Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.1128287
Vol. 11, No. 2, April 2024, hlm. 421-428 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 36/E/KPT/2019 e-ISSN: 2528-6579

IMPLEMENTASI HPACK DAN GZIP PADA OPTIMASI APLIKASI DENGAN ARSITEKTUR *MICROSERVICE*

Rizal Tjut Adek[*1], Syibbran Mulaesyi[2], Zara Yunizar[3], M. Fauzan[4]

[1,2,3,4] Universitas Malikussaleh, Lhokseumawe
Email: [1]rizal@unimal.ac.id, [2]syibbran.190170067@mhs.unimal.ac.id, [3]zarayunizar@unimal.ac.id, [4]mfauzan@unimal.ac.id,
[*]Penulis Korespondensi

(Naskah masuk: 14 Desember 2023, diterima untuk diterbitkan: 24 April 2023)

Abstrak

Penelitian ini mengkaji penerapan arsitektur *microservice* dalam pengembangan aplikasi web, dengan fokus pada optimasi komunikasi antar layanan melalui kompresi data. Dengan latar belakang permasalahan *bottleneck* dalam transfer data antar layanan, tujuan utama penelitian ini adalah mengatasi tantangan tersebut dengan memanfaatkan teknologi kompresi HPack dan Gzip, terutama dalam konteks penggunaan format data JSON yang umum dalam komunikasi antar layanan. Penelitian dilakukan melalui tahapan analisis kebutuhan, desain sistem, implementasi, *deployment* dan pengujian. Data uji yang merupakan respons dari *microservice* yang berupa 34 item JSON *Array* yang dikompresi menggunakan Hpack, Gzip dan gabungan keduanya. Proses pengukuran waktu dan ukuran respons untuk setiap jenis kompresi dilakukan menggunakan *Chrome DevTools*, dengan analisis data yang dilakukan menggunakan alat bantu *pandas*. Hasil pengujian menunjukkan penurunan sebanyak ~79,4% dari 12991 *bytes* dalam ukuran respons dan penurunan waktu respons sebanyak ~8,7% dari 641,401 *ms*. Hal ini menegaskan efektivitas HPack dan Gzip dalam meningkatkan performa aplikasi *microservice*. Penelitian ini memberikan wawasan dalam optimasi aplikasi *microservice* dan rekomendasi praktik terbaik dalam pemilihan metode kompresi data.

Kata kunci: *arsitektur microservice, kompresi data, HPack, Gzip, optimasi komunikasi.*

Implementation of HPack and Gzip in Application Optimization on Microservice Architecture

Abstract

This study examines the implementation of microservice architecture in web application development, with a focus on optimizing communication between services through data compression. Given the background issue of bottleneck in data transfer between services, the primary objective of this research is to address this challenge by leveraging HPack and Gzip compression technologies, especially in the context of using the common JSON data format for inter-service communication. The research was conducted through stages of requirements analysis, system design, implementation, deployment, and testing. Test data consists of responses from microservices in the form of a 34-item JSON Array compressed using HPack, Gzip, and a combination of both. The process of measuring response time and size for each compression type was carried out using Chrome DevTools, with data analysis performed using pandas tools. The test results show a reduction of approximately 79.4% from 12991 bytes in response size and a decrease in response time by about 8.7% from 641.401 ms. This confirms the effectiveness of HPack and Gzip in enhancing microservice application performance. The study provides insights into microservice application optimization and recommendations for best practices in selecting data compression methods.

Keywords: *microservice architecture, data compression, HPack, Gzip, communication optimization*.

1. PENDAHULUAN

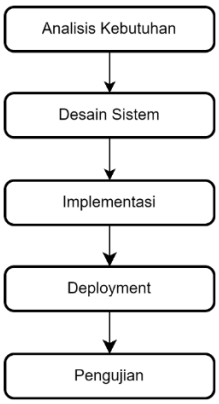
Microservice memecah aplikasi menjadi unit-unit mandiri dengan tugas dan fungsi spesifik, berkomunikasi melalui antarmuka yang jelas (Lewis & Fowler, 2014). Arsitektur *microservice* adalah aplikasi terdistribusi di mana semua modulnya adalah *microservice* (Dragoni, et al., 2017). Kekurangan arsitektur ini muncul ketika data yang ditransfer antar layanan membesar, dapat menyebabkan *bottleneck* dalam komunikasi.

Salah satu solusi adalah mengoptimalkan proses komunikasi antar layanan adalah dengan mengurangi ukuran data yang ditransfer antar layanan, terutama dalam format JSON yang umum digunakan dalam komunikasi *microservice* (Salah, et al., 2017). Metode kompresi yang digunakan pada penelitian ini adalah HPack dan Gzip. HPack, digunakan dalam protokol HTTP/2, menghilangkan *header redundant*, dan membutuhkan memori terbatas (Peon & Ruellan, 2015). Sementara itu, Gzip, sebuah metode kompresi data populer, mengurangi ukuran data dengan menghilangkan redundansi, menghasilkan pengurangan ukuran data hingga ~95% (Objelean, 2011).

Penelitian tentang pengaruh kompresi objek JSON pada sistem dengan arsitektur *microservice* belum banyak dilakukan. Demikian pula, penggunaan HPack belum tersebar luas dan perlu diteliti lebih lanjut. Kombinasi antara arsitektur *microservice* dan metode kompresi untuk tujuan optimasi juga menjadi topik menarik untuk penelitian lebih lanjut.

2. METODE PENELITIAN

Pengembangan dalam penelitian ini menggunakan metodologi *waterfall* yang dibagi menjadi lima bagian utama yaitu analisis kebutuhan, desain sistem, implementasi, *deployment* dan pengujian Pada Gambar 1. dijelaskan secara visual tahap-tahap dari penelitian ini.



Gambar 1. Bagan Tahapan Penelitian

2.1 Analisis kebutuhan

SocialSphere, sebuah *microservice* media sosial, dipilih sebagai objek uji. Metode optimasi melibatkan kompresi setiap *input* dan *output* dari setiap layanan, dengan implementasi algoritma HPack dan GZIP. Proses kompresi dan dekompresi input dan output akan diintegrasikan sebagai modul yang dapat di-*install* dan digunakan sebagai *middleware* pada setiap layanan, memastikan konsistensi program.

2.2 Desain Sistem

Identifikasi dan pemahaman komponen-komponen utama, seperti Modul Kompresi Dekompresi JSON menjadi bagian integral dari perancangan sistem (Adek, et al., 2023).

Dalam perancangan modul kompresi dekompresi JSON, perhatian utama adalah kemudahan penggunaan dan fleksibilitas. Modul ini memiliki dua fungsi utama, yaitu kompresi *(json object*) dan dekompresi (*compressed object*). Struktur internal modul terdiri dari dua sub-modul, yaitu Modul Kompresi dan Modul Dekompresi, yang digunakan oleh fungsi kompresi dan dekompresi pada modul utama. Fungsi kompresi maupun dekompresi memiliki kemampuan pemilihan tipe kompresi yang di inginkan, di mana pemilihan tipe kompresi ini dapat di pilih oleh pengguna.

Desain implementasi modul ini ke dalam *microservice* berada pada bagian *middleware* pada setiap layanan di mana akan memiliki kemampuan untuk mengompresi dan dekompresi data pada saat *request* diterima dan sebelum *request* dikirimkan.

2.3 Implementasi

Implementasi teknologi HPACK dan Gzip melalui Modul *JavaScript* dirancang khusus untuk mengompresi *JSON* dalam konteks server dan browser. Proses implementasi dimulai dengan pembuatan modul *JavaScript* untuk mengompresi *JSON* menggunakan HPACK dan Gzip. Selanjutnya, tahap *building* modul ini memungkinkan penggunaan efektif di server. *Frontend* juga di implementasi untuk berinteraksi dengan modul ini pada browser. Keseluruhan modul kemudian dapat digunakan di server dan browser untuk memberikan kompresi yang efisien terhadap data *JSON*.

2.4 *Deployment*

Konteks penelitian ini didorong oleh kebutuhan untuk mempublikasikan modul yang telah dikembangkan ke dalam ekosistem NPM, agar dapat diakses dan digunakan setiap layanan *microservice* secara mudah. Proses *deployment* ini memberikan dasar untuk tahap pengujian selanjutnya, yang akan lebih mengevaluasi performa dan efisiensi dari aplikasi secara keseluruhan.

2.5 Pengujian

Pengujian dilakukan terhadap masing-masing tipe kompresi yaitu Hpack, Gzip dan gabungan antara Hpack dan Gzip, sebagai pembanding, pengujian di lakukan juga tanpa menggunakan kompresi apapun.

Pengujian di lakukan sebanyak 20 kali di setiap tipe kompresi, nilai yang di analisa adalah durasi respons dan ukuran respons dari setiap percobaan.

Alat pengujian yang digunakan adalah Chrome DevTools, memberikan data akurat dan dapat

diandalkan mengenai waktu pemuatan halaman dan ukuran permintaan.

Hasil pengujian disimpan dalam format file HAR, dan rata-rata hasil diperoleh untuk setiap jenis kompresi. File HAR kemudian diproses menggunakan skrip Python untuk mengonversi data menjadi format tabel yang mudah dipahami. Evaluasi dilakukan dengan mengumpulkan data atau informasi dan membandingkannya dengan standar tertentu untuk membuat kesimpulan (Ula, et al., 2021).

3. HASIL DAN PEMBAHASAN

3.1 Hasil Implementasi Modul Kompresi JSON

Modul kompresi JSON yang dikembangkan dalam penelitian ini merupakan alat yang efisien dalam mengompresi data JSON untuk penggunaan di server dan browser, mendukung aplikasi berbasis arsitektur *microservice*. Alat ini mengintegrasikan algoritma Gzip dan HPack, menawarkan fleksibilitas dalam pemilihan tipe kompresi yang sesuai dengan kebutuhan spesifik. Dikembangkan menggunakan *JavaScript* dan Rollup.js, modul ini memanfaatkan efisiensi *bundling* dan kompatibilitas dengan format ES6 *modules*. Informasi detail tentang modul, termasuk nama, versi, dan ketergantungan, diatur dalam *file* "*package.json*", memudahkan integrasi dan pemahaman oleh pengembang lain.

Setelah dipublikasikan di NPM sebagai "*@mkdierz/json-compressor*", modul ini menjadi mudah diakses dan dapat diintegrasikan ke dalam berbagai proyek. Di dalamnya, *file* index.js berfungsi sebagai titik masuk utama, menyediakan berbagai fungsi seperti kompresi, dekompresi. Ini memungkinkan pengguna untuk memilih antara kompresi Gzip, HPack, atau kombinasi keduanya sesuai dengan kebutuhan. Khususnya, pada penelitian ini akan di implementasikan kepada setiap masing-masing layanan.

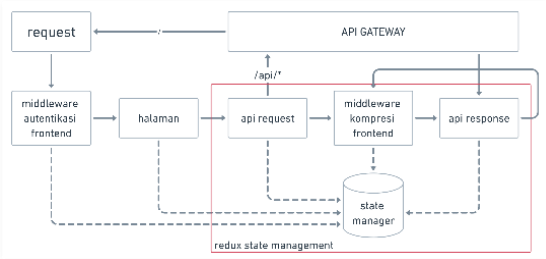
3.2 Logika Sistem dan Optimasi *Microservice*

Dalam konteks *microservices* terdapat layanan-layanan yang terpisah. Setiap layanan terbagi menjadi 2 tipe yaitu, aplikasi server yang selanjutnya akan di sebut *backend* dan aplikasi klien yang selanjutnya akan di sebut *frontend*. Untuk implementasi kompresi dibedakan berdasarkan dua tipe tersebut.

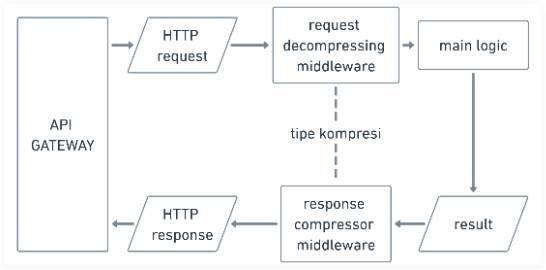
Implementasi kompresi dan dekompresi pada bagian *fontend* dapat di lihat pada Gambar 2., di mana setiap respons dari *backend* akan melalui *middleware* dekompresi untuk di dekompresi yang selanjutnya data akan di konsumsi oleh aplikasi.

Backend memiliki logika yang dapat dilihat pada Gambar 3., di mana setiap permintaan yang masuk akan melalui *middleware* dekompresi. *Middleware* ini memproses permintaan yang terkompresi, menentukan jenis kompresi untuk respons berdasarkan *header* "*Compressed-Response*". Setelah logika utama dari setiap *endpoint* dijalankan, data yang dihasilkan dikompresi sesuai pilihan pengguna sebelum dikirimkan kembali sebagai respons HTTP. Proses autentikasi dan validasi *token* juga terjadi di *backend*. Sebagai contoh, proses *login* melibatkan verifikasi kredensial pengguna di layanan autentikasi, dan jika berhasil, menghasilkan *token* JWT. Untuk validasi *token*, *token* di-*decode*, dan jika berhasil, informasi pengguna dikembalikan.

Untuk optimalisasi, terdapat *middleware* kompresi di *frontend* yang bertanggung jawab atas kompresi permintaan dan respons dari *backend*. *Middleware* ini memilih metode kompresi berdasarkan konfigurasi pengguna dan melakukan dekompresi data yang diterima dari *backend* sesuai kebutuhan.



Gambar 2. Diagram Alur *Frontend*



Gambar 3. Diagram Alur dasar logika dari layanan *backend*

Secara keseluruhan, SocialSphere memanfaatkan keunggulan arsitektur *microservice*, dengan masing-masing layanan bekerja secara mandiri namun tetap terintegrasi secara efisien melalui API *Gateway*. Hal ini menghasilkan sistem yang fleksibel, dapat diandalkan, dan mudah dikelola serta dikembangkan lebih lanjut.

3.3 Kompresi Menggunakan Hpack

Salah satu karakteristik utama dari objek JSON adalah keberadaan pasangan *key* (kunci) dan *value* (nilai). Jika terdapat sejumlah besar objek JSON, hal tersebut dapat membentuk apa yang dikenal sebagai JSON *Array*. Setiap kunci dalam objek bersifat unik, namun dalam konteks JSON *Array*, terdapat kunci yang sama pada setiap objek yang terdapat di dalam *array* tersebut. Hal ini merupakan *redundancy* data dikarenakan informasi yang berulang tetap dikirimkan, di mana data yang berulang ini menambah ukuran dari data tersebut.

Oleh karena itu Hpack menghadapi ini dengan pengurangan perulangan *key*. Pada Gambar 4 memperlihatkan respons yang di butuh kan oleh

halaman *home* yang akan di kompresi menggunakan Hpack.

Untuk memulai kompresi HPack, diperlukan kondisi bahwa setiap item *JSON Array* memiliki struktur *JSON* yang sama. Untuk memvisualisasikan pengecekan ini, dapat dilihat pada Gambar 5 dan Gambar 6 yang memperlihatkan dua objek.


```
[
  {
    "id": 2,
    "content": "thanks",
    "parentId": 1,
    "createdAt": "2023-08-29T05:58:30.422Z",
    "updatedAt": "2023-08-29T05:58:30.422Z",
    "user": {
      "id": 2,
      "name": "test",
      "username": "test",
      "avatar": null,
      "createdAt": "2023-08-29T05:58:09.487Z",
      "updatedAt": "2023-11-20T09:40:38.874Z"
    },
    "tags": [
      {
        "name": "newbie",
        "id": 2
      }
    ],
    "comment": 0
  },
  {
    "id": 1,
    "content": "Welcome ",
    "parentId": null,
    "createdAt": "2023-08-27T19:00:05.444Z",
    "updatedAt": "2023-08-27T19:00:05.444Z",
    "user": {
      "id": 1,
      "name": "syibbran",
      "username": "syibbran",
      "avatar": null,
      "createdAt": "2023-08-27T18:59:27.404Z",
      "updatedAt": "2024-01-18T09:43:35.617Z"
    },
    "tags": [
      {
        "name": "welcome",
        "id": 1
      }
    ],
    "comment": 3
  }
]
```


Gambar 4. Respons JSON *Array* dari halaman *home*


```
{
  "id": 2,
  "content": "thanks",
  "parentId": 1,
  "createdAt": "2023-08-29T05:58:30.422Z",
  "updatedAt": "2023-08-29T05:58:30.422Z",
  "user": {
    "id": 2,
    "name": "test",
    "username": "test",
    "avatar": null,
    "createdAt": "2023-08-29T05:58:09.487Z",
    "updatedAt": "2023-11-20T09:40:38.874Z"
  },
  "tags": [
    {"name": "newbie", "id": 2}
  ],
  "comment": 0
}
```


Gambar 5. Perbandingan antara *JSON* objek pertama


```
{
  "id": 1,
  "content": "Welcome ",
  "parentId": null,
  "createdAt": "2023-08-27T19:00:05.444Z",
  "updatedAt": "2023-08-27T19:00:05.444Z",
  "user": {
    "id": 1,
    "name": "syibbran",
    "username": "syibbran",
    "avatar": null,
    "createdAt": "2023-08-27T18:59:27.404Z",
    "updatedAt": "2024-01-18T09:43:35.617Z"
  },
  "tags": [
    {"name": "welcome", "id": 1}
  ],
  "comment": 3
}
```


Gambar 6. Perbandingan antara *JSON* objek kedua

Struktur yang dimaksud adalah setiap item dari *array* memiliki jumlah *key* yang sama dan nama *key* yang sama. Pada objek pertama dan kedua, dapat dilihat pada Gambar 5 dan Gambar 6, keduanya memiliki jumlah *key* yang sama (8) dan nama *key* yang sama, yaitu: *id*, *content*, *parentId*, *createdAt*, *updatedAt*, *user*, *tags*, *comment*. Hal ini memenuhi kondisi yang diperlukan untuk memulai kompresi HPack.

Sebelum memulai proses kompresi, diperlukan *array* untuk menampung seluruh hasil kompresi, karena hasil akhir dari kompresi HPack akan berbentuk *array* satu dimensi.

Pada indeks 0 dari *array* HPack, informasi jumlah *key* yang ada pada objek disimpan. Jumlah *key* yang telah terhitung sebelumnya (8) dimasukkan ke dalam *array* pada indeks 0, sebagaimana terlihat pada Tabel 1.

Pada indeks 1 sampai dengan sejumlah isi dari indeks 0, *key* yang ada dalam setiap objek disimpan. Oleh karena itu, semua nama *key* diisi dari indeks 1 sampai dengan 8, sesuai dengan Tabel 2. Pada tahap ini, *array* HPack telah memiliki informasi mengenai jumlah *key* dan *key* yang dibutuhkan bagi setiap item dari JSON *array* sebelumnya. Tahap terakhir adalah memasukkan semua *value* dari setiap item *array* sebelumnya tanpa *key* ke dalam *array* HPack. Untuk menyelesaikan tahap ini, semua item diiterasi. Hasil pada iterasi pertama dapat dilihat pada Tabel 3.

Pada

Tabel 4 memperlihatkan *array* HPack pada iterasi pertama, yaitu terdapat 8 item baru dimulai dari indeks 9 sampai dengan 16, yang berisi data dari objek pertama dari JSON *Array*. Iterasi selanjutnya dilanjutkan untuk objek seterusnya.

Tabel 1. Representasi *Array* Hpack pada tahap pertama

Indeks	Nilai
0	8
...	...

Tabel 2. Representasi *Array* Hpack pada tahap kedua

Indeks	Nilai
...	....
1	*id*
2	*content*
3	*parentId*
4	*createdAt*
5	*updatedAt*
6	*user*
7	*tags*
8	*comment*
...	...

Tabel 3. Representasi *Array* Hpack pada tahap ketiga iterasi pertama

Indeks	Nilai
...	...
9	2
10	*thanks*
11	1
12	2023-08-29T05:58:30.422Z
13	2023-08-29T05:58:30.422Z

Indeks	Nilai
14	{"id":2,"name":"test","username":"test","avatar":null,"createdAt":"2023-08-29T05:58:09.487Z","updatedAt":"2023-11-20T09:40:38.874Z"}
15	[{"name":"newbie","id":2}]
16	0
...	...

Tabel 4. Representasi *Array* Hpack pada tahap ketiga iterasi kedua

Indeks	Nilai
...	...
17	1
18	*Welcome*
19	null
20	2023-08-27T19:00:05.444Z
21	2023-08-27T19:00:05.444Z
22	{"id":1,"name":"syibbran","username":"syibbran","avatar":null,"createdAt":"2023-08-27T18:59:27.404Z","updatedAt":"2024-01-18T09:43:35.617Z"}
23	[{"name":"welcome","id":1}]
24	3
...	...

Pada

Tabel 4 memperlihatkan hasil dari iterasi kedua dari JSON *Array* yang menghasilkan 8 item baru dimulai dari indeks 17 hingga indeks 24, yang berisi data dari objek kedua. Dengan demikian, proses kompresi JSON *Array* menggunakan metode HPack telah selesai dilakukan.

3.4 Kompresi Menggunakan Gzip

Program ini menampilkan implementasi kompresi dan dekompresi objek *JSON* menggunakan algoritma yang dirujuk sebagai Gzip. Gzip Mengombinasikan algoritma LZ77 dan *Huffman coding* menghasilkan sistem kompresi yang optimal bagi data berbentuk teks. Pada proses kompresi, Gzip menerima *input* berupa *text* dan *output* berupa *binary*.


```
[{"id":2,"content":"thanks","parentId":1,"createdAt":"2023-08-29T05:58:30.422Z","updatedAt":"2023-08-29T05:58:30.422Z","user":{"id":2,"name":"test","username":"test","avatar":null,"createdAt":"2023-08-29T05:58:09.487Z","updatedAt":"2023-11-20T09:40:38.874Z"},"tags":[{"name":"newbie","id":2}],"comment":0},{"id":1,"content":"Welcome","parentId":null,"createdAt":"2023-08-27T19:00:05.444Z","updatedAt":"2023-08-27T19:00:05.444Z","user":{"id":1,"name":"syibbran","username":"syibbran","avatar":null,"createdAt":"2023-08-27T18:59:27.404Z","updatedAt":"2024-01-18T09:43:35.617Z"},"tags":[{"name":"welcome","id":1}],"comment":3}]
```

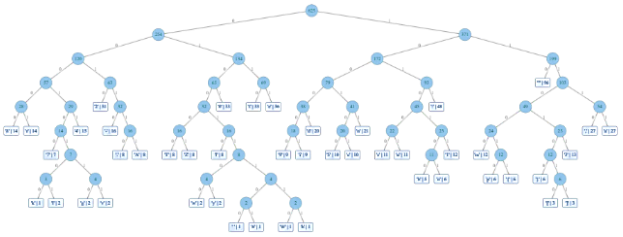

Gambar 7. Representasi Teks dari JSON *Array*

Untuk memulai kompresi Gzip pertama-tama Objek *JSON* di ubah menjadi bentuk teks. *JSON Array* yang telah di ubah menjadi teks dapat di lihat pada Gambar 7 Setelah data di ubah ke dalam bentuk teks dilakukan perhitungan frekuensi kemunculan karakter yang dapat dilihat pada Tabel 5.

Setelah mendapatkan tabel frekuensi karakter, langkah selanjutnya melibatkan proses pembuatan *Huffman tree*. Pembuatan *Huffman tree* merupakan suatu tahap kritis dalam teori kompresi data yang bertujuan untuk menyusun pohon biner yang merepresentasikan struktur pembagian panjang kode biner *Huffman* berdasarkan frekuensi karakter-karakter yang terdapat. Proses ini diterapkan dengan menggunakan algoritma *Greedy*, di mana karakter-karakter dengan frekuensi terendah dikelompokkan bersama dan membentuk cabang-cabang pohon yang lebih tinggi. Setiap langkah dalam proses ini melibatkan penggabungan dua simpul dengan frekuensi terendah, dan hasilnya ditempatkan kembali dalam himpunan karakter dengan frekuensi yang diperbarui. Proses ini berlanjut hingga hanya terdapat satu simpul pada tingkat tertinggi yang mewakili seluruh himpunan karakter yang dapat di lihat pada Gambar 8.

Tabel 5. Frekuensi kemunculan karakter

No.	*Char*	Frekuensi	No.	*Char*	Frekuensi
1	6	1	24	9	9
2	h	1	25	i	9
3	k	1	26	5	10
4	W	1	27	c	10
5	' '	1	28	s	11
6	I	2	29	u	11
7	v	2	30	1	12
8	g	2	31	m	12
9	w	2	32	3	13
10	y	2	33	8	14
11	[	3	34	r	14
12	]	3	35	4	15
13	b	5	36	-	16
14	{	6	37	d	20
15	o	6	38	n	21
16	p	6	39	,	27
17	}	6	40	a	27
18	7	7	41	2	31
19	A	8	42	0	33
20	T	8	43	t	33
21	.	8	44	e	36
22	Z	8	45	:	48
23	l	8	46	"	96



Gambar 8. Hasil Pembuatan *Huffman Tree*

Setelah *Huffman tree* terbentuk, langkah berikutnya dalam proses *Huffman encoding* adalah menetapkan kode biner. Kode biner adalah kode yang unik bagi tiap karakter di mana kode biner ini akan mencerminkan jalur yang diikuti dalam *Huffman tree* dari akar ke daun yang mewakili karakter tersebut. Proses penyandian *Huffman* melibatkan *recursive* melintasi *Huffman tree* dan memberikan nilai biner yang unik untuk setiap karakter. Proses ini menghasilkan sebuah tabel atau kamus yang menghubungkan setiap karakter dengan kode biner *Huffman* itu sendiri. Hasil dari Pengkodean *Huffman* dapat di lihat pada Tabel 6.

Setelah setiap karakter telah memiliki representasi binernya masing-masing representasi teks dari JSON *array* di ubah menjadi biner

berdasarkan ketentuan yang telah di dapatkan sehingga menghasilkan biner yang dapat dilihat pada Gambar 9. Dengan demikian proses kompresi Gzip selesai di lakukan.

Tabel 6. Ketetapan kode biner bagi setiap karakter

No.	*Char*	Biner	No.	*Char*	Biner
1	0	101	24	h	10011101
2	1	101011	25	k	10011110
3	2	10	26	W	10011111
4	3	111011	27	t	110
5	4	11	28	e	111
6	5	100100	29	i	100001
7	6	10011100	30	d	10001
8	7	100	31	c	100101
9	8	0	32	n	10011
10	9	100000	33	s	101000
11	r	1	34	u	101001
12	' '	10100	35	b	1010100
13	I	10101	36	{	1010101
14	v	10110	37	:	1011
15	g	10111	38	"	110
16	-	110	39	m	111000
17	A	1110	40	o	1110010
18	T	1111	41	p	1110011
19	.	10000	42	}	1110100
20	Z	10001	43	[	11101010
21	l	10010	44	]	11101011
22	w	1001100	45	,	11110
23	y	1001101	46	a	11111

1110101010101011101000011000111010110010111101101001011110010100110110
0111100110110110101111001100100111011111110011010011110101000110111101
1011100111111100001011110011011000010101100011101011101011111101101001
0100001011111111011001111000100111001101101011110001001010010111011001
1001010000000110001010000000111101011001001011100100000001011111011010
1010000000110010001001000111011110110101001111001110001111110110011110
0010011100110110101111000100101001011101100110010100000001100010100000
0011110101100100101110010000000101111101101010100000001100100010010001
1101111011010100110100001110000111010111010101110100001100011101011001
0111101101001111111111000011111010111100110011110100001101101111011010
1001101000011100001100111111111100001111101011110011001111010000110110
1111011011111000101101111101101111100001110101110011101001010010010010
1111011010010100001011111111011001111000100111001101101011110001001010
0101110110011001010000000110001010000000111101011001001011100100000001
0110101100000010000000110000000010001000111011110110101001111001110001
1111101100111100010011100110110101111000100101001011101100110101011101
0110011000100101001111010110000010110001101011011111011000000100000000
0000100000110100011101110100111101100110111110001011110100011010111110
10101010101110100111111111110000111110101111010011011101001100101010010
0001011111011110110100001100011101011001011101001110101111110110100101
1110010111000111000011110011011011010110101111010011110101010111010000
1100011101011101011111101101001011110010100110110011110011011011010111
1001001111101110100101001011110010111000011100010100110111101101110011
1111100001011110011011000010101100011101011100111010010100100100101111
0110100101000010111111110110011110001001110011011010111100010010100101
1101100110010100000000110001000010000111110101110000010110101010101101
0110010001000000001100011000110100011101111011010100111100111000111110
1100111100010011100110110101111000100101001011101100110010100000001100
01000010000111110101110000010110101010110110101100100010000000011000110
0011010001110111101101010011010000111000011101011101010111010000110001
11010111010111111011010011111111111000011111010111101010000100110110000
1101010010101000000111111100111101111011010100110100001110000110011111
11111000011111010111101010000100110110000110101001010100000001111111001
11101111011011111100010110111110110111110000111010111001110100101001001
0010111101101001010000101111111101100111100010011100110110101111000100
10100101110110011001010000000011000100001000011111010110000010111001001
00000101100100001000100000000110101000110100011101111011010100111100111
0001111110110011110001001110011011010111100010010100100001100110010110
1011001101010110000000111101011000001011000111110111011111011100100010
00001001110010101100010001000111011101001110110011011111000101111010 0
0110101111101010101010111010011111111111000011111010111001001100011101
0010100101111001011100001111011110110100001100011101011101011111101001
1101011111101101001011110010111000111000011110011011011010111110111110
10011101011

Gambar 9. Representasi Biner Hasil Kompresi Gzip

3.5 Hasil Pengujian

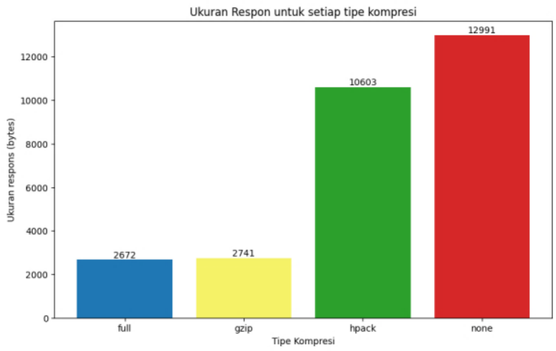
Penelitian ini bertujuan untuk mengevaluasi dampak kompresi respons dan *request* pada *microservice* menggunakan HPack, Gzip, dan kombinasi keduanya. Pengujian dilakukan pada durasi yang dapat dilihat pada **Error! Reference source not found.** dan ukuran respons untuk menyimpulkan pengaruh penggunaan kompresi terhadap performa sistem tersaji pada Tabel 7**Error! Reference source not found.**. Dan hasil pengujian terhadap ukuran respons dapat di lihat pada Tabel 8.

Tabel 7. Hasil Pengujian Terhadap Waktu Respons (*millisecond*)

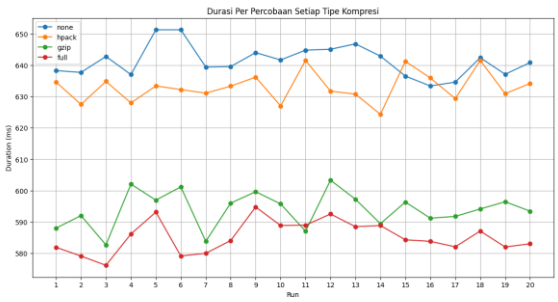
Percobaan ke-	*full*	*gzip*	*hpack*	*none*
1	581,872	587,981	634,640	638,299
2	579,066	592,004	627,496	637,715
3	576,099	582,622	634,866	642,826
4	586,117	602,051	627,960	637,020
5	593,116	596,936	633,417	651,335
6	579,099	601,164	632,231	651,335
7	579,951	583,766	631,096	639,438
8	584,024	595,975	633,334	639,605
9	594,756	599,630	636,206	644,039
10	588,820	595,721	627,032	641,704
11	588,944	587,011	641,564	644,830
12	592,502	603,295	631,739	645,120
13	588,448	597,218	630,780	646,835
14	588,821	589,410	624,333	642,933
15	584,254	596,321	641,265	636,545
16	583,766	591,165	635,960	633,397
17	582,016	591,740	629,349	634,620
18	587,052	594,107	641,564	642,430
19	581,969	596,421	630,947	637,130
20	582,994	593,404	634,189	640,856

Tabel 8. Hasil Pengujian Terhadap Ukuran Respons

Tipe Kompresi	Ukuran Respon (*bytes*)
full	2672
gzip	2741
hpack	10603
none	12991



Gambar 10. Grafik perbandingan ukuran respons



Gambar 11. Grafik garis dari durasi tiap kompresi dari seluruh percobaan

3.6 Analisa Hasil

Ukuran respons dari setiap tipe kompresi tetap konsisten pada setiap percobaan, menunjukkan bahwa hasil kompresi hanya mempengaruhi ukuran dasar respons. Pada Tabel 8 dan Gambar 10**Error!**

Reference source not found.Error! Reference source not found., terlihat dengan jelas efektivitas berbagai metode kompresi terhadap ukuran respons data dalam hal ukuran. Tanpa penerapan teknik kompresi, ukuran respons adalah 12,991 *byte*. Penggunaan HPack mengurangi ukuran ini menjadi 10,603 *byte*, mencerminkan penurunan sebesar 18,382% dari ukuran asli.

Lebih lanjut, penerapan Gzip mengurangi ukuran menjadi 2,741 *byte*, sebuah pemangkasan yang signifikan sebesar 78,901% dibandingkan dengan data yang tidak terkompresi. Metode kompresi gabungan, yang mengombinasikan HPack dan Gzip, berhasil menghasilkan ukuran respons yang lebih kecil lagi, yaitu 2,672 *byte*. Ini menunjukkan pengurangan sebesar 79,432% dibandingkan dengan kondisi tanpa kompresi, dan penurunan sebanyak 74,800% dibandingkan dengan HPack. Meskipun penurunan ukuran dari Gzip ke metode gabungan tidak signifikan hanya sebesar 2,517%, hal ini menegaskan temuan yang diungkapkan oleh (Objelean, 2011), yang menyatakan bahwa tidak ada peningkatan efisiensi yang signifikan ketika kedua metode kompresi ini digabungkan. Analisis ini mengindikasikan bahwa Gzip sendiri sudah sangat efisien dalam mengurangi ukuran respons data. Dengan demikian, penambahan HPack ke dalam proses kompresi tidak memberikan kontribusi yang signifikan terhadap pengurangan ukuran lebih lanjut. Detail persentase perubahan ukuran dari setiap tipe kompresi dapat dilihat pada Tabel 9**Error! Reference source not found.**.

Tabel 9. Persentase Perbandingan Ukuran Dari Setiap Tipe Kompresi

Pembanding	*none*	*hpack*	*gzip*	*full*
none	0,000%	-18,382%	-78,901%	-79,432%
hpack	22,522%	0,000%	-74,149%	-74,800%
gzip	373,951%	286,830%	0,000%	-2,517%
full	386,190%	296,819%	2,582%	0,000%

Tabel 10. Rangkuman Hasil Durasi Respons Dari Setiap Tipe Kompresi dalam satuan *ms*

Tipe Kompresi	*none*	*hpack*	*gzip*	*full*
Rata-rata	641,401	632,998	593,897	585,184
Median	641,280	632,783	594,914	584,139
Minimum	633,397	624,333	582,622	576,099
Maksimum	651,335	641,564	603,295	594,756
Range	17,938	17,231	20,673	18,657
Variasi	23,331	21,831	31,447	24,422
Standar Devariasi	4,830	4,672	5,608	4,942

Tabel 11. Persentase Perbandingan Durasi Respons Dari Setiap Tipe Kompresi

Pembanding	*none*	*hpack*	*gzip*	*full*
none	0,000%	-1,310%	-7,406%	-8,765%
hpack	1,327%	0,000%	-6,177%	-7,555%
gzip	7,998%	6,584%	0,000%	-1,468%
full	9,607%	8,172%	1,490%	0,000%

Membahas hasil durasi respons, walaupun ukuran *request* tetap pada setiap 20 percobaan, durasi respons dari masing-masing tipe kompresi menunjukkan sedikit variasi yang dapat diamati pada **Error! Reference source not found.**, di mana terdapat selisih yang cukup besar antar tipe kompresi. Tabel 10**Error! Reference source not found.** memberikan gambaran terperinci tentang statistik deskriptif untuk setiap tipe kompresi data pada durasi respons.

Tipe Kompresi *'None'* menunjukkan tingkat konsistensi yang relatif tinggi dalam durasi respons, dengan standar deviasi sebesar 4,83 ms. Data menunjukkan konsentrasi yang cukup signifikan di sekitar nilai rata-rata 641,40 ms, menandakan homogenitas durasi respons tanpa adanya kompresi.

Tipe Kompresi 'HPack' menunjukkan durasi respons yang sangat terpusat dan konsisten, dengan rata-rata sebesar 633,00 ms dan median yang hampir identik pada 632,78 ms. Standar deviasi yang sedikit lebih rendah dibandingkan dengan *'None'*, yakni 4,67 ms, serta rentang yang hampir serupa, menegaskan stabilitas dalam performa kompresi HPack.

Tipe Kompresi 'Gzip' menunjukkan rata-rata durasi respons terendah di antara semua metode, yakni 593,90 ms, menandakan bahwa Gzip memberikan waktu respons yang lebih cepat secara konsisten. Meskipun memiliki rentang yang lebih luas dibandingkan dengan *'None'* dan 'HPack' sebesar 20,67 ms, nilai standar deviasi pada 5,61 ms tetap menunjukkan bahwa Gzip memiliki variabilitas yang terbatas.

Tipe Kompresi *'Full'*, sebagai kombinasi HPack dan Gzip, mendekati performa Gzip dengan nilai rata-rata durasi respons sebesar 585,18 ms. Variabilitas yang serupa dengan Gzip, ditunjukkan oleh standar deviasi sebesar 4,94 ms, menunjukkan bahwa metode ini memberikan konsistensi yang baik dengan adanya penggabungan kompresi.

Dari analisis ini, Gzip dan *Full* menunjukkan durasi respons rata-rata yang lebih baik dibandingkan dengan 'None' dan HPack. Gzip menonjol dengan waktu respons rata-rata tercepat, meskipun dengan variabilitas yang sedikit lebih tinggi. Metode *Full* menunjukkan tingkat konsistensi yang sebanding dengan HPack namun dengan kecepatan yang lebih baik.

Berdasarkan tabel data durasi rata-rata yang telah dihitung, kompresi 'HPack' menghasilkan penurunan rata-rata durasi sebesar 1,31% dibandingkan dengan *'None'*. 'Gzip' menunjukkan penurunan yang lebih signifikan, yaitu sebesar 7,41% dibandingkan dengan *'None'* dan 6,18% dibandingkan dengan 'HPack'. *'Full'*, sebagai kombinasi dari 'HPack' dan 'Gzip', mencatat penurunan rata-rata durasi sebesar 8,77% dibandingkan dengan 'None', 7,55% dibandingkan dengan 'HPack', dan 1,47% dibandingkan dengan 'Gzip'.

Ketika dilihat dari perspektif sebaliknya, *'None'* menunjukkan peningkatan durasi rata-rata sebesar 1,33% ketika dibandingkan dengan 'HPack', 8% dengan 'Gzip', dan 9,61% dengan *'Full'*,

mengindikasikan bahwa *'None'* memiliki durasi yang lebih panjang karena tidak ada proses kompresi yang diterapkan. Detail persentase perbandingan rata-rata durasi dari tiap metode kompresi dapat dilihat pada Tabel 11.

Tabel 11 menyajikan persentase perbandingan durasi respons untuk empat tipe kompresi: none (tanpa kompresi), hpack, gzip, dan full (kombinasi dari hpack dan gzip). Persentase ini menggambarkan perubahan waktu respons saat menggunakan metode kompresi yang berbeda.

Melalui analisis ini, dapat disimpulkan bahwa 'Gzip' dan *'Full'* memberikan peningkatan efisiensi yang signifikan dalam hal durasi rata-rata jika dibandingkan dengan 'none' dan 'Hpack'. Metode *'Full'*, yang menggabungkan dua metode kompresi, menunjukkan performa terbaik dengan durasi rata-rata yang paling rendah. Meskipun 'Gzip' sudah mengalami penurunan yang besar dibanding 'none', penurunan tambahan yang diberikan oleh metode *'Full'* menunjukkan bahwa kombinasi kedua teknik tersebut menghasilkan kinerja yang lebih optimal.

4. KESIMPULAN DAN SARAN

4.1 Kesimpulan

Setelah melakukan pengujian dengan menggunakan kompresi Hpack dan Gzip pada arsitektur *microservice* dapat ditarik kesimpulan sebagai berikut:

Optimasi komunikasi antar layanan melalui kompresi HPack dan Gzip telah diuji dan hasilnya menunjukkan peningkatan performa yang signifikan melalui pengurangan ukuran dari 12991 *bytes* sebayak ~79,432% dan durasi respons sebanyak ~8,765% dari 641,401 *millisecond*.

Penambahan HPack tidak memberikan kontribusi yang signifikan terhadap pengurangan ukuran respons dibandingkan dengan hanya menggunakan Gzip ~0,531% dari segi durasi respons dan sebanyak ~1,47% dari ukuran respons

Kompresi Gzip secara signifikan menurunkan ukuran dari respons sebanyak ~78,901% yaitu 2741 *bytes* dibandingkan dengan tanpa menggunakan kompresi apa pun sebanyak 12991 *bytes*. Sedangkan pada sisi waktu terdapat penurunan sebanyak ~7,406% dengan nilai 593,897 *millisecond* dibanding dengan tanpa menggunakan kompresi mana pun dengan durasi 641,401 *millisecond*.

Penggabungan Gzip dan Hpack tidak memiliki peningkatan signifikan dibandingkan dengan hanya menggunakan kompresi Gzip. Pada segi ukuran respons Gzip ukuran sebesar 2741 *bytes*, sedangkan Gabungan Gzip dan Hpack memiliki ukuran respons sebesar 2672 *bytes* di mana perubahannya hanya ~2,582% dibanding Gzip. Jika di amati pada durasi respons Gzip (593,897 *millisecond*) dan Gabungan Gzip dan Hpack (585,184 *millisecond*) hanya memiliki penurunan sebanyak ~1,490%.

4.2 Saran

Untuk penelitian lebih lanjut, peneliti menyarankan beberapa area penting yang dapat dijelajahi.

Optimasi Kompresi Data; Meskipun penelitian ini menemukan bahwa HPack tidak signifikan dalam mengurangi ukuran respons ketika digunakan bersama Gzip, peneliti menyarankan pengujian lebih lanjut untuk mengeksplorasi kasus penggunaan yang mungkin di mana HPack dapat memberikan manfaat lain, seperti kecepatan dekompresi atau efisiensi dalam skenario transmisi data tertentu.

Pengujian dalam Berbagai Kondisi Jaringan; Pengujian lebih lanjut di lingkungan dengan berbagai kondisi jaringan dan beban pengguna yang berbeda dapat membantu dalam memahami performa arsitektur *microservice* dalam kondisi operasional yang realistis.

DAFTAR PUSTAKA

ADEK, R. T., BUSTAMI, B. & ULA, M., 2023. Systematics Review on Detecting Cyberattack Threat by Social Network Analysis and Machine Learning. *Lecture Notes in Networks and Systems*, Volume 448, pp. 567-577.

BRAY, T., 2017. *The JavaScript Object Notation (JSON) Data Interchange Format*, s.l.: RFC Editor.

DRAGONI, N. et al., 2017. Microservices: Yesterday, today, and tomorrow. *Present and Ulterior Software Engineering*, 11.pp. 195-216.

LEWIS, J. & FOWLER, M., 2014. *Microservices.* [Online] Available at: https://martinfowler.com/articles/microservices.html [Diakses 22 March 2023].

OBJELEAN, A., 2011. *JSON Compression Algorithms.* [Online] Available at: http://repository.utm.md/handle/5014/6418

PEON, R. & RUELLAN, H., 2015. *HPACK: Header Compression for HTTP/2*, s.l.: s.n.

RICHARDS, M., 2022. *Software architecture patterns.* Second Edition penyunt. Sebastopol: O'Reilly Media, Inc..

RICHARDSON, C., 2018. *Microservices patterns: with examples in Java.* s.l.:Simon and Schuster.

SALAH, T. et al., 2017. The evolution of distributed systems towards microservices architecture. *2016 11th International Conference for Internet Technology and Secured Transactions, ICITST 2016*, 2.pp. 318-325.

ULA, M., TJUT ADEK, R. & BUSTAMI, B., 2021. Emarketplace Performance Analysis Using PIECES Method. *International Journal of Engineering, Science and Information Technology*, 8, 1(4), pp. 1-6.

YUNIZAR, Z., 2023. Sistem Monitoring Revass (Revanue Assurance) Pelanggan Pasang Baru Di PT. PLN ULP Krueng Geukuh Lhokseumawe Berbasis Web. *Jurnal Teknologi Terapan and Sains 4.0*, 5, 4(1), pp. 923-937.