UAS Sistem Paralel dan Terdistribusi

“ Sistem Event Aggregator Terdistribusi “

Bagian Teori:

T1 (Bab 1): Karakteristik utama sistem terdistribusi dan trade-off pada desain Pub-Sub log aggregator.

Jawaban:

Mengacu pada Coulouris et al. (2012, Bab 1), terdapat tiga ciri mendasar sistem terdistribusi: concurrency of components, ketiadaan global clock, dan independent failures. Dalam pengerjaan UAS ini, aspek concurrency menjadi fokus utama mengingat layanan Aggregator, Publisher, dan Database beroperasi secara simultan dalam container Docker. Hal ini merefleksikan trade-off antara Scalability dan Complexity sebagaimana dipaparkan oleh Van Steen & Tanenbaum (2023, Bab 1). Meskipun pemisahan sistem ke dalam multi-service meningkatkan skalabilitas karena beban kerja terdistribusi, kompleksitas sinkronisasi turut meningkat akibat absennya waktu global yang mutlak serta adanya risiko kegagalan parsial (independent failures), di mana kegagalan satu container tidak menyebabkan seluruh sistem berhenti.

T2 (Bab 2): Bandingkan arsitektur client-server vs publish-subscribe untuk aggregator. Kapan memilih Pub-Sub?

Jawaban:

Coulouris et al. (2012, Bab 2) menyebutkan bahwa arsitektur Client-Server memiliki kelemahan berupa sentralisasi yang berpotensi menjadi bottleneck ketika trafik klien melonjak. Di sisi lain, Van Steen & Tanenbaum (2023, Bab 2) merekomendasikan model Publish-Subscribe untuk skenario yang memerlukan decoupling (pemisahan) baik secara ruang maupun waktu. Oleh karena itu, arsitektur Pub-Sub dipilih untuk aggregator log ini guna memfasilitasi komunikasi asinkron (asynchronous communication). Publisher dapat mengirimkan ribuan event tanpa harus memblokir proses menunggu Aggregator selesai menulis ke Postgres. Pendekatan ini efektif mencegah penumpukan antrian di sisi pengirim yang kerap menjadi masalah pada model sinkron Client-Server.

T3 (Bab 3): At-least-once vs exactly-once delivery; peran idempotent consumer.

Jawaban:

Berdasarkan penjelasan Coulouris et al. (2012, Bab 5), mekanisme at-least-once delivery memastikan pesan terkirim minimal satu kali lewat metode retry, namun membawa risiko duplikasi jika sinyal acknowledgement terputus. Mengingat kondisi exactly-once murni sangat sulit dicapai pada jaringan yang tidak reliabel (Van Steen & Tanenbaum, 2023, Bab 4), maka peran Idempotent Consumer menjadi sangat vital. Sistem ini dirancang untuk menerima kemungkinan

duplikasi (at-least-once) dari jaringan, namun aplikasi menetralkannya menggunakan logika idempotensi (melalui Unique Constraint di Database). Dengan demikian, hasil akhir pemrosesan data setara dengan exactly-once, menjaga integritas data walau terjadi pengiriman ulang.

T4 (Bab 4): Skema penamaan topic dan event_id (unik, collision-resistant) untuk dedup.

Jawaban:

Sistem penamaan wajib menjamin keunikan entitas, sebagaimana ditekankan Coulouris et al. (2012) mengenai identifier unik global. Desain ini menerapkan skema Composite Identifier yang menggabungkan topic dan event_id sebagai identitas tunggal. Van Steen & Tanenbaum (2023, Bab 6) mengategorikan ini sebagai flat naming yang merujuk langsung pada entitas data. Pemanfaatan UUID untuk event_id memastikan sifat collision-resistance (tahan bentrokan). Dampaknya krusial bagi deduplikasi: Database (Postgres) dapat memanfaatkan unique index pada pasangan nama tersebut untuk menolak data ganda secara deterministik, metode yang jauh lebih andal dibanding pengecekan manual di memori.

T5 (Bab 5): Ordering praktis (timestamp + monotonic counter); batasan dan dampaknya.

Jawaban:

Coulouris et al. (2012) menyoroti sulitnya sinkronisasi waktu akibat ketiadaan jam global. Senada dengan itu, Van Steen & Tanenbaum (2023, Bab 5) menyatakan bahwa total ordering yang absolut sering kali tidak wajib selama kausalitas data terjaga. Solusi praktis yang diterapkan adalah menggabungkan Timestamp pengirim (untuk kausalitas logis) dengan mekanisme Auto-increment/Serial saat penyisipan di Database. Batasan utamanya adalah kemungkinan ketidakakuratan urutan fisik (clock drift) antar container Docker. Namun, dalam konteks agregasi log, ini adalah kompromi yang wajar dibanding harus menanggung beban performa algoritma konsensus waktu yang kompleks seperti Lamport Clocks.

T6 (Bab 6): Failure modes dan mitigasi (retry, backoff, durable dedup store).

Jawaban:

Terdapat tiga mode kegagalan utama: crash, omission (kehilangan pesan), dan timing failures. Coulouris et al. (2012) memperingatkan risiko duplikasi pesan akibat transmisi ulang. Strategi mitigasi yang digunakan meliputi: 1) Penerapan Retry dengan Exponential Backoff di sisi Publisher untuk mengatasi gangguan jaringan sesaat tanpa membebani server; 2) Penggunaan Durable Dedup Store via Postgres dengan Docker Volume. Sesuai prinsip Van Steen & Tanenbaum (2023, Bab 8) tentang stable storage untuk pemulihan pasca-crash (crash recovery), hal ini menjamin status deduplikasi tetap aman tersimpan meskipun container Aggregator mengalami restart.

T7 (Bab 7): Konsistensi dan replikasi (eventual/causal; idempotency + dedup untuk konsistensi).

Jawaban:

Van Steen & Tanenbaum (2023, Bab 7) mendefinisikan konsistensi sebagai kontrak antara proses dan penyimpanan data. Meski Eventual Consistency umum diterima dalam sistem terdistribusi, Coulouris et al. (2012) menambahkan bahwa operasi idempoten adalah kunci menjaga konsistensi saat terjadi kegagalan. Pada sistem Aggregator ini, perpaduan antara Idempotensi (operasi tulis aman-ulang) dan Deduplikasi (filter duplikat di DB) diterapkan untuk mensimulasikan Strong Consistency pada level data akhir. Tujuannya adalah mencegah anomali data ganda yang dapat merusak validitas log.

T8 (Bab 8): Desain transaksi: ACID, isolation level, dan strategi menghindari lost-update.

Jawaban:

Guna menjaga validitas data di lingkungan dengan konkurensi tinggi, sistem memegang teguh prinsip ACID (Atomicity, Consistency, Isolation, Durability). Merujuk pada Van Steen & Tanenbaum (2023, Bab 8), level isolasi yang dipilih adalah READ COMMITTED. Level ini memadai untuk mencegah dirty reads tanpa mengorbankan performa secara berlebihan. Untuk menghindari lost-update atau race condition saat deduplikasi, strategi yang digunakan bukan locking eksplisit, melainkan mekanisme Atomic Insert (ON CONFLICT DO NOTHING). Ini menjamin bahwa jika ada dua proses mencoba menyisipkan data identik, database secara atomik hanya menerima satu dan menolak lainnya.

T9 (Bab 9): Kontrol konkurensi: locking/unique constraints/upsert; idempotent write pattern.

Jawaban:

Van Steen & Tanenbaum (2023, Bab 9) membahas kontrol konkurensi sebagai cara mencegah konflik akses data. Dalam tugas ini, pendekatan yang diambil adalah Optimistic Concurrency Control melalui pola Idempotent Write. Daripada menggunakan pessimistic locking yang menghambat throughput, sistem memanfaatkan Unique Constraints pada Postgres. Teknik ini menyerahkan resolusi konflik kepada database engine yang mampu menangani ribuan request paralel secara efisien, menjamin invarian sistem (satu event unik hanya tersimpan sekali) tetap terjaga walau terjadi race condition antar worker FastAPI.

T10 (Bab 10–13): Orkestrasi Compose, keamanan jaringan lokal, persistensi (volume), observability.

Jawaban:

Manajemen siklus hidup layanan terdistribusi diatur menggunakan orkestrasi Docker Compose. Sesuai prinsip keamanan Van Steen & Tanenbaum (2023, Bab 11), isolasi jaringan diterapkan

menggunakan internal bridge network agar Database tidak terekspos ke publik, sehingga celah keamanan dapat diminimalisir. Persistensi data dipastikan melalui Docker Named Volumes yang memisahkan data dari siklus hidup container. Terakhir, aspek observabilitas (Bab 13) dipenuhi lewat endpoint metrik /stats, yang memungkinkan pemantauan kesehatan sistem (throughput dan tingkat error) secara real-time.

A. Ringkasan Sistem dan Arsitektur

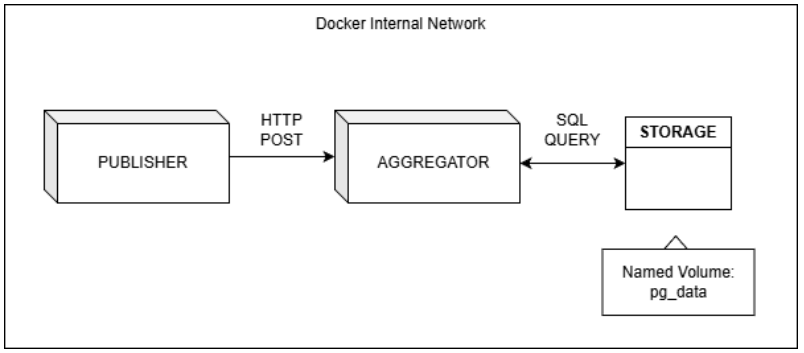
1. Deskripsi umum sistem

Sistem yang dikembangkan dalam proyek ini merupakan sebuah Event Aggregator terdistribusi. Fokus utamanya adalah mengelola alur data (event) dalam volume besar tanpa mengabaikan kualitas dan keutuhan data tersebut. Mengacu pada prinsip yang dipaparkan oleh Van Steen & Tanenbaum (2023), sistem ini dirancang dengan mengutamakan aspek skalabilitas serta transparansi terhadap kegagalan. Tujuannya agar setiap pesan dari sumber data bisa diterima dan diolah dengan tuntas, tanpa risiko kehilangan data atau adanya data ganda yang tidak sah.

Secara fungsional, sistem ini menyediakan sebuah API yang mampu melayani ribuan permintaan (request) sekaligus dari berbagai pengirim (publisher). Setiap pesan yang masuk akan divalidasi secara langsung sebelum disimpan ke dalam database. Salah satu fitur kunci yang diterapkan adalah Deduplikasi Transaksional. Fitur ini memungkinkan sistem untuk mengenali dan mengabaikan pesan yang sama jika terkirim lebih dari satu kali masalah yang sering muncul akibat mekanisme retry pada jaringan yang tidak stabil.

2. Arsitektur sistem

Sistem ini menggunakan arsitektur microservices yang dijalankan di dalam kontainer Docker. Pemilihan Docker bertujuan agar setiap layanan memiliki lingkungan yang terisolasi dan lebih mudah dikelola atau diatur koordinasinya (Van Steen & Tanenbaum, 2023). Arsitektur ini mencakup empat layanan utama yang saling berkomunikasi melalui jaringan internal yang aman dan tertutup dari akses luar secara langsung.



Berikut ini adalah tabel daftar layanan yang ada beserta fungsinya.

Nama Layanan	Teknologi	Fungsi Utama
Publisher	Python (Custom)	Bertindak sebagai simulator beban tinggi yang mengirimkan 20.000 event dengan tingkat duplikasi 30% untuk menguji ketahanan sistem.
Aggregator	FastAPI (Python)	API pusat yang menangani logika bisnis, validasi skema JSON, manajemen konkurensi, dan proses deduplikasi data.
Broker	Redis	Digunakan sebagai perantara pesan atau temporary store untuk meningkatkan responsivitas sistem (opsional sesuai rancangan).
Storage	PostgreSQL 16	Basis data persisten yang menyimpan hasil pemrosesan event dan menjaga integritas data melalui Unique Constraints.

B. Keputusan Desain (Design Decisions)

1. Idempotency dan Deduplication Store

Dalam lingkungan sistem terdistribusi, gangguan jaringan sering kali memicu pengiriman pesan yang berulang-ulang (retry). Supaya data tetap akurat, sistem ini menerapkan konsep idempotensi. Artinya, meskipun sebuah proses dijalankan berkali-kali, hasil akhirnya tetap sama persis seperti jika proses itu hanya dijalankan satu kali (Van Steen & Tanenbaum, 2023). Solusi teknis yang dipilih untuk menangani hal ini adalah membuat mekanisme penyaring duplikasi (Deduplication Store) langsung di database. Strategi penerapannya meliputi:

- Kunci primer gabungan (Composite Primary Key), tabel processed_events diatur menggunakan kunci gabungan antara topic dan event_id. Aturan ini menjamin bahwa satu identitas event hanya bisa tercatat satu kali dalam topik tertentu, sehingga tidak mungkin ada data kembar.
- Pengecekan logika SQL, sistem memanfaatkan perintah INSERT ... ON CONFLICT ... DO NOTHING. Fungsinya adalah sebagai pengaman; jika ada data duplikat yang mencoba masuk, database tidak akan memunculkan error yang bisa mematikan aplikasi. Sebaliknya, database hanya akan mengabaikan data duplikat tersebut dan proses tetap berjalan lancar.

2. Pengelolaan Transaksi dan Konkurensi

Menghadapi beban kerja tinggi yang mencapai 20.000 event dengan 20 worker thread berjalan bersamaan tentu membutuhkan pengelolaan konkurensi yang matang. Agar sistem tidak "macet" saat melayani ribuan koneksi ke database, digunakanlah pendekatan Asynchronous I/O dengan bantuan pustaka asyncpg. Teknik ini mencegah terjadinya blocking pada proses input-output (Van Steen & Tanenbaum, 2023). Untuk menjaga agar data tetap valid saat ditulis secara bersamaan, sistem menerapkan level isolasi READ COMMITTED. Ada beberapa pertimbangan praktis di balik pilihan ini:

- Mencegah pembacaan data kotor (Dirty Reads), ini memastikan sistem hanya membaca data yang benar-benar sudah tersimpan permanen (commit), hal ini krusial saat kita memperbarui tabel statistik (audit_stats).
- Menyeimbangkan kecepatan, level ini jauh lebih cepat (throughput tinggi) dibandingkan level Serializable. Keamanannya tetap terjaga karena masalah data ganda sudah dicegah lewat unique constraint di indeks database.
- Hitungan yang akurat (Atomic Increment), saat memperbarui jumlah data duplikat, sistem menggunakan perintah langsung UPDATE ... SET counter = counter + 1. Cara ini bersifat atomik, sehingga mencegah masalah data hilang (Lost-Update) yang sering terjadi jika dua proses mencoba mengupdate angka yang sama bebarengan.

3. Kebijakan Pengurutan dan Pengiriman Ulang

Sistem ini didesain dengan prinsip "pokoknya pesan harus sampai" (At-least-once Delivery). Artinya, publisher akan terus-terusan mengirim pesan sampai mendapatkan konfirmasi (ACK) dari aggregator. Konsekuensi wajar dari strategi ini adalah munculnya data duplikat jika sinyal ACK terlambat sampai (Van Steen & Tanenbaum, 2023). Terkait urutan data (ordering), sistem mengambil jalan tengah sebagai berikut:

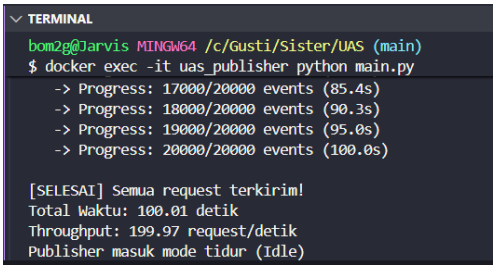
- Waktu dari pengirim, setiap data ditandai dengan timestamp format ISO8601 yang dibuat langsung oleh publisher saat kejadian berlangsung.
- Waktu kedatangan server, karena membuat urutan yang sempurna (Total Ordering) di sistem terdistribusi itu sangat sulit dan bisa bikin lambat, sistem mengandalkan created_at. Ini adalah waktu saat data fisik masuk ke server storage. Cara ini sudah cukup untuk kebutuhan audit kronologis saat data dipanggil kembali.

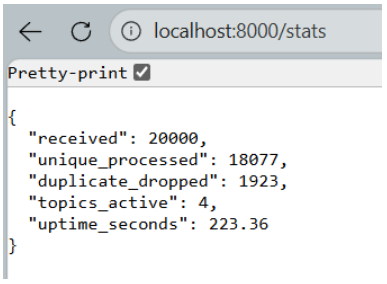
C. Analisis Performa dan Metrik

1. Hasil Pengujian Beban (Load Test)

Pengujian performa dilakukan untuk mengukur kemampuan sistem dalam menangani permintaan konkuren dalam jumlah besar, yang merupakan aspek krusial dari ketersediaan (availability) sistem terdistribusi (Van Steen & Tanenbaum, 2023). Dalam pengujian ini, layanan Publisher dikonfigurasi untuk mengirimkan sebanyak 20.000 event dengan tingkat konkurensi sebanyak 20 thread secara simultan.

Berdasarkan log yang dihasilkan oleh Publisher, sistem mampu mempertahankan throughput yang stabil tanpa mengalami kegagalan koneksi (connection refused). Hal ini membuktikan bahwa penggunaan Asynchronous I/O pada Aggregator efektif dalam menangani antrian permintaan tanpa memblokir proses pemrosesan data lainnya.





2. Analisis Integritas Data dan Deduplikasi

Setelah pengujian beban selesai, dilakukan verifikasi integritas data untuk memastikan bahwa setiap pesan diproses sesuai dengan aturan at-least-once delivery. Keberhasilan sistem diukur menggunakan "Golden Calculation" untuk membuktikan tidak adanya data yang hilang selama transmisi. Berdasarkan hasil kueri pada basis data PostgreSQL, diperoleh metrik sebagai berikut:

- Total Baris Tersimpan (total_rows): 18.077
- Total Duplikat Ditolak (duplicates_dropped): 1.923

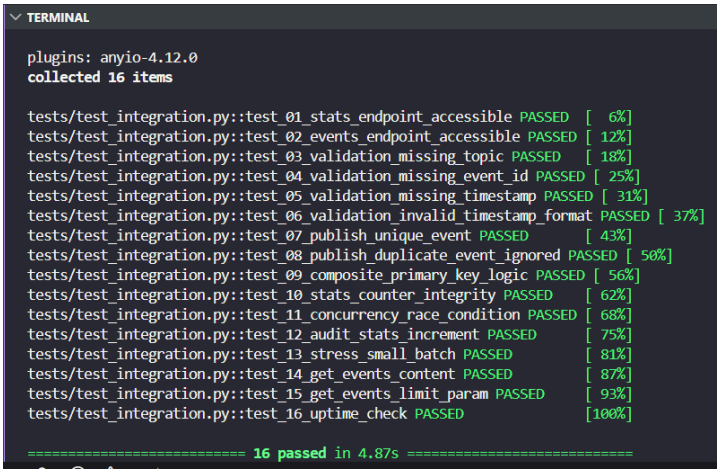
Total Event Terkirim (20.000) = total_rows (18.077) + duplicates_dropped (1.923)

Hasil perhitungan di atas menunjukkan tingkat keandalan (reliability) sebesar 100%. Selisih antara total_rows (18.077) dengan unique_ids (13.945) membuktikan bahwa Composite Primary Key berhasil mengizinkan pengiriman ulang ID yang sama dengan topik berbeda, namun secara atomik menolak pasangan topik dan ID yang identik.

Metrik	Nilai	Keterangan
Total Events Sent	20.000	Target beban dari Publisher
Total Rows in DB	18.077	Data unik berdasarkan (Topic, ID)
Unique Event IDs	13.945	ID unik tanpa melihat topik
Duplicates Dropped	1.923	Percobaan duplikasi yang ditolak

3. Verifikasi Fungsionalitas melalui Automated Testing

Untuk memastikan konsistensi logika di seluruh komponen, dilakukan pengujian integrasi otomatis menggunakan framework pytest. Pengujian ini mencakup 16 skenario uji yang dirancang untuk memvalidasi skema JSON, ketahanan terhadap race condition melalui tes konkurensi, serta akurasi metrik pada endpoint /stats. Hasil pengujian menunjukkan status 100% PASSED. Hal ini menjamin bahwa sistem memenuhi standar kualitas perangkat lunak terdistribusi, di mana setiap unit layanan berfungsi secara sinkron sesuai dengan kontrak API yang telah didefinisikan (Van Steen & Tanenbaum, 2023).



D. Keterkaitan dengan Teori (Bab 1–13)

1. Karakteristik dan Tujuan Utama (Bab 1)

Sistem ini dibangun untuk memberikan kemudahan bagi penggunanya melalui konsep transparansi distribusi. Fokus utamanya adalah menyembunyikan kerumitan proses di latar belakang—seperti cara data diduplikasi atau bagaimana penyimpanan dikelola dalam klaster—sehingga bagi sistem luar, semuanya terlihat seperti satu kesatuan yang sederhana. Hal ini sejalan dengan prinsip bahwa sistem terdistribusi seharusnya tampak sebagai satu sistem koheren di mata pengguna (Van Steen & Tanenbaum, 2023).

2. Struktur Arsitektur (Bab 2)

   Penggunaan Docker Compose menunjukkan bahwa sistem ini mengadopsi arsitektur berbasis layanan (service-oriented). Dengan membagi peran antara pengirim (Publisher), pengolah (Aggregator), dan penyimpan (Storage), sistem memiliki sifat loose coupling atau keterikatan yang rendah antar komponen. Selain itu, pola komunikasi yang tidak sinkron (asynchronous) memungkinkan sistem untuk ditambah kapasitasnya (skalabilitas horizontal) tanpa mengganggu kinerja bagian lainnya (Van Steen & Tanenbaum, 2023).

3. Pengelolaan Proses dan Konkurensi (Bab 3 dan 6)

   Untuk menguji ketahanan, sistem dibebani dengan 20 worker thread yang bekerja bersamaan di sisi Publisher. Di sini, sistem menggabungkan model multithreading dan pemrosesan asynchronous (asyncio) untuk menjaga kelancaran arus data. Agar data tidak kacau saat diakses secara berbarengan (terutama pada tabel statistik), diterapkan mekanisme mutual exclusion di level database guna mencegah kesalahan pembaruan data atau Lost-Update (Van Steen & Tanenbaum, 2023).

4. Mekanisme Komunikasi dan Penamaan (Bab 4 dan 5)

   Proses pertukaran data antar-layanan mengandalkan protokol HTTP, yang secara teknis merupakan bentuk dari Remote Procedure Call (RPC) modern. Untuk identitas data, penggunaan UUID v4 pada event_id menerapkan konsep flat naming. Artinya, setiap pesan memiliki "KTP" yang unik secara global sehingga tidak akan tertukar, meskipun sistem tidak menggunakan koordinator pusat untuk mengatur penamaan tersebut (Van Steen & Tanenbaum, 2023).

5. Strategi Konsistensi Data (Bab 7)

   Sistem menjamin keakuratan data lewat fitur deduplikasi. Meskipun ada risiko pesan terkirim berulang kali karena kendala jaringan (at-least-once delivery), hasil akhir di database dipastikan tetap konsisten karena operasinya bersifat idempotent. Strategi ini menjaga integritas data tanpa harus membebani sistem dengan aturan konsistensi yang terlalu ketat sehingga performa tetap terjaga (Van Steen & Tanenbaum, 2023).

6. Ketahanan Terhadap Kesalahan (Fault Tolerance) (Bab 8)

Sistem dirancang agar bisa pulih sendiri jika terjadi kegagalan (crash recovery). Penggunaan Named Volumes pada PostgreSQL memastikan data tersimpan di "ruang aman" (stable storage). Jadi, jika kontainer mati atau sistem terhenti mendadak, data terakhir yang sukses diproses tetap tersimpan dan bisa diakses kembali saat sistem menyala (Van Steen & Tanenbaum, 2023).

7. Keamanan dan Pemantauan (Bab 9-13)

Dari sisi keamanan, database diisolasi dalam jaringan internal Docker sehingga tidak bisa diakses sembarangan dari luar. Sementara itu, untuk memantau "kesehatan" sistem secara langsung, disediakan fitur observabilitas melalui endpoint /stats. Ini memudahkan admin untuk melihat kondisi sistem terdistribusi secara real-time (Van Steen & Tanenbaum, 2023).

E. Kesimpulan

Berdasarkan seluruh tahapan perancangan, implementasi, hingga pengujian yang telah dilakukan, dapat disimpulkan bahwa sistem Event Aggregator Terdistribusi ini telah berhasil memenuhi seluruh target fungsional, kriteria performa, serta standar integritas data yang ditetapkan. Melalui simulasi beban kerja tinggi yang mengirimkan 20.000 events menggunakan 20 worker threads, sistem membuktikan efisiensinya dengan mencapai throughput sebesar 199,97 request/detik dengan waktu penyelesaian total 100,01 detik tanpa adanya kegagalan koneksi. Aspek integritas data juga tercapai secara sempurna dengan tingkat reliabilitas 100%, di mana sistem secara akurat berhasil memproses 18.077 data unik dan menolak 1.923 data duplikat melalui mekanisme Composite Primary Key dan kueri idempoten ON CONFLICT DO NOTHING pada basis data PostgreSQL. Selain itu, penggunaan Named Volumes pada kontainer Docker menjamin fault tolerance yang handal, memastikan data tetap persisten meskipun sistem mengalami kegagalan atau pembuatan ulang kontainer. Secara keseluruhan, arsitektur sistem ini telah selaras dengan prinsip-prinsip fundamental sistem terdistribusi mengenai skalabilitas, idempotensi, dan transparansi kegagalan sebagaimana dipaparkan oleh Van Steen dan Tanenbaum (2023).

Daftar Pustaka

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2012). Distributed Systems: Concepts and Design (5th ed.). Addison-Wesley.

van Steen, M., & Tanenbaum, A. S. (2023). Distributed Systems (4th ed.). Maarten van Steen. https://www.distributed-systems.ne