JURNAL ILMU KOMPUTER
Volume 4 Nomor 2, Juli, 2026
ISSN: 3031-125X (ONLINE)
Penerbit: Program Studi Teknik Informatika S-2 Universitas Pamulang

Evaluasi Performa API Berbasis Arsitektur Microservices pada Sistem Logistik Menggunakan k6 dan Apache JMeter

*Mufti Ridwan

[1)] Program Studi Teknik Informatika S-2, Program Pascasarjana, Universitas Pamulang, Tangerang Selatan, Banten

Email: 1muftiridwan@gmail.com

ABSTRACT

The growth of Indonesia's e-commerce and logistics industry demands high reliability of microservices-based systems, making API performance a critical factor of service quality. This research evaluates logistics API performance under various load scenarios while comparing two load testing tools, k6 and Apache JMeter. Testing was conducted in a staging environment on the pickup request and shipment manifest endpoints through load, stress, and spike test scenarios with identical configurations, comprising 36 sessions (18 per tool), measuring response time (p95, p99), throughput, error rate, and the CPU and memory overhead of both tools. The results reveal two distinct degradation mechanisms: pickup request degrades with concurrency, its error rate rising from 6.68% (5 VU) to 31.48% (50 VU) and p95 from 2,569 ms to 3,365 ms, whereas manifest degrades with load duration, keeping a low error rate (0.00–0.14%) while p95 rises from 2,831 ms to 4,623 ms. Under normal load, the p95 of both endpoints (2,249–3,263 ms) remains far above the 1,000 ms service level threshold adopted in this study. Both tools cross-validate each other, with relative p95 differences of 2.6–4.0% and 4.5–13.5%; k6 is more resource-efficient (2.27% CPU, 35.6 MB) than Apache JMeter (3.90% CPU, 454.9 MB), whereas Apache JMeter offers greater data depth through its JTL format. The contribution of this research is the identification of two fundamentally different degradation mechanisms on microservices endpoints residing on the same infrastructure.

Keywords: API performance; Apache JMeter; k6; load testing; microservices

ABSTRAK

Pertumbuhan industri e-commerce dan logistik nasional menuntut keandalan tinggi pada sistem berbasis microservices, sehingga performa API menjadi faktor kritis kualitas layanan. Penelitian ini mengevaluasi performa API sistem logistik pada berbagai skenario beban sekaligus membandingkan dua tools load testing, yaitu k6 dan Apache JMeter. Pengujian dilakukan di lingkungan staging pada endpoint pickup request dan manifest pengiriman melalui skenario load, stress, dan spike test dengan konfigurasi identik sebanyak 36 sesi (18 sesi tiap tools), dengan metrik response time (p95 dan p99), throughput, error rate, serta overhead CPU dan memori kedua tools. Hasilnya mengungkap dua mekanisme degradasi yang berbeda: pickup request terdegradasi oleh konkurensi dengan error rate naik dari 6,68% (5 VU) menjadi 31,48% (50 VU) dan p95 dari 2.569 ms menjadi 3.365 ms, sedangkan manifest terdegradasi oleh durasi beban dengan error rate tetap rendah (0,00–0,14%) namun p95 naik dari 2.831 ms menjadi 4.623 ms. Pada beban normal, p95 kedua endpoint (2.249–3.263 ms) masih jauh di atas ambang tingkat layanan 1.000 ms yang ditetapkan pada penelitian ini. Kedua tools saling mengonfirmasi dengan selisih relatif p95 2,6–4,0% dan 4,5–13,5%; k6 lebih efisien sumber daya (CPU 2,27%, memori 35,6 MB) dibanding Apache JMeter (3,90%, 454,9 MB), sedangkan Apache JMeter unggul pada kedalaman data melalui format JTL. Kontribusi penelitian ini adalah identifikasi dua mekanisme degradasi yang berbeda secara fundamental pada endpoint microservices di infrastruktur yang sama.

Kata kunci: Apache JMeter; k6; load testing; microservices; performa API

1. PENDAHULUAN

Industri logistik di Indonesia tumbuh signifikan seiring meningkatnya aktivitas perdagangan elektronik. Jumlah usaha e-commerce pada 2024 mencapai 4,4 juta unit dengan nilai transaksi Rp1.288,93 triliun atau naik 15,3% dibandingkan tahun sebelumnya, dan 99,08% transaksinya melibatkan pengiriman barang dalam negeri [1]. Sistem informasi logistik modern umumnya dibangun di atas arsitektur microservices, yaitu pendekatan yang memecah aplikasi besar menjadi sejumlah layanan kecil berdiri sendiri yang berkomunikasi melalui API berbasis HTTP/REST [2]. Tantangan utamanya adalah memastikan setiap API mampu menangani beban tinggi secara bersamaan tanpa penurunan performa, karena kegagalan satu service berimbas pada seluruh alur bisnis yang bergantung padanya [3].

Performance testing adalah evaluasi perilaku sistem di bawah beban kerja tertentu untuk mengidentifikasi bottleneck serta mengukur response time, throughput, dan penggunaan sumber daya [4]. Dua tools yang banyak digunakan adalah k6, framework open-source berbasis JavaScript yang mudah diintegrasikan ke pipeline CI/CD [5], dan Apache JMeter berbasis Java yang mencatat hasil per-request melalui format JTL [6].

Sebagian besar penelitian terdahulu menggunakan satu tools tunggal tanpa perbandingan. Sahputra mengevaluasi performa layanan berbasis web hanya dengan Apache JMeter sebagai instrumen tunggal [7], sedangkan Blinowski et al. membandingkan arsitektur monolitik dan microservices namun tidak membandingkan instrumen pengujiannya [8]. Pola serupa tampak pada penelitian terbaru Rachman et al. yang mengevaluasi performa API hanya dengan satu instrumen [9]. Kajian pemetaan sistematis Hui et al. menegaskan bahwa studi komparatif antar-tools pengujian pada domain microservices masih terbatas [10], sementara studi Sulistiyani dan Rosul terhadap Apache JMeter, Gatling, dan k6 belum menempatkan sistem logistik nyata sebagai objek pengujiannya [11].

Kebaruan penelitian ini terletak pada tiga aspek. Pertama, studi komparatif k6 dan Apache JMeter dilakukan pada lingkungan staging sistem logistik yang sebenarnya, bukan aplikasi simulasi. Kedua, eksperimen terkontrol menerapkan konfigurasi skenario identik pada kedua tools sehingga perbedaan hasil dapat diatribusikan pada karakteristik

tools itu sendiri. Ketiga, objek evaluasi adalah dua endpoint yang merepresentasikan alur bisnis krusial domain logistik, yaitu pickup request sebagai titik awal masuknya data dan manifest sebagai tahap konsolidasi akhir.

2. METODE

Penelitian ini menggunakan pendekatan kuantitatif eksperimental komparatif dengan seluruh data berupa metrik numerik hasil eksekusi pengujian sebagai data primer [12]. Eksperimen terkontrol dipilih karena memberikan validitas internal tertinggi pada rekayasa perangkat lunak [13]. Alur penelitian mencakup penetapan layanan API, perancangan skenario beban, pengembangan skrip, eksekusi pada lingkungan terkontrol, pengumpulan metrik, hingga analisis komparatif.

2.1. Lingkungan dan Objek Pengujian

Pengujian dijalankan dari satu mesin load generator berupa Macbook Air M4 yang menjalankan k6 dan Apache JMeter 5.6 secara bergantian. Targetnya adalah lingkungan staging sistem logistik yang diakses melalui VPN OpenVPN ke jaringan internal perusahaan dengan protokol HTTP/HTTPS REST API, dipilih untuk meminimalkan gangguan terhadap produksi sekaligus menjaga relevansi latensi yang terukur [4].

Objek pengujian adalah dua layanan API. Endpoint /partner/requestpickuppackage (pickreq) merupakan titik masuk alur bisnis logistik, yaitu pembuatan permintaan penjemputan paket dari mitra, dengan volume request tertinggi pada jam puncak. Endpoint /partner/manifest (manifest) merupakan proses konsolidasi data sebelum sortasi sehingga komputasinya lebih intensif karena melibatkan agregasi beberapa paket sekaligus [14]. Keduanya operasi tulis berbasis metode POST.

2.2. Skenario dan Parameter Pengujian

Skenario disusun berdasarkan tiga jenis pengujian performa baku, yaitu load test untuk performa baseline pada kondisi normal, stress test untuk mengidentifikasi titik degradasi melalui peningkatan beban bertahap, dan spike test untuk mengevaluasi elastisitas serta pemulihan terhadap lonjakan mendadak [4], [5]. Kombinasi 2 endpoint, 3 skenario, 2 tools, dan 3 pengulangan menghasilkan 36 sesi pengujian dengan parameter final pada Tabel 1.

Tabel 1. Parameter Skenario Pengujian yang Dieksekusi

Skenario	Endpoint	Profil beban (VU)	Durasi/ulangan
Load Test	pickreq	5–20 VU	5 menit / 3×
Load Test	manifest	5–30 VU	5 menit / 3×
Stress Test	pickreq	5–50 VU, +5 VU per 90 detik	±15 menit / 3×
Stress Test	manifest	5–60 VU, +5 VU per 60 detik	±12 menit / 3×
Spike Test	pickreq	5 → 40 VU dalam 30 detik	±5 menit / 3×
Spike Test	manifest	5 → 60 VU dalam 30 detik	±5 menit / 3×

Kedua skrip menerapkan jeda (pacing) 1 detik antar-request per virtual user serta pembangkitan nomor identifier unik yang identik (receipt_number untuk pickreq dan awbNumber untuk manifest), sehingga laju request dan karakteristik payload kedua tools sepadan. Threshold penilaian mengacu pada standar tingkat layanan Google SRE [15], yaitu response time p95 maksimum 1.000 ms dan p99 maksimum 2.000 ms untuk operasi tulis, error rate maksimum 1%, serta throughput stabil tanpa degradasi lebih dari 20%.

2.3. Teknik Analisis

Analisis dilakukan secara deskriptif dan komparatif. Data mentah keluaran k6 (JSON) dan Apache JMeter (JTL) dibersihkan dengan membuang 30 detik pertama dan terakhir setiap sesi untuk menghindari bias ramp-up, lalu rata-rata tiga pengulangan digunakan sebagai nilai final [4]. Persentil tinggi (p95 dan p99) diprioritaskan karena lebih sensitif terhadap outlier [15], sejalan dengan praktik pengujian REST API yang dirangkum Zhang et al. [16].

Analisis komparatif difokuskan pada konsistensi hasil pengukuran antar tools serta overhead pengukuran, dengan sumber daya dipantau melalui perintah top tiap 2 detik pada proses k6 dan java. Validitas dijaga melalui isolasi waktu pelaksanaan di luar jam puncak, identikalitas parameter kedua skrip, dan replikasi tiga kali [13].

3. HASIL DAN PEMBAHASAN

3.1. Hasil Pengujian Load Test

Load test mengukur performa baseline kedua endpoint pada kondisi operasional normal dengan ringkasan hasil rata-rata tiga pengulangan pada Tabel 2.

Tabel 2. Ringkasan Hasil Load Test (Rata-rata Tiga Pengulangan)

Metrik	k6 – pickreq	JMeter – pickreq	k6 – manifest	JMeter – manifest
Response time p50 (ms)	1.207	1.367	771	977
Response time p90 (ms)	1.988	2.141	1.870	1.971
Response time p95 (ms)	2.249	2.343	2.876	3.263
Response time p99 (ms)	3.505	2.726	5.265	5.338
Throughput (request/detik)	7,43	7,86	12,20	12,31
Error rate (%)	2,46	2,06	1,64	0,50

Response time p95 pickreq tercatat 2.249 ms (k6) dan 2.343 ms (JMeter), sedangkan manifest 2.876 ms dan 3.263 ms, sehingga keempatnya jauh di atas threshold 1.000 ms untuk operasi tulis. Manifest konsisten menunjukkan latensi dasar lebih tinggi daripada pickreq pada kedua tools, sejalan dengan sifat komputasinya yang agregatif [14]. Dari sisi error rate, pickreq mencatat 2,46% (k6) dan 2,06% (JMeter), sedangkan manifest 1,64% dan 0,50% — nilai terakhir satu-satunya kombinasi yang memenuhi threshold 1%.

Analisis data per-request Apache JMeter menunjukkan degradasi progresif pada pickreq meskipun VU konstan, yaitu response time rata-rata naik dari 325 ms menjadi 1.780–1.950 ms pada dua menit terakhir dengan error rate naik dari 0% menjadi 3,6–6,6%, sedangkan pada manifest pola serupa lebih ringan (471 ms menjadi 1.200–1.400 ms) dengan error rate tetap rendah (0–1,6%).

3.2. Hasil Pengujian Stress Test

Stress test menaikkan beban secara bertahap untuk mengidentifikasi pola degradasi dan breaking point. Ringkasan hasil keseluruhan sesi disajikan pada Tabel 3.

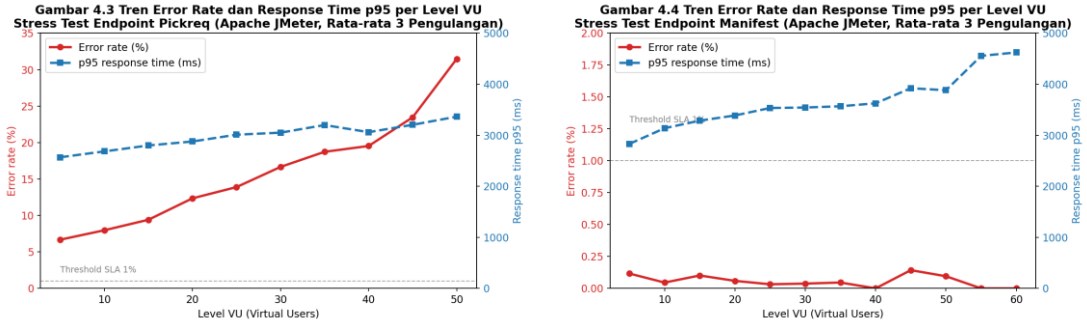
Tabel 3. Ringkasan Hasil Stress Test (Rata-rata Tiga Pengulangan, Keseluruhan Sesi)

Endpoint	Tools	p95 (ms)	p99 (ms)	Maks (ms)*	Throughput (req/detik)	Error rate (%)
pickreq	k6	2.987	3.927	5.038	10,26	14,25
pickreq	JMeter	2.903	3.832	6.783	10,64	11,80
manifest	k6	3.558	5.988	10.255	13,78	1,46
manifest	JMeter	3.398	5.231	10.032	14,02	0,07

*Nilai tertinggi yang teramati dari ketiga pengulangan, bukan rata-rata.

Perbedaan paling mencolok pada Tabel 3 adalah error rate. Pickreq mencatat 14,25% (k6) dan 11,80% (JMeter) pada rentang 5–50 VU, sedangkan manifest hanya

1,46% dan 0,07% meskipun beban puncaknya lebih tinggi dan durasinya lebih panjang. Response time p95 keduanya sebanding, namun p99 dan maksimum manifest jauh lebih tinggi (5.988 ms dan 10.255 ms) dibandingkan pickreq (3.927 ms dan 6.783 ms), yang mengindikasikan tail latency lebih berat meskipun tingkat keberhasilan request-nya jauh lebih baik [3]. Tren per level VU dari data JTL disajikan pada Gambar 1(a) dan 1(b); rincian setara untuk k6 tidak tersedia karena keluaran ringkasannya tidak memuat metrik per-level VU.



Gambar 1. Tren Error Rate dan Response Time p95 per Level VU Stress Test: (a) pickreq dan (b) manifest (Apache JMeter)

Gambar 1(a) menunjukkan degradasi pickreq yang gradual pada dua dimensi sekaligus: error rate naik dari 6,68% (5 VU) menjadi 31,48% (50 VU) sejalan dengan kenaikan p95 dari 2.569 ms menjadi 3.365 ms — pola klasik breaking point akibat keterbatasan konkurensi. Sebaliknya, Gambar 1(b) menunjukkan error rate manifest tetap 0,00–0,14% sepanjang 5 hingga 60 VU sedangkan p95 tetap naik dari 2.831 ms menjadi 4.623 ms — bukti kuantitatif bahwa mekanisme degradasi keduanya berbeda fundamental.

3.3. Hasil Pengujian Spike Test

Spike test dijalankan dengan baseline 5 VU, lonjakan mendadak dalam 30 detik, lalu turun kembali ke baseline. Ringkasan hasil k6 dan Apache JMeter yang dipilah per fase disajikan pada Tabel 4.

Tabel 4. Ringkasan Hasil Spike Test Kedua Endpoint (Rata-rata Tiga Pengulangan)

Metrik	Endpoint	k6 (keseluruhan)	JMeter – baseline (5 VU)	JMeter – lonjakan	JMeter (keseluruhan)
Response time p95 (ms)	pickreq	2.544	2.018	2.631	2.479
Response time p95 (ms)	manifest	2.390	2.188	4.391	4.095
Response time p99 (ms)	pickreq	3.323	-	-	3.394

Response time p99 (ms)	manifest	4.488	5.027	6.674	6.356
Response time maks (ms)*	pickreq	4.663	-	-	4.624
Response time maks (ms)*	manifest	12.557	-	-	11.592
Throughput (request/detik)	pickreq	6,51	-	-	7,50
Throughput (request/detik)	manifest	10,20	-	-	9,96
Error rate (%)	pickreq	8,29	2,15	7,80	5,78
Error rate (%)	manifest	0,00	0,35	0,50	0,45

Fase lonjakan pickreq 35–40 VU dan manifest 55–60 VU. *Nilai tertinggi dari ketiga pengulangan; k6 tidak menyajikan pemilahan per-fase.

Pada pickreq, error rate naik dari 2,15% (baseline) menjadi 7,80% (lonjakan). Pada manifest, kenaikan error rate jauh lebih kecil (0,35% menjadi 0,50%), namun kenaikan p95 lebih tajam secara relatif (2.188 ms menjadi 4.391 ms atau dua kali lipat) dibandingkan pickreq (2.018 ms menjadi 2.631 ms atau sekitar 30%). Perilaku pemulihan juga berbeda: error rate pickreq kembali ke 0,00% dalam kurang dari 30 detik, sedangkan response time rata-rata manifest pada fase pemulihan (543–979 ms) tidak sepenuhnya kembali ke level sebelum lonjakan (226–281 ms) dalam jendela pengamatan dua menit.

3.4. Overhead Sumber Daya dan Evaluasi Threshold SLA/SLO

Overhead sumber daya dipantau pada seluruh 36 sesi pengujian (18 sesi per tools) dengan hasil disajikan pada Tabel 5.

Tabel 5. Perbandingan Overhead Sumber Daya k6 dan Apache JMeter (18 Sesi per Tools)

Aspek	k6	Apache JMeter
CPU rata-rata (%)	2,27	3,90
CPU puncak (%)	6,50	102,70
Memori rata-rata (MB)	35,6	454,9
Memori puncak (MB)	49,0	822,0

k6 menggunakan CPU rata-rata 2,27% dan memori 35,6 MB, sedangkan Apache JMeter 3,90% dan 454,9 MB atau sekitar 13 kali lipat memori k6. Pola tersebut konsisten pada kedua endpoint secara terpisah, yaitu 1,79% dan 33,3 MB berbanding 3,29% dan 446,8 MB pada pickreq, serta 2,75% dan 37,9 MB berbanding 4,51% dan 463,1 MB pada manifest. Perbedaan ini sejalan dengan arsitektur kedua tools, yaitu k6

yang dikompilasi menjadi binari native dengan virtual user berupa goroutine ringan [5] berbanding Apache JMeter yang berjalan di atas Java Virtual Machine [6]. Kedua nilai CPU rata-rata tetap jauh di bawah ambang kewajaran 80%, sehingga mesin load generator tidak menjadi bottleneck yang membiaskan hasil.

Metrik load test yang merepresentasikan kondisi operasional normal kemudian dievaluasi terhadap threshold SLA/SLO [15] dengan kedua endpoint diklasifikasikan sebagai operasi tulis. Rekapitulasi pencapaiannya disajikan pada Tabel 6.

Tabel 6. Evaluasi Hasil Load Test terhadap Threshold SLA/SLO

Endpoint	Metrik	Threshold	Hasil k6	Hasil JMeter	Status
pickreq	Response time p95	≤ 1.000 ms	2.249 ms	2.343 ms	Tidak terpenuhi
pickreq	Response time p99	≤ 2.000 ms	3.505 ms	2.726 ms	Tidak terpenuhi
pickreq	Error rate	≤ 1%	2,46%	2,06%	Tidak terpenuhi
pickreq	Stabilitas throughput	Penurunan ≤ 20%	Tidak dapat dihitung*	17,3%	Terpenuhi (JMeter)
manifest	Response time p95	≤ 1.000 ms	2.876 ms	3.263 ms	Tidak terpenuhi
manifest	Response time p99	≤ 2.000 ms	5.265 ms	5.338 ms	Tidak terpenuhi
manifest	Error rate	≤ 1%	1,64%	0,50%	Campuran**
manifest	Stabilitas throughput	Penurunan ≤ 20%	Tidak dapat dihitung*	5,7%	Terpenuhi (JMeter)

*Keluaran ringkasan k6 tidak memuat data time-series per-request. **Tidak terpenuhi pada k6 (1,64%) namun terpenuhi pada Apache JMeter (0,50%).

Pickreq tidak memenuhi kriteria response time dan error rate pada kedua tools, sedangkan manifest lebih beragam: p95 dan p99 tetap tidak terpenuhi, namun error rate Apache JMeter 0,50% memenuhi threshold 1% dan stabilitas throughput dengan penurunan 5,7% jauh lebih baik dibandingkan pickreq yang turun 17,3%. Tidak satupun endpoint memenuhi seluruh kriteria, dengan response time paling konsisten tidak terpenuhi.

3.5. Pembahasan

Kedua tools menghasilkan pengukuran response time yang konvergen dengan selisih relatif p95 berkisar 2,6–4,0% pada pickreq dan 4,5–13,5% pada manifest di seluruh skenario (Tabel 2 sampai Tabel 4), sejalan dengan hasil Sulistiyani dan Rosul [11]. Selisih yang lebih besar pada manifest kemungkinan dipengaruhi jarak waktu pelaksanaan kedua sesi, mengingat variasi run-to-run pada satu tools saja (645 ms atau

19,8% dari rata-rata p95 JMeter pada load test manifest) sudah melebihi selisih antar tools (387 ms atau 13,5%).

Kontribusi utama penelitian ini adalah identifikasi dua mekanisme degradasi yang berbeda secara fundamental pada dua endpoint dalam satu sistem microservices yang sama. Pickreq menunjukkan degradasi berbasis konkurensi, yaitu error rate naik hampir linear dari 6,68% menjadi 31,48% seiring kenaikan VU, konsisten dengan definisi breaking point klasik [3]. Manifest sebaliknya mempertahankan error rate di bawah 0,15% namun p95 naik dari 2.831 ms menjadi 4.623 ms, sehingga merespons kenaikan beban dengan memperlambat request alih-alih menggagalkannya — konsisten dengan komputasi agregatif dan hipotesis bottleneck throughput-kumulatif pada jalur asinkron [14], sejalan dengan pengamatan bahwa perantara pesan mengubah karakteristik performa layanan [17], [18].

Dibandingkan penelitian terdahulu, kenaikan response time tanpa kenaikan throughput yang sepadan sejalan dengan temuan Blinowski et al. bahwa hubungan pengguna bersamaan dan throughput bersifat non-linear setelah titik jenuh [8]; perbedaannya, penelitian ini memperlihatkan titik jenuh dapat berbeda antar-layanan pada infrastruktur yang sama. Pendekatan identifikasi batas kapasitas serupa dengan Maestro dan Surantha pada sistem perbankan di public cloud [19], dengan perbedaan skala beban yang menegaskan bahwa breaking point merupakan properti kapasitas lingkungan yang diuji — diperkuat temuan Ismail et al. bahwa lingkungan virtual tetap memunculkan kegagalan meskipun spesifikasi server dinaikkan [20].

Secara teoretis, degradasi performa pada arsitektur microservices tidak bersifat tunggal, melainkan dapat mengambil setidaknya dua bentuk: degradasi berbasis konkurensi yang bermanifestasi sebagai kenaikan error rate dan degradasi berbasis durasi yang bermanifestasi sebagai kenaikan response time pada tingkat kegagalan yang tetap rendah. Konsekuensinya, error rate rendah tidak dapat dijadikan satu-satunya indikator kesehatan layanan sehingga persentil tinggi menjadi penting [15].

Secara praktis, prioritas perbaikan sebaiknya dibedakan menurut pola degradasi masing-masing layanan: endpoint yang terdegradasi oleh konkurensi memerlukan penambahan kapasitas permintaan simultan seperti connection pool atau replika layanan, sedangkan endpoint yang terdegradasi oleh durasi beban memerlukan

perbaikan pada jalur pemrosesan asinkron — sebagaimana ditunjukkan penerapan klaster Redis yang memangkas latency hingga 40,6% dan menurunkan beban basis data hingga 58% [21], distribusi beban adaptif [22], dan pemantauan otomatis [23]. Kesenjangan antara p95 pada beban normal (2.249–3.263 ms) dan ambang 1.000 ms juga terlalu besar untuk ditutup melalui penyetelan kecil. Kedua instrumen pun dapat dipakai komplementer: k6 pada pipeline CI/CD dan Apache JMeter untuk sesi diagnostik berkala.

Keterbatasan penelitian ini mencakup keluaran ringkasan k6 yang teragregasi sehingga sejumlah metrik granular hanya dapat disajikan dari data Apache JMeter, serta lingkungan staging bersama yang kondisinya tidak sepenuhnya terkontrol. Akar penyebab kedua mekanisme degradasi memerlukan data observabilitas sisi server, sejalan dengan catatan bahwa penelusuran akar masalah performa merupakan kesulitan utama praktisi microservices [3], celah studi komparatif antar-tools [10], dan kebutuhan penguatan praktik pengujian microservices yang dirangkum Ponce et al. [24].

4. KESIMPULAN

Performa API pickreq dan manifest yang diukur menggunakan k6 dan Apache JMeter pada tiga skenario beban menunjukkan dua mekanisme degradasi yang berbeda secara fundamental. Pickreq mengalami degradasi berbasis konkurensi dengan error rate naik hampir linear dari 6,68% (5 VU) menjadi 31,48% (50 VU) dan p95 dari 2.569 ms menjadi 3.365 ms, sedangkan manifest mengalami degradasi berbasis durasi dengan error rate tetap rendah (0,00–0,14%) namun p95 naik dari 2.831 ms menjadi 4.623 ms. Pada beban normal, p95 kedua endpoint berada pada 2.249–3.263 ms atau jauh di atas threshold 1.000 ms dengan error rate 0,50% hingga 2,46%.

Perbandingan kedua tools pada skenario identik menunjukkan konsistensi tinggi dengan selisih relatif p95 sebesar 2,6–4,0% pada pickreq dan 4,5–13,5% pada manifest, sehingga keduanya menghasilkan penilaian yang konvergen dan dapat dipercaya secara silang. Perbedaan mendasar terletak pada konsumsi sumber daya dan pelaporan: k6 unggul dalam efisiensi (CPU 2,27% dan memori 35,6 MB berbanding 3,90% dan 454,9 MB), sedangkan Apache JMeter unggul dalam kedalaman data melalui format JTL per-request.

Berdasarkan hasil tersebut, direkomendasikan penambahan kapasitas konkurensi untuk endpoint pickreq, optimasi jalur pemrosesan asinkron untuk endpoint manifest, serta pemilihan tools sesuai kebutuhan. Penelitian lanjutan disarankan melakukan diagnosis akar penyebab menggunakan data observabilitas sisi server dan mereplikasi pengujian pada lingkungan yang merepresentasikan produksi.

5. DAFTAR PUSTAKA

[1] Badan Pusat Statistik, Statistik E-Commerce 2024. Jakarta: Badan Pusat Statistik, 2025.

[2] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed. Sebastopol: O'Reilly Media, 2021.

[3] X. Zhou et al., "Revisiting the Practices and Pains of Microservice Architecture in Reality: An Industrial Inquiry," J. Syst. Softw., vol. 195, p. 111521, 2023, doi: 10.1016/j.jss.2022.111521.

[4] J. D. Meier, C. Farre, P. Bansode, S. Barber, and D. Rea, Performance Testing Guidance for Web Applications. Redmond: Microsoft Press, 2007.

[5] Grafana Labs, "k6 Documentation," 2023. [Online]. Available: https://k6.io/docs/

[6] Apache Software Foundation, "Apache JMeter User's Manual," 2023. [Online]. Available: https://jmeter.apache.org/usermanual/index.html

[7] I. Sahputra, "Analisis Performa Website Perpustakaan Universitas Malikussaleh Menggunakan Metode Load Testing dengan Apache JMeter," JATISI, vol. 12, no. 4, 2025, doi: 10.35957/jatisi.v12i4.13652.

[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] A. N. Rachman, R. N. Shofa, I. N. Sjamsuddin, G. N. Tarempa, I. T. Julianto, and B. A. Athoillah, "Load Testing-Based Performance Evaluation of the SiUKT API System," JISTICS, vol. 2, no. 1, pp. 22–29, 2026, doi: 10.64878/jistics.v2i1.89.

[10] M. Hui et al., "Unveiling the Microservices Testing Methods, Challenges, Solutions, and Solutions Gaps: A Systematic Mapping Study," J. Syst. Softw., vol. 220, p. 112232, 2024, doi: 10.1016/j.jss.2024.112232.

[11] E. Sulistiyani and Q. M. Rosul, "Analisis Perbandingan Kinerja Alat Pengujian Beban Perangkat Lunak: Apache JMeter, Gatling, dan K6," J. Inform. Polinema, vol. 12, no. 2, pp. 393–400, 2026, doi: 10.33795/jip.v12i2.8693.

[12] Sugiyono, Metode Penelitian Kuantitatif, Kualitatif, dan R&D, 2nd ed. Bandung: Alfabeta, 2019.

[13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in Software Engineering. Berlin: Springer, 2012.

[14] C. Richardson, Microservices Patterns: With Examples in Java. Shelter Island: Manning Publications, 2018.

[15] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Site Reliability Engineering: How Google Runs Production Systems. Sebastopol: O'Reilly Media, 2016.

[16] M. Zhang, B. Marculescu, and A. Arcuri, "Testing RESTful APIs: A Survey," ACM Trans. Softw. Eng. Methodol., vol. 33, no. 3, 2024, doi: 10.1145/3617175.

[17] S. A. Asri, I. N. G. A. Astawa, I. G. A. M. Sunaya, I. M. R. A. Nugroho, and W. Setiawan, "Implementation of Asynchronous Microservices Architecture on Smart Village Application," Int. J. Adv. Sci. Eng. Inf. Technol., vol. 12, no. 3, 2022, doi: 10.18517/ijaseit.12.3.13897.

[18] R. Bux and G. S. Shenoy, "Performance Analysis of RESTful Web Services and RabbitMQ for Microservices Based Systems on Cloud Environment," in Proc. 3rd Int. Conf. Innov. Technol. (INOCON), 2024, pp. 1–6, doi: 10.1109/INOCON60754.2024.10511747.

[19] A. Maestro and N. Surantha, "Scalability Evaluation of Microservices Architecture for Banking Systems in Public Cloud," in Proc. 3PGCIC 2023, LNDECT vol. 189, Cham: Springer, 2023, pp. 103–115, doi: 10.1007/978-3-031-46970-1_10.

[20] A. Ismail, A. Y. Ananta, S. N. Arief, and E. N. Hamdana, "Performance Testing Sistem Ujian Online Menggunakan JMeter pada Lingkungan Virtual," J. Inform. Polinema, vol. 9, no. 2, pp. 159–164, 2023, doi: 10.33795/jip.v9i2.1190.

[21] F. Dipraja and A. Rahman, "Penerapan Redis Cluster Meningkatkan Efisiensi Caching Arsitektur Microservices," Intellect, vol. 4, no. 1, pp. 171–179, 2025, doi: 10.57255/intellect.v4i1.1445.

[22] P. Zhang, Y. Yang, Z. Song, and L. Xiang, "Adaptive Load Balancing and Fault-Tolerant Microservices Architecture for High-Availability Web Systems Using Docker and Spring Cloud," Discov. Appl. Sci., vol. 7, art. 705, 2025, doi: 10.1007/s42452-025-07320-7.

[23] C. F. Henao Villa et al., "Optimizing Microservices Performance and Scalability Through Automated Monitoring with Kubernetes and Prometheus," Ing. Syst. Inf., vol. 30, no. 2, pp. 551–563, 2025, doi: 10.18280/isi.300225.

[24] F. Ponce, R. Verdecchia, B. Miranda, and J. Soldani, "Microservices Testing: A Systematic Literature Review," Inf. Softw. Technol., vol. 188, p. 107870, 2025, doi: 10.1016/j.infsof.2025.107870.