Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

EVALUASI KINERJA ARSITEKTUR WEBSOCKET-REDIS PADA LINGKUNGAN VPS DENGAN INTEGRASI *PIPELINE* CI/CD

Akhmad Mukti Setiaji*, Fitriasih
1,2) Teknik Informatika, Politeknik Purbaya, Indonesia

Article Info

Article history:

Received: 16 April 2026
Revised: 20 April 2026
Accepted: 21 April 2026

ABSTRACT

Abstrak

Penelitian ini mengevaluasi kinerja arsitektur WebSocket berbasis Redis yang dideploy pada *Virtual Private Server* (VPS) dengan satu vCPU dan 4 GB RAM menggunakan pipeline CI/CD GitHub Actions dan Docker Compose. Tiga konfigurasi diuji: C1 sebagai baseline tanpa Redis pub/sub, C2 dengan Redis pub/sub aktif, dan C3 dengan Redis pub/sub ditambah cache HTTP yang dipanaskan penuh. Pengujian beban menggunakan Grafana K6 mensimulasikan 500 pengguna virtual secara bersamaan. Hasil menunjukkan server mampu mempertahankan 335 koneksi WebSocket bersamaan dengan *zero real errors*. Redis pub/sub meningkatkan rata-rata latensi *broadcast* dari 23,78 ms menjadi 42,05 ms dan memunculkan *spike rate* 1,65%, yang disebabkan oleh *fan-out* O(N) pada *event loop* Node.js libuv, bukan oleh Redis yang latensinya tetap *sub-milidetik*. Redis HTTP caching mengurangi *response time cache-miss* sebesar 39,6–51,4%. Plugin JMeter eu.luminis terbukti tidak kompatibel dengan *endpoint* wss://, menghasilkan HTTP 400 pada 902 koneksi, sehingga Grafana K6 dipilih sebagai alat pengujian. Studi ini berkontribusi pada kuantifikasi *overhead* Redis pub/sub dalam arsitektur WebSocket terkendala satu *core* CPU.

Kata Kunci: : CI/CD, Docker, Event Loop, Grafana K6, Node.js, Redis Pub/Sub, WebSocket

Abstract

This study evaluates the performance characteristics of a real-time WebSocket and Redis architecture deployed on a resource-constrained Virtual Private Server (1 vCPU, 4 GB RAM) using a GitHub Actions CI/CD pipeline and Docker Compose. Three configurations are tested: C1 as a baseline without Redis pub/sub, C2 with Redis pub/sub enabled, and C3 combining Redis pub/sub with a fully warmed HTTP cache. Load tests using Grafana K6 simulate 500 concurrent virtual users. Results show the server sustains 335 simultaneous WebSocket connections with zero real errors. Redis pub/sub increases average broadcast delivery latency from 23.78 ms to 42.05 ms and introduces a 1.65% spike rate, caused by O(N) broadcast fan-out on the Node.js libuv event loop rather than Redis itself, whose round-trip latency remained sub-millisecond. Redis HTTP caching reduces cache-miss response time by 39.6–51.4%. JMeter's eu.luminis plugin proved incompatible with wss:// endpoints, returning HTTP 400 for all 902 connections, motivating adoption of Grafana K6. This work contributes quantitative measurement of Redis pub/sub overhead trade-offs in single-core constrained WebSocket architectures with automated CI/CD performance gates

Keywords: *CI/CD, Docker, Event Loop, Grafana K6, Node.js, Redis Pub/Sub, WebSocket*

Corresponding Author:
E-mail : muktiempires@gmail.com

Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

1. PENDAHULUAN

Perkembangan pesat aplikasi web *real-time* menuntut kemampuan komunikasi yang responsif dan efisien untuk berbagai domain seperti dasbor e-commerce, platform monitoring, dan sistem kolaborasi daring. Protokol WebSocket menyediakan kanal komunikasi *full-duplex* persisten di atas satu koneksi TCP, menghilangkan *overhead* HTTP *polling* berulang. Keunggulan ini telah dibuktikan secara empiris oleh sejumlah penelitian, antara lain Sathya dan Swetha [1] yang menunjukkan WebSocket mencapai latensi rata-rata 30 ms dibandingkan 100 ms pada HTTP *polling*, serta penggunaan bandwidth yang 50% lebih efisien. Sejalan dengan itu, Fernando dan Engel [2] melakukan *benchmarking* library WebSocket di Node.js dan Golang dengan skenario *broadcast* pada 1.000 klien bersamaan, memperkuat posisi WebSocket sebagai pilihan utama untuk komunikasi *real-time* berskala tinggi. Implementasi WebSocket berbasis PHP juga telah menunjukkan kemampuan koneksi *real-time* yang andal dalam lingkungan web modern [3].

Meskipun WebSocket menyediakan mekanisme koneksi yang efisien, arsitektur *single-instance* tidak mampu mendistribusikan *broadcast* secara horizontal ke banyak proses server. Redis, sebagai penyimpan *key-value in-memory* berkinerja tinggi, mengisi peran ganda: sebagai *message broker* publish/subscribe (pub/sub) untuk koordinasi *broadcast* WebSocket lintas instance, dan sebagai cache response HTTP untuk mengurangi tekanan query database. Sammir et al. [4] mengimplementasikan Redis pub/sub untuk sistem monitoring IoT secara *real-time* dan membuktikan kemampuannya menangani pengiriman pesan latensi rendah dalam arsitektur terdistribusi. Selain itu, Mali et al. [5] merancang platform pencarian kerja *real-time* menggunakan Redis pub/sub yang mampu memfasilitasi notifikasi instan kepada banyak pengguna secara bersamaan. Dari perspektif caching, Shi [6] membuktikan bahwa Redis sebagai *distributed cache* secara signifikan mengurangi beban disk I/O database pada skenario akses data frekuensi tinggi, dan Dipraja serta Rahman [7] mengkonfirmasi penurunan latensi API sebesar 40,6% menggunakan Redis Cluster pada arsitektur *microservices* berbasis Docker Compose.

Dari sisi pengelolaan *deployment*, containerisasi menggunakan Docker dan *pipeline Continuous Integration/Continuous Deployment* (CI/CD) telah menjadi standar industri. Chintagunta [8] mendokumentasikan bahwa containerisasi melalui Docker dan orkestrasi CI/CD memungkinkan *deployment* yang konsisten dan bebas perbedaan konfigurasi lingkungan. Sobieraj dan Kotynski [9] mengevaluasi performa Docker di berbagai sistem operasi dan mengkonfirmasi *overhead* yang minimal pada platform Linux, menjadikannya pilihan optimal untuk *deployment* VPS. Dalam konteks *pipeline* CI/CD, Faqih et al.[10] menganalisis penggunaan empiris GitHub Actions dalam *workflow* CI/CD dan membuktikan efisiensinya sebagai alat otomasi DevOps modern. Rostami Mazrae et al. [11] mengkaji migrasi dan ko-penggunaan alat CI/CD, menegaskan tren adopsi GitHub Actions sebagai platform utama DevOps modern. Pinyagin dan Sadovykh [12] mengevaluasi berbagai alat pengujian beban dalam *pipeline* CI/CD, termasuk

Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

perbandingan K6 dan JMeter, sebagai metodologi inti yang harus terotomasi untuk mencegah regresi performa memasuki lingkungan produksi.

Meskipun penelitian-penelitian di atas membuktikan keunggulan masing-masing komponen secara terpisah, kombinasi WebSocket, Redis pub/sub, Redis HTTP cache, dan *pipeline* CI/CD terotomasi dalam satu sistem yang diuji secara kuantitatif pada *hardware* terkendala masih merupakan celah penelitian yang belum terjawab. Fernando dan Engel [2] tidak menyertakan Redis pub/sub dalam jalur *broadcast* WebSocket mereka. Pinyagin dan Sadovykh [12] membandingkan alat pengujian beban dalam *pipeline* Jenkins namun tanpa workload WebSocket maupun Redis. Privalov dan Stupina [13] membuktikan efektivitas Redis caching pada aplikasi web namun tanpa evaluasi beban WebSocket bersamaan. Dari kajian tersebut teridentifikasi tiga celah penelitian: (1) belum ada pengukuran kuantitatif *overhead* latensi Redis pub/sub pada *broadcast* WebSocket di bawah beban pengguna bersamaan pada hardware satu vCPU; (2) belum ada kerangka eksperimen yang menggabungkan WebSocket *stress test*, pengukuran HTTP cache, dan *pipeline* gate CI/CD dalam satu studi empiris; dan (3) inkompatibilitas plugin JMeter eu.luminis dengan *endpoint* wss:// belum terdokumentasi dalam literatur akademik.

Tabel 1 Pemetaan Penelitian Terdahulu dan Celah Penelitian

Penelitian	**Protokol/Stack**	**Peran Redis**	**Gate CI/CD**	**Celah vs Studi Ini**
Sathya & Swetha [1], IJSRST 2025	WebSocket vs HTTP	Tidak digunakan	Tidak ada	Tanpa Redis broker, beban massal, atau *pipeline* CI/CD
Fernando & Engel [2], Sinkron 2024	ws, socket.io vs Golang gorilla	Tidak digunakan	Tidak ada	Tanpa Redis pub/sub dalam jalur *broadcast*; tanpa CI/CD
Privalov & Stupina [13], IJEECS 2024	Web app, Spring Boot + Redis	Cache HTTP	Tidak ada	Tanpa beban WebSocket bersamaan atau *pipeline* CI/CD
Pinyagin & Sadovykh [12], Springer 2026	Spring Boot HTTP	Tidak digunakan	Jenkins + 5 tools	HTTP only; tanpa WebSocket atau Redis
Faqih et al. [10], JIWE 2024	GitHub Actions vs manual	Tidak digunakan	GitHub Actions	Tanpa WebSocket, tanpa Redis, tanpa pengujian beban
Studi Ini, 2026	Node.js, Redis 7, Nginx, Docker, GitHub Actions	Pub/sub + cache HTTP (keduanya diukur)	GitHub Actions + K6 gate	**Novel: *overhead* pub/sub + cache + CI/CD gate di VPS 1-vCPU**

Sumber: Penulis, 2026

2. METODE PENELITIAN

Sistem yang dievaluasi adalah aplikasi web *full-stack* yang dideploy pada *Virtual Private Server* (VPS) berbasis Ubuntu 22.04 LTS dengan spesifikasi 1 vCPU dan 4 GB RAM. Stack teknologi mencakup Node.js 20 LTS, Express 4.18, Redis 7.2, Next.js 14.2, dan Nginx 1.24. Seluruh komponen dikontainerisasi menggunakan Docker 26.0 dan dikelola oleh Docker Compose v2 dengan tiga service: nginx sebagai *reverse proxy* TLS *termination* (port 443), express-app sebagai WebSocket dan REST API server (port 3002), dan redis menggunakan image redis:7-alpine dengan parameter --maxmemory 512mb --maxmemory-policy allkeys-lru. Pendekatan containerisasi ini memungkinkan

Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

reprodusibilitas *deployment* dan konsistensi lingkungan eksekusi [8], sekaligus meminimalkan *overhead* sistem operasi yang dikonfirmasi oleh Sobieraj dan Kotynski [9] pada platform Linux.

Temuan konfigurasi kritis ditemukan selama *deployment* awal: direktif http2 harus dihapus dari blok server HTTPS Nginx. Dokumentasi resmi Nginx [14] menjelaskan bahwa WebSocket adalah protokol *hop-by-hop* yang memerlukan penanganan khusus pada proxy, di mana Nginx harus meneruskan header Upgrade secara eksplisit menggunakan proxy_http_version 1.1. HTTP/2 menggunakan *binary framing layer* pada koneksi TCP yang termultipleks, yang secara struktural tidak kompatibel dengan mekanisme Upgrade WebSocket yang memerlukan pertukaran header HTTP/1.1 *plaintext*. Mempertahankan direktif http2 menyebabkan *worker* Nginx *crash* saat menerima permintaan upgrade WebSocket, menghasilkan HTTP 400 pada semua klien. Konfigurasi blok location /ws menggunakan proxy_http_version 1.1, header Upgrade: websocket, Connection: Upgrade, dan proxy_read_timeout 86400s untuk mencegah pemutusan koneksi idle jangka panjang.

Pipeline CI/CD diimplementasikan menggunakan GitHub Actions yang dipicu pada setiap push ke *branch* main. Pendekatan ini sejalan dengan temuan Faqih et al. [10] dan studi CI/CD strategies oleh Divya [15] yang menegaskan bahwa integrasi pengujian otomatis dalam *pipeline* secara signifikan mengurangi waktu deteksi *bug* dan meningkatkan keandalan *deployment*. *Pipeline* menjalankan tujuh tahapan berurutan: (1) checkout repository via actions/checkout@v4; (2) build Docker image dan push ke registry; (3) SSH ke VPS menggunakan secret HOST, USERNAME, SSH_KEY; (4) deploy dengan docker compose pull && up -d --no-deps express-app; (5) verifikasi health check melalui wget ke *endpoint* /api/health; (6) eksekusi K6 smoke test minimal 5 VU selama 30 detik; dan (7) evaluasi gate *pipeline* dengan kriteria gagal build jika ws_delivery_latency_ms p(95) > 100 ms atau ws_real_errors > 0%. Pendekatan *shift-left* ini sesuai dengan kerangka Pinyagin dan Sadovykh [12] untuk mencegah regresi performa memasuki produksi.

Tiga konfigurasi eksperimen dirancang untuk mengisolasi kontribusi independen setiap kapabilitas Redis. Konfigurasi C1 menggunakan in-process Node.js EventEmitter *broadcasting* tanpa Redis dalam jalur *broadcast* WebSocket, berfungsi sebagai *baseline*. Konfigurasi C2 mengaktifkan Redis pub/sub secara penuh: server Express mempublikasikan system_stats ke channel Redis setiap 1 detik. Konfigurasi C3 menambahkan Redis product cache yang telah dipanaskan penuh dengan TTL 60 detik, merepresentasikan kondisi produksi *steady-state*. Pendekatan multi-konfigurasi ini memungkinkan pengukuran *overhead* yang presisi, konsisten dengan metodologi eksperimen terkontrol yang diadvokasi dalam penelitian kinerja Redis [13].

Pengujian awal dengan Apache JMeter 5.6.3 menggunakan plugin WebSocket eu.luminis v0.2.3 menghasilkan 100% kegagalan pada 902 koneksi dengan respons HTTP 400 terhadap *endpoint* wss://macdevtech.site/ws. Log akses Nginx mengkonfirmasi permintaan tiba tanpa header Upgrade: websocket yang diperlukan, menunjukkan

Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

bahwa plugin eu.luminis tidak menyelesaikan sesi TLS sebelum mengirimkan permintaan upgrade HTTP. Pinyagin dan Sadovykh [12] mengkonfirmasi superioritas K6 berbasis *goroutine* Go untuk pengujian terotomasi dalam pipeline CI/CD, dengan konsumsi resource yang lebih efisien dibandingkan JMeter yang berbasis thread-per-user. Grafana K6 v0.52 menggunakan paket crypto/tls native Go yang mereplikasi perilaku TLS + WebSocket *handshake* setara browser sehingga berhasil terkoneksi [16].

Latensi pengiriman WebSocket diukur menggunakan timestamp Unix millisecond publishedAt dalam setiap *broadcast* server. Handler pesan K6 merekam Date.now() - msg.publishedAt untuk mengukur latensi *end-to-end*. Rumus analisis yang diterapkan mencakup:

Persentase peningkatan latensi:

$$\boldsymbol{Improvement}(\%) = \left(\frac{\left(\boldsymbol{C1\_miss_{avg}} - \boldsymbol{Cx\_miss_{avg}}\right)}{\boldsymbol{C1\_miss_{avg}}}\right) \times \mathbf{100}$$

Overhead pub/sub Redis:

$$\boldsymbol{PubSub_{overhead}} = \boldsymbol{avg_{latency(C2)}} - \boldsymbol{avg_{latency(C1)}} = \mathbf{42.05} - \mathbf{23.78} = \mathbf{18.27}\ \boldsymbol{ms}$$

Spike rate *broadcast*:

$$\boldsymbol{Spike_{rate}}(\%) = \left(\frac{\boldsymbol{spikes\_over\_500ms}}{\boldsymbol{total\_messages}} \times \mathbf{100}\right)$$

Kompleksitas algoritmik *fan-out*:

$$\boldsymbol{Fan-out\ complexity} = \boldsymbol{O(N)}, \boldsymbol{N} = \boldsymbol{jumlah\ subscriber\ WebSocket\ bersamaan}$$

Manfaat konteksi CPU:

$$\boldsymbol{CPU\_benefit} = \boldsymbol{HTTP_{fail(C2)}} - \boldsymbol{HTTP_{fail(C3)}} = \mathbf{6.77}\% - \mathbf{5.93}\% = \mathbf{0.84}$$

Tabel 2 Konfigurasi Lingkungan Pengujian dan *Deployment*

Parameter	**Nilai**
Server *hardware*	1 vCPU, 4 GB RAM, Ubuntu 22.04 LTS
Application stack	Node.js 20 LTS, Express 4.18, Redis 7.2, Next.js 14.2, Nginx 1.24
Container runtime	Docker 26.0 + Docker Compose v2; services: nginx, express-app, redis
Redis config	redis:7-alpine, --maxmemory 512mb, --maxmemory-policy allkeys-lru, TTL 60s
Nginx WS config	proxy_http_version 1.1; Upgrade: websocket; Connection: Upgrade; http2 dihapus
CI/CD pipeline	GitHub Actions: 7 tahap — build Docker → SSH deploy → K6 smoke gate

Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

Load testing tool	Grafana K6 v0.52 — Go runtime, native TLS, k6/ws module
C1 configuration	In-process broadcast via EventEmitter; Redis aktif; HTTP cache dinonaktifkan
C2 configuration	Redis pub/sub aktif; PUBLISH system_stats tiap 1 detik; cache HTTP parsial
C3 configuration	Redis pub/sub + cache HTTP dipanaskan penuh; TTL 60s; allkeys-lru
Stress test profile	500 VU: ramp 100 (30s) → sustain 100 (1m) → ramp 300 (30s) → sustain 300 (1m) → spike 500 (30s)
HTTP cache test profile	300 VU: ramp 100 (30s) → sustain 300 (2m) → ramp-down (30s)

Sumber: Penulis, 2026

3. HASIL DAN PEMBAHASAN

Pengujian stress test 500 VU menghasilkan data komprehensif mengenai kapasitas dan karakteristik latensi sistem. Tabel 3 merangkum seluruh metrik utama dari ketiga konfigurasi. Koneksi WebSocket bersamaan maksimum yang teramati adalah 335–337 di seluruh konfigurasi, meskipun target VU adalah 500. Hal ini merepresentasikan batas kapasitas aktual server di bawah beban HTTP dan WebSocket gabungan pada satu core CPU, konsisten dengan pernyataan Fernando dan Engel [2] bahwa *bottleneck* pada beban *broadcast* berat berpindah ke limitasi runtime platform. Sekitar 165 VU berada dalam antrian *handshake* TLS pada titik puncak, dengan 163–170 iterasi terganggu karena *timeout* sebelum tes selesai.

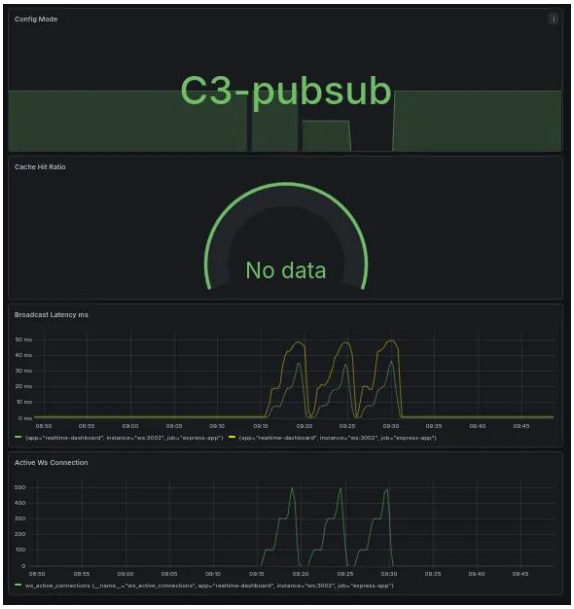
Tabel 3 Hasil Stress Test WebSocket 500 VU — Tiga Konfigurasi

Metrik	**C1 (Baseline)**	**C2 (Pub/Sub)**	**C3 (Pub/Sub+Cache)**
Avg delivery latency (ms)	23,78	42,05	40,13
Median delivery latency (ms)	21	22	23
p(95) delivery latency (ms)	48	67	61
Max delivery latency (ms)	171 ✓	1030 ✗	1000 ✗
Spikes > 500 ms (count / %)	0 / 0,00% ✓	920 / 1,65% ✗	929 / 1,67% ✗
Broadcast msgs/s	199,9	206,6 ✓	206,0 ✓
Real WebSocket errors	0,00% ✓	0,00% ✓	0,00% ✓
Redis round-trip avg (ms)	0,457	0,447	0,508
Express event loop avg (ms)	7,14	7,46	6,68

Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

Max simultaneous WS connections	335	336	337
HTTP failure rate (beban gabungan)	6,31% ✗	6,77% ✗	5,93%
WS handshake median — same VPS (ms)	4,93 ✓	4,76 ✓	5,30 ✓

Sumber: Data primer hasil pengujian K6, 2026



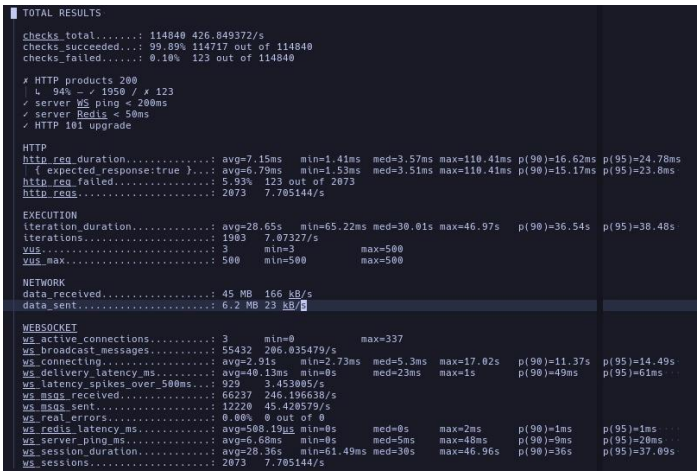
Gambar 1 Tampilan dashboard Grafana selama pengujian stress 500 VU

Sumber: Tangkapan layar Grafana dashboard, 2026

Konfigurasi C1 mencapai latensi pengiriman terendah (23,78 ms rata-rata, 48 ms p(95), 0 spike) karena menggunakan in-process Node.js EventEmitter tanpa jalur Redis dalam alur *broadcast*. Konfigurasi C2 dan C3 menunjukkan latensi lebih tinggi (40–42 ms rata-rata) dengan 920–929 spike di atas 500 ms per 55.000 pesan total. Krusialnya, Redis round-trip tetap *sub-milidetik* (0,447–0,508 ms) di semua konfigurasi, mengkonfirmasi bahwa Redis bukan *bottleneck* konsisten dengan karakteristik latensi Redis yang terdokumentasi [17]. Penyebab spike adalah kompleksitas O(N) *fan-out* broadcast pada *event loop* Node.js libuv [18]. Ketika Redis mengirimkan notifikasi pub/sub, *callback* Node.js mengiterasi seluruh 335 objek koneksi dan mengeksekusi ws.send() secara berurutan. Setiap pemanggilan send() memerlukan operasi sinkron CPU-bound: alokasi memori, serialisasi JSON, dan WebSocket *frame masking*. Operasi XOR bitwise masking tidak dapat didelegasikan ke *thread* pool I/O, sehingga 335 operasi masking sekuensial pada VPS 1 vCPU menciptakan *event loop lag* yang bermanifestasi sebagai spike latensi mengkonfirmasi temuan Fernando dan Engel [2] bahwa workload *broadcast* berat menggeser *bottleneck* ke platform runtime. Overhead pub/sub sebesar 18,27 ms (Persamaan 2) harus dievaluasi sebagai biaya skalabilitas horizontal, bukan regresi performa, karena tanpa Redis pub/sub, konfigurasi C1 tidak dapat mendukung multi-

Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

instance Express yang berbagi channel *broadcast* yang sama sebagaimana diimplementasikan Sammir et al. [4] dan Mali et al. [5] dalam sistem mereka.



Gambar 2 Output lengkap pengujian K6 konfigurasi C3 — 500 VU

Sumber: Terminal output K6, 2026

Tabel 4 menyajikan hasil pengujian latensi HTTP API terpisah dengan 300 VU. Menggunakan Persamaan 1, peningkatan C2 = ((103,67 – 50,33) / 103,67) × 100 = 51,4%; peningkatan C3 = ((103,67 – 62,56) / 103,67) × 100 = 39,6%. Temuan ini konsisten dengan Shi [6] yang melaporkan Redis *distributed cache* mampu mengurangi beban disk I/O secara signifikan, serta Privalov dan Stupina [13] yang membuktikan peningkatan efisiensi sistem web berbasis caching Redis. Rata-rata *cache miss* C2 (50,33 ms) lebih rendah dari C3 (62,56 ms) karena cache C2 baru dipopulasi tanpa TTL expiry. Pada C3, beberapa key telah kedaluwarsa sehingga memerlukan re-fetch dan re-store database perilaku realistis yang merepresentasikan *churn* TTL di bawah beban berkelanjutan. Hartanto et al. [19] menemukan fenomena serupa di mana *warming state* cache secara langsung memengaruhi efisiensi web service berbasis Redis.

Tabel 4 Hasil Pengujian Latensi HTTP API — 300 VU, C1 vs C2 vs C3

Metrik	**C1 (no cache)**	**C2 (parsial)**	**C3 (dipanaskan)**
Cache miss avg (ms)	103,67	50,33 ✓	62,56
Cache miss p(95) (ms)	309,70	150,05 ✓	180,09
Cache hit avg (ms)	8,52†	4,13 ✓	5,59
Cache hit p(95) (ms)	27,45†	14,59 ✓	21,41
HTTP error rate	0,00% ✓	0,00% ✓	0,00% ✓
Throughput (req/s)	328	329	329
Peningkatan vs C1 miss avg	— (baseline)	51,4% lebih cepat ✓	39,6% lebih cepat ✓

Sumber: Penulis, 2026

Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

Failure rate HTTP sebesar 5,93–6,77% pada pengujian beban gabungan tidak muncul dalam pengujian HTTP khusus (0,00% error), mengkonfirmasi CPU *contention* sebagai mekanisme kegagalan. VPS satu core harus secara bersamaan menangani TLS *termination* untuk koneksi HTTP baru dan mengeksekusi O(N) WebSocket *broadcast fan-out*. C3 mencapai failure rate terendah (5,93%) karena product caching menghilangkan siklus CPU untuk serialisasi query database, membebaskan *event loop* menerima lebih banyak koneksi bersamaan. Manfaat kumulatif ini sebesar 0,84 poin persentase (Persamaan 5) hanya terlihat di bawah beban gabungan temuan yang tidak dapat diobservasi dari pengujian *single-workload* secara terpisah. Fenomena CPU *contention* ini relevan dengan strategi caching dan batching yang dibahas dalam studi kinerja microservice [20], di mana penghematan sumber daya komputasi pada satu lapisan memberikan dampak positif berlipat pada lapisan lain.

Dibandingkan dengan penelitian terdahulu, studi ini memberikan kontribusi empiris yang lebih spesifik. Fernando dan Engel [2] mengidentifikasi *event loop* Node.js sebagai bottleneck broadcast berat; studi ini mengkuantifikasinya secara presisi: 1,65% *spike rate* di atas 500 ms pada N = 335 subscriber bersamaan, dengan Redis round-trip terbukti tidak berkontribusi pada spike (0,447–0,508 ms konstan). Privalov dan Stupina [13] membuktikan efektivitas Redis caching; studi ini menambahkan dimensi beban WebSocket bersamaan yang menunjukkan manfaat cache berupa pengurangan CPU *contention* hanya terukur pada pengujian gabungan. Pinyagin dan Sadovykh [12] mengkonfirmasi superioritas K6 dalam CI/CD; studi ini melengkapi dengan mendokumentasikan inkompatibilitas JMeter eu.luminis pada wss:// yang belum terdokumentasi dalam literatur akademik. Rostami Mazrae et al. [11] mengkaji evolusi penggunaan alat CI/CD; studi ini mengimplementasikan dan mengevaluasi *pipeline* GitHub Actions dengan K6 threshold gate secara empiris pada aplikasi *real-time*.

Sistem yang dievaluasi memiliki kekuatan: zero real WebSocket errors di seluruh konfigurasi, Redis round-trip *sub-milidetik* yang membuktikan broker bukan *bottleneck*, dan konsistensi peningkatan *cache miss* 39,6–51,4%. Keterbatasan mencakup: eksperimen single-node tidak memvalidasi skalabilitas horizontal secara empiris; tidak adanya *soak test* 24 jam untuk deteksi *memory leak* WebSocket; dan seluruh pengujian berasal dari satu lokasi geografis.

4. SIMPULAN

Penelitian ini membuktikan secara empiris bahwa VPS satu core 4 GB RAM yang menjalankan arsitektur WebSocket-Redis berbasis Docker dalam pipeline CI/CD GitHub Actions mampu mempertahankan 335 koneksi WebSocket bersamaan dengan *zero real errors* di seluruh tiga konfigurasi yang diuji, menjawab pertanyaan utama penelitian mengenai kapasitas dan karakteristik kinerja sistem terkendala sumber daya . Redis pub/sub terukur menambah *overhead* latensi *broadcast* sebesar 18,27 ms rata-rata dengan *spike rate* 1,65% pada N = 335 subscriber bersamaan, yang disebabkan oleh kompleksitas O(N) *fan-out* pada *event loop* libuv Node.js bukan oleh Redis itu sendiri yang

Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

round-trip-nya tetap *sub-milidetik* , sehingga *overhead* tersebut merupakan biaya skalabilitas horizontal yang dapat diterima bukan regresi performa . Redis HTTP caching mengurangi *response time cache-miss* sebesar 39,6–51,4% dengan manfaat kumulatif tambahan berupa pengurangan HTTP failure rate 0,84 poin persentase di bawah beban gabungan WebSocket dan HTTP melalui mekanisme pengurangan CPU *contention*, sebuah temuan yang tidak dapat diobservasi dari pengujian *single-workload*. Inkompatibilitas plugin JMeter eu.luminis dengan *endpoint* wss:// yang menghasilkan 100% kegagalan (902 dari 902 koneksi, HTTP 400) terdokumentasikan untuk pertama kalinya dalam literatur akademik, mengarahkan praktisi pada Grafana K6 sebagai alat yang tepat untuk pengujian WebSocket berproteksi TLS dalam *pipeline* CI/CD terotomasi.

REFERENCES

[1] Dr. S. Sathya and Ms. S. Swetha, "Real-Time Chat Application with Web Socket for Network Efficiency," *Int. J. Sci. Res. Sci. Technol.*, vol. 12, no. 4, pp. 830–835, Aug. 2025, doi: 10.32628/IJSRST251362.

[2] L. Fernando and M. M. Engel, "Comparative Performance Benchmarking of WebSocket Libraries on Node.js and Golang," *sinkron*, vol. 9, no. 4, pp. 2051–2060, Oct. 2025, doi: 10.33395/sinkron.v9i4.15266.

[3] Friendly, A. P. Sembiring, S. Faza, A. Lukcyhasnita, and R. Destiadi, "Design and Implementation of IOT Connection With Websocket Using PHP," *International Journal of Research in Vocational Studies (IJRVOCAS)*, vol. 2, no. 4, pp. 94–98, Jan. 2023, doi: 10.53893/ijrvocas.v2i4.173.

[4] H. Sammir and K. Hamdi, "Implementasi Protokol Redis Pub/Sub Menggunakan Python untuk Sistem Monitoring Suhu IoT Secara Real-Time," 2026. [Online]. Available: http://jurnal.unidha.ac.id/index.php/jiska

[5] A. B. Mali, I. Khan, M. M. K. Dandu, Prof. (Dr) P. Goel, Prof. (Dr.) A. Jain, and Er. A. Shrivastav, "Designing Real-Time Job Search Platforms with Redis Pub/Sub and Machine Learning Integration," *Journal of Quantum Science and Technology*, vol. 1, no. 3, Aug. 2024, doi: 10.63345/jqst.v1i3.115.

[6] L. Shi, "Research and Application of Distributed Cache Based on Redis," *Journal of Software*, pp. 1–8, Jan. 2024, doi: 10.17706/jsw.19.1.1-8.

[7] F. Dipraja and A. Rahman, "Penerapan Redis Cluster Meningkatkan Efisiensi Caching Arsitektur Microservices," *Intellect : Indonesian Journal of Learning and Technological Innovation*, vol. 4, no. 1, pp. 171–179, Oct. 2025, doi: 10.57255/intellect.v4i1.1445.

[8] Satyadhar Kumar Chintagunta, "Survey of Containerization, Orchestration, and CI/CD Integration on DevOps in Modern Software Development," *International Journal of Current Engineering and Technology*, vol. 13, no. 6, pp. 610–618, [Online]. Available: https://ijcet.evegenis.org/index.php/ijcet/article/view/825

[9] M. Sobieraj and D. Kotyński, "Docker Performance Evaluation across Operating Systems," *Applied Sciences*, vol. 14, no. 15, p. 6672, Jul. 2024, doi: 10.3390/app14156672.

[10] A. R. Faqih, A. Taufiqurrahman, J. H. Husen, and M. K. Sabariah, "Empirical Analysis of CI/CD Tools Usage in GitHub Actions Workflows," *Journal of Informatics and Web Engineering*, vol. 3, no. 2, pp. 251–261, Jun. 2024, doi: 10.33093/jiwe.2024.3.2.18.

Djtechno : Jurnal Teknologi Informasi
Vol. 7, No. 1, April 2026
E-ISSN: 2745-3758, P-ISSN : 2776-8546 DOI: 10.46576/djtechno

[11] P. Rostami Mazrae, T. Mens, M. Golzadeh, and A. Decan, "On the usage, co-usage and migration of CI/CD tools: A qualitative analysis," *Empir. Softw. Eng.*, vol. 28, no. 2, p. 52, Mar. 2023, doi: 10.1007/s10664-022-10285-5.

[12] M. Pinyagin and A. Sadovykh, "Automating Performance Testing in CI/CD - Tools Evaluation," 2026, pp. 195–209. doi: 10.1007/978-3-032-05188-2_13.

[13] M. V. Privalov and M. V. Stupina, "Improving web-oriented information systems efficiency using Redis caching mechanisms," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 33, no. 3, p. 1667, Mar. 2024, doi: 10.11591/ijeecs.v33.i3.pp1667-1675.

[14] Nginx Inc, "WebSocket proxying." Accessed: Apr. 06, 2026. [Online]. Available: https://nginx.org/en/docs/http/websocket.html

[15] Divya Kodi, "Efficient CI/CD Strategies: Integrating Git with automated testing and deployment," *World Journal of Advanced Research and Reviews*, vol. 20, no. 2, pp. 1517–1530, Nov. 2023, doi: 10.30574/wjarr.2023.20.2.2363.

[16] Grafana Labs, "WebSockets | Grafana k6 documentation." Accessed: Apr. 06, 2026. [Online]. Available: https://grafana.com/docs/k6/latest/using-k6/protocols/websockets/

[17] Redis Ltd, "Redis Pub/sub | Docs." Accessed: Apr. 06, 2026. [Online]. Available: https://redis.io/docs/latest/develop/pubsub/

[18] NodeJs Foundation, "Don't Block the Event Loop (or the Worker Pool) | Node.js v24.14.1 Documentation." Accessed: Apr. 06, 2026. [Online]. Available: https://nodejs.org/learn/asynchronous-work/dont-block-the-event-loop

[19] Mb. Hartanto, T. Muhammad Fawa, and D. P. Eko Hendro, "ANALISA KINERJA DATABASE DAN IMPLEMENTASI CACHE REDIS PADA WEB SERVICE LUMEN," 2023. [Online]. Available: https://lumen.laravel.com/,https://github.com/laravel/lumen,https://laravel.com/docs

[20] G. Swapna, E. Susmitha, N. Satyanandaram, and S. V. Punith Kumar, "Enhancing Microservice Performance: A Hybrid Approach Using Caching and Batching Techniques," *International Research Journal on Advanced Engineering Hub (IRJAEH)*, vol. 4, no. 02, pp. 766–775, Feb. 2026, doi: 10.47392/IRJAEH.2026.0110.