OPEN ACCESS

Volume 10 (4), October 2026, 1444-1452

E-ISSN:2580-1643

Jurnal JTIK (Jurnal Teknologi Informasi dan Komunikasi)

DOI: https://doi.org/10.35870/jtik.v10i4.6709

Analisis Performa Arsitektur *Microservices* pada Platform *E-Commerce* Waltmart Menggunakan *Protocol Buffers*, *Redis Caching*, *Message Queue*, dan *Kubernetes*

Dennise Adrianto [1*], Herman [2], William Chang [3], Richard Delbert Tannady [4]

[1*] *Computer Science Department, Binus Online Learning, Bina Nusantara University, Kota Jakarta Barat, Daerah Khusus Ibukota Jakarta, Indonesia.*

[2,3,4] *Computer Science Department, School of Computer Science, Bina Nusantara University, Kota Jakarta Barat, Daerah Khusus Ibukota Jakarta, Indonesia.*

a r t i c l e i n f o

Article history:
Received 6 February 2026
Received in revised form 3 March 2026
Accepted 25 April 2026
Available online October 2026.

Keywords:
Microservices Architecture; Scalability; Protocol Buffers.

Kata Kunci:
Arsitektur Microservices; Skalabilitas; Protocol Buffers.

a b s t r a c t

Modern e-commerce platforms are required to handle increasing transaction volumes and user traffic while maintaining high performance and scalability. This study evaluates the implementation of a microservices architecture on the Waltmart e-commerce platform to improve performance, scalability, and system reliability. The architecture is designed using independent services supported by Protocol Buffers for efficient inter-service communication, Redis caching for faster data retrieval, and Message Queue (MQ) mechanisms for asynchronous processing. Kubernetes is applied to manage container orchestration and autoscaling under varying traffic loads. Performance evaluation is conducted through load testing by measuring serialization time, response time, and system throughput. The results show that Protocol Buffers reduce serialization and deserialization time by more than 50% compared to JSON, while Redis caching significantly lowers response times and increases throughput. Although the performance improvement from MQ is relatively modest, it enhances system stability during high-volume transactions.

a b s t r a k

Penelitian ini mengevaluasi penerapan arsitektur microservices pada platform e-commerce Waltmart untuk meningkatkan performa, skalabilitas, dan keandalan sistem. Platform e-commerce modern dituntut mampu menangani peningkatan volume transaksi dan trafik pengguna yang tinggi secara efisien. Arsitektur dirancang menggunakan layanan independen yang didukung Protocol Buffers untuk komunikasi antarlayanan yang efisien, Redis caching untuk mempercepat pengambilan data, serta mekanisme Message Queue (MQ) untuk pemrosesan asynchronous. Kubernetes digunakan untuk mengelola orkestrasi container dan autoscaling pada kondisi beban trafik yang beragam. Evaluasi performa dilakukan melalui pengujian load testing dengan mengukur waktu serialisasi, response time, dan throughput sistem. Hasil pengujian menunjukkan bahwa Protocol Buffers mampu mengurangi waktu serialisasi dan deserialisasi lebih dari 50% dibandingkan JSON, sementara Redis caching menurunkan waktu respons dan meningkatkan throughput secara signifikan. Meskipun peningkatan MQ relatif kecil, mekanisme ini meningkatkan stabilitas saat transaksi tinggi.

Corresponding Author. Email: dennise@binus.ac.id [1].

1. Pendahuluan

Saat ini, *platform e-commerce* dituntut untuk memiliki performa tinggi, reliabilitas yang konsisten, dan kemampuan skalabilitas untuk menangani lonjakan trafik dan transaksi dalam jumlah besar. Pada arsitektur monolitik, semua fungsi aplikasi digabungkan dalam satu kesatuan, yang dapat menyebabkan bottleneck, proses pemeliharaan yang sulit, dan risiko kegagalan sistem secara menyeluruh ketika beban meningkat (Kleppmann, 2017; Newman, 2015). Pendekatan arsitektur *microservices* kemudian berkembang sebagai salah satu solusi. Arsitektur *microservices* diterapkan dengan memecah aplikasi menjadi layanan-layanan kecil yang dapat dikembangkan, diuji, dan di-deploy secara independen (Guntakandla, 2025). Hal ini memberikan fleksibilitas teknologi, peningkatan isolasi kesalahan, serta kemampuan skalabilitas yang lebih adaptif terhadap kebutuhan bisnis (Sharma, 2025; Oyeniran *et al.*, 2024). Penelitian mengenai sistem informasi dan pengembangan aplikasi e-commerce di Indonesia menunjukkan kebutuhan nyata terhadap arsitektur yang skalabel dan efisien. Penelitian-penelitian tersebut menekankan pentingnya modularitas, efisiensi pemrosesan, dan desain basis data yang terstruktur untuk mendukung proses transaksi digital (Setiadi *et al.*, 2022; Sugiarto & Triandini, 2022; Wati *et al.*, 2025).

Dalam arsitektur *microservices*, komunikasi antar layanan menjadi komponen penting yang harus efisien dan berlatensi rendah. Berbagai penelitian menunjukkan bahwa *Protocol Buffers* menawarkan kinerja lebih tinggi dibandingkan *JSON*. Hal ini terlihat dari sisi ukuran payload, kecepatan serialisasi, dan efisiensi energi, sehingga ideal digunakan untuk komunikasi internal layanan berskala besar (Shatnawi *et al.*, 2025). Di sisi lain, optimasi performa layanan juga dapat dicapai dengan penerapan proses *Redis caching* (Privalov & Stupina, 2024), yang terbukti meningkatkan throughput dan menurunkan latensi pada aplikasi web dinamis serta layanan terdistribusi. Studi yang dilakukan oleh Dipraja dan Rahman menunjukkan peningkatan signifikan pada efisiensi caching berbasis *Redis Cluster* (Dipraja & Rahman, 2025), sementara penelitian lainnya menemukan penurunan waktu respons aplikasi *PHP* lebih dari 70% setelah *Redis caching* diterapkan (Suwarjono & Averoes, 2025). Pada proses asinkron, penggunaan *Message Queue* (MQ) menjadi mekanisme penting karena mampu mengurangi beban sinkron, menstabilkan proses transaksi, serta meningkatkan keandalan pemrosesan. Penelitian sebelumnya pada pengembangan *REST API* dan deployment multiplatform menunjukkan bahwa MQ mampu menjaga konsistensi operasi dan menghindari blocking pada layanan inti (Choirudin & Adil, 2019; Putra *et al.*, 2025). Untuk orkestrasi dan pengelolaan layanan, aplikasi e-commerce *Walmart* yang dikembangkan menggunakan *Kubernetes* digunakan terkait autoscaling, manajemen sumber daya, dan efisiensi kontainer. Namun, penggunaan *Kubernetes* ini tetap akan menghadapi tantangan pada penjadwalan, overhead di komponen datastore, serta kebutuhan adaptasi untuk lingkungan edge (Jeffery *et al.*, 2021; Rejiba & Chamanara, 2023). Di sisi lain, pemodelan performa *microservices* dalam konteks *Kubernetes* dapat memberikan pendekatan analitis untuk proses estimasi kapasitas layanan dan mengidentifikasi bottleneck sebelum dan sesudah deployment (Jindal *et al.*, 2019). Meskipun berbagai penelitian telah membahas penerapan arsitektur *microservices*, *Redis caching*, serta mekanisme *Message Queue*, sebagian besar penelitian tersebut hanya mengevaluasi teknologi tersebut secara terpisah. Penelitian sebelumnya umumnya berfokus pada konsep arsitektur atau pengujian performa pada satu komponen sistem saja. Masih terdapat keterbatasan penelitian yang mengevaluasi integrasi beberapa teknologi sekaligus seperti *Protocol Buffers* untuk komunikasi data, *Redis caching* untuk optimasi pengambilan data, *Message Queue* untuk pemrosesan asinkron, serta orkestrasi kontainer menggunakan *Kubernetes* dalam satu platform e-commerce yang terintegrasi. Oleh karena itu, penelitian ini bertujuan untuk mengevaluasi efektivitas penerapan arsitektur *microservices* pada platform e-commerce *Walmart* dengan mengintegrasikan teknologi *Protocol Buffers*, *Redis caching*, *Message Queue*, serta *Kubernetes* untuk meningkatkan performa, skalabilitas, dan keandalan sistem.

2. Metode Penelitian

Penelitian ini menggunakan pendekatan rekayasa perangkat lunak berbasis *Domain-Driven Design* (DDD)

untuk memetakan kebutuhan dari platform e-commerce *Walmart* ke dalam enam proses utama yaitu *Product*, *Search*, *User*, *Promo*, *Transaction*, dan *Payment*. Setiap konteks kemudian direalisasikan sebagai *microservice* independen. Tahapan penelitian yang dilakukan adalah sebagai berikut:

Analisis Sistem dan Kebutuhan

Tahap awal dalam proses penelitian ini dilakukan dengan melakukan analisis proses bisnis e-commerce *Walmart* dan identifikasi terkait fitur penting yang dibutuhkan oleh tiga jenis pengguna: *customer*, *merchant*, dan *admin*. Kendala utama yang dihadapi meliputi lambatnya pengolahan data, keterbatasan pengelolaan transaksi, dan kebutuhan akan modularitas.

Perancangan Arsitektur Microservices

Arsitektur *microservices* akan dirancang berbasis *API Gateway* untuk dapat melakukan komunikasi internal dengan menggunakan *REST + Protocol Buffers*. Komponen pendukung seperti *Redis* untuk caching dan *message queue* diintegrasikan dalam sistem ini agar dapat mendukung proses asinkron.

Pemodelan Sistem

Pemodelan sistem akan dilakukan dengan menggunakan *UML* (use case, sequence, dan class diagram) agar dapat memetakan alur proses. Dalam hal ini, setiap *microservice* akan memiliki basis data terpisah yang sesuai dengan prinsip basis data per layanan.

Implementasi Sistem

Setelah semua perancangan telah dibuat, maka akan dilakukan tahap pengembangan aplikasi dengan metode *Waterfall* sesuai dengan requirement yang telah ditentukan di awal. Setiap layanan akan dikembangkan dalam kontainer dengan orkestrasi *Kubernetes* untuk mendukung horizontal pod autoscaling. Sistem *Walmart* diimplementasikan menggunakan layanan cloud *Amazon Web Services* (AWS). Infrastruktur sistem terdiri dari tujuh instance server yang digunakan untuk menjalankan masing-masing *microservice* serta satu instance tambahan yang digunakan untuk menjalankan sistem *message queue*. Setiap instance *microservice* menggunakan konfigurasi t2.medium dengan spesifikasi 2 vCPU dan 4 GB RAM, sedangkan instance untuk *message queue* menggunakan konfigurasi t2.micro dengan spesifikasi 1 vCPU dan 1 GB RAM. Basis data sistem menggunakan konfigurasi db.t3.micro dengan kapasitas memori 2 GB. Infrastruktur ini memungkinkan sistem untuk melakukan scaling secara horizontal apabila terjadi peningkatan jumlah pengguna.

Pengujian Blackbox dan Load Testing

Pada proses pengujian, akan dilakukan proses untuk mengukur waktu respons, throughput, dan penggunaan sumber daya. Pengujian performa sistem dilakukan melalui beberapa skenario pengujian utama yaitu:

1) Perbandingan performa komunikasi data antara *JSON* dan *Protocol Buffers*, untuk mengukur efisiensi proses serialisasi dan deserialisasi data.
2) Perbandingan performa sistem dengan dan tanpa *Redis caching*, untuk mengevaluasi pengaruh caching terhadap waktu respons dan throughput sistem.
3) Perbandingan performa sistem dengan dan tanpa *Message Queue*, untuk mengevaluasi pengaruh pemrosesan asinkron terhadap stabilitas dan performa sistem.

Pengujian serialisasi data antara *JSON* dan *Protocol Buffers* dilakukan dengan memproses sebanyak 5 juta data dengan ukuran rata-rata 473 byte untuk setiap data. Pengujian ini dilakukan menggunakan perangkat dengan spesifikasi Intel Core i7-8750H, RAM 8 GB, dan sistem operasi Windows 10 64-bit. Evaluasi *UI/UX* juga dilakukan dengan 8 aturan emas *Shneiderman*.

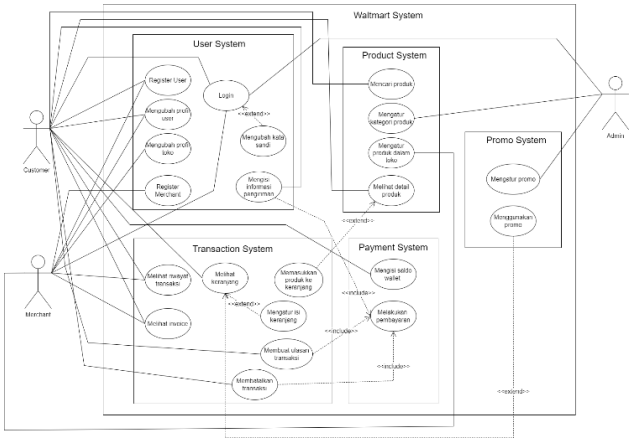
Analisis dan Dokumentasi

Tahap terakhir dalam penelitian ini adalah melakukan analisis terhadap hasil pengujian yang telah dilakukan. Hasil pengujian dianalisis untuk mengetahui seberapa besar peningkatan performa sistem setelah penerapan arsitektur *microservices* serta penggunaan teknologi pendukung seperti *Protocol Buffers*, *Redis caching*, dan *Message Queue*. Hasil analisis tersebut kemudian digunakan untuk mengevaluasi efektivitas penerapan arsitektur *microservices* pada platform *Walmart* dalam meningkatkan performa, skalabilitas, dan stabilitas sistem e-commerce.

3. Hasil dan Pembahasan

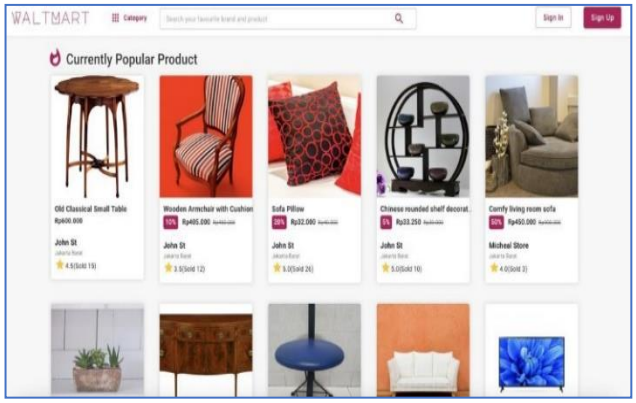
Hasil

Platform e-commerse Waltmart akhirnya dapat dibangun dengan menggunakan arsitektur microservices yang memisahkan domain layanan menjadi beberapa service independen seperti Product, Search, User, Promo, Transaction, dan Payment. Seperti yang dapat terlihat pada Gambar 1. berikut.



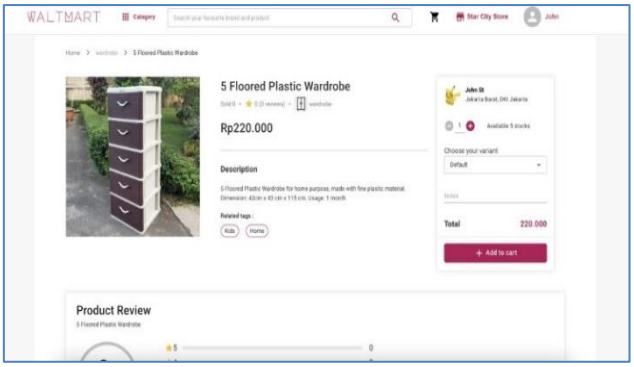
Gambar 1. Use Case Diagram Aplikasi Waltmart

Setiap service dari platform Waltmart berjalan pada container terisolasi dan berkomunikasi menggunakan REST API dengan dua format data: JSON dan Protocol Buffers. Dalam proses pengembangan, aplikasi Waltmart dapat diselesaikan dengan baik dengan fitur-fitur yang disebutkan, yaitu diantaranya adalah halaman produk seperti yang dilihat di Gambar 2. Dimana user dapat melihat keseluruhan produk yang ditawarkan dalam e-commerce ini.

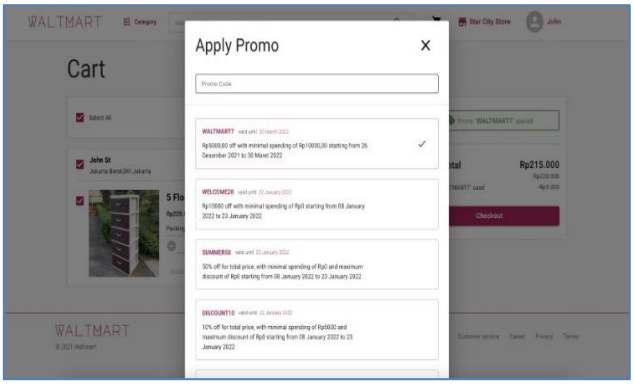


Gambar 2. Halaman Produk Aplikasi Waltmart

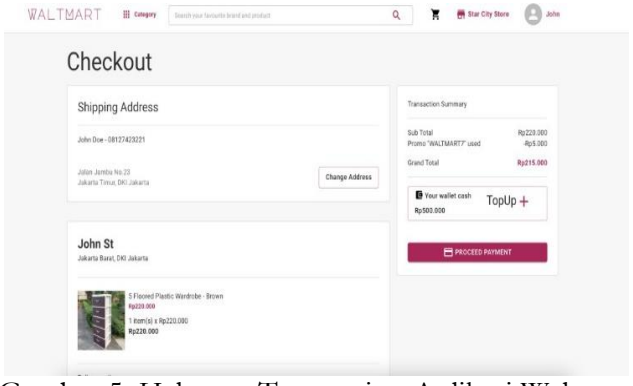
Kemudian fitur-fitur utama dalam aplikasi ini dapat dilihat yaitu utamanya adalah fitur pencarian produk yang dapat diaplikasikan dengan memberikan kata kunci pada bagian header aplikasi. Dengan memberikan kata kunci, aplikasi akan mencari produk yang sesuai, seperti yang terlihat pada Gambar 3. Beberapa fitur lain juga dapat dilihat seperti halaman promo, pada Gambar 4. Halaman transaksi yaitu pada Gambar 5. Dan halaman pembayaran seperti yang terlihat pada Gambar 6.



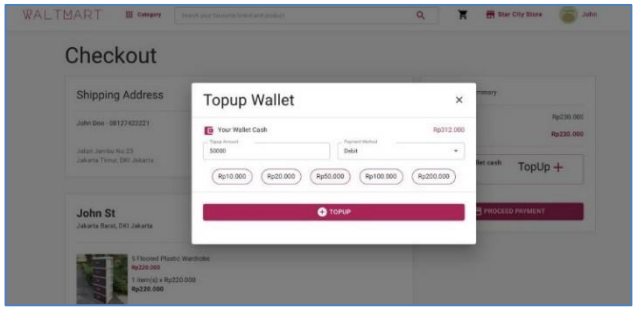
Gambar 3. Halaman Search Aplikasi Waltmart



Gambar 4. Halaman Promo Aplikasi Waltmart



Gambar 5. Halaman Transaction Aplikasi Waltmart



Gambar 6. Halaman Payment Aplikasi Waltmart

Dalam hal peningkatan performa pemanggilan data yang sering digunakan, *Redis caching* diterapkan pada layer tertentu, sementara *Message Queue* (MQ) diterapkan dalam melakukan proses pada bagian proses bisnis yang bersifat asinkron, yang diterapkan dengan melakukan implementasi antrian internal sederhana sesuai kebutuhan fungsi sistem. Aplikasi kemudian di-deploy menggunakan kontainer yang dikelola dengan *Kubernetes* sehingga mendukung pengaturan layanan, load balancing, dan autoscaling. Secara keseluruhan, seluruh fungsi inti seperti transaksi, checkout, pengelolaan produk, ulasan, dan pembuatan promo dapat berjalan dengan baik dan stabil. Pengujian dari segi sistem dilakukan dengan melakukan *blackbox testing* dan *load testing*. Tabel 1. di bawah ini menunjukkan hasil dari *blackbox testing* terhadap platform e-commerce *Walmart*.

Tabel 1. Hasil Pengujian Blackbox Testing

Fitur	Hasil
Login	Lulus pengecekan
Register user	Lulus pengecekan
Mengirimkan *email* pembaharuan *password*	Lulus pengecekan
Memperbaharui *password* dari tautan di *email* yang dikirimkan	Lulus pengecekan
Menampilkan rekomendasi produk pada halaman utama	Lulus pengecekan
Menampilkan rekomendasi produk untuk *customer*	Lulus pengecekan
Menampilkan kategori pada menu pencarian	Lulus pengecekan
Mencari produk berdasarkan kategori	Lulus pengecekan
Menampilkan rekomendasi *tag* dan nama produk pada pencarian	Lulus pengecekan
Mencari produk berdasarkan *tag* dan nama produk	Lulus pengecekan
Mengubah *tag* pada hasil pencarian dan mengubah hasil pencarian	Lulus pengecekan
Menampilkan detail produk beserta ulasannya	Lulus pengecekan
Menambahkan produk ke keranjang dengan catatan	Lulus pengecekan
Menampilkan profil dan produk toko	Lulus pengecekan
Menampilkan data pengguna di halaman profil	Lulus pengecekan
Memperbaharui *password* di halaman profil	Lulus pengecekan
Memperbaharui data pengguna	Lulus pengecekan
Menambah dan mengubah alamat pengiriman	Lulus pengecekan
Menampilkan riwayat *wallet*	Lulus pengecekan
Mencari riwayat *wallet*	Lulus pengecekan
Melakukan *top up*	Lulus pengecekan
Melihat keranjang	Lulus pengecekan
Mengubah isi keranjang	Lulus pengecekan
Menggunakan promo pada halaman keranjang	Lulus pengecekan
Menampilkan dan mencari promo yang berlaku	Lulus pengecekan
Menampilkan alamat pada halaman *checkout*	Lulus pengecekan
Mengubah alamat pada halaman *checkout*	Lulus pengecekan
Menggunakan dan menampilkan opsi pengiriman	Lulus pengecekan
Melakukan *top up* halaman *checkout*	Lulus pengecekan
Melakukan *checkout*	Lulus pengecekan

Mengirim *email* setelah *checkout*	Lulus pengecekan
Menampilkan, mencari dan melihat riwayat transaksi	Lulus pengecekan
Melihat status pengiriman dan detail transaksi	Lulus pengecekan
Membatalkan transaksi saat produk belum dikirim	Lulus pengecekan
Menyelesaikan transaksi saat produk sudah sampai	Lulus pengecekan
Mengulas produk saat transaksi sudah diselesaikan	Lulus pengecekan
Melakukan registrasi sebagai *merchant*	Lulus pengecekan
Melihat dan mengubah profil toko	Lulus pengecekan
Mengatur produk pada toko	Lulus pengecekan
Menerapkan diskon pada beberapa produk dalam toko	Lulus pengecekan
Melihat riwayat transaksi pada toko	Lulus pengecekan
Mengirim transaksi yang masuk pada toko	Lulus pengecekan
Melihat detail transaksi pada toko	Lulus pengecekan
Mengatur promo yang berlaku	Lulus pengecekan
Mengatur kategori	Lulus pengecekan

Kemudian di bawah ini merupakan hasil *load testing* sistem di berbagai bidang yaitu:

1) Uji Format *Protobuf* dengan *JSON*

Uji format *Protobuf* dan *JSON* dilakukan dengan cara meng-serialize/deserialize data sebanyak 5 juta dengan ukuran 473-byte untuk setiap data, seperti terlihat pada Tabel 2. berikut:

Tabel 2. Hasil Perbandingan Pengujian JSON dan Protobuf

Jenis Pengujian	JSON	Protobuf
Serialize berdasarkan jumlah data	33.905,3 ms	17.621 ms
Deserialize berdasarkan jumlah data	30.799,6 ms	16.055,3 ms

Jika dilihat dari hasil uji pada Tabel 2, maka dapat disimpulkan bahwa penggunaan *Protobuf* mempercepat proses komputasi dengan persentase 52% lebih cepat jika dibandingkan dengan *JSON*. Perbedaan performa ini disebabkan oleh mekanisme serialisasi *Protocol Buffers* yang menggunakan format data biner sehingga menghasilkan ukuran payload yang lebih kecil dibandingkan dengan *JSON*. Selain itu, proses encoding dan decoding pada *Protocol Buffers* juga lebih efisien karena tidak memerlukan proses parsing teks seperti pada *JSON*. Oleh karena itu, penggunaan *Protocol Buffers* sangat cocok digunakan dalam komunikasi antar layanan pada arsitektur *microservices* yang membutuhkan latensi rendah dan efisiensi pertukaran data.

2) Uji dengan Cache atau Tanpa Cache

Uji dengan cache atau tanpa cache menunjukkan perbedaan kecepatan secepat 52% di mana penggunaan cache lebih cepat jika dibandingkan dengan proses tanpa penggunaan cache. Hasil ini dapat dilihat pada Tabel 3. berikut.

Tabel 3. Hasil Perbandingan Pengujian Tanpa Caching dan dengan Caching

Keterangan	Tanpa *Caching*	Dengan *Caching*
Rata-rata *response time*	563 ms	291 ms
Jumlah *request* yang dapat ditangani	26663 *request*	51464 *request*

Hasil pengujian menunjukkan bahwa penggunaan *Redis caching* mampu menurunkan waktu respons sistem secara signifikan. *Redis caching* bekerja dengan menyimpan data yang sering diakses pada memori sehingga proses pengambilan data tidak selalu harus dilakukan melalui query ke basis data utama. Dengan adanya mekanisme caching ini, sistem dapat merespon permintaan pengguna dengan lebih cepat

serta mampu menangani jumlah *request* yang lebih besar dalam waktu yang sama. Hal ini menunjukkan bahwa penggunaan caching merupakan salah satu teknik optimasi yang efektif dalam meningkatkan performa aplikasi e-commerce.

3) Uji Perbandingan dengan *Message Queue* atau Tanpa *Message Queue*

Uji perbandingan dengan *Message Queue* atau tanpa *Message Queue* menunjukkan penggunaan *message queue* lebih cepat 0.99% jika dibandingkan dengan proses yang tidak menggunakan *message queue*. Dapat dilihat pada Tabel 4. berikut.

Tabel 4. Hasil Perbandingan pengujian tanpa Message Queue dan dengan Message Queue

Keterangan	Tanpa *Message Queue* (Percobaan 1)	Dengan *Message Queue* (Percobaan 1)	Tanpa *Message Queue* (Percobaan 2)	Dengan *Message Queue* (Percobaan 2)
Rata-rata *response time*	386 ms	368 ms	392 ms	408 ms
Jumlah *request* yang dapat ditangani	39090 *request*	40952 *request*	38540 *request*	36900 request
Jumlah data transaksi dalam *database*	± 12 ribu	± 32 ribu	± 51 ribu	± 70 ribu

Hasil pengujian menunjukkan bahwa penggunaan *Message Queue* memberikan peningkatan performa yang relatif kecil dibandingkan pengujian lainnya. Pada percobaan kedua terlihat bahwa waktu respons sistem dengan *Message Queue* sedikit lebih lambat dibandingkan tanpa *Message Queue*. Hal ini dapat disebabkan oleh adanya overhead tambahan pada proses pengiriman dan pengambilan pesan dalam sistem antrian. Meskipun demikian, penggunaan *Message Queue* tetap memberikan manfaat penting dalam meningkatkan stabilitas sistem karena proses transaksi dapat diproses secara asinkron sehingga mengurangi kemungkinan terjadinya blocking pada layanan utama. Selain itu, *Message Queue* juga memungkinkan sistem untuk menangani lonjakan transaksi secara lebih stabil karena pesan dapat diproses secara bertahap oleh layanan yang tersedia.
Evaluasi antarmuka pengguna dilakukan dengan menggunakan pendekatan *Eight Golden Rules of Interface Design* yang dikemukakan oleh *Shneiderman*. Prinsip ini digunakan untuk mengevaluasi kualitas desain antarmuka aplikasi *Walmart* dari segi kemudahan penggunaan serta pengalaman pengguna.

Pembahasan

Pembahasan ini bertujuan untuk mengevaluasi penerapan arsitektur *microservices* pada platform e-commerce *Walmart* dengan fokus pada penggunaan *Redis caching* dan *Message Queue* (MQ) untuk meningkatkan performa sistem. Hasil pengujian menunjukkan bahwa penerapan *Redis caching* secara signifikan menurunkan waktu respons sistem, memungkinkan pengambilan data yang sering diakses dilakukan dengan lebih cepat. Hal ini sejalan dengan penelitian yang dilakukan oleh Privalov dan Stupina (2024), yang menekankan bahwa *Redis caching* dapat meningkatkan throughput dan menurunkan latensi pada aplikasi web dinamis. Dengan menyimpan data dalam memori, sistem tidak perlu selalu melakukan query ke basis data utama, yang berkontribusi pada efisiensi operasional. Di sisi lain, penerapan *Message Queue* menunjukkan peningkatan performa yang relatif kecil dibandingkan dengan pengujian lainnya, dengan waktu respons yang sedikit lebih lambat pada percobaan kedua. Penelitian oleh Choirudin dan Adil (2019) mendukung temuan ini, yang menyatakan bahwa meskipun *Message Queue* dapat menambah overhead pada proses pengiriman dan pengambilan pesan, mekanisme ini tetap penting dalam meningkatkan stabilitas sistem. Proses transaksi yang diproses secara asinkron mengurangi kemungkinan terjadinya blocking pada layanan utama, memungkinkan sistem untuk menangani lonjakan transaksi dengan lebih stabil. Evaluasi antarmuka pengguna yang dilakukan dengan pendekatan *Eight Golden Rules of Interface Design* oleh *Shneiderman* juga menunjukkan bahwa desain antarmuka aplikasi *Walmart* telah mempertimbangkan kemudahan penggunaan dan pengalaman pengguna. Dengan mengintegrasikan teknologi seperti *Protocol Buffers*, *Redis caching*, dan *Message Queue*, penelitian ini menunjukkan bahwa arsitektur *microservices* dapat

memberikan solusi yang efisien dan scalable untuk kebutuhan e-commerce modern. Secara keseluruhan, penerapan arsitektur *microservices* pada platform e-commerce *Walmart* tidak hanya meningkatkan performa dan responsivitas, tetapi juga memastikan stabilitas sistem dalam menghadapi beban transaksi yang tinggi. Penelitian ini memberikan kontribusi penting dalam memahami bagaimana integrasi berbagai teknologi dapat memperkuat infrastruktur e-commerce, serta membuka jalan untuk penelitian lebih lanjut dalam pengembangan sistem yang lebih kompleks.

4. Kesimpulan

Penelitian ini menunjukkan bahwa penerapan arsitektur *microservices* pada platform e-commerce *Walmart* memberikan peningkatan yang signifikan terhadap performa, stabilitas, dan skalabilitas sistem. Proses pemecahan layanan dalam platform e-commerce *Walmart* menjadi enam domain independen memungkinkan pengembangan dan deployment dilakukan secara terpisah tanpa mengganggu layanan lainnya. Hasil pengujian performa membuktikan bahwa penggunaan *Protocol Buffers* mampu meningkatkan efisiensi hingga 52% dibandingkan dengan penggunaan *JSON*. Hal ini menjadikan format *Protocol Buffers* lebih sesuai untuk komunikasi antar layanan dengan intensitas data tinggi. Integrasi *Redis caching* juga memiliki dampak besar terhadap peningkatan performa, terlihat dari penurunan waktu respons sistem hampir setengahnya dan peningkatan throughput server hingga 52%. Di sisi lain, penerapan *Message Queue* (MQ) juga dirasakan memiliki peran penting dalam platform e-commerce *Walmart*. Meskipun penerapan MQ hanya menghasilkan peningkatan kinerja sekitar 1%, mekanisme MQ tetap berperan penting dalam menjaga stabilitas proses asinkron, khususnya dalam mengurangi beban pada *Transaction Service*. Hasil *load testing* memperlihatkan bahwa *Kubernetes Horizontal Pod Autoscaler* mampu menyesuaikan kapasitas layanan secara otomatis, menjaga sistem tetap stabil pada beban lebih dari 30.000 permintaan per menit. Secara keseluruhan, kombinasi *microservices*, *Protocol Buffers*, *Redis caching*, *MQ generic*, dan *Kubernetes* memberikan fondasi arsitektur yang efisien, tangguh, dan skalabel untuk pengembangan platform e-commerce berskala besar. Meskipun hasil penelitian menunjukkan peningkatan performa dan stabilitas sistem yang signifikan, penelitian ini masih memiliki beberapa keterbatasan. Pengujian performa sistem dilakukan pada lingkungan cloud dengan jumlah instance yang terbatas, sehingga belum sepenuhnya merepresentasikan kondisi sistem pada skala produksi dengan jumlah pengguna yang sangat besar. Selain itu, penelitian ini lebih berfokus pada evaluasi performa *backend services* dan belum mengevaluasi secara mendalam pengaruh faktor lain seperti latensi jaringan atau distribusi layanan pada beberapa wilayah geografis yang berbeda. Untuk penelitian selanjutnya, pengujian sistem dapat dilakukan dengan jumlah pengguna simultan yang lebih besar untuk mengevaluasi kemampuan skalabilitas sistem secara lebih komprehensif. Selain itu, penelitian berikutnya juga dapat mengeksplorasi pendekatan arsitektur lain seperti *serverless computing* atau penggunaan *service mesh* untuk meningkatkan observability, keamanan layanan, serta manajemen komunikasi antar *microservices* pada sistem e-commerce berskala besar.

5. Daftar Pustaka

Choirudin, R., & Adil, A. (2019). Implementasi Rest API Web Service dalam Membangun Aplikasi Multiplatform untuk Usaha Jasa. *MATRIK: Jurnal Manajemen, Teknik Informatika Dan Rekayasa Komputer, 18*(2), 284–293. https://doi.org/10.30812/matrik.v18i2.407.

Dipraja, F., & Rahman, A. (2025). Penerapan Redis Cluster Meningkatkan Efisiensi Caching Arsitektur Microservices. *Intellect: Indonesian Journal of Learning and Technological Innovation, 4*(1), 171–179. https://doi.org/10.57255/intellect.v4i1.1445.

Guntakandla, A. R. (2025). Microservices and Modular Architecture: Revolutionizing E-Commerce Scalability. *Journal of Computer Science and Technology Studies*, 133–137.

Jeffery, A., Howard, H., & Mortier, R. (2021). Rearchitecting Kubernetes for the Edge. *EdgeSys 2021 - Proceedings of the 4th International Workshop on Edge Systems, Analytics and*

Networking, Part of EuroSys 2021, 7–12. https://doi.org/10.1145/3434770.3459730.

Jindal, A., Podolskiy, V., & Gerndt, M. (2019). Performance modeling for cloud microservice applications. *ICPE 2019 - Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 25–32. https://doi.org/10.1145/3297663.3310309.

Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media.

Newman, S. (2015). *Building microservices*. O'Reilly.

Oyeniran, O. C., Adewusi, A. O., Adeleke, A. G., Akwawa, L. A., & Azubuko, C. F. (2024). Microservices architecture in cloud-native applications: Design patterns and scalability. *Computer Science & IT Research Journal, 5*(9), 2107–2124. https://doi.org/10.51594/csitrj.v5i9.1554.

Privalov, M. V., & Stupina, M. V. (2024). Improving web-oriented information systems efficiency using Redis caching mechanisms. *Indonesian Journal of Electrical Engineering and Computer Science, 33*(3), 1667–1675. https://doi.org/10.11591/ijeecs.v33.i3.pp1667-1675.

Putra, K. R., Nurjaman, A. R., & Haniifah, A. F. (2025). Perancangan dan Evaluasi Arsitektur Microservices Menggunakan Microservices Migration Pattern dan Microservices Scorecard. *Jurnal Aplikasi Teknologi Informasi Dan Manajemen (JATIM), 6*(2), 70–83.

Rejiba, Z., & Chamanara, J. (2023). Custom Scheduling in Kubernetes: A Survey on Common Problems and Solution Approaches. *ACM Computing Surveys, 55*(7). https://doi.org/10.1145/3544788.

Setiadi, T., Rajendra, L., Fajri, H. A., & Ilhami, S. D. (2022). Penerapan Sistem Informasi Untuk survei Marketplace dalam Bisnis Kreatif UMKM Berbasis E-commerce. *JURNAL ILMIAH SISTEM INFORMASI (JUSI), 1*(2), 16–29.

Sharma, S. (2025). The Impact of Microservices Architecture on System Scalability. *American Academic Scientific Research Journal for Engineering, Technology, and Sciences*.

Shatnawi, A., Bahri, A., Niang, B., & Verhaeghe, B. (2025). Enhancing Data Serialization Efficiency in REST Services: Migrating from JSON to Protocol Buffers. *Proceedings of the 20th International Conference on Software Technologies (ICSOFT2025)*, 193–200. https://doi.org/10.5220/0013459500003964.

Sugiarto, & Triandini, E. (2022). Pengembangan Database E-commerce De Janggelan Menggunakan Metode Database Life Cycle. *JURNAL SISTEM DAN INFORMATIKA (JSI), 16*(2), 122–132.

Suwarjono, & Averoes, F. (2025). Peningkatan Performa Aplikasi Web Dinamis Berbasis PHP melalui Implementasi Redis Caching. *JINTIKOM: JURNAL INFORMASI TEKNOLOGI DAN KOMPUTER, 1*(1), 37–44.

Wati, R., Andriyani, N., Priyono, P., & Susilowati, T. (2025). Rancang Bangun Sistem Informasi Akademik Berbasis Microservices. *RIGGS: Journal of Artificial Intelligence and Digital Business, 4*(4), 4906–4912. https://doi.org/10.31004/riggs.v4i4.4006.