Teknologi: Jurnal Ilmiah Sistem Informasi 14 (1) Januari 2024 37-47

ISSN 2527-3671 (*Online*)
ISSN 2087-8893 (*Print*)

Tersedia *online* di www.journal.unipdu.ac.id

Unipdu

Halaman jurnal di www.journal.unipdu.ac.id/index.php/teknologi

Analisis Kinerja Streaming RabbitMQ dan Nats untuk Komunikasi di Layanan Mikro

Agung Nur Aprianto[a], Abba Suganda Girsang[b], Yulianto Nugroho[c], Widjaya Kumala Putra[d]
a,b,c,d Department of Computer Science, Bina Nusantara University, Jakarta, Indonesia

email: [a,]agung.aprianto@binus.ac.id*

*Korespondensi

Dikirim 29 Februari 2024; **Direvisi** 03 Maret 2024; **Diterima** 13 Maret 2024; **Diterbitkan** 15 Maret 2024

Abstrak

Dalam penelitian ini, pengujian kinerja dilakukan antara dua broker pesan yang umum digunakan di perusahaan, yaitu RabbitMQ dan Nats Streaming. REST adalah metode yang mengimplementasikan permintaan protokol HTTP untuk mengakses dan menggunakan data. REST adalah salah satu metode gaya sinkron dalam layanan mikro, gaya lainnya adalah asinkron yang dapat diimplementasikan melalui perantara pesan. REST dapat digunakan untuk komunikasi antar layanan dalam microservice, karena menggunakan protokol HTTP, kinerjanya akan menurun ketika jumlah permintaan melimpah dan kurang dapat diandalkan karena komunikasinya yang sinkron. Dengan menggunakan message broker sebagai media komunikasi antar layanan di microservice, maka setiap layanan yang terhubung tidak akan saling bergantung satu sama lain dan akan membuat pengiriman pesan lebih terjamin. Oleh karena itu, penelitian ini akan mengimplementasikan perantara pesan untuk komunikasi antar proses (IPC) pada layanan mikro. Saat ini sudah banyak broker pesan yang dikembangkan oleh berbagai perusahaan atau komunitas. Dalam penelitian ini, kami melakukan eksperimen dengan kedua perantara pesan tersebut. Ketiga aspek yang akan diuji yaitu throughput, latensi berdasarkan jumlah pesan, dan latensi berdasarkan ukuran pesan. Kinerjanya akan dievaluasi model arsitektur yang bertindak sebagai produsen dan konsumen. Modelnya adalah salah satu produsen dan konsumen jasa. Layanan ini akan diterapkan pada kontainer buruh pelabuhan

Kata Kunci: Layanan mikro, asinkron, message broker, docker container, RabbitMQ, Nats Streaming

Performance Analysis of RabbitMQ and Nats Streaming for Communication in Microservice

Abstract

In this research, performance testing is performed between the two message brokers, which commonly used in the enterprise, namely RabbitMQ and Nats Streaming. REST is a method that implements the HTTP protocol requests to access and use data. REST is one of the synchronous style methods in microservice, the other style is asynchronous that can be implemented through message broker. REST can be used for communication between services in microservice, since it is using HTTP protocol, the performance will degrade when the amount of request is abundant and less reliable due to its synchronous communication. By using a message broker as the medium of communication between services in microservice, each connected service will not rely on each other and will make the message delivery more guaranteed. By reason, this research will implement a message broker for inter process communication (IPC) in microservice. Today there are many message brokers developed by various companies or communities. In this research, we do experiments with both message brokers. The three aspects will be tested, they are throughput, latency by number of messages and latency by message size. The performance will be evaluated the architecture model that act as producer and consumer. The model is one producer and consumer service. The service will be deployed on docker container

Keywords: *: Microservice, Asynchronous, Message Broker, Docker Container, RabbitMQ, Nats Streaming*

Untuk mengutip artikel ini dengan *APA Style*:
Aprianto. N. A, Girsang. S.A, Nugroho. Y & Putra K. W (2024). Analisis Kinerja Streaming RabbitMQ dan Nats untuk Komunikasi di Layanan Mikro. TEKNOLOGI: Jurnal Ilmiah Sistem Informasi, 14(1), 37-47 : https://doi.org/10.26594/teknologi.v14i1.4498

1. Pendahuluan

Microservices adalah gaya arsitektur yang terinspirasi oleh komputasi berorientasi layanan yang baru-baru ini mulai mendapatkan popularitas [1]. Layanan mikro dapat diterapkan secara independen satu sama lain dan digabungkan secara longgar dan masing-masing layanan mikro ini berfokus pada penyelesaian satu tugas saja. Layanan mikro mendukung banyak klien dalam arsitektur yang khas [2].

Microservice Architecture (MSA) adalah pola desain arsitektur sistem yang pendekatannya membangun aplikasi sebagai rangkaian layanan [3]. Alternatif untuk MSA adalah gaya Arsitektur

ISSN 2502-3357 (*Online*)
N. A Aprianto dkk./TEKNOLOGI 14 (1) 2024 37-47 ISSN 2503-0477 (*Print*)

Berorientasi Layanan (SOA), yang juga memanfaatkan layanan sebagai blok sistem bawaan perangkat lunak terdistribusi [4]. SOA memiliki Enterprise Service Buses (ESB) sebagai middleware, bukan MSA yang menggunakan teknologi ringan untuk mendistribusikan tugas ke setiap komponen layanan. Layanan merupakan suatu unit yang dapat berjalan secara independen dan berkomunikasi satu sama lain dengan mekanisme sederhana seperti Application Programming Interface (API) [5]. Dalam MSA ada dua jenis gaya interaksi yang disebut komunikasi antar proses (IPC) yang dapat dipilih: sinkron dan asinkron [6]. Komunikasi sinkron sering dianggap sebagai gaya interaksi permintaan/tanggapan. Satu layanan mikro membuat permintaan ke layanan lain dan menunggu layanan tersebut memproses hasilnya dan mengirimkan respons kembali. REST adalah salah satu kerangka kerja paling umum untuk mengimplementasikan bentuk komunikasi sinkron.

REST (Representational state transfer) adalah arsitektur API (Application Programming Interface) yang menyediakan komunikasi client-server untuk Aplikasi Web melalui protokol HTTP, sehingga mudah diimplementasikan karena tidak terikat pada protokol transfer apa pun [7]. RESTful API adalah gaya arsitektur untuk API yang menggunakan permintaan Hypertext Transfer Protocol (HTTP) untuk mengakses dan menggunakan data [8]. Gaya layanan mikro IPC lainnya adalah asinkron. Bentuk komunikasi asinkron dapat diimplementasikan dalam layanan mikro ketika layanan bertukar pesan satu sama lain melalui perantara pesan. Dalam bentuk interaksi ini, perantara pesan bertindak sebagai perantara antar layanan untuk mengoordinasikan permintaan dan tanggapan [6]. Protokol AMQP dirancang untuk memberikan kemampuan interoperabilitas antara aplikasi dan sistem yang berbeda, memungkinkan pertukaran pesan antara platform yang berbeda, diimplementasikan dalam bahasa yang berbeda [9]. AMQP (Advanced Message Queuing Protocol) adalah protokol pengiriman pesan yang memungkinkan penyesuaian aplikasi klien untuk berkomunikasi dengan broker middleware pengiriman pesan yang sesuai [10]. AMQP adalah lapisan aplikasi standar terbuka untuk layanan middleware berorientasi pesan (MOM) yang berorientasi protokol, memungkinkan interoperabilitas pengiriman pesan antar sistem, apa pun platform yang digunakan. AMQP adalah protokol lapisan aplikasi terbuka yang lazim untuk protokol pesan middleware [11]. Salah satu keuntungan utama AMQP adalah sangat ringan sehingga sangat cepat, mengkonsumsi lebih sedikit waktu CPU dan mudah untuk dikonfigurasi dan dijalankan [12].

Terdapat penelitian sebelumnya yang membandingkan kinerja REST dengan RabbitMQ [13], mengevaluasi kinerja broker pesan untuk komunikasi pada komputasi kabut dan mengevaluasi kinerja arsitektur layanan mikro menggunakan container [14]. Dalam penelitian ini akan dievaluasi kinerja broker pesan (RabbitMQ dengan broker pesan modern) dengan empat aspek yaitu throughput, latensi berdasarkan jumlah pesan, latensi ukuran pesan dan total waktu untuk memproduksi dan mengkonsumsi pesan di empat model dan diterapkan menggunakan wadah buruh pelabuhan. Kinerja akan dievaluasi pada empat model arsitektur. Empat model arsitektur tersebut terdiri dari jasa yang bertindak sebagai produsen dan konsumen. Modelnya adalah jasa satu produsen dan konsumen, jasa satu produsen dan banyak konsumen, jasa banyak produsen dan satu konsumen, dan jasa banyak produsen dan banyak konsumen. Layanan akan diterapkan pada kontainer buruh pelabuhan.

2. State of the Art

Penelitian mengenai MSA telah banyak dilakukan sebelumnya, baik yang dilakukan oleh pelaku industri, praktisi, maupun akademisi dengan sudut pandang dan skenario yang berbeda-beda. Dalam penelitian Hong tentang mengevaluasi kinerja RESTful API dan RabbitMQ untuk komunikasi antar layanan dalam layanan mikro. Pada percobaan ini diberikan masukan dari jumlah pengguna yang bervariasi dari 50, 100, 150, 200, 250 hingga 300 pengguna yang mengirimkan request secara bersamaan dalam waktu 15 menit. Hasilnya, kecepatan respons metode RESTful API secara bertahap lebih cepat dibandingkan mode RabbitMQ. Ketika jumlah pengguna meningkat menjadi 200 pengguna, kecepatan respons Restful API hampir dua kali lipat dari mode RabbitMQ. Ketika jumlah pengguna bertambah menjadi 300 pengguna, metode RESTful API memiliki kinerja yang sangat buruk dan mode RabbitMQ relatif stabil. Disimpulkan bahwa untuk menangani data dalam jumlah besar, AMQP relatif lebih baik dibandingkan REST [15].

Dalam penelitian Hasselbring & Steinacker tentang arsitektur layanan mikro untuk skalabilitas, kelincahan, dan keandalan dalam e-commerce. Penelitian ini membangun kembali website ecommerce otto.de dengan menggunakan microservices dan menggunakan REST API sebagai IPC antar layanan. Hasil penelitian meningkatkan skalabilitas dan jumlah penerapan per minggu serta mengurangi jumlah insiden [16].

Penelitian yang dilakukan oleh Ionescu membahas tentang tantangan analisis kinerja RabbitMQ dan Analisis Kinerja Broker Pesan ActiveMQ untuk Komunikasi dalam Komputasi Kabut. Pada penelitian ini dilakukan evaluasi kinerja message broker pada RabbitMQ & ActiveMQ. Berdasarkan hasil penelitian yang

ISSN 2502-3357 (*Online*)
N. P. Satu dkk./TEKNOLOGI volume (no) Tahun Halaman Awal-Halaman Akhir ISSN 2503-0477 (*Print*)

telah dilakukan diperoleh bahwa ActiveMQ menerima pesan lebih cepat sedangkan RabbitMQ lebih cepat dalam menghasilkan pesan [17].

Dalam penelitian yang dilakukan Bagaskara, peneliti membahas mengenai Analisis Kinerja Perantara Pesan untuk Komunikasi pada Fog Computing. Dalam studi ini, peneliti menguji broker pesan RabbitMQ dan Apache Kafka untuk mengukur latensi dan throughput dengan mengirimkan data sebanyak mungkin menggunakan alat RabbitMQ-Perftest dan Kafka-Perftest. Dari penelitian diperoleh bahwa Kafka memiliki throughput yang lebih baik dibandingkan RabbitMQ ketika pesannya kecil. Untuk pengujian latensi RabbitMQ lebih baik dibandingkan Kafka walaupun hanya sedikit berbeda [18].

Penelitian ini membahas tentang Evaluasi Kinerja Arsitektur Microservices Menggunakan Container. Dalam studi ini, peneliti membandingkan kinerja CPU & Jaringan dari dua model layanan mikro berbasis container: master-slave, atau nested-container. Dari hasil evaluasi kinerja kedua model yaitu master-slave & nested-container, nested-container merupakan model yang tepat [14].

Fernandes melakukan evaluasi kinerja layanan web RESTful dan protokol AMQP untuk pertukaran pesan antara klien dan server. Dalam penelitian ini, percobaan dilakukan dua puluh empat pengguna untuk mengirim pesan selama 30 menit. Percobaan yang dilakukan sebagai berikut, pertama aplikasi pengguna mengirimkan pesan selama 30 menit ke RabbitMQ menggunakan protokol AMQP, kedua aplikasi pengguna mengirimkan pesan selama 30 menit ke server Web service menggunakan protokol HTTP, ketiga aplikasi pengguna mengirimkan pesan selama 30 menit ke RabbitMQ menggunakan protokol AMQP. Oleh karena itu, pendekatan terbaik untuk bertukar data dalam jumlah besar adalah dengan menggunakan protokol AMQP. Pendekatan ini akan memungkinkan penghematan sumber daya, mencegah kehilangan data, dan pengorganisasian pesan yang lebih baik [13].

Penelitian ini mengkaji pemanfaatan sumber daya secara efisien dengan berbagai model penerapan layanan mikro. Para peneliti menganalisis pengaruh model penerapan terhadap kinerja layanan mikro. Data dibangun dengan menggunakan 1 server dan 3 server. Dari hasil penelitian penerapan menggunakan 1 server memiliki utilisasi CPU dan bandwidth jaringan yang lebih baik [19].

Penelitian ini membahas tentang perancangan berbasis container dan berbasis microservices untuk infrastruktur cloud DevOps. Para peneliti sedang membangun layanan infrastruktur di cloud menggunakan teknologi container dan layanan mikro. berdasarkan hasil penelitian yang dilakukan teknologi container mempercepat waktu inisialisasi dan mengurangi utilitas CPU selama penerapan [20].

Penelitian ini membahas tentang pesan modern dimana terdapat beberapa sistem pesan modern yang pernah muncul di masa lalu, yang semuanya memiliki kelebihan dan kekurangannya masing-masing. Ini telah menjadi masalah bagi industri untuk memutuskan sistem pesan mana yang paling cocok untuk aplikasi tertentu. Studi mendalam diperlukan untuk memutuskan fitur sistem pesan mana yang memenuhi kebutuhan aplikasi. Makalah survei ini menguraikan teknologi perpesanan modern dan menggali lebih dalam tiga sistem penerbit/pelanggan populer- Apache Kafka, RabbitMQ, dan NATS Streaming. Makalah ini memberikan gambaran singkat tentang cara kerja tiga kerangka kerja perpesanan populer- Apache Kafka, NATS Streaming, dan RabbitMQ. Setelah itu, dibahas fitur perbandingan yang berkisar pada persistensi pesan, reservasi pesan, throughput, latensi, skalabilitas, dan ketersediaan [21].

Dalam penelitian ini, peneliti melakukan salah satu implementasi alat untuk memantau broker RabbitMQ dan antrian mereka di sistem rumah pintar yang disajikan. Alat ini memantau jumlah pesan dalam antrian dan mengirimkan pemberitahuan yang tepat ketika masalah terdeteksi. Pemberitahuan ini memicu komponen sistem yang dapat menindaklanjuti untuk mengatasi masalah tersebut. Alat tersebut telah divalidasi dan terbukti dapat mengurangi masalah dengan beban pesan yang tinggi dalam sistem [22].

Penelitian ini membahas tentang penggunaan docker untuk mengimplementasikan arsitektur microservices, dimana Docker telah menjadi teknologi disruptif yang mengubah cara aplikasi dikembangkan dan didistribusikan. Dengan banyak keunggulan, Docker sangat cocok untuk menerapkan arsitektur layanan mikro. Dalam makalah ini kita telah membahas beberapa konsep tentang arsitektur layanan mikro dan bagaimana Docker dapat membantu menerapkan gaya arsitektur yang berguna dengan studi kasus yang berfungsi dengan baik. Meskipun Docker sendiri bukanlah solusi terbaik yang dapat mengatasi setiap tantangan dalam membangun arsitektur layanan mikro, namun seiring dengan alat-alat yang ada di sekitarnya, Docker akan sangat membantu dalam meningkatkan efisiensi, otomatisasi, dan hal-hal mendasar lainnya yang diperlukan untuk mencapai prinsip-prinsip terpenting dalam membangun arsitektur layanan mikro [23].

Penelitian ini membahas tentang merancang seperangkat solusi kontrol akses, yang dapat memenuhi persyaratan otentikasi terpadu, meningkatkan efisiensi verifikasi otoritas dan mempercepat kecepatan pengembangan sistem di bawah arsitektur layanan mikro, yang memiliki referensi untuk pentingnya desain sistem aplikasi perusahaan . Berdasarkan analisis persyaratan kontrol akses sumber daya dalam arsitektur layanan mikro, diusulkan skema kontrol akses berdasarkan protokol OAuth2. Dalam skema ini, JWT digunakan untuk mewujudkan verifikasi otoritas lokal pada server sumber daya untuk

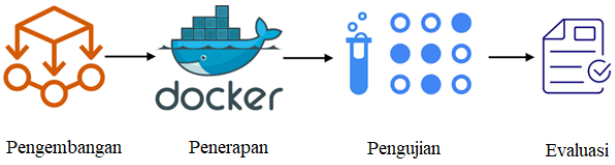
ISSN 2502-3357 (*Online*)
N. A Aprianto dkk./TEKNOLOGI 14 (1) 2024 37-47 ISSN 2503-0477 (*Print*)

meningkatkan efisiensi verifikasi otoritas. Gateway digunakan untuk mewujudkan otentikasi terpadu, yang memudahkan manajemen otentikasi. Modul keamanan bersama dipisahkan untuk memfasilitasi implementasi cepat kontrol akses layanan mikro bisnis [24].

Makalah ini membahas pendatang baru dalam memahami arsitektur layanan mikro, peneliti dan praktisi untuk verifikasi desain konseptual yang berkembang dan prospek masa depan. Arsitektur layanan mikro menawarkan keunggulan dibandingkan berbagai masalah arsitektur yang mengganggu mulai dari monolitik hingga SOA. Meskipun penyebarannya muncul di banyak bidang sistem perangkat lunak, industri dan literatur tidak memiliki konsensus mengenai konsep dan desain arsitektur layanan mikro yang diusulkan secara konseptual. Implementasi praktis dari modul yang diusulkan hadir di industri secara terpisah. Meskipun desain dapat memiliki variasi sesuai dengan penerapan dan persyaratan, namun desain konseptual harus tetap sama. Upaya telah dilakukan untuk mengkonsolidasikan tantangan dan mengusulkan validasi desain konseptual melalui praktik industri dan selanjutnya memerlukan standarisasi di masa depan [25].

3. Metode Penelitian

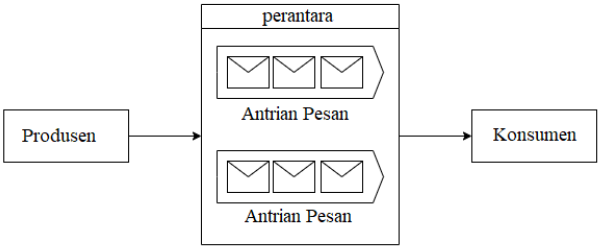
Secara umum proses penelitian akan dibagi menjadi empat sub bab yaitu Developing, Deploying, Testing dan Evaluation



Gambar 1. Proses riset

3.1 Pengembangan

Dalam penelitian ini akan mengembangkan model arsitektur tunggal untuk mengevaluasi kinerja perantara pesan. Keempat model arsitektur tersebut akan dijelaskan pada Gambar 2.



Gambar 2. Model arsitektur

Dari Gambar 2 model arsitektur yang dibuat adalah satu produsen ke satu konsumen. Dimana satu produsen dapat mengirimkan data ke perantara pesan dan satu konsumen dapat membaca data tersebut .

3.2 Deployment

Deployment akan dilakukan setelah proses pengembangan. Penyebaran akan dihosting menggunakan wadah buruh pelabuhan di komputer lokal. Spesifikasi akan dijelaskan pada Tabel I dan Tabel II.

Tabel 1. Spesifikasi komputer

Sistem Operasi	Mac OS monterey version 12.4
RAM	16 GB
SSD	512 GB

Tabel 2I. Spesifikasi Desktop Docker

Versi doker	Docker Desktop 4.9.1
CPU	4 Thread
Memory	3 GB

ISSN 2502-3357 (*Online*)
N. P. Satu dkk./TEKNOLOGI volume (no) Tahun Halaman Awal-Halaman Akhir ISSN 2503-0477 (*Print*)

3.3 Pengujian

Dalam penelitian ini, kinerja perantara pesan akan dievaluasi berdasarkan throughput, latensi dan total waktu yang diperlukan untuk semua parameter transfer. Pengujian dilakukan untuk mengetahui banyaknya data yang akan dikirim dalam satuan waktu. variasi waktu yang digunakan dalam penelitian ini adalah 1, 10, 100, 1000 dan 10000, 100000 data. Pengujian akan dilakukan terhadap model arsitektur yang telah dibuat menggunakan RabbitMQ dan NATS Streaming. Untuk mengukur kinerja pesan ada tiga parameter aspek yang akan diuji:

a. Hasil

   Throughput diukur sebagai jumlah total pesan yang dihasilkan atau dikonsumsi per detik. Percobaan akan dilakukan dengan menghasilkan dan mengkonsumsi ukuran pesan yang berbeda dari 100 byte per detik.

b. Latensi berdasarkan Jumlah Pesan

   Latensi diukur dengan berapa lama suatu pesan dapat diproduksi dan dikonsumsi.

c. Latensi berdasarkan Ukuran Pesan
   Latensi menurut ukuran pesan adalah rata-rata latensi dalam mili detik untuk total data yang didistribusikan menurut ukuran pesan sebesar 100 byte.

3.4 Evaluasi

Kinerja perantara pesan akan dievaluasi dengan beberapa parameter di atas. Setiap parameter akan dihitung menggunakan broker pesan yang berbeda.

4. Hasil dan Pembahasan

4.1 Pembahasan

Dalam studi ini, kami mengevaluasi kinerja RabbitMQ dan NATS Streaming untuk perantara pesan. Dalam penelitian ini menggunakan ukuran data 1, 10, 100, 1000, 10000. Berikut hasil pengujian variasi data.

1. Latensi berdasarkan Jumlah Pesan

Parameter pengujian pertama adalah latensi berdasarkan nomor pesan. Pada pengujian ini dilakukan dengan variasi data 1, 10, 100, 1000 dan 10000 menggunakan software RabbitMQ dan NATS Streaming untuk mengetahui latency berdasarkan jumlah pesan yang dibutuhkan dalam melakukan perantara pesan. Gambar 3 sedang menguji parameter latency by number message menggunakan 1 data pada RabbitMQ. Dari hasil tersebut diketahui bahwa untuk mengirim 1 data dibutuhkan rata-rata latency dengan jumlah pesan sebesar 2.50900 ms. Pengujian ini akan dilakukan pada semua variasi data menggunakan RabbitMQ dan NATS Streaming.

```
2022/12/01 10:10:34 Begin rabbitmq test
2022/12/01 10:10:34 Sent 1 messages in 0.042000 ms
2022/12/01 10:10:34 Sent 23809.523438 per second
2022/12/01 10:10:34 Mean latency for 1 messages: 2.509000 ms
2022/12/01 10:10:34 End rabbitmq test
```

Gambar 3. Pengujian Latensi dengan Pesan Nomor di RabbitMQ

Langkah selanjutnya adalah menguji latensi berdasarkan parameter pesan nomor menggunakan NATS Streaming. Gambar 4 menunjukkan bahwa pengujian latensi berdasarkan parameter pesan angka dengan 1 data memerlukan waktu 1.820000 ms.

```
2022/12/01 10:21:39 Begin nats test
2022/12/01 10:21:39 Sent 1 messages in 0.006000 ms
2022/12/01 10:21:39 Sent 166666.671875 per second
2022/12/01 10:21:39 Mean latency for 1 messages: 1.820000 ms
2022/12/01 10:21:39 End nats test
```

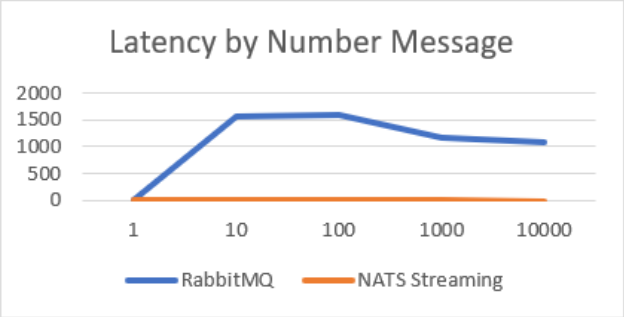
Gambar 1. Menguji Latensi berdasarkan Pesan Nomor pada Streaming NATS

Hasil pengujian parameter latency by number message secara keseluruhan pada RabbitMQ dan NATS Streaming dengan variasi data 1, 10, 100, 1000 dan 10000 diperoleh hasil seperti pada Tabel 3.

ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)

Tabel 3. Latensi Hasil Tes berdasarkan Pesan Nomor

Nilai data	Latensi berdasarkan Pesan Nomor	
	RabbitMQ	NATS Streaming
1	2.509000	1.820000
10	1577.940186	2.015900
100	1599.461060	1.824140
1000	1168.534546	1.563516
10000	1087.672241	1.352136



Gambar 2. Diagram Latensi berdasarkan Pesan Nomor (ms)

Dari Gambar 5 diagram latensi berdasarkan nomor pesan menunjukkan bahwa di RabbitMQ latensi berdasarkan nomor pesan terus meningkat seiring dengan jumlah pesan yang dikirim. Namun penggunaan data 1000 dan 10000 mengalami penurunan karena banyaknya pesan yang dikirim. Sedangkan pada NATS Streaming grafiknya tidak stabil, jika menggunakan 10000 data rata-rata latency yang dibutuhkan lebih sedikit yaitu 1.352136.

Dari hasil pengujian latency by number message dapat disimpulkan bahwa parameter latency by number message NATS Streaming dapat bekerja lebih cepat pada data dalam jumlah besar. Sedangkan di RabbitMQ besarnya data yang digunakan akan mempengaruhi latensi yang dibutuhkan untuk mengirimkan data tersebut.

1. Latensi berdasarkan Ukuran Pesan

Parameter pengujian ini adalah latency berdasarkan ukuran pesan pada broker pesan menggunakan RabbitMQ dan NATS Streaming dengan variasi data yang sama yaitu 1, 10, 100, 1000 dan 10000. Pada pengujian ini dilakukan untuk mengetahui waktu yang dibutuhkan untuk mengirim pesan berdasarkan ukuran pesan per 100 byte.

Pada Gambar 6 merupakan hasil pengujian parameter latency berdasarkan ukuran pesan pada penggunaan 1 data menggunakan RabbitMQ. Dari hasil tersebut rata-rata latency untuk 1 pesan adalah 1228847.750000 ms. Pengujian ini dilakukan terhadap seluruh variasi data yang telah ditentukan.

```
2022/12/01 10:31:03 Begin rabbitmq test
2022/12/01 10:31:03 Sent 1 messages in 0.044000 ms
2022/12/01 10:31:03 Sent 22727.273438 per second
2022/12/01 10:31:03 Mean latency for 1 messages: 1228847.750000 ms
2022/12/01 10:31:03 End rabbitmq test
```

Gambar 3. Menguji Latensi berdasarkan Ukuran Pesan di RabbitMQ

Selanjutnya dilakukan pengujian menggunakan NATS Streaming dengan penggunaan 1 data per 100 byte yang dapat dilihat pada Gambar 7. Hasilnya menunjukkan bahwa pengiriman 1 pesan memerlukan latency rata-rata dengan ukuran pesan sebesar 1.650000 ms.

```
2022/12/01 10:28:49 Begin nats test
2022/12/01 10:28:49 Sent 1 messages in 0.004000 ms
2022/12/01 10:28:49 Sent 249999.984375 per second
2022/12/01 10:28:49 Mean latency for 1 messages: 1.650000 ms
2022/12/01 10:28:49 End nats test
```

Gambar 4. Menguji Latensi berdasarkan Ukuran Pesan pada Streaming NATS

Hasil pengujian latency keseluruhan dengan parameter ukuran pesan sebesar 100 byte menggunakan RabbitMQ dan NATS Streaming dengan variasi data 1, 10, 100, 1000 dan 10000 diperoleh hasil seperti pada Tabel 4.

ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)

Tabel 4. Latensi Hasil Tes berdasarkan Ukuran Pesan

Nilai data	Latensi berdasarkan Ukuran Pesan	
	RabbitMQ	NATS Streaming
1	1228847.750000	1.650000
10	1227786.500000	1.518500
100	1218241.875000	1.713840
1000	1213073.250000	1.309811
10000	1210213.875000	1.309649

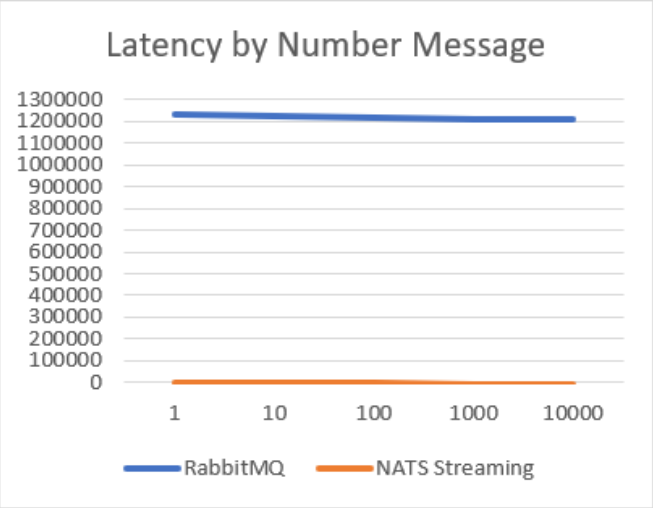


Figure 5. Diagram Latency by Message Size (ms)

Dari Gambar 8 latensi berdasarkan diagram ukuran pesan dengan 100 byte pada RabbitMQ mendapatkan grafik yang semakin menurun seiring bertambahnya jumlah data. Semakin besar jumlah data yang digunakan, semakin lama waktu yang dibutuhkan untuk mengirimkan pesan. Sedangkan NATS Streaming juga mendapatkan rata-rata yang menurun seiring bertambahnya jumlah data yang digunakan.

Dari pengujian latensi berdasarkan ukuran pesan yang telah dilakukan pada RabbitMQ dan NATS Streaming, dapat disimpulkan bahwa NATS Streaming memerlukan waktu yang lebih sedikit untuk mengirimkan sejumlah data sehingga NATS Streaming memiliki kecepatan yang baik dalam pengiriman berapa pun jumlah data yang dikirim. .

1. Hasil

Pada pengujian ini dilakukan untuk mengetahui throughput yang dibutuhkan dalam pengiriman pesan. Pengujian dilakukan menggunakan RabbitMQ dan NATS Streaming pada variasi data 1, 10, 100, 1000 dan 10000.

Pengujian pada 1 pesan menggunakan RabbitMQ memerlukan throughput pengirim 18867.923828 dan pada saat data diterima pada penerima memerlukan throughput +Inf per detik. Hasil pengujian ini dapat dilihat pada Gambar 9.

```
2022/12/01 10:16:22 Begin rabbitmq test
2022/12/01 10:16:22 Sent 1 messages in 0.053000 ms
2022/12/01 10:16:22 Sent 18867.923828 per second
2022/12/01 10:16:22 Received 1 messages in 0.000000 ms
2022/12/01 10:16:22 Received +Inf per second
2022/12/01 10:16:22 End rabbitmq test
```

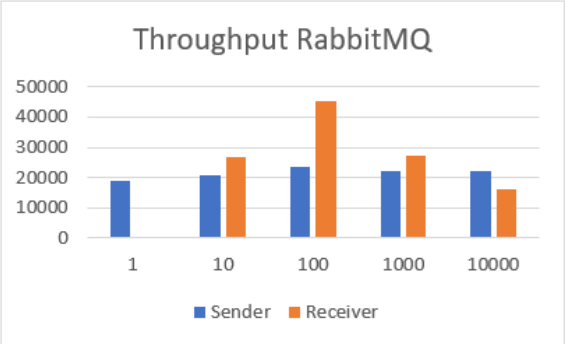
Gambar 6. Pengujian Throughput pada RabbitMQ

Pengujian yang dilakukan pada RabbitMQ terhadap seluruh variasi data yang telah ditentukan dapat dilihat pada Tabel 5.

ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)

Tabel 5. Throughput Hasil Uji pada RabbitMQ

Nilai data	Hasil (pesan/detik)	
	Pengirim	Penerima
1	18867.923828	+Inf per detik
10	20920.501953	26881.720703
100	23601.605469	45085.664062
1000	22327.908203	27278.429688
10000	21952.115234	16278.745117



Gambar 7. Diagram Throughput RabbitMQ (pesan/dtk)

Dari hasil pengujian throughput pada RabbitMQ dapat dilihat pada Gambar 10 Pada RabbitMQ dengan 100 pesan, pesan terbanyak yang dapat diterima per detiknya, namun pesan yang dikirimkan sangat lambat. Pada 10.000 pesan, throughput pengirim mendapat lebih banyak daripada throughput penerima.
Pengujian selanjutnya dilakukan menggunakan NATS Streaming untuk menentukan throughput yang diperlukan saat mengirim dan menerima pesan. Pada Gambar 4.9 menggunakan 1 pesan membutuhkan throughput pengirim sebesar 142857.140625 dan penerima membutuhkan throughput +Inf per detik. Pengujian ini akan dilakukan terhadap seluruh variasi data untuk mengetahui throughput yang dibutuhkan pada saat pengiriman dan penerimaan pesan.

```
2022/12/01 10:23:00 Begin nats test
2022/12/01 10:23:00 Sent 1 messages in 0.007000 ms
2022/12/01 10:23:00 Sent 142857.140625 per second
2022/12/01 10:23:00 Received 1 messages in 0.000000 ms
2022/12/01 10:23:00 Received +Inf per second
2022/12/01 10:23:00 End nats test
```

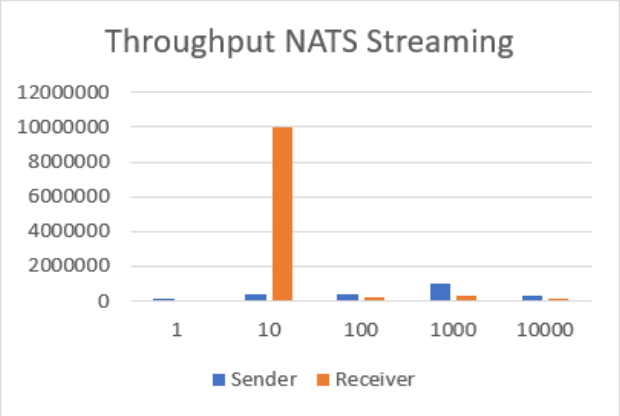
Gambar 8. Menguji Throughput pada Streaming NATS

Hasil pengujian throughput yang diperlukan pada saat mengirim dan menerima pesan dengan berbagai data yang telah digunakan menggunakan NATS Streaming dapat dilihat pada Tabel 6.

Tabel 6II. Throughput Hasil Uji pada NATS Streaming

Nilai data	Hasil (pesan/detik)	
	Pengirim	Penerima
1	142857.140625	+Inf
10	370370.375000	10000000.000000
100	413223.156250	208768.265625
1000	984251.937500	263435.187500
10000	302425.437500	150652.328125

ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)



Gambar 9. Diagram Throughput NATS Streaming (pesan/dtk)

4.2 Hasil

Dari hasil pengujian throughput pada NATS Streaming dapat dilihat pada Gambar 12 At 10 pesan dapat menerima pesan dengan cepat namun pada saat mengirim pesan throughput yang dibutuhkan berbeda jauh dengan throughput penerima. Hasil yang hampir stabil diperoleh pada 10000 pesan karena dapat mengirim pesan terkirim dalam jumlah besar per detik dan menerima pesan dengan cepat.

4.3 Diskusi

Dari hasil penelitian yang telah dilakukan dalam menganalisa performa message broker mana yang lebih baik menggunakan RabbitMQ dan NATS Streaming dengan variasi data 1, 10, 100, 1000 dan 10000 maka dapat dilihat hasil performa kedua software tersebut pada Tabel 7.

Tabel 7. Pesan Perbandingan Kinerja

Nilai data	Latensi berdasarkan Jumlah Pesan		Latensi berdasarkan Ukuran Pesan		Hasil	
	RabbitMQ	NATS Streaming	RabbitMQ	NATS	Sender	Receiver
1	-	✓	-	✓	N	+Inf
10	-	✓	-	✓	R	R
100	-	✓	-	✓	R	N
1000	-	✓	-	✓	R	N
10000	-	✓	-	✓	R	N

Informasi:
N = NATS Streams
R = RabbitMQ

5. Kesimpulan

Dari pengujian yang telah dilakukan untuk mengetahui kinerja RabbitMQ dan NATS Streaming pada broker pesan dapat disimpulkan bahwa.

1. Dalam pengujian latensi RabbitMQ berdasarkan nomor pesan, jumlah data yang digunakan berpengaruh pada perantara pesan. Kemudian, pengujian latensi berdasarkan ukuran pesan menggunakan 10.000 pesan mendapatkan rata-rata latensi yang rendah sehingga RabbitMQ dapat bekerja dengan data dalam

ISSN 2502-3357 (*Online*)
ISSN 2503-0477 (*Print*)

jumlah besar. Sedangkan throughput yang digunakan pada 100 pesan dapat menerima data per detik terbanyak. Namun proses pengiriman datanya lambat dibandingkan dengan penerima.

2. Pada NATS Streaming, pada pengujian latency dengan jumlah pesan pada 10.000 pesan diperoleh rata-rata latency sebesar 1.352136 sehingga banyaknya data yang digunakan tidak mempengaruhi latency yang dibutuhkan pada message broker. Pada parameter latency by message size menggunakan 10000 pesan, rata-rata latency by number pesan semakin berkurang, sehingga semakin besar jumlah data yang digunakan maka semakin sedikit waktu yang dibutuhkan. Sedangkan throughput yang menggunakan 10 pesan dapat menerima pesan dengan cepat, namun pada saat mengirim pesan throughput yang dibutuhkan berbeda jauh dengan throughput penerima. Hasil yang hampir stabil diperoleh pada 10000 pesan karena dapat mengirim pesan terkirim dalam jumlah besar per detik dan menerima pesan dengan cepat.

6. *Declaration of Competing Interest*

Penulis menyatakan tidak ada konflik kepentingan.

7. Referensi

Bashish, D. A., Braik, M., & Ahmad, S. B. (2010). A framework for detection and classification of plant leaf and stem diseases. *Signal and Image Processing (ICSIP), 2010 International Conference on* (hal. 113-118). Chennai: IEEE.

Busin, L., Vandenbroucke, N., & Macaire, L. (2008). Color spaces and image segmentation. *Advances in Imaging and Electron Physics, 151*, 65-168.

Chaudhary, P., Chaudhari, A. K., Cheeran, A. N., & Godara, S. (2012). Color transform based approach for disease spot detection on plant leaf. *International Journal of Computer Science and Telecommunications, 3*(6), 65-70.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory, 13*(1), 21-27.

Haralick, R. M., Shanmugam, K., & Dinstein, I. (1973). Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics, 3*(6), 610-621.

Huang, K.-Y. (2007). Application of artificial neural network for detecting Phalaenopsis seedling diseases using color and texture features. *Computers and Electronics in Agriculture, 57*(1), 3–11.

Kadir, A., Nugroho, L. E., Susanto, A., & Santosa, P. I. (2013). Leaf classification using shape, color, and texture features. *International Journal of Computer Trends and Technology*, 225-230.

Kusuma, A. P., & Darmanto. (2016). Pengenalan angka pada sistem operasi android dengan menggunakan metode template matching. *Register: Jurnal Ilmiah Teknologi Sistem Informasi, 2*(2), 68-78.

Mendoza, F., Dejmek, P., & Aguilera, J. M. (2006). Calibrated color measurements of agricultural foods using image analysis. *Postharvest Biology and Technology, 41*(3), 285–295.

Meunkaewjinda, A., Kumsawat, P., Attakitmongcol, K., & Srikaew, A. (2008). Grape leaf disease detection from color imagery using hybrid intelligent system. *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2008. ECTI-CON 2008. 5th International Conference on* (hal. 513-516). Krabi: IEEE.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics, 9*(1), 62-66.

ISSN 2502-3357 (*Online*)
N. P. Satu dkk./TEKNOLOGI volume (no) Tahun Halaman Awal-Halaman Akhir ISSN 2503-0477 (*Print*)

Rathod, A. N., Tanawal, B., & Shah, V. (2013). Image processing techniques for detection of leaf disease. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(11), 397-399.

Ratnasari, E. K., Ginardi, R. V., & Fatichah, C. (2014). Pengenalan penyakit noda pada citra daun tebu berdasarkan ciri tekstur fractal dimension co-occurrence matrix dan L*a*b* color moments. *JUTI, 12*(2), 27– 36.

Rott, P. (2000). *A guide to sugarcane diseases.* Paris: Quae.

Sa'diyah, N., & Aeny, T. N. (2012). Keragaman dan heritabilitas ketahanan tebu populasi F1 terhadap penyakit bercak kuning di PT. Gunung Madu Plantations Lampung. *Jurnal Hama dan Penyakit Tumbuhan Tropika, 12*(1), 71-77.

Sungkur, R. K., Baichoo, S., & Poligadu, A. (2013). An automated system to recognise fungi-caused diseases on sugarcane leaves. *Proceedings of Global Engineering, Science and Technology Conference.* Bencoolen, Singapura: Global Institute of Science & Technology.

Vibhute, A., & Bodhe, S. K. (2012). Applications of image processing in agriculture: A survey. *International Journal of Computer Applications, 52*(2), 34-40.