleh organisasi Ecma International. Standarisasi ini untuk memastikan bahwa berbagai versi JavaScript tetap sesuai (McPeak, 2015).

JavaScript tidak berhubungan dengan bahasa Java walaupun namanya cukup mirip. Kesamaan nama ini hanya sebagai strategi pemasaran. Hal ini karena pada saat JavaScript diperkenalkan. bahasa Java sedang mendapatkan popularitas yang tinggi. Meskipun awalnya JavaScript dikembangkan untuk peramban, tetapi sekarang JavaScript telah banyak digunakan di luar aplikasi peramban. Misalnya

pada bahasa *query* basis data MongoDB dan CouchDB atau aplikasi desktop dan peladen web dengan menggunakan Node.js (Haverbeke, 2018).

JavaScript merupakan bahasa *interpreter* bukan sebuah bahasa *compile*. Sehingga komputer tidak bisa langsung mengeksekusi perintah dari program JavaScript. Untuk itu, JavaScript membutuhkan sebuah penerjemah untuk mengubah kode JavaScript menjadi sesuatu yang dimengerti oleh komputer yaitu *machine code*. Proses penerjemah ini langsung dilakukan pada saat kode JavaScript tersebut dijalankan. Proses tersebut terus dilakukan ketika akan menjalankan kode JavaScript. Selain JavaScript, beberapa bahasa pemrograman lain yang bersifat interpreted adalah Python, PHP dan Ruby (McPeak, 2015).

2.6 NodeJs

Node.js adalah *runtime environment* atau platform untuk mengeksekusi kode-kode yang ditulis menggunakan bahasa JavaScript (Raharjo, 2019). Sehingga dengan menggunakan Node.js, kode JavaScript dapat dijalankan di luar aplikasi peramban (Haverbeke, 2018). Node.js menggunakan arsitektur *even-drivent* untuk menangani permintaan dari klien. Dengan arsitektur tersebut, semua permintaan dari klien akan dimasukkan ke antrian. Teknologi ini menggunakan Mesin JavaScript V8 dengan terintegrasi modul tambahan (Prayogi dkk, 2019). Salah satu module tambahan pada Node.js adalah http. Module http memberikan kemampuan Node.js untuk menjalankan program peladen HTTP maupun mengirimkan *request* ke sebuah peladen HTTP lain (Havebeke, 2018).

V8 adalah sebuah mesin JavaScript yang bertugas untuk mengubah kode JavaScript ke dalam bentuk *bytecode*. File *bytecode* tersebut yang kemudian akan

dieksekusi oleh Node.js. Sehingga, proses eksekusi dengan file *bytecode* bisa lebih cepat dibanding eksekusi pada kode JavaScript langsung. V8 adalah mesin JavaScript yang dikembangkan oleh Google menggunakan bahasa C++ untuk digunakan pada peramban web Chrome. Hal ini karena biasanya masing-masing aplikasi peramban web memiliki mesin untuk mengeksekusi kode JavaScript yang berbeda-beda. Misalnya pada Mozilla Firefox yang menggunakan Spidermonkey sedangkan Apple Safari menggunakan JavaScriptCore (Raharjo, 2019).

Node.js awalnya dibuat oleh Ryan Dahl pada tahun 2009 yang hanya dapat dijalankan pada sistem operasi Linux. Namun, saat ini Node.js sudah dapat berjalan di sistem operasi Windows, Mac OS, dan beberapa varian Unix Lainnya. Pada awal kemunculannya, Node.js lebih difokuskan untuk mengembangkan program-program yang berkaitan dengan permasalahan jaringan komputer. Namun, akhir-akhir ini Node.js telah banyak digunakan untuk mengembangkan aplikasi web. Hal ini karena Node.js memiliki performa yang baik untuk memproses permintaan dari klien dalam jumlah besar (Raharjo, 2019). Performa yang baik ini didukung dari hasil penelitian Prayogi yang menganalisa performa node.js dibandingkan PHP pada sebuah web *service*. Pada penelitian tersebut mereka mendapatkan bahwa kinerja Node.js unggul baik pada parameter *response time* maupun *throughput* dari pada PHP (Prayogi dkk, 2019).

2.7 Redis

Redis adalah database NoSQL berbasis *key-value* yang dikembangkan oleh Salvatore Sanfilippo. Salvatore memperkenalkan redis pada tahun 2009. *Key* pada Redis dapat digunakan untuk mencari sebuah *value* atau menghapusnya

dengan menghapus *key* tersebut. Nama Redis sendiri Berasal dari singkatan *Remote Dictionary Server*. Redis adalah *server single thread* yang berkinerja tinggi dan ditulis dengan bahasa C. Redis bekerja dengan menggunakan protokol Redis dengan model *client-server* (Das, 2015).

Redis memiliki Kinerja yang tinggi karena didukung oleh penyimpanan data yang berada di memori. Redis mendukung lima jenis “*value*” yaitu *strings*, *lists*, *sets*, *hashes* dan *sorted sets*. Selain itu Redis juga mendukung penyimpanan persisten pada disk, replikasi untuk menskalakan kinerja baca, dan *sharding* untuk menskalakan kinerja tulis. Sebagai database NoSQL atau *non-relasional*, di Redis tidak terdapat tabel untuk mendefinisikan maupun menentukan bagaimana sebuah data berhubungan (Carlson, 2013).

2.8 AWS EC2

AWS EC2 (Amazon Web Service Elastic Compute Cloud) adalah salah satu layanan *cloud* dari Amazon Web Service yang menyediakan kapasitas komputasi yang dapat diskalakan. Penggunaan AWS EC2 dapat menghilangkan kebutuhan investasi untuk biaya *hardware*. Sehingga dapat mempercepat proses pengembangan dan penyebaran sebuah aplikasi. AWS EC2 mempermudah meluncurkan *Virtual Machine* sebanyak dan sesuai kebutuhan. Selain itu AWS EC2 juga memudahkan pengaturan keamanan dan jaringan hingga pengelolaan penyimpanan. AWS EC2 memiliki beberapa tipe seperti *general purpose*, *compute optimized*, *memory optimize*, *acceleration optimize* dan *storage optimized* (Saini & Behl, 2020).

AWS EC2 sebagai lingkungan komputasi virtual yang mudah dijalankan juga menyediakan template pra konfigurasi yang dikenal sebagai Amazon Machine Images (AMI). Template tersebut sudah termasuk sistem operasi dan perangkat lunak tambahan. Volume penyimpanan untuk data sementara akan dihapus pada saat *instance* AWS EC2 dihentikan atau dihibernasi. Sedangkan Volume penyimpanan data yang *persistence* disimpan menggunakan Amazon Elastic Block Store (Amazon EBS). Firewall pada AWS EC2 digunakan untuk menentukan protokol, port, dan rentang IP yang dapat menjangkau *instance* tersebut. AWS EC2 menggunakan jaringan virtual yang dapat dibuat terisolasi secara logis dari jaringan AWS *cloud* lainnya (Amazon, tanpa tahun)

2.9 Apache Jmeter

Apache JMeter adalah program untuk melakukan pengujian kinerja *server*. Teknologi ini adalah sebuah proyek *open source* dan dapat berjalan diberbagai jenis platform. Sebagai proyek *open source*, teknologi ini telah terus diperbaiki sejak diperkenalkan pada tahun 2001. Apache JMeter dapat dijalankan dalam "mode terdistribusi" di layanan *cloud* dan menghasilkan ribuan pengguna virtual. Apache JMeter menggunakan lisensi Apache yang tidak memilikinya pembatasan dalam penggunaan, pendistribusian, ataupun modifikasinya. Apache JMeter memiliki format standar untuk menulis pengujian kinerja. Yaitu format JMX yang mana juga telah didukung oleh sebagian besar alat uji kinerja lainnya (Matam & Jain, 2017).

Apache JMeter memungkinkan kita menjalankan beberapa user secara simultan untuk mencapai target skenario test yang kita inginkan. Hal ini karena JMeter

mendukung *full multi trading* yang dapat dijalankan dengan *thread grup* yang terpisah. Aplikasi Apache JMeter dapat dijalankan secara *Graphical User Interface (GUI)* maupun *Command Line Interface (CLI)*. Apache JMeter mampu melakukan pengujian untuk beberapa tipe peladen seperti web, SAOP, Database, LDAP, JMS, Mail hingga Shell Script. Untuk menampilkan hasil pengujian, JMeter dapat mengubah hasil uji kinerja menjadi file HTML untuk ditampilkan di peramban (Erinle, 2017).