Implementation and Analysis of Cloud-Native Approaches to Enhance Scalability and Performance in Dwi Jaya Understel Workshop Information System
(Implementasi dan Analisis Pendekatan Cloud-Native untuk Meningkatkan Skalabilitas dan Kinerja pada Sistem Informasi Bengkel Dwi Jaya Understel)

Moch Bagus Tri Cahyo[1)], Hamzah Setiawan [2)], Ika Ratna Indra Astutik[3)], Rohman Dijaya[4)]

[1)] Program Studi Informatika, Universitas Muhammadiyah Sidoarjo, Indonesia
[2)] Program Studi Informatika, Universitas Muhammadiyah Sidoarjo, Indonesia
[3)] Program Studi Informatika, Universitas Muhammadiyah Sidoarjo, Indonesia
[4)] Program Studi Informatika, Universitas Muhammadiyah Sidoarjo, Indonesia
*Email Penulis Korespondensi: *hamzah@umsida.ac.id*

Abstract. *This study aims to analyze the differences in scalability and performance between a traditional monolithic system hosted on a Virtual Private Server (VPS) and a cloud-native serverless architecture using AWS services for an automotive workshop information system. An experimental method was employed using a post-test only control group design. Performance testing was conducted with K6 as the stress testing tool under a ramp-up load pattern of up to 60 Virtual Users (VU) to simulate peak traffic conditions, while Grafana was used for real-time monitoring and visualization of system metrics.*

The results indicate that under peak load scenarios, the cloud-native architecture reduced the average response time by 89.1% (from 6.05 seconds to 657.10 milliseconds) and eliminated the error rate completely (from 0.154% to 0%), compared to the monolithic system. Additionally, the throughput improved by 38.2%, demonstrating better responsiveness and stability. These findings confirm that serverless cloud-native systems offer superior scalability and reliability in handling dynamic and high-demand workloads, making them well-suited for public service platforms such as automotive workshop information systems.

Keywords - *cloud-native architecture, monolithic architecture, AWS Lambda, Amazon RDS, scalability, system performance, serverless computing, information system.*

Abstrak. *Penelitian ini bertujuan untuk menganalisis perbedaan skalabilitas dan kinerja antara sistem monolitik tradisional yang dihosting pada Virtual Private Server (VPS) dan arsitektur cloud-native tanpa server (serverless) menggunakan layanan AWS untuk sistem informasi bengkel otomotif. Metode eksperimen digunakan dengan desain post-test only control group. Pengujian kinerja dilakukan menggunakan K6 sebagai alat stress testing dengan pola ramp-up load hingga 60 Virtual Users (VU) untuk mensimulasikan kondisi lalu lintas puncak, sementara Grafana digunakan untuk pemantauan waktu nyata dan visualisasi metrik sistem.*

Hasil penelitian menunjukkan bahwa dalam skenario beban puncak, arsitektur cloud-native berhasil menurunkan waktu respons rata-rata sebesar 89,1% (dari 6,05 detik menjadi 657,10 milidetik) dan menghilangkan tingkat kesalahan sepenuhnya (dari 0,154% menjadi 0%), dibandingkan dengan sistem monolitik. Selain itu, throughput meningkat sebesar 38,2%, yang menunjukkan responsivitas dan stabilitas yang lebih baik. Temuan ini menegaskan bahwa sistem cloud-native serverless menawarkan skalabilitas dan keandalan yang lebih unggul dalam menangani beban kerja dinamis dan permintaan tinggi, sehingga sangat cocok untuk platform layanan publik seperti sistem informasi bengkel otomotif.

Kata Kunci - *Arsitektur cloud-native, arsitektur monolitik, AWS Lambda, Amazon RDS, skalabilitas, kinerja sistem, komputasi tanpa server, sistem informasi.*

I. PENDAHULUAN

Kemajuan teknologi informasi telah mendorong sistem informasi modern tidak hanya untuk beroperasi secara stabil tetapi juga memiliki kemampuan adaptif dalam menangani lonjakan beban kerja secara tiba-tiba dan permintaan pengguna yang berfluktuasi. Hal ini sangat penting terutama pada layanan publik seperti sistem informasi bengkel otomotif, di mana transaksi, data pelanggan, dan permintaan layanan dapat meningkat secara mendadak dan bersamaan. Kegagalan dalam menangani beban tersebut dapat menyebabkan antrean, kehilangan data transaksi, serta

penurunan kepuasan pelanggan [1]. Integrasi layanan cloud ke dalam sistem informasi berbasis mobile juga terbukti meningkatkan efisiensi manajemen layanan dan fleksibilitas dalam menangani permintaan pengguna yang dinamis, mendukung adaptabilitas yang dibutuhkan oleh lingkungan bengkel [2].
Dibandingkan dengan praktik komputasi awan tradisional yang biasanya menjalankan aplikasi monolitik pada infrastruktur virtual, pendekatan cloud-native menawarkan efisiensi sumber daya dan fleksibilitas yang lebih besar untuk kebutuhan bisnis yang dinamis [3]. Masalah umum di bengkel yang masih mengandalkan pemrosesan data manual—seperti catatan transaksi yang tidak lengkap atau laporan keuangan yang tidak akurat—dapat diatasi melalui penerapan sistem berbasis cloud yang dilengkapi dengan fitur real-time [4].

Arsitektur cloud-native memberikan solusi yang lebih fleksibel dan efisien dengan memanfaatkan layanan seperti AWS Lambda, Amazon S3, dan Amazon RDS, yang mendukung auto-scaling dan pemantauan secara real-time. Alat otomatisasi seperti AWS CodeDeploy juga memungkinkan proses deployment yang lebih tangguh dan efisien. Pendekatan ini cocok untuk sistem informasi bengkel yang melayani pengguna dalam volume dinamis dan membutuhkan tingkat keandalan tinggi [5].

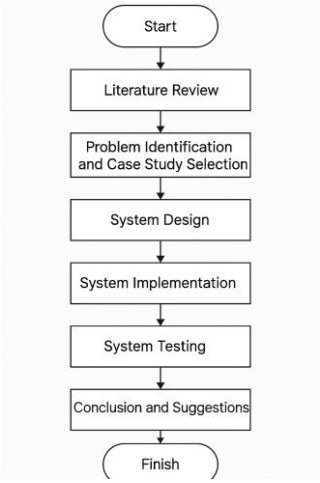
Beberapa penelitian sebelumnya telah mengeksplorasi kinerja arsitektur cloud-native; namun, sebagian besar bersifat umum dan tidak secara spesifik membahas konteks sistem informasi bengkel otomotif. Misalnya, pendekatan benchmarking untuk mengevaluasi skalabilitas aplikasi cloud-native telah diusulkan dalam [6], tetapi tidak divalidasi di lingkungan dengan transaksi tinggi seperti bengkel. Perbandingan antara aplikasi monolitik dan mikroservis dalam [7] hanya berfokus pada throughput dan tidak mempertimbangkan faktor penting lain seperti cold start dan tingkat kesalahan. Selain itu, evaluasi dalam [8] mengecualikan teknologi serverless seperti AWS Lambda, yang semakin relevan dalam konteks skalabilitas modern. Keterbatasan ini menunjukkan bahwa penelitian lebih lanjut diperlukan untuk menilai secara komprehensif bagaimana arsitektur serverless bekerja dalam sistem dengan permintaan dinamis dan kompleksitas transaksi harian seperti yang terdapat di lingkungan bengkel.

Penelitian ini bertujuan untuk mengisi kekosongan tersebut dengan menyajikan investigasi eksperimental berbasis sistem informasi bengkel otomotif nyata, menggunakan arsitektur serverless dan cloud-native. Tujuan utamanya adalah mengevaluasi kemampuan autoscaling dan stabilitas operasional sistem saat menghadapi lonjakan pengguna secara mendadak, dengan fokus khusus pada throughput dan tingkat kesalahan—faktor yang belum banyak dibahas dalam literatur sebelumnya. Penelitian ini diharapkan dapat memberikan wawasan praktis bagi pengembang dan arsitek sistem dalam memilih strategi implementasi untuk beban kerja dengan kebutuhan skalabilitas tinggi.

II. METODE

Penelitian ini menggunakan pendekatan eksperimen terapan dengan tujuan membandingkan kinerja dan skalabilitas antara arsitektur monolitik berbasis Virtual Private Server (VPS) dan arsitektur cloud-native yang menggunakan layanan AWS. Desain penelitian yang digunakan adalah post-test only control group design, di mana observasi dilakukan pada dua arsitektur berbeda tanpa fase pre-test. Penelitian ini dilaksanakan di Bengkel Otomotif Dwi Jaya Understel, Pasuruan, Jawa Timur, selama tiga bulan.

Pengujian dilakukan dengan menerapkan beban lalu lintas secara bertahap menggunakan stress test dengan pola ramp, yang berarti peningkatan beban dilakukan secara bertahap seiring waktu. Metode ini serupa dengan pendekatan yang digunakan oleh [7], yang juga menerapkan pola ramp untuk membandingkan kinerja sistem berbasis monolitik dan mikroservis.



Gambar 1. Proses penelitian

A. Variables in This Study

Penelitian ini melibatkan tiga jenis variabel utama. Variabel independen adalah lingkungan *hosting* yang digunakan dalam eksperimen, yaitu *Virtual Private Server* (VPS) dan layanan *cloud-native* AWS. Variabel dependen adalah performa sistem, yang diukur berdasarkan metrik seperti waktu respons, *throughput*, dan skalabilitas. Untuk memastikan konsistensi dan keandalan dalam proses eksperimen, beberapa variabel kontrol dipertahankan, termasuk data uji yang digunakan, skenario beban kerja yang diterapkan, dan alat pengujian yang diimplementasikan di seluruh evaluasi performa.

B. Teknik Pengumpulan Data

Pengumpulan data dalam penelitian ini dilakukan melalui beberapa metode terintegrasi. Observasi langsung dilakukan untuk menangkap respons sistem secara *real-time* dan mengidentifikasi anomali selama pengujian. Simulasi beban kerja dilakukan menggunakan K6 dengan pola *ramp-up* hingga 60 *Virtual Users (VUs)*. Jumlah ini didasarkan pada estimasi wajar pengguna bersamaan saat puncak, mengikuti rasio DAU-to-PCU (Daily Active Users to Peak Concurrent Users) yang umum digunakan, yaitu 10:1 hingga 20:1. Aturan praktis ini banyak dipakai dalam perencanaan kinerja untuk memperkirakan beban puncak realistis ketika tidak tersedia log lalu lintas yang detail. Tujuannya adalah mensimulasikan skenario dengan tingkat *concurrency* tinggi yang biasanya terjadi pada jam sibuk dan mengevaluasi sejauh mana arsitektur sistem dapat menangani kondisi tersebut. Metode ini sejalan dengan praktik terbaik dalam pengujian kinerja *cloud-native*, di mana *stress testing* dirancang untuk memvalidasi skalabilitas dan ketahanan dalam kondisi penggunaan yang realistis namun menuntut [13].

Selama pengujian, metrik kinerja seperti waktu respons, *throughput*, dan tingkat kesalahan dikirim ke InfluxDB dan divisualisasikan menggunakan dashboard Grafana untuk pemantauan interaktif. Selain itu, AWS CloudWatch juga digunakan secara paralel untuk memvalidasi dan memastikan konsistensi data dengan memantau metrik sisi server seperti durasi eksekusi Lambda, jumlah kesalahan, dan latensi API Gateway. Integrasi K6, Grafana, dan CloudWatch memungkinkan evaluasi kinerja yang komprehensif serta perbandingan yang akurat antara arsitektur monolitik dan *cloud-native*.

C. Teknik Analisis Data

Data dalam penelitian ini dianalisis secara kuantitatif dengan berfokus pada beberapa indikator kinerja utama. Indikator tersebut mencakup perhitungan rata-rata waktu respons untuk menentukan tingkat responsivitas sistem, waktu respons maksimum untuk mengidentifikasi keterlambatan puncak di bawah beban, serta tingkat kesalahan untuk menilai keandalan sistem selama skenario pengujian. Metrik-metrik ini memberikan evaluasi komprehensif terhadap kinerja sistem pada kedua implementasi arsitektur.

D. Desain Arsitektur Sistem

a. Arsitektur Sistem yang Ada (Monolitik Sederhana)

Sistem yang ada berjalan pada server VPS dengan menggunakan pendekatan monolitik berbasis LAMP stack (Linux, Nginx, MySQL, Flask). Semua komponen sistem dijalankan pada satu mesin, yang menyederhanakan proses *deployment* dan *debugging*. Namun, pendekatan ini memiliki keterbatasan signifikan dalam hal skalabilitas, distribusi beban, dan kemampuan pemantauan. Hal

ini sejalan dengan temuan Blinowski et al. [8], yang menyatakan bahwa sistem monolitik, meskipun lebih mudah dikembangkan pada tahap awal, menjadi tidak efisien dan sulit untuk diskalakan seiring meningkatnya kompleksitas sistem. Pernyataan ini juga didukung oleh penelitian Srinivasan et al. [14], yang menyimpulkan bahwa arsitektur monolitik cenderung tidak efisien pada lingkungan yang membutuhkan skalabilitas tinggi, serta sering menimbulkan *bottleneck* pada sistem kompleks karena keterkaitan erat dalam satu unit *deployment* dan terbatasnya isolasi layanan.

Gambar 2. Monolitik Sederhana

b. Arsitektur Sistem Cloud-Native

Sistem informasi bengkel otomotif ditransformasikan menjadi arsitektur *cloud-native* dengan memanfaatkan berbagai layanan dari Amazon Web Services (AWS) untuk meningkatkan skalabilitas dan efisiensi manajemen infrastruktur. Pada sisi *frontend*, situs web dihosting menggunakan Amazon S3 dengan *static website hosting*, yang memungkinkan akses cepat dan biaya operasional lebih rendah. Pada sisi *backend*, sistem dibangun menggunakan *framework* Flask dan dideploy di AWS Lambda, terhubung melalui API Gateway, sehingga mendukung konsep *serverless* dan memungkinkan permintaan diproses secara elastis sesuai kebutuhan. Basis data yang digunakan adalah Amazon RDS dengan MySQL, atau sebagai alternatif DynamoDB untuk kebutuhan yang lebih fleksibel pada skala besar. Selain itu, pemantauan dan pencatatan log dilakukan menggunakan Amazon CloudWatch, yang memungkinkan pelacakan kinerja dan aktivitas secara *real-time* sekaligus mempermudah proses *troubleshooting*. Pendekatan arsitektur *cloud* ini selaras dengan praktik terbaik dalam migrasi *cloud-native* untuk mencapai elastisitas layanan, otomatisasi proses bisnis, dan efisiensi biaya sebagaimana dijelaskan oleh Bagir et al. [15]. Selain itu, Kumar et al. [16] menekankan bahwa komputasi *serverless* menjadi paradigma evolusioner dalam pengembangan aplikasi modern karena kemampuannya menyediakan *automatic scaling*, mengurangi beban manajemen infrastruktur, serta mengoptimalkan biaya dalam lingkungan *cloud-native*.

Gambar 3. Cloud-Native

c. Perbandingan Arsitektur Sistem

Tabel 1. Perbandingan Arsitektur Sistem

Aspect	Monolithic (Before)	Cloud-Native (AWS Lambda)	Aspect	Monolithic (Before)
Hosting	VPS (Single Server)	AWS Lambda (Serverless)	Hosting	VPS (Single Server)
Frontend Hosting	Stored on VPS	Amazon S3 (Static Hosting)	Frontend Hosting	Stored on VPS
Backend	Flask on VPS	Flask via AWS Lambda + API Gateway	Backend	Flask on VPS
Database	MySQL on the server	Amazon RDS (MySQL) or DynamoDB	Database	MySQL on the server

Tabel 1 menyajikan perbandingan antara arsitektur sistem sebelum dan sesudah migrasi, khususnya antara pendekatan monolitik berbasis VPS dan arsitektur *cloud-native* yang dibangun di atas AWS Lambda. Perbandingan mencakup beberapa aspek penting seperti *hosting*, skalabilitas, *deployment*, kemampuan pemantauan, dan model biaya.

Tabel tersebut secara jelas menyoroti keterbatasan arsitektur monolitik yang bergantung pada satu server dan konfigurasi manual, sehingga menimbulkan tantangan dalam hal skalabilitas dan pemeliharaan sistem. Sebaliknya, model *cloud-native* memungkinkan *automatic scaling*, *serverless deployment*, serta struktur biaya berbasis penggunaan (*pay-per-use*), yang menjadikannya lebih sesuai untuk lingkungan dinamis dan permintaan tinggi seperti sistem informasi bengkel otomotif. Perbedaan ini membenarkan adanya transisi arsitektur dan mendukung alasan dilakukannya evaluasi kinerja dalam penelitian ini.

III. HASIL DAN PEMBAHASAN

A. Hasil

a. Hasil Pengujian Fungsional

Tabel 2. Hasil Pengujian Fungsional

Test Case ID	Feature	Test Steps	Tes data	Status
TC F01	Access Dashboard	Enter email & password, then login	Username:q Password: q	✓
TC F02	Add Transaction	Input license plate, select spare part, save	B1234XYZ, Engine Oil, Rp80,000	✓
TC F03	Search Spare Part	Type "Oli" in the search field	Oli	✓
TC F04	Logout	Click the logout button	-	✓

Pengujian fungsional dilakukan untuk memastikan bahwa semua fitur inti dari sistem informasi bengkel otomotif beroperasi sesuai dengan kebutuhan pengguna dan spesifikasi sistem. Pengujian mencakup proses login, entri transaksi layanan, pencarian suku cadang, dan fungsi logout. Semua fitur berhasil lulus pengujian dengan status *Pass*, yang menunjukkan bahwa sistem berfungsi sebagaimana mestinya tanpa adanya kesalahan. Menurut Jia et al. [17], pengujian fungsional merupakan bagian penting dari manajemen kualitas perangkat lunak, karena bertujuan memverifikasi apakah sistem memenuhi kebutuhan pengguna dan spesifikasi yang telah ditentukan. Pendekatan ini termasuk dalam kategori *black-box testing*, di mana pengujian dilakukan tanpa mempertimbangkan struktur internal sistem dan berfokus pada input serta output sistem.

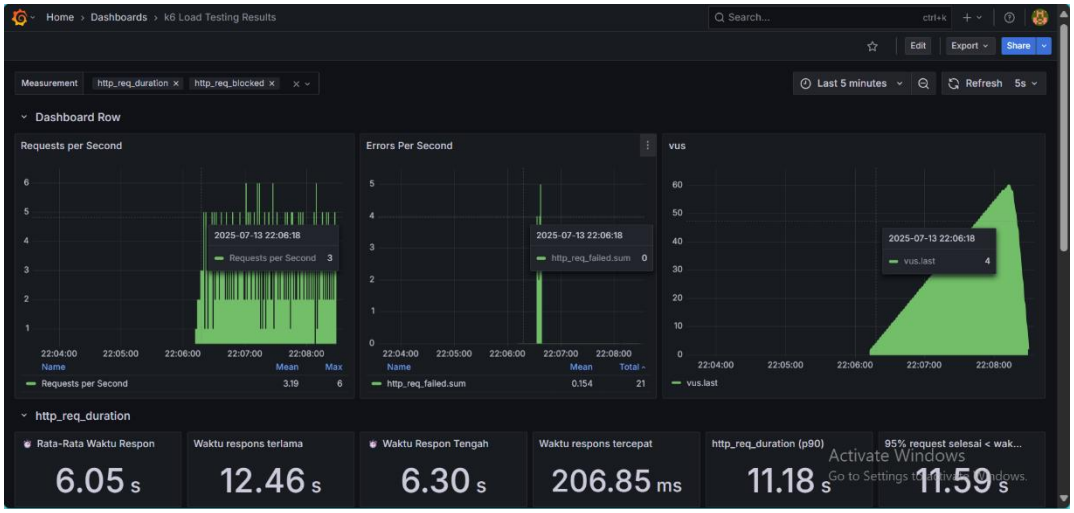
b. Hasil Uji Stres

Tabel 3. Hasil Uji Stres

Test Case	Scenario	Virtual Users (VU)	SLA Target
TC-K01	Normal Load (open page, search spare parts) duration 1 minute	10 VU	Response < 1500ms, Error < 1%
TC-K02	Medium Load (input service transaction) duration 1 minute	20 VU	Response < 1000ms, Error < 1%
TC-K03	Peak Load (register simulation 1–60 VU) duration 1 minute	1-60 VU	Response < 1000ms, Error < 2%

Pengujian *stress testing* dilakukan untuk mengevaluasi kinerja sistem informasi bengkel otomotif pada berbagai skenario beban, yaitu beban normal, beban sedang, dan beban puncak, dengan mensimulasikan jumlah *Virtual Users (VUs)* yang berbeda. Pengujian ini mengukur metrik seperti waktu respons, tingkat kesalahan, dan *throughput* berdasarkan target *Service Level Agreement (SLA)*. Menurut Castro dan Harman [18], pengujian kinerja pada arsitektur *serverless* seperti AWS Lambda menghadapi tantangan dalam kestabilan hasil, terutama ketika jumlah permintaan meningkat secara drastis. Hal ini menekankan

pentingnya merancang skenario beban yang representatif dan menggunakan alat uji yang mampu mensimulasikan kondisi nyata secara konsisten.



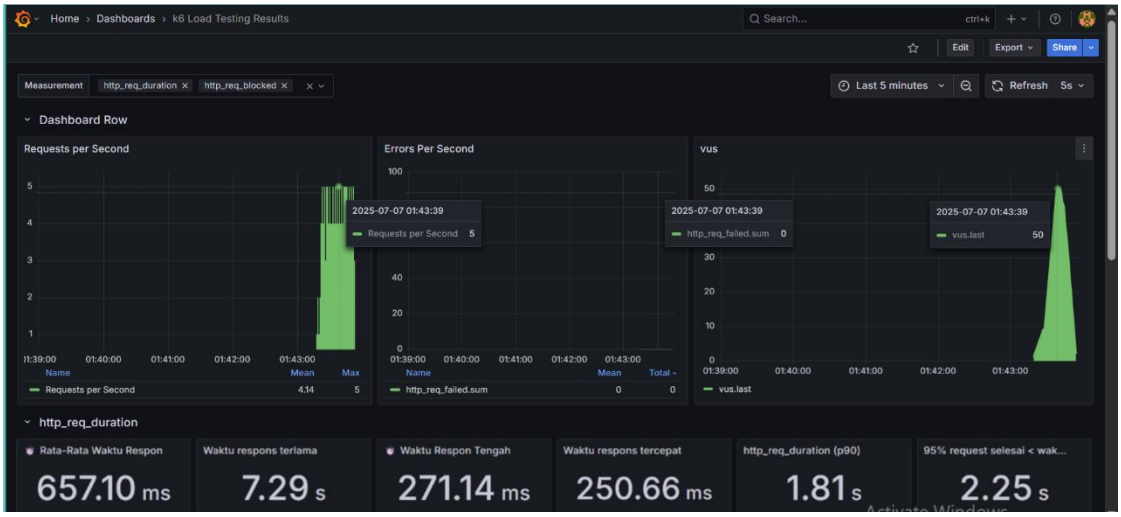
Gambar 4. Hasil pengujian monolit

Gambar tersebut menyajikan hasil pengujian kinerja sistem monolitik yang dihosting dalam lingkungan VPS tradisional. Pada panel “Errors per Second”, terlihat lonjakan sebanyak 5 kesalahan di awal pengujian. Meskipun tidak ada kesalahan lebih lanjut, sistem mencatat waktu respons rata-rata 6,05 detik dan waktu respons puncak 12,46 detik, keduanya jauh melebihi target SLA <2 detik. Selain itu, panel "Throughput" menunjukkan bahwa sistem kesulitan mempertahankan tingkat pemrosesan permintaan yang konsisten. Hasil ini menunjukkan keterbatasan arsitektur monolitik dalam menangani beban sedang sekalipun, serta kurangnya elastisitas dalam merespons fluktuasi lalu lintas. Hal ini sejalan dengan temuan Blinowski et al. [12], yang menyoroti kelemahan sistem monolitik dibandingkan arsitektur *cloud-native* yang terdistribusi dan dapat diskalakan.

Tabel 4. Hasil pengujian monolit

Test Case	AVG Response	MAX Response	Error Rate	Status
TC-K01	1.25s	2.95s	0%	Failed
TC-K02	271.62ms	1.16s	0%	passed
TC-K03	6.05s	12.46s	0.154%	Failed

Hasil pengujian menunjukkan kinerja sistem monolitik yang berjalan pada VPS. Pada skenario beban normal (TC-K01), sistem mencatat waktu respons rata-rata 1,25 detik (1250 ms), yang masih berada dalam ambang SLA <1500 ms. Namun, waktu respons maksimum mencapai 2,95 detik (2950 ms), yang menunjukkan inkonsistensi kinerja karena melebihi batas toleransi signifikan dalam kondisi tertentu. Pada skenario beban sedang (TC-K02), sistem berhasil memenuhi SLA dengan waktu respons rata-rata 271,62 ms dan tingkat kesalahan 0%, yang menunjukkan kinerja stabil dalam kondisi ini. Sementara itu, pada beban puncak (TC-K03), sistem mengalami penurunan kinerja yang signifikan dengan waktu respons rata-rata 6,05 detik dan waktu respons maksimum 12,46 detik, sehingga gagal memenuhi SLA untuk skenario ini. Hasil ini menunjukkan bahwa arsitektur monolitik masih memiliki keterbatasan dalam menangani lonjakan lalu lintas secara elastis, meskipun tetap stabil dan responsif pada kondisi beban sedang.



Gambar 5. Hasil pengujian Cloud - Native

Gambar 5 menggambarkan hasil uji kinerja sistem *cloud-native* berbasis AWS Lambda pada skenario TC-K03, di mana sistem diuji dengan beban puncak 60 pengguna simultan. Pengujian ini dilakukan untuk mengevaluasi kemampuan sistem dalam menangani lonjakan lalu lintas secara tiba-tiba.

Hasilnya menunjukkan bahwa sistem berhasil mempertahankan waktu respons rata-rata sebesar 657,10 milidetik, yang berada jauh di bawah ambang SLA 2 detik. Selain itu, tidak ada kesalahan yang tercatat selama pengujian (tingkat kesalahan 0%), yang menunjukkan keandalan sistem dalam kondisi lalu lintas tinggi.

Grafik juga memperlihatkan bahwa latensi dan *throughput* tetap stabil, tanpa lonjakan signifikan selama periode ketika 60 pengguna mengakses sistem secara bersamaan. Hal ini mencerminkan kemampuan *automatic scaling* dari arsitektur *serverless*, di mana AWS Lambda secara dinamis menyesuaikan kapasitasnya untuk memenuhi permintaan pengguna.

Temuan ini semakin menegaskan keunggulan pendekatan *cloud-native*, yang memungkinkan sistem tetap responsif dan stabil bahkan saat terjadi lonjakan lalu lintas—selaras dengan temuan Blinowski et al. [8] terkait elastisitas dan skalabilitas arsitektur mikroservis dan *serverless*.

Tabel 5. Hasil pengujian Cloud - Native

Test Case	AVG Response	MAX Response	Error Rate	Status
TC-K01	2.14s	2.56s	0%	Failed
TC-K02	871.62ms	1.16s	0%	Passed
TC-K03	657.10ms	72.9s	0%	Passed

Bagian ini menyajikan hasil pengujian kinerja sistem *cloud-native* yang di-*deploy* menggunakan layanan AWS. Pada skenario beban normal (TC-K01), sistem gagal memenuhi *Service Level Agreement (SLA)* karena waktu respons rata-rata mencapai 2,14 detik, melebihi target yang telah ditetapkan. Namun, pada skenario beban sedang (TC-K02) dan beban puncak (TC-K03), sistem berhasil memenuhi SLA dengan waktu respons rata-rata masing-masing 871,62 ms dan 657,10 ms, serta tingkat kesalahan 0%.

Hasil ini menunjukkan bahwa arsitektur *cloud-native* menawarkan skalabilitas yang lebih unggul dalam menangani beban tinggi. Temuan ini konsisten dengan penelitian Blinowski et al. [8], yang menunjukkan bahwa sistem berbasis mikroservis dan *cloud-native* memiliki keunggulan dalam elastisitas dan kinerja pada kondisi lalu lintas tinggi. Sementara itu, kegagalan yang diamati pada skenario beban ringan dapat dikaitkan dengan latensi *cold start*, yang merupakan masalah umum dalam lingkungan *serverless* seperti AWS Lambda [18].

B. Pembahasan

erdasarkan pengujian yang dilakukan, terdapat perbedaan kinerja yang signifikan antara arsitektur monolitik dan *cloud-native*. Dari perspektif fungsional, semua fitur inti—termasuk login, pembuatan transaksi, pencarian suku cadang, dan logout—berhasil dijalankan pada kedua arsitektur, yang menunjukkan bahwa sistem memenuhi kebutuhan fungsionalnya.

Dari sisi kinerja, arsitektur monolitik menunjukkan hasil baik pada skenario beban ringan dan sedang. Pada TC-K01 (beban normal), sistem mencatat waktu respons rata-rata 1,23 detik, dan 271,62 ms pada TC-K02 (beban sedang), tetap menjaga responsivitas sistem. Namun, pada kondisi lonjakan lalu lintas di TC-K03 (beban puncak), sistem

mengalami penurunan kinerja signifikan dengan waktu respons rata-rata 6,05 detik dan tingkat kesalahan 0,154%, yang mencerminkan keterbatasan skalabilitasnya. Hal ini konsisten dengan penelitian Fan et al. [19], yang menekankan bahwa sistem monolitik dapat menjadi *bottleneck* kinerja pada lalu lintas tinggi akibat arsitektur yang saling terikat erat dan ketidakmampuan untuk melakukan penskalaan komponen secara terpisah.
Sebaliknya, arsitektur *cloud-native* menunjukkan ketahanan lebih baik dalam menangani beban yang lebih berat. Meskipun mengalami waktu respons rata-rata sedikit lebih lama, yaitu 2,14 detik pada TC-K01 akibat latensi *cold start* AWS Lambda, sistem tetap stabil tanpa kesalahan. Pada TC-K02, sistem *cloud-native* tetap sesuai SLA dengan 871,62 ms, dan pada TC-K03 (beban puncak), berhasil menangani 60 pengguna virtual bersamaan dengan waktu respons rata-rata 657,10 ms dan tingkat kesalahan 0%.
Temuan ini mendukung klaim Menéndez et al. [20], yang membandingkan arsitektur mikroservis menggunakan AWS Lambda dan RDS, dan menyimpulkan bahwa konfigurasi *cloud-native* semacam itu menunjukkan stabilitas kinerja dan keandalan yang kuat di bawah lalu lintas tinggi, meskipun menghadapi tantangan terkait inisialisasi dan *cold start*. Selain itu, Christian dan Bisma [7] menyoroti bahwa sistem berbasis mikroservis lebih stabil dan dapat diskalakan dibandingkan sistem monolitik, meskipun berpotensi mengonsumsi lebih banyak sumber daya CPU. Hal ini semakin memperkuat kesimpulan bahwa arsitektur *cloud-native* lebih tangguh dalam kondisi permintaan tinggi.
Pola kinerja yang kontras ini berasal dari sifat dasar masing-masing arsitektur. Sistem monolitik paling cocok untuk lingkungan berskala kecil dengan variabilitas rendah karena kesederhanaannya. Namun, kekakuannya menjadi keterbatasan ketika menghadapi lalu lintas yang tidak terduga atau volume tinggi. Sebaliknya, sistem *cloud-native* secara inheren bersifat elastis, dengan arsitektur seperti AWS Lambda yang memungkinkan komponen individu diskalakan sesuai permintaan.
Meskipun demikian, satu kelemahan utama tetap ada: latensi *cold start*. Brasoveanu et al. [21] mencatat bahwa *cold start* dalam lingkungan *serverless* menimbulkan keterlambatan tergantung pada lingkungan *runtime*, ukuran *deployment*, dan konfigurasi, biasanya berkisar dari beberapa ratus milidetik hingga beberapa detik. Meskipun *cold start* bukan fokus utama penelitian ini, hal tersebut tetap menjadi tantangan umum pada aplikasi *serverless*. AWS menawarkan fitur bernama *provisioned concurrency*, yang memungkinkan fungsi Lambda tetap dalam kondisi terinisialisasi untuk mengurangi latensi awal, meskipun dengan tambahan biaya operasional. Untuk sistem yang membutuhkan waktu respons konsisten rendah, strategi mitigasi semacam ini layak dipertimbangkan, sebagaimana disoroti dalam studi terbaru yang mengevaluasi efektivitas pendekatan tersebut dalam secara signifikan mengurangi dampak *cold start* [22].

V. SIMPULAN

Berdasarkan hasil penelitian ini, dapat disimpulkan bahwa arsitektur *cloud-native* memiliki kinerja yang lebih baik dibandingkan arsitektur monolitik dalam hal skalabilitas dan ketahanan sistem pada kondisi beban tinggi, berkat dukungan *autoscaling* dan pemantauan *real-time*. Di sisi lain, arsitektur monolitik memberikan waktu respons yang lebih cepat pada beban ringan hingga sedang, tetapi mengalami penurunan kinerja yang signifikan ketika menghadapi skenario beban puncak.

Untuk sistem informasi bengkel otomotif yang beroperasi di lingkungan dinamis dan berpotensi mengalami lonjakan lalu lintas secara tiba-tiba, pendekatan *cloud-native* lebih direkomendasikan. Namun, optimalisasi diperlukan untuk mengurangi latensi *cold start* pada Lambda, guna meningkatkan waktu respons pada kondisi beban ringan.

Selain itu, peningkatan dalam responsivitas sistem, skalabilitas, dan stabilitas diharapkan dapat memberikan kontribusi positif terhadap kepuasan pelanggan dan efisiensi layanan secara keseluruhan. Bagi bengkel otomotif yang bergantung pada pemrosesan transaksi yang tepat waktu dan akurat, penurunan tingkat kesalahan serta kinerja yang konsisten di bawah beban tinggi dapat meminimalkan keterlambatan layanan, mengurangi waktu antrean, dan meningkatkan kepercayaan pelanggan. Faktor-faktor ini juga dapat memperbaiki metrik bisnis utama seperti *throughput* layanan, ketersediaan sistem, dan tingkat retensi pelanggan.

Untuk penelitian selanjutnya, disarankan agar mencakup parameter evaluasi terkait biaya operasional dan efisiensi sumber daya, guna memberikan pemahaman yang lebih komprehensif tentang manfaat migrasi ke arsitektur *cloud-native*, baik dari sisi teknis maupun ekonomis.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Bengkel Otomotif Dwi Jaya Understel, Pasuruan, Jawa Timur, yang telah menyediakan fasilitas dan dukungan selama proses penelitian berlangsung. Ucapan terima kasih juga disampaikan kepada pihak-pihak yang membantu dalam penyediaan data serta pengujian sistem, sehingga penelitian ini dapat terlaksana dengan baik.

REFERENSI

[1] S. Henning and W. Hasselbring, "A configurable method for benchmarking scalability of cloud-native applications," *Empir Softw Eng*, vol. 27, no. 6, Nov. 2022, doi: 10.1007/s10664-022-10162-1.

[2] K. Maydiva Heriansaputri and M. A. Romli, "Application for Mental Health Consultation with Scheduling Function at the Counseling Guidance of Universitas Teknologi Yogyakarta," *Journal of Information and Technology) Accredited Sinta*, vol. 4.

[3] J. Lin, D. Xie, J. Huang, Z. Liao, and L. Ye, "A multi-dimensional extensible cloud-native service stack for enterprises," *Journal of Cloud Computing*, vol. 11, no. 1, Dec. 2022, doi: 10.1186/s13677-022-00366-7.

[4] B. H. Lavenia, W. Hayuhardhika, N. Putra, and B. T. Hanggara, "Pengembangan Sistem Informasi Point of Sales untuk Bengkel berbasis Cloud Computing (Studi Kasus: Bengkel Mas Pur Baturaja)," 2019. [Online]. Available: http://j-ptiik.ub.ac.id

[5] D. Desmulyati and M. R. Perdana Putra, "Load Balance Design of Google Cloud Compute Engine VPS with Round Robin Method in PT. Lintas Data Indonesia," *SinkrOn*, vol. 3, no. 2, p. 147, Mar. 2019, doi: 10.33395/sinkron.v3i2.10064.

[6] A. A'fa Nafasha, I. Putu, E. Indrawan, and G. I. Setiawan, "ANALISIS PERBANDINGAN BIAYA DAN SERVERLESS COMPUTING PADA GOOGLE CLOUD PLATFORM," *Jurnal Manajemen dan Teknologi Informasi (JMTI)*, vol. 14, pp. 1–09, doi: 10.59819.

[7] Y. Christian and R. Bisma, "Studi Perbandingan Performa Aplikasi Web Monolitik Dan Microservice Berbasis Apache Kafka," *Journal of Informatics and Computer Science*, vol. 03, 2021.

[8] G. Blinowski, A. Ojdowska, and A. Przybylek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," *IEEE Access*, vol. 10, pp. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[9] D. Nurlaila and H. Mulyono, "Sistem Informasi Manajemen Bengkel Berbasis Web pada Bengkel Ikhsan Jaya Motor," 2023.

[10] R. Rakhman, M. Shadiq, D. Syaddad, and V. R. Sesa, "Implementasi Cloud Native dan Multi Cloud Pada Sistem Operasi Windows Dengan Menggunakan Docker dan Cara Penggunaannya," vol. 2, no. 5, pp. 819–826, 2024, doi: 10.5281/zenodo.13119907.

[11] S. Deng *et al.*, "Cloud-Native Computing: A Survey From the Perspective of Services," *Proceedings of the IEEE*, vol. 112, no. 1, pp. 12–46, Jan. 2024, doi: 10.1109/JPROC.2024.3353855.

[12] A. M. Ștefan, N. R. Rusu, E. Ovreiu, and M. Ciuc, "Empowering Healthcare: A Comprehensive Guide to Implementing a Robust Medical Information System—Components, Benefits, Objectives, Evaluation Criteria, and Seamless Deployment Strategies," *Applied System Innovation*, vol. 7, no. 3, Jun. 2024, doi: 10.3390/asi7030051.

[13] J. S. Patel, "Cloud-Native Performance Testing: Strategies for Scalability and Reliability in Modern Applications," *European Journal of Computer Science and Information Technology*, vol. 13, no. 8, pp. 32–49, Mar. 2025, doi: 10.37745/ejcsit.2013/vol13n83249.

[14] D. Narsina, "Microservices vs. Monoliths: Comparative Analysis for Scalable Software Architecture Design", doi: 10.18034/ei.v11i2.734.

[15] M. Bagir *et al.*, "Migrasi Cloud Native Architecture API Development untuk Memaksimalkan 5G Monetization pada Jaringan AXIATA." [Online]. Available: https://maklumatika.i-tech.ac.id/index.php/reklamasi

[16] T. Bodner, T. Radig, D. Justen, D. Ritter, and T. Rabl, "An Empirical Evaluation of Serverless Cloud Infrastructure for Large-Scale Data Processing," Jan. 2025, [Online]. Available: http://arxiv.org/abs/2501.07771

[17] X. Jia, "The Role and Importance of Software Testing in Software Quality Management." [Online]. Available: http://www.stemmpress.com

[18] S. Eismann *et al.*, "A Review of Serverless Use Cases and their Characteristics," Jan. 2021, [Online]. Available: http://arxiv.org/abs/2008.11110

[19] C. F. Fan, A. Jindal, and M. Gerndt, "Microservices vs serverless: A performance comparison on a cloud-native web application," in *CLOSER 2020 - Proceedings of the 10th International Conference on Cloud Computing and Services Science*, SciTePress, 2020, pp. 204–215. doi: 10.5220/0009792702040215.

[20] J. M. Menéndez, J. E. L. Gayo, E. R. Canal, and A. E. Fernández, "A comparison between traditional and Serverless technologies in a microservices setting," May 2023, [Online]. Available: http://arxiv.org/abs/2305.13933

[21] A. Brasoveanu, M. Moodie, and R. Agrawal, “Textual evidence for the perfunctoriness of independent medical reviews,” in *CEUR Workshop Proceedings*, CEUR-WS, 2020, pp. 1–9. doi: 10.1145/nnnnnnn.nnnnnnn.
[22] A. Brasoveanu, M. Moodie, and R. Agrawal, “Textual evidence for the perfunctoriness of independent medical reviews,” in *CEUR Workshop Proceedings*, CEUR-WS, 2020, pp. 1–9. doi: 10.1145/nnnnnnn.nnnnnnn.

Conflict of Interest Statement:
The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.