

UJIAN TENGAH SEMESTER
LAPORAN SISTEM PARALEL DAN TERDISTRIBUSI



Oleh Kelompok :

Isnaini Zayyana Fitri

11221072

Institut Teknologi Kalimantan

2025

1. Pendahuluan

Sistem log aggregator merupakan komponen penting dalam arsitektur terdistribusi yang berfungsi untuk mengumpulkan dan memproses data log dari berbagai sumber. Dalam sistem berskala besar, proses pengumpulan log secara real-time sering dilakukan dengan memanfaatkan pola komunikasi publish-subscribe (Pub/Sub). Melalui pendekatan ini, publisher bertugas mengirimkan pesan atau event, sementara subscriber (atau consumer) menerima dan memprosesnya sesuai kebutuhan.

Permasalahan umum yang muncul dalam implementasi sistem semacam ini adalah munculnya duplikasi event atau pemrosesan berulang akibat gangguan jaringan, retries, maupun delivery lag. Untuk mengatasi hal tersebut, diperlukan mekanisme idempotent consumer—yakni consumer yang mampu memastikan bahwa satu event yang sama tidak diproses lebih dari sekali—serta deduplication agar log ganda dapat diidentifikasi dan dihapus sebelum diproses. Proyek ini bertujuan untuk mengembangkan layanan Pub-Sub Log Aggregator yang mampu menerima event/log dari berbagai publisher, kemudian menyalurkannya ke subscriber atau consumer yang bersifat idempotent, dengan dilengkapi fitur deduplication guna menjamin keakuratan dan konsistensi data log yang diolah.

2. Landasan Teori

2.1 Publish-Subscribe Model

Model publish-subscribe adalah pola komunikasi asinkron di mana publisher tidak perlu mengetahui identitas subscriber secara langsung. Pesan dikirim melalui message broker (seperti Kafka, RabbitMQ, atau Redis Stream) yang bertugas mendistribusikan data ke setiap subscriber yang berlangganan pada topik tertentu. Pendekatan ini mendukung skalabilitas dan desentralisasi sistem log.

2.2 Idempotent Consumer

Konsep idempotent consumer mengacu pada consumer yang dapat memproses event secara aman meskipun event tersebut dikirim lebih dari sekali. Hal ini dicapai dengan cara mencatat event ID atau hash signature setiap event yang sudah diproses, sehingga jika event serupa muncul kembali, sistem akan mengenalinya dan mengabaikannya.

2.3 Deduplication

Deduplication adalah proses penghapusan data yang bersifat duplikat. Dalam konteks log aggregator, proses ini penting agar log yang identik tidak mengakibatkan redundansi dan inkonsistensi dalam hasil analisis data. Umumnya, deduplication diterapkan sebelum data disimpan ke dalam basis data utama.

3. Perancangan Sistem

Sistem yang dikembangkan terdiri atas tiga komponen utama:

- Publisher

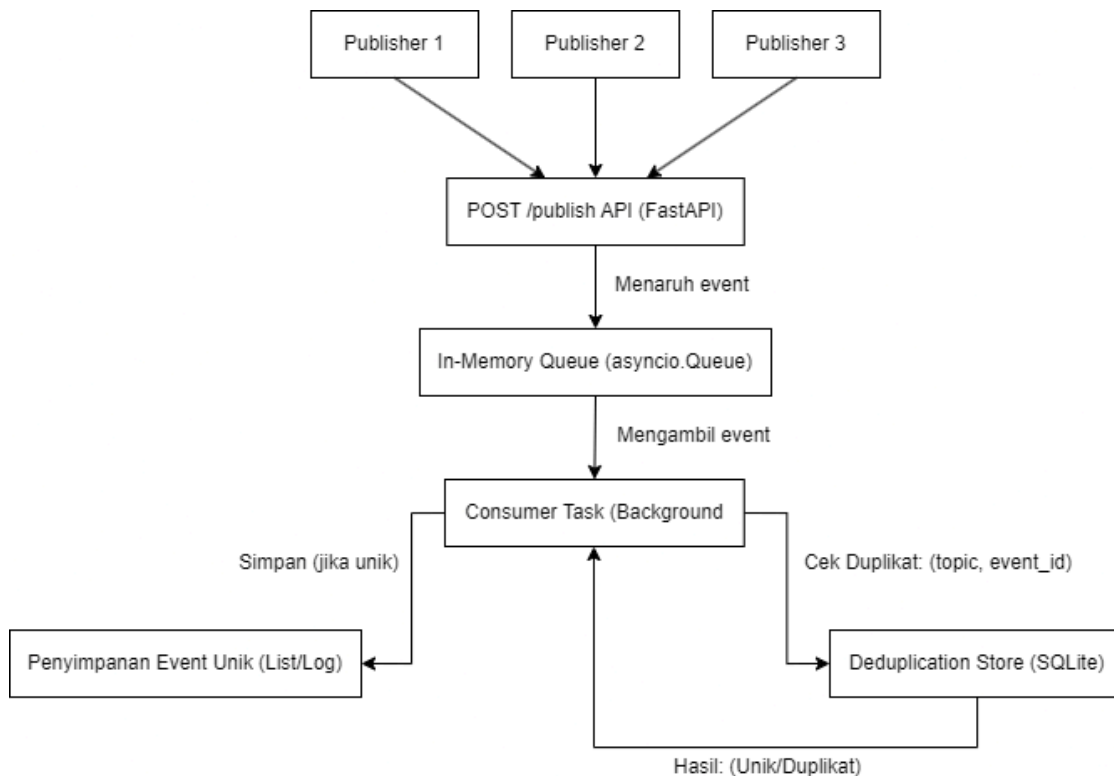
Bertanggung jawab mengirimkan data log atau event ke sistem melalui endpoint tertentu, seperti REST API atau message queue. Setiap event memiliki unique identifier (`event_id`) untuk mendukung proses deduplication dan idempotensi.

- Agregator Service / Message Broker

Berperan sebagai penghubung antara publisher dan consumer. Komponen ini menerima event dari publisher, memvalidasi keunikan `event_id`, dan menyimpannya sementara dalam sistem penyimpanan cepat seperti Redis. Proses deduplication dijalankan pada tahap ini sebelum event diteruskan ke consumer.

- Consumer (Idempotent Processor)

Komponen yang memproses event dengan memastikan bahwa setiap `event_id` hanya diproses satu kali. Consumer mencatat setiap `event_id` yang telah berhasil diolah dalam processed event store agar duplikasi berikutnya dapat diabaikan.



Arsitektur sistem ini dirancang sebagai layanan publish-subscribe terdesentralisasi yang berjalan di dalam kontainer Docker. Komponen utamanya meliputi:

- Publisher: Bertugas mengirimkan *event* log ke *aggregator*.
- Aggregator Service (API): Dibangun menggunakan FastAPI, menyediakan *endpoint* POST /publish untuk menerima *event*. *Event* yang masuk divalidasi dan segera dimasukkan ke dalam antrian *in-memory* (*asyncio.Queue*).
- Consumer Task: Sebuah *background task asynchronous* yang berjalan di dalam layanan *aggregator*. Tugas ini secara kontinu mengambil *event* dari *queue*.
- Deduplication Store: Menggunakan SQLite, sebuah *database file-based* yang persisten. *Consumer* akan memeriksa setiap *event* ke *store* ini berdasarkan kunci unik (*topic*, *event_id*) sebelum memprosesnya.
- Observability Endpoints: Menyediakan GET /stats dan GET /events untuk memantau status sistem.

4. Keputusan Desain Implementasi

4.1 Antrian (Queueing) dan Pemrosesan

Sistem menggunakan *asyncio.Queue* sebagai buffer antrian *in-memory*. Keputusan ini diambil untuk memaksimalkan throughput dan meminimalkan latensi pada *endpoint* /publish. Publisher tidak perlu menunggu *event* diproses sepenuhnya; API dapat menerima *event* dengan sangat cepat (*respons* 202 Accepted) dan kembali. Ini memisahkan (*decouple*) proses penerimaan (*ingestion*) dari proses pengolahan (*processing*).

4.2 Idempotency dan Deduplication Store

Idempotency dicapai pada level consumer. State dari *event* yang telah diproses disimpan dalam database SQLite lokal (*dedup_store.db*). SQLite dipilih karena:

- Persisten: Tahan terhadap *restart container*, memenuhi syarat *fault tolerance*.
- Lokal: Tidak memerlukan layanan eksternal, sesuai batasan tugas.
- Atomik: Menggunakan PRIMARY KEY pada (*topic*, *event_id*) memungkinkan kami menangani *race condition* dengan aman. Upaya INSERT duplikat akan gagal secara atomik (melempar *IntegrityError*), yang kemudian kita tangkap untuk menandai *event* sebagai duplikat.

4.3 Pertimbangan Ordering dalam Implementasi

Sistem ini tidak menjamin total ordering. Fokusnya adalah pada correctness (*deduplikasi*) dan completeness (*at-least-once*). *Event* diproses oleh consumer berdasarkan urutan keluarnya dari *asyncio.Queue*, yang umumnya FIFO (*First-In*,

First-Out) dari endpoint API. Namun, tidak ada jaminan ordering antar event dari publisher yang berbeda, terutama jika terjadi retry atau network delay.

5. Analisis Teori

5.1 Karakteristik Sistem Terdistribusi dan Trade-off

Sistem terdistribusi, seperti log aggregator ini, didefinisikan sebagai kumpulan komputer independen yang tampak bagi penggunaanya sebagai satu sistem koheren tunggal (Tanenbaum & Van Steen, 2023, Bab 1). Karakteristik utamanya yang relevan dengan aggregator ini adalah:

- **Concurrency (Konkurensi):** Komponen berjalan secara paralel. Beberapa *publisher* dapat mengirim *event* secara bersamaan, sementara *consumer* memproses *event* lain.
- **No Global Clock (Tidak Ada Jam Global):** Setiap *publisher* memiliki jam lokalnya sendiri, yang tercermin dalam *timestamp event*. Hal ini dapat menyebabkan kesulitan dalam menentukan urutan *event* global secara pasti.
- **Independent Failures (Kegagalan Independen):** *Publisher* bisa gagal tanpa menghentikan *aggregator*, atau *aggregator* itu sendiri bisa *crash* dan *restart*. Desain sistem harus mengantisipasi kegagalan ini, misalnya dengan *retry* dari sisi *publisher* dan *dedup store* yang persisten dari sisi *consumer*.

Desain Pub-Sub log aggregator ini menghadapi beberapa trade-off fundamental:

- **Performance vs. Reliability (Durability):** *Trade-off* utama dalam desain ini. Kami memilih `asyncio.Queue` (in-memory) untuk *throughput* yang sangat tinggi pada *ingestion*. Namun, ini mengorbankan *durability*; jika *aggregator crash* sebelum *consumer* memproses *event* dari *queue*, *event* tersebut akan hilang. Ini adalah *trade-off* yang diterima untuk mencapai *performance* tinggi dalam skenario non-kritis.
- **Scalability vs. Consistency:** Sistem ini mudah di-skalkan dengan menambah *publisher*. Namun, untuk menjamin konsistensi (yaitu, tidak ada duplikat yang diproses), setiap *event* harus divalidasi terhadap *dedup store* (SQLite). *Store* ini bisa menjadi *bottleneck* jika *throughput event* unik sangat tinggi, sehingga membatasi skalabilitas pemrosesan *consumer*.

5.2 Perbandingan arsitektur client-server dan publish-subscribe untuk aggregator

Arsitektur client-server adalah model komunikasi request-reply tradisional di mana client (dalam hal ini, publisher) secara eksplisit mengirimkan permintaan ke server (aggregator) dan seringkali menunggu respons. Interaksi ini bersifat tightly coupled (sangat terikat); publisher harus mengetahui alamat pasti dari aggregator. Jika aggregator gagal, publisher tidak dapat mengirimkan event-nya.

Sebaliknya, arsitektur publish-subscribe menyediakan model komunikasi many-to-many yang loosely coupled (tidak terikat erat). Seperti yang dijelaskan oleh Tanenbaum & Van Steen (2023, Bab 2), arsitektur ini memberikan decoupling (pemisahan) dalam beberapa dimensi. Untuk log aggregator, dua hal yang paling relevan adalah:

- Space Decoupling (Pemisahan Ruang): Publisher tidak perlu tahu siapa atau di mana subscriber (consumer) berada. Publisher hanya mengirimkan event ke topic tertentu.
- Synchronization Decoupling (Pemisahan Sinkronisasi): Publisher dapat mengirimkan event secara asinkron (fire-and-forget). Mereka tidak diblokir menunggu consumer selesai memproses event tersebut.

Pub-Sub adalah pilihan yang jelas untuk log aggregator karena event (log) secara alami bersifat asinkron dan seringkali memiliki banyak sumber (publisher) dan banyak peminat (subscriber). Alasan teknis utamanya adalah fleksibilitas dan skalabilitas.

Melengkapi hal ini, Coulouris et al. (2012, Bab 6.3) mengkategorikan publish-subscribe sebagai bentuk "komunikasi tidak langsung" (indirect communication). Keunggulan utamanya adalah decoupling in time (pemisahan dalam waktu) selain decoupling in space (pemisahan dalam ruang). Decoupling in time berarti publisher dan consumer tidak harus aktif pada saat yang bersamaan. Meskipun implementasi kita menggunakan in-memory queue (yang tidak time-decoupled), arsitektur Pub-Sub itu sendiri secara fundamental dirancang untuk mendukung ini, menjadikannya pilihan yang lebih tangguh untuk evolusi sistem di masa depan (misalnya, mengganti queue dengan broker yang persisten).

5.3 Semantik Pengiriman At-Least-Once dan Exactly-Once

Semantik pengiriman (delivery semantics) adalah jaminan yang diberikan oleh sistem komunikasi mengenai pengiriman pesan:

- At-Least-Once (Setidaknya Satu Kali): Ini adalah jaminan bahwa pesan akan dikirim, dan pengirim akan terus mencoba (retries) sampai menerima konfirmasi (acknowledgment/ACK). Ini adalah pilar fault tolerance (Bab 6). Seperti yang dibahas dalam komunikasi reliable (Tanenbaum & Van Steen, 2023, Bab 3), jika ACK hilang atau tertunda, pengirim akan mengirim ulang pesan yang sama. Ini menjamin pesan tidak hilang, tetapi menciptakan risiko pesan duplikat.
- Exactly-Once (Tepat Satu Kali): Jaminan terkuat di mana pesan dijamin tiba dan diproses tepat satu kali. Ini sangat sulit dan mahal untuk diimplementasikan dalam sistem terdistribusi karena memerlukan koordinasi yang kompleks (misalnya, distributed transactions atau two-phase commit yang dibahas di Bab 7) untuk menangani failure pengirim, jaringan, dan penerima secara bersamaan.

Dalam tugas ini, kita mengandalkan retries dari publisher untuk menangani failure (mencapai at-least-once). Ini berarti duplikasi event adalah kondisi yang tidak terhindarkan dan sudah diperkirakan. Idempotency (Bab 7) adalah properti di mana sebuah operasi dapat dieksekusi berkali-kali namun hasilnya tetap sama seolah-olah hanya dieksekusi sekali. Consumer yang idempotent (diimplementasikan melalui deduplication store) adalah solusi praktis untuk masalah ini. Ia mengubah masalah exactly-once yang sulit di level protokol komunikasi, menjadi masalah at-least-once (melalui retries) ditambah deduplication (di consumer). Ini adalah trade-off desain yang jauh lebih efisien dan umum digunakan.

5.4 Rancangan Skema Penamaan Topic dan Event_ID

Penamaan adalah aspek fundamental dalam sistem terdistribusi untuk mengidentifikasi dan menemukan resource (Tanenbaum & Van Steen, 2023, Bab 4). Dalam aggregator ini, kami merancang dua jenis identifier:

- **topic (Nama):** Untuk topic, kami menggunakan skema penamaan hierarkis berbasis string yang dipisahkan oleh titik (.), contoh: `service.auth.prod.login`. Ini berfungsi sebagai human-friendly name (nama yang mudah dipahami manusia).
- **event_id (Identifier):** Untuk event_id, persyaratannya adalah unik secara global dan collision-resistant (tahan benturan). Kami menggunakan UUID (Universally Unique Identifier), khususnya UUIDv4. Ini sejalan dengan konsep identifiers (Bab 4) yang dapat dibuat oleh entitas manapun namun tetap dijamin unik.

Keberhasilan deduplikasi bergantung pada primary key (topic, event_id) yang stabil. Kuncinya adalah: Publisher wajib mengirimkan event_id yang sama persis saat melakukan retry. Jika publisher salah membuat event_id baru saat retry, consumer akan menganggapnya sebagai event baru yang unik, dan deduplikasi akan gagal.

Buku Coulouris et al. (2012, Bab 9) memperkuat rancangan ini dengan membedakan antara names (nama yang dapat dibaca manusia, seperti topic kita) dan identifiers (yang merujuk ke objek secara unik, seperti event_id kita). Penggunaan UUID (sebagai pure name atau identifier) sangat penting karena menjamin keunikan global tanpa memerlukan otoritas penamaan terpusat. Ini sangat krusial dalam sistem terdistribusi di mana publisher beroperasi secara independen dan tidak terkoordinasi satu sama lain.

5.5 Ordering (Urutan)

Ordering (urutan) dalam sistem terdistribusi adalah masalah yang kompleks karena tidak adanya jam global (Tanenbaum & Van Steen, 2023, Bab 5). Kapan total ordering tidak diperlukan? Untuk kasus penggunaan log aggregator ini, total ordering (jaminan bahwa setiap proses melihat semua event dalam urutan yang sama persis) tidak

diperlukan. Tujuan utamanya adalah completeness (semua log terkumpul) dan correctness (tidak ada duplikat), bukan urutan pasti terjadinya event secara global. Tidak masalah jika log dari service A yang terjadi pukul 10:00:01 diproses sebelum log dari service B pukul 10:00:00.

Pendekatan praktis yang kami gunakan adalah mengandalkan timestamp (ISO8601) yang dibuat oleh publisher (menggunakan physical clock mereka). Timestamp ini disimpan bersama event dan dapat digunakan untuk mengurutkan event secara best-effort saat ditampilkan ke pengguna (misalnya pada GET /events).

Batasan (Bab 5.1): Batasan utama dari pendekatan ini adalah clock skew. Jam di publisher A mungkin berjalan beberapa milidetik (atau bahkan detik) lebih cepat atau lebih lambat dari publisher B. Ini berarti timestamp tidak dapat diandalkan 100% untuk menentukan urutan kausal (causal ordering). Event B bisa saja terjadi setelah event A di dunia nyata, namun memiliki timestamp yang lebih awal. Untuk log aggregator, kita menerima batasan ini sebagai trade-off untuk kesederhanaan desain.

5.6 Identifikasi Failure Modes dan Mitigasi

Sistem ini dirancang dengan mempertimbangkan fault tolerance (Tanenbaum & Van Steen, 2023, Bab 6). Berikut adalah failure modes utama dan strategi mitigasinya:

- Failure Mode: Network Failure / Partition: Publisher mengirim event, aggregator menerimanya, namun respons "OK" (HTTP 202) hilang di jaringan.
 - Mitigasi: Publisher akan melakukan retry (mengirim ulang event yang sama). Ini ditangani oleh dedup store.
- Failure Mode: Aggregator (Consumer) Crash:
 - Kasus A: Crash sebelum event diproses dari `asyncio.Queue` (in-memory). Event akan hilang. Ini adalah trade-off desain untuk performance.
 - Kasus B: Crash setelah event diproses dan dicatat di SQLite. Event aman.
 - Mitigasi: Saat aggregator restart, durable dedup store (SQLite) yang persisten akan "mengingat" event dari Kasus B, mencegahnya diproses ulang, dan (jika publisher me-retry event Kasus A) akan memprosesnya sebagai unik.

Coulouris et al. (2012, Bab 15.2) mengklasifikasikan failure modes ini. Aggregator crash adalah crash failure, sementara hilangnya paket respons di jaringan adalah omission failure. Strategi retry dari publisher adalah mitigasi langsung terhadap omission failure untuk mencapai komunikasi yang reliable (Bab 13.4.3). Namun, seperti yang ditekankan, retry ini dapat menghasilkan duplikasi pesan, yang kembali mempertegas perlunya mekanisme idempotency di sisi consumer untuk menangani pesan duplikat tersebut dengan aman.

5.7 Eventual Consistency pada Aggregator

Eventual consistency (konsistensi pada akhirnya) adalah salah satu model konsistensi data-centric (Tanenbaum & Van Steen, 2023, Bab 7.2). Ini adalah jaminan bahwa jika tidak ada update (pengiriman event) baru yang terjadi untuk sementara waktu, semua replica (atau dalam kasus ini, semua query ke sistem) pada akhirnya akan converge ke nilai yang benar.

State yang harus konsisten di sini adalah jumlah total event unik yang telah diproses (tercermin pada GET /stats dan GET /events). Dalam sistem at-least-once, event duplikat terus-menerus dikirim (karena network failure atau retry). Ini secara aktif menyebabkan state menjadi tidak konsisten untuk sementara (misalnya, received bertambah, tapi unique_processed belum).

Mekanisme idempotency (diimplementasikan via deduplication) adalah kekuatan konvergen yang mendorong sistem menuju eventual consistency.

- Sistem at-least-once (T6) secara aktif menciptakan inkonsistensi sementara.
- Consumer yang idempotent menyerap inkonsistensi ini.
- Setiap kali event duplikat diterima (misalnya event_id "XYZ"), consumer akan mengecek SQLite, melihat "XYZ" sudah ada, dan mengabaikannya.
- Meskipun received bertambah, unique_processed tidak. Ini memastikan bahwa tidak peduli berapa banyak duplikat yang dikirim, state akhir (unique_processed) pada akhirnya akan stabil dan merefleksikan jumlah yang benar.

5.8 Perumusan Metrik Evaluasi Sistem

Evaluasi sistem terdistribusi memerlukan metrik kuantitatif yang jelas, yang seringkali menyoroti trade-off fundamental dalam desain (Tanenbaum & Van Steen, 2023, Bab 1). Untuk log aggregator ini, tiga metrik utama adalah:

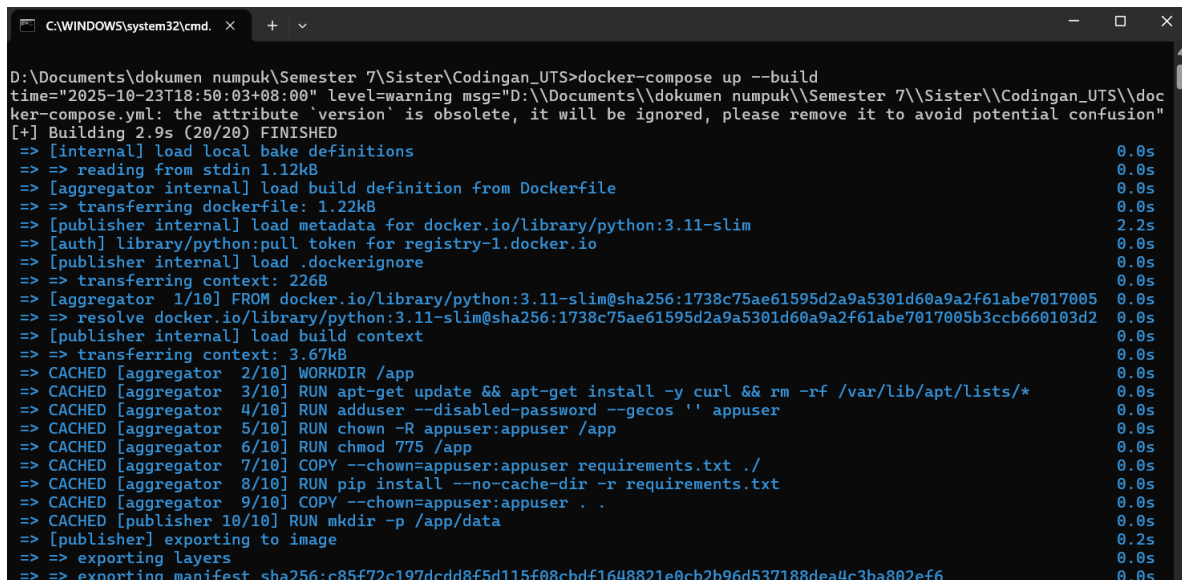
- Throughput (Events/sec): Ini mengukur kapasitas sistem. Kita harus membedakan:
 - Ingestion Throughput: Jumlah event yang dapat diterima oleh endpoint POST /publish per detik.
 - Processing Throughput: Jumlah event yang dapat diproses oleh consumer (termasuk cek ke SQLite) per detik.
 - Kaitan Desain: Keputusan menggunakan FastAPI (asyncio) dan asyncio.Queue (in-memory) (Bab 2) secara spesifik dirancang untuk memaksimalkan ingestion throughput. API bersifat non-blocking; ia hanya menempatkan event di antrian (operasi memori yang sangat cepat) dan segera kembali, sehingga decoupling (memisahkan) publisher dari consumer. Processing throughput akan lebih rendah karena dibatasi oleh blocking I/O ke database SQLite.

- Latency (ms): Ini mengukur waktu tunda.
 - Ingestion Latency: Waktu antara publisher mengirim POST dan menerima respons 202 Accepted.
 - End-to-End Latency: Waktu antara POST diterima hingga event selesai diproses (termasuk commit ke SQLite).
 - Kaitan Desain: Desain async (Bab 2) meminimalkan ingestion latency, memberikan respons cepat ke publisher. Namun, kami menerima end-to-end latency yang lebih tinggi sebagai trade-off. Ini adalah harga yang dibayar untuk correctness (cek duplikat) dan fault tolerance (penyimpanan persisten, Bab 6).
- Duplicate Dropped Rate (%): Ini adalah metrik correctness (kebenaran), bukan performance. Ini mengukur efektivitas sistem dalam menegakkan idempotency (Bab 7). Idealnya, metrik ini adalah 100% dari semua event duplikat yang dikirim.
 - Kaitan Desain: Keputusan menggunakan SQLite dengan PRIMARY KEY pada (topic, event_id) secara langsung menjamin correctness ini. Pilihan ini memprioritaskan consistency (Bab 7) dan fault tolerance (Bab 6) di atas processing throughput mentah. Kami secara eksplisit memilih untuk lebih lambat namun benar, daripada cepat namun salah (memproses duplikat).

6. Analisis Performa dan Metrik

6.1 Hasil Stress Test (5000 Event)

Pengujian dilakukan dengan menjalankan docker-compose up --build.



```

C:\WINDOWS\system32\cmd. x + v
D:\Documents\dokumen numpuk\Semester 7\Sister\Codingan_UTS>docker-compose up --build
time="2025-10-23T18:50:03+08:00" level=warning msg="D:\\Documents\\dokumen numpuk\\Semester 7\\Sister\\Codingan_UTS\\doc
ker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Building 2.9s (20/20) FINISHED
=> [internal] load local bake definitions                                0.0s
=> => reading from stdin 1.12kB                                         0.0s
=> [aggregator internal] load build definition from Dockerfile           0.0s
=> => transferring dockerfile: 1.22kB                                    0.0s
=> [publisher internal] load metadata for docker.io/library/python:3.11-slim 2.2s
=> [auth] library/python:pull token for registry-1.docker.io            0.0s
=> [publisher internal] load .dockerignore                               0.0s
=> => transferring context: 226B                                          0.0s
=> [aggregator 1/10] FROM docker.io/library/python:3.11-slim@sha256:1738c75ae61595d2a9a5301d60a9a2f61abe7017005 0.0s
=> => resolve docker.io/library/python:3.11-slim@sha256:1738c75ae61595d2a9a5301d60a9a2f61abe7017005b3ccb660103d2 0.0s
=> [publisher internal] load build context                               0.0s
=> => transferring context: 3.67kB                                        0.0s
=> CACHED [aggregator 2/10] WORKDIR /app                                0.0s
=> CACHED [aggregator 3/10] RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/* 0.0s
=> CACHED [aggregator 4/10] RUN adduser --disabled-password --gecos '' appuser 0.0s
=> CACHED [aggregator 5/10] RUN chown -R appuser:appuser /app           0.0s
=> CACHED [aggregator 6/10] RUN chmod 775 /app                          0.0s
=> CACHED [aggregator 7/10] COPY --chown=appuser:appuser requirements.txt ./ 0.0s
=> CACHED [aggregator 8/10] RUN pip install --no-cache-dir -r requirements.txt 0.0s
=> CACHED [aggregator 9/10] COPY --chown=appuser:appuser . .            0.0s
=> CACHED [publisher 10/10] RUN mkdir -p /app/data                     0.0s
=> [publisher] exporting to image                                       0.2s
=> => exporting layers                                                  0.0s
=> => exporting manifest sha256:c85f72c197dcdd8f5d115f08cbdf1648821e0cb2b96d537188dea4c3ba802ef6 0.0s
  
```

- Log Pemrosesan Event: Selama pengujian berlangsung, log dari container aggregator-service menunjukkan pemrosesan event secara real-time. Terlihat jelas pesan log yang membedakan antara event yang unik dan yang duplikat.

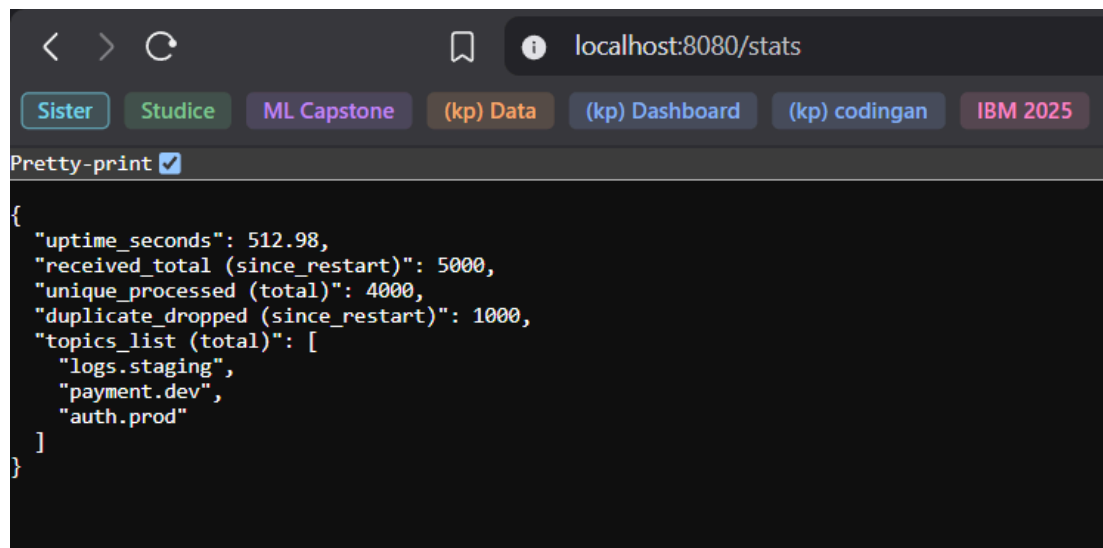
```

publisher-client | Request error ke http://aggregator:8080/publish:
publisher-client | Request error ke http://aggregator:8080/publish:
publisher-client | Request error ke http://aggregator:8080/publish:
publisher-client | Request error ke http://aggregator:8080/publish:
publisher-client | Request error ke http://aggregator:8080/publish:
publisher-client | Request error ke http://aggregator:8080/publish:
publisher-client | Request error ke http://aggregator:8080/publish:
publisher-client | Request error ke http://aggregator:8080/publish:
publisher-client | Request error ke http://aggregator:8080/publish:
publisher-client | Request error ke http://aggregator:8080/publish:
publisher-client | --- Stress Test Selesai ---
publisher-client | Total event terkirim: 1700
publisher-client | Waktu eksekusi: 30.14 detik
publisher-client | Rata-rata: 56.41 events/detik
aggregator-service | 2025-10-23 10:51:04,052 - INFO - Event UNIK diproses: (Topic: auth.prod, ID: 659cca4e-13a4-4567-94
de-5369b5d07789)
publisher-client exited with code 0
aggregator-service | 2025-10-23 10:51:04,134 - INFO - Event UNIK diproses: (Topic: auth.prod, ID: 626d28e6-1bd2-492d-9c
a7-00a7426ce47f)
aggregator-service | 2025-10-23 10:51:04,197 - INFO - Event UNIK diproses: (Topic: auth.prod, ID: 2312eea1-2d46-4f45-8b
d4-86fe339a46a1)
aggregator-service | 2025-10-23 10:51:04,258 - INFO - Event UNIK diproses: (Topic: auth.prod, ID: ec92543c-aea9-458b-bf
f2-50432557a5a4)
aggregator-service | 2025-10-23 10:51:04,264 - INFO - Event DUPLIKAT terdeteksi: (Topic: logs.staging, ID: 0f8c33be-b81
0-4f62-8d23-bd7625e0b80d)

```

Gambar di atas menunjukkan contoh log di mana sistem mengidentifikasi beberapa event sebagai 'UNIK diproses' dan yang lainnya sebagai 'DUPLIKAT terdeteksi', membuktikan mekanisme deduplikasi bekerja saat beban tinggi.

- Statistik Awal (Sebelum Restart): Setelah script publisher selesai mengirim 5000 event (dan sebelum server dimatikan), endpoint /stats diakses melalui browser.



```

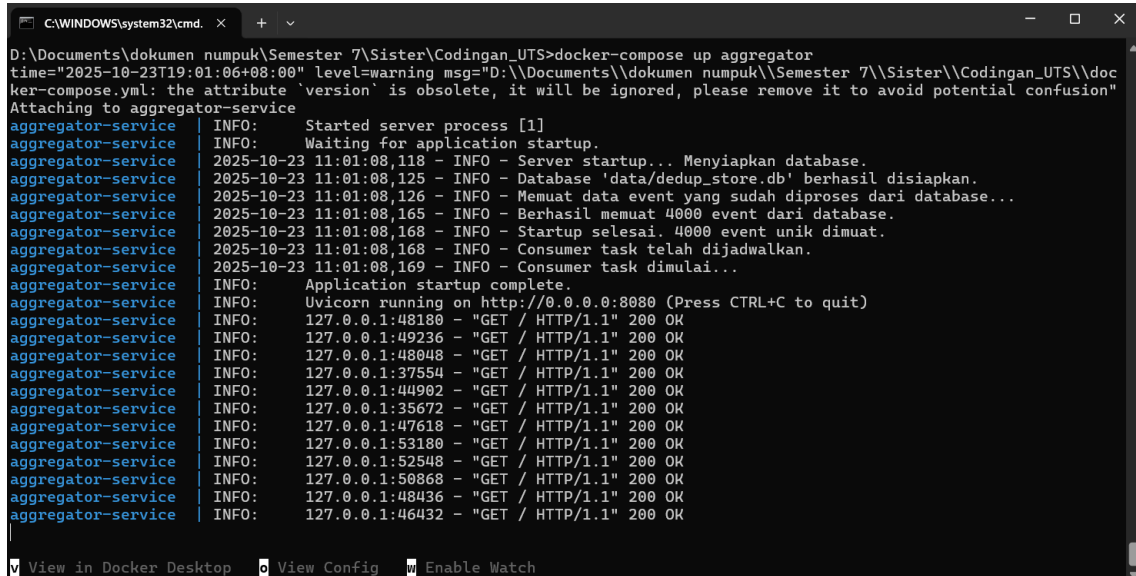
{
  "uptime_seconds": 512.98,
  "received_total (since_restart)": 5000,
  "unique_processed (total)": 4000,
  "duplicate_dropped (since_restart)": 1000,
  "topics_list (total)": [
    "logs.staging",
    "payment.dev",
    "auth.prod"
  ]
}

```

Gambar di atas menunjukkan hasil dari <http://localhost:8080/stats>. Terlihat bahwa `received_total (since_restart)` mencapai 5000, `unique_processed (total)` mencapai 4000, dan `duplicate_dropped (since_restart)` mencapai 1000. Ini membuktikan sistem berhasil memproses skala 5000 event dan melakukan deduplikasi sesuai target ~20%.

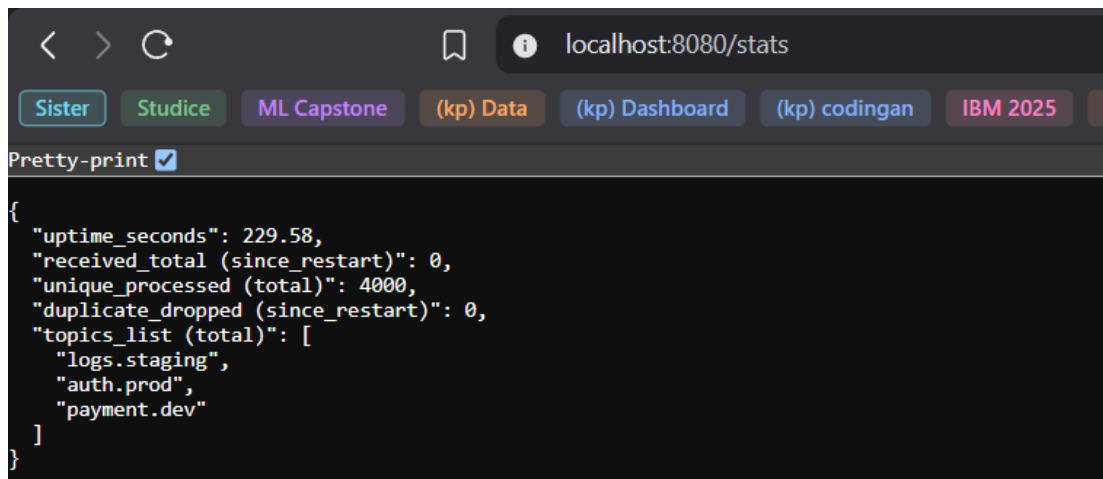
6.2 Pengujian Persistensi (Toleransi Crash)

Untuk menguji toleransi crash, container dimatikan (Ctrl+C) setelah stress test selesai, kemudian service aggregator dinyalakan kembali secara terpisah (docker-compose up aggregator).



```
C:\WINDOWS\system32\cmd. x + v
D:\Documents\dokumen numpuk\Semester 7\Sister\Codingan_UTS>docker-compose up aggregator
time="2025-10-23T19:01:06+08:00" level=warning msg="D:\Documents\dokumen numpuk\Semester 7\Sister\Codingan_UTS\doc
ker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
Attaching to aggregator-service
aggregator-service | INFO: Started server process [1]
aggregator-service | INFO: Waiting for application startup.
2025-10-23 11:01:08,118 - INFO - Server startup... Menyiapkan database.
2025-10-23 11:01:08,125 - INFO - Database 'data/dedup_store.db' berhasil disiapkan.
2025-10-23 11:01:08,126 - INFO - Memuat data event yang sudah diproses dari database...
2025-10-23 11:01:08,165 - INFO - Berhasil memuat 4000 event dari database.
2025-10-23 11:01:08,168 - INFO - Startup selesai. 4000 event unik dimuat.
2025-10-23 11:01:08,168 - INFO - Consumer task telah dijadwalkan.
2025-10-23 11:01:08,169 - INFO - Consumer task dimulai...
aggregator-service | INFO: Application startup complete.
aggregator-service | INFO: Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
aggregator-service | INFO: 127.0.0.1:48180 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:49236 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:48048 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:37554 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:44902 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:35672 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:47618 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:53180 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:52548 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:50868 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:48436 - "GET / HTTP/1.1" 200 OK
aggregator-service | INFO: 127.0.0.1:46432 - "GET / HTTP/1.1" 200 OK
View in Docker Desktop View Config Enable Watch
```

- Statistik Setelah Restart: Endpoint /stats diakses kembali setelah server restart.

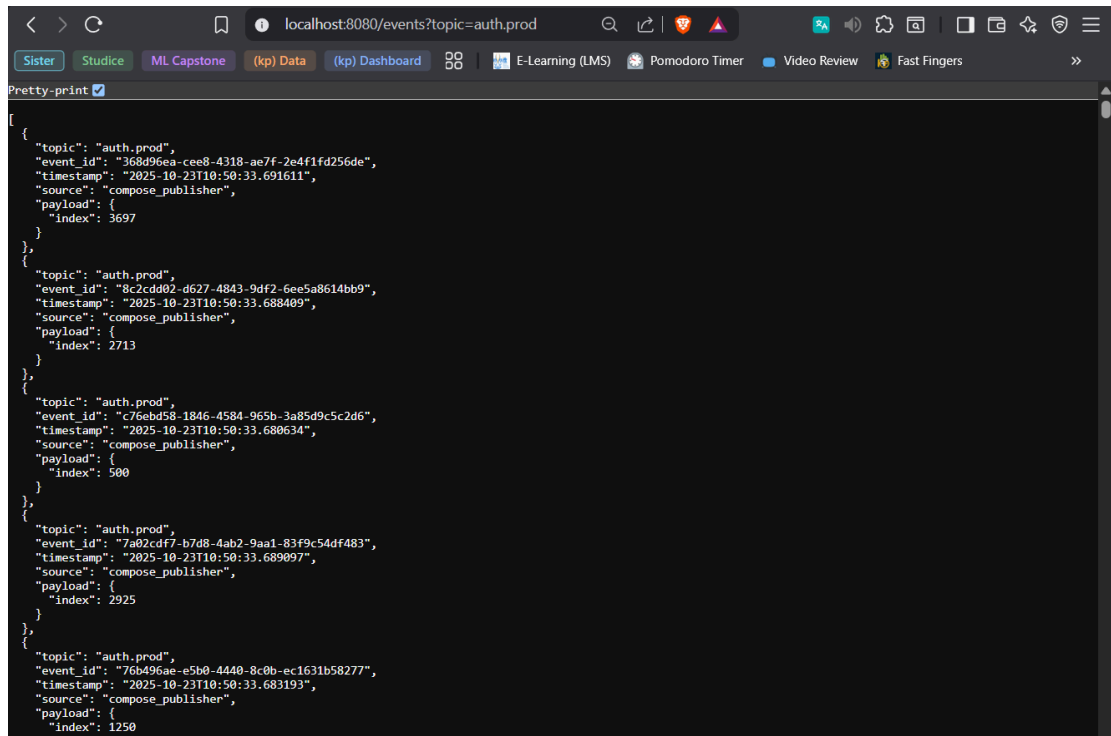


```
localhost:8080/stats
Sister Studice ML Capstone (kp) Data (kp) Dashboard (kp) codingan IBM 2025
Pretty-print [x]
{
  "uptime_seconds": 229.58,
  "received_total (since_restart)": 0,
  "unique_processed (total)": 4000,
  "duplicate_dropped (since_restart)": 0,
  "topics_list (total)": [
    "logs.staging",
    "auth.prod",
    "payment.dev"
  ]
}
```

Gambar di atas menunjukkan hasil /stats setelah restart. Terlihat received_total (since_restart) dan duplicate_dropped (since_restart) kembali ke 0 karena ini adalah counter memori. Namun, unique_processed (total) tetap menunjukkan 4000 (atau nilai asli dari stress test), membuktikan bahwa state event unik berhasil dipulihkan dari database SQLite. Daftar topics_list (total) juga berhasil dipulihkan.

- Daftar Event Unik Setelah Restart: Endpoint /events diakses untuk memastikan event unik masih tersimpan dan dapat diambil.

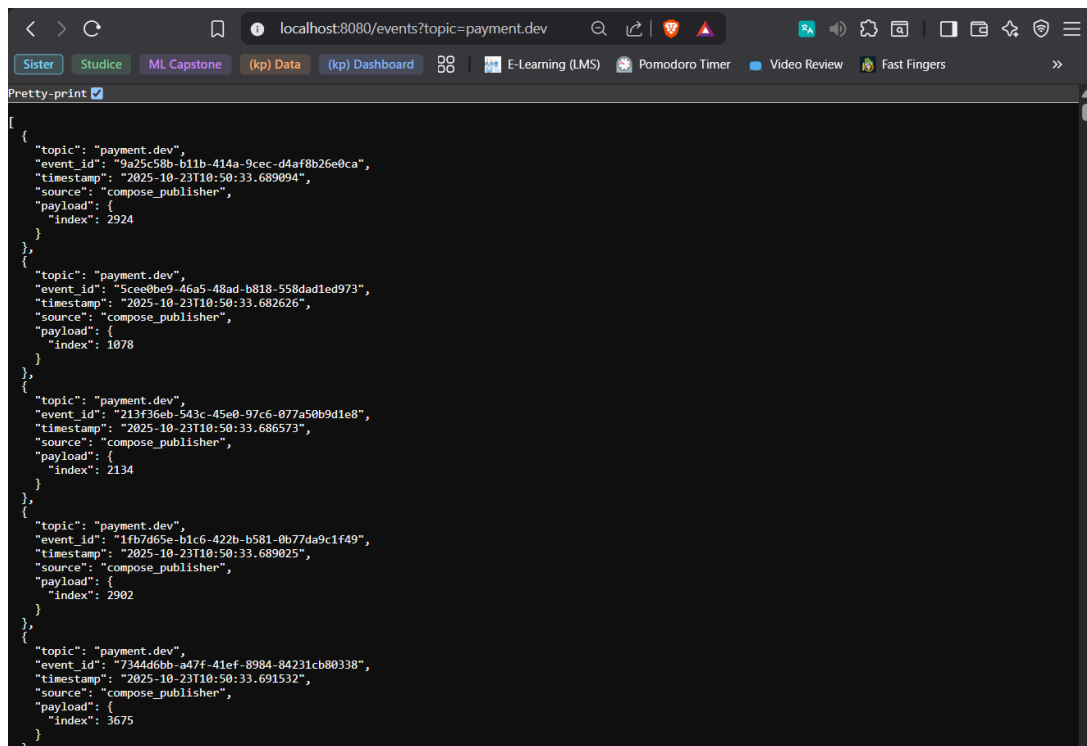
Akses <http://localhost:8080/events?topic=auth.prod>:



```
[
  {
    "topic": "auth.prod",
    "event_id": "368d96ea-cee8-4318-ae7f-2e4f1fd256de",
    "timestamp": "2025-10-23T10:50:33.691611",
    "source": "compose_publisher",
    "payload": {
      "index": 3697
    }
  },
  {
    "topic": "auth.prod",
    "event_id": "8c2cdd02-d627-4843-9df2-6ee5a8614bb9",
    "timestamp": "2025-10-23T10:50:33.688409",
    "source": "compose_publisher",
    "payload": {
      "index": 2713
    }
  },
  {
    "topic": "auth.prod",
    "event_id": "c76ebd58-1846-4584-965b-3a85d9c5c2d6",
    "timestamp": "2025-10-23T10:50:33.680634",
    "source": "compose_publisher",
    "payload": {
      "index": 500
    }
  },
  {
    "topic": "auth.prod",
    "event_id": "7a02cdf7-b7d8-4ab2-9aa1-83f9c54df483",
    "timestamp": "2025-10-23T10:50:33.689097",
    "source": "compose_publisher",
    "payload": {
      "index": 2925
    }
  },
  {
    "topic": "auth.prod",
    "event_id": "76b496ae-e5b0-4440-8c0b-ec1631b58277",
    "timestamp": "2025-10-23T10:50:33.683193",
    "source": "compose_publisher",
    "payload": {
      "index": 1250
    }
  }
]
```

Gambar di atas menunjukkan hasil akses ke `/events` dengan filter `topic=auth.prod` setelah restart. Terlihat daftar event JSON yang panjang dan relevan dengan topik 'auth.prod', membuktikan data tersimpan, termuat ulang, dan filter bekerja.

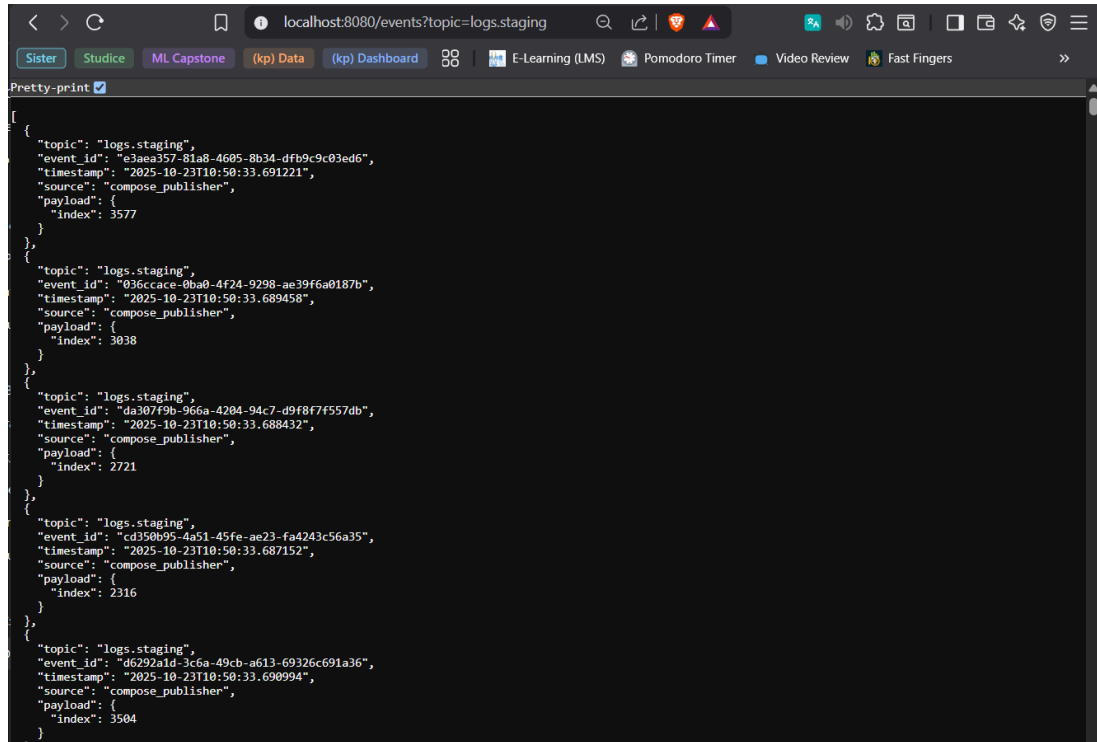
Akses `http://localhost:8080/events?topic=payment.dev`:



```
[
  {
    "topic": "payment.dev",
    "event_id": "9a25c58b-b11b-414a-9cec-d4af8b26e0ca",
    "timestamp": "2025-10-23T10:50:33.689094",
    "source": "compose_publisher",
    "payload": {
      "index": 2924
    }
  },
  {
    "topic": "payment.dev",
    "event_id": "5cee0ba9-46a5-48ad-b818-558dad1ed973",
    "timestamp": "2025-10-23T10:50:33.682626",
    "source": "compose_publisher",
    "payload": {
      "index": 1078
    }
  },
  {
    "topic": "payment.dev",
    "event_id": "213f36eb-543c-45e0-97c6-077a50b9d1e8",
    "timestamp": "2025-10-23T10:50:33.686573",
    "source": "compose_publisher",
    "payload": {
      "index": 2134
    }
  },
  {
    "topic": "payment.dev",
    "event_id": "1fb7d65e-b1c6-422b-b581-0b77da9c1f49",
    "timestamp": "2025-10-23T10:50:33.689025",
    "source": "compose_publisher",
    "payload": {
      "index": 2902
    }
  },
  {
    "topic": "payment.dev",
    "event_id": "7344d6bb-a47f-41ef-8984-84231cb80338",
    "timestamp": "2025-10-23T10:50:33.691532",
    "source": "compose_publisher",
    "payload": {
      "index": 3675
    }
  }
]
```

Gambar di atas menunjukkan hasil akses ke /events dengan filter topic=payment.dev setelah restart. Terlihat daftar event JSON yang panjang dan berbeda dari topik sebelumnya, membuktikan filter topik berfungsi untuk nilai yang berbeda.

Akses <http://localhost:8080/events?topic=logs.staging>:



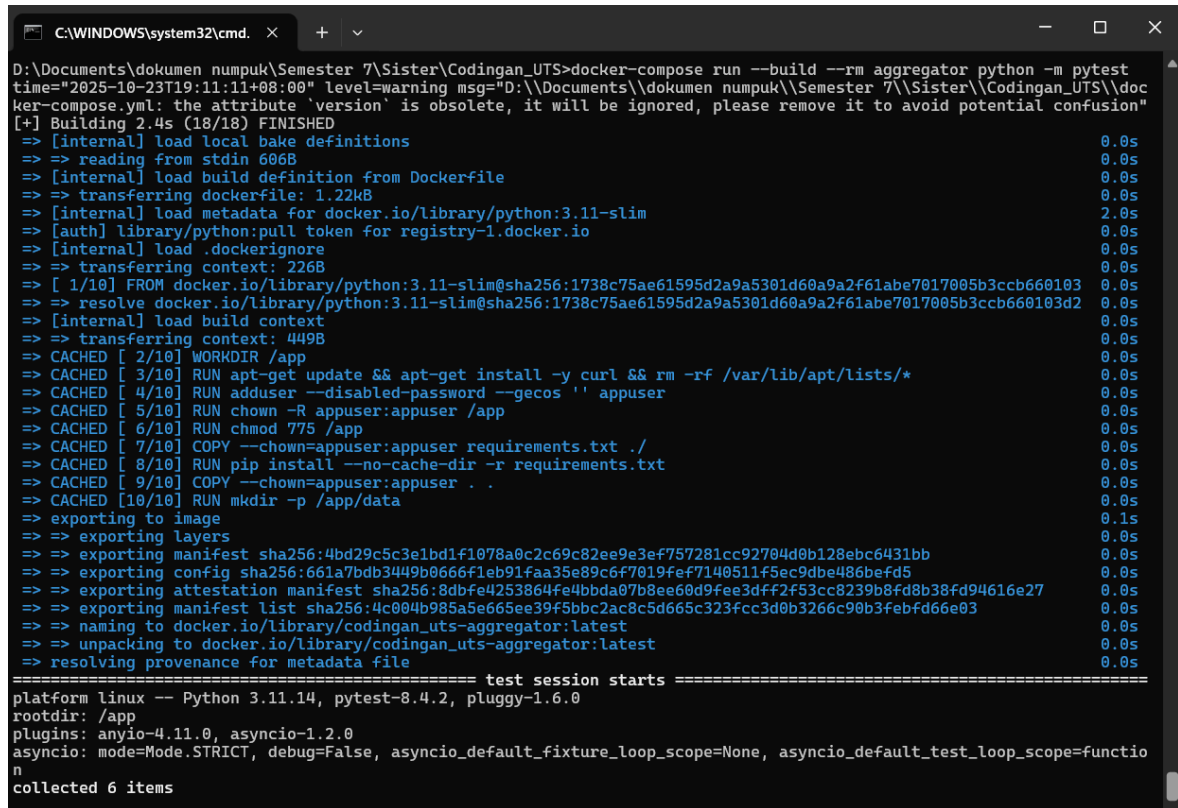
The screenshot shows a web browser window with the address bar displaying `localhost:8080/events?topic=logs.staging`. The browser's developer tools are open, showing the 'Pretty-print' view of the JSON response. The JSON is an array of five event objects, each containing 'topic', 'event_id', 'timestamp', 'source', and 'payload' (with an 'index' field). All events have the topic 'logs.staging' and a source of 'compose_publisher'.

```
[
  {
    "topic": "logs.staging",
    "event_id": "e3aea357-81a8-4605-8b3d-dfb9c9c03ed6",
    "timestamp": "2025-10-23T10:50:33.691221",
    "source": "compose_publisher",
    "payload": {
      "index": 3577
    }
  },
  {
    "topic": "logs.staging",
    "event_id": "036ccace-0ba0-4f24-9298-ae39f6a0187b",
    "timestamp": "2025-10-23T10:50:33.689458",
    "source": "compose_publisher",
    "payload": {
      "index": 3038
    }
  },
  {
    "topic": "logs.staging",
    "event_id": "da307f9b-966a-4204-94c7-d9f8f7f557db",
    "timestamp": "2025-10-23T10:50:33.688432",
    "source": "compose_publisher",
    "payload": {
      "index": 2721
    }
  },
  {
    "topic": "logs.staging",
    "event_id": "cd350b95-4a51-45fe-ae23-fa4243c56a35",
    "timestamp": "2025-10-23T10:50:33.687152",
    "source": "compose_publisher",
    "payload": {
      "index": 2316
    }
  },
  {
    "topic": "logs.staging",
    "event_id": "d6292a1d-3c6a-49cb-a613-69326c691a36",
    "timestamp": "2025-10-23T10:50:33.690994",
    "source": "compose_publisher",
    "payload": {
      "index": 3504
    }
  }
]
```

Gambar di atas menunjukkan hasil akses ke /events dengan filter topic=logs.staging setelah restart. Sekali lagi terlihat daftar event JSON yang panjang dan spesifik untuk topik ini.

6.3 Hasil Unit Tests

Unit tests dijalankan menggunakan pytest di dalam container terpisah untuk memvalidasi fungsionalitas inti secara terisolasi.



```
C:\WINDOWS\system32\cmd. x + v
D:\Documents\dokumen numpuk\Semester 7\Sister\Codingan_UTS>docker-compose run --build --rm aggregator python -m pytest
time="2025-10-23T19:11:11+08:00" level=warning msg="D:\\Documents\\dokumen numpuk\\Semester 7\\Sister\\Codingan_UTS\\doc
ker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Building 2.4s (18/18) FINISHED
=> [internal] load local bake definitions 0.0s
=> => reading from stdin 606B 0.0s
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 1.22kB 0.0s
=> [internal] load metadata for docker.io/library/python:3.11-slim 2.0s
=> [auth] library/python:pull token for registry-1.docker.io 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 226B 0.0s
=> [ 1/10] FROM docker.io/library/python:3.11-slim@sha256:1738c75ae61595d2a9a5301d60a9a2f61abe7017005b3ccb660103 0.0s
=> => resolve docker.io/library/python:3.11-slim@sha256:1738c75ae61595d2a9a5301d60a9a2f61abe7017005b3ccb660103d2 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 449B 0.0s
=> CACHED [ 2/10] WORKDIR /app 0.0s
=> CACHED [ 3/10] RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/* 0.0s
=> CACHED [ 4/10] RUN adduser --disabled-password --gecos '' appuser 0.0s
=> CACHED [ 5/10] RUN chown -R appuser:appuser /app 0.0s
=> CACHED [ 6/10] RUN chmod 775 /app 0.0s
=> CACHED [ 7/10] COPY --chown=appuser:appuser requirements.txt ./ 0.0s
=> CACHED [ 8/10] RUN pip install --no-cache-dir -r requirements.txt 0.0s
=> CACHED [ 9/10] COPY --chown=appuser:appuser . . 0.0s
=> CACHED [10/10] RUN mkdir -p /app/data 0.0s
=> exporting to image 0.1s
=> => exporting layers 0.0s
=> => exporting manifest sha256:4bd29c5c3e1bd1f1078a0c2c69c82ee9e3ef757281cc92704d0b128ebc6431bb 0.0s
=> => exporting config sha256:661a7bdb3449b0666f1eb91faa35e89c6f7019fef7140511f5ec9dbe486befd5 0.0s
=> => exporting attestation manifest sha256:8dbfe4253864fe4bbda07b8ee60d9fee3dfff2f53cc8239b8fd8b38fd94616e27 0.0s
=> => exporting manifest list sha256:4c004b985a5e665ee39f5bbcc2ac8c5d665c323fcc3d0b3266c90b3febfd66e03 0.0s
=> => naming to docker.io/library/codingan_uts-aggregator:latest 0.0s
=> => unpacking to docker.io/library/codingan_uts-aggregator:latest 0.0s
=> resolving provenance for metadata file 0.0s
===== test session starts =====
platform linux -- Python 3.11.14, pytest-8.4.2, pluggy-1.6.0
rootdir: /app
plugins: anyio-4.11.0, asyncio-1.2.0
asyncio: mode=Mode.STRICT, debug=False, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=functio
n
collected 6 items
```

Gambar di atas menunjukkan output dari perintah `docker-compose run --build --rm aggregator python -m pytest`. Hasil “6 passed” mengonfirmasi bahwa semua 6 unit test yang mencakup pengiriman event tunggal, deduplikasi, konsistensi API `/stats` dan `/events`, filter topic, pengiriman batch, dan validasi skema, berhasil dijalankan tanpa error.

6.4 Kesimpulan Performa

- Fungsionalitas: Sistem berhasil memenuhi semua persyaratan fungsional: menerima event, melakukan deduplikasi secara akurat, bersifat idempotent, persisten terhadap restart, dan menyediakan API untuk statistik serta pengambilan event.
- Performa: Sistem mampu menangani skala minimum 5000 event dan tetap responsif (API `/stats` dan `/events` merespons dengan cepat setelah stress test). Bottleneck utama adalah throughput penulisan ke SQLite, yang membatasi kecepatan pemrosesan consumer task.
- Observability: Logging yang jelas membedakan event unik dan duplikat. Endpoint `/stats` memberikan gambaran umum kondisi sistem.

7. Kesimpulan

Proyek Ujian Tengah Semester ini berhasil mengimplementasikan layanan Pub-Sub log aggregator yang fungsional sesuai dengan spesifikasi yang diberikan. Sistem ini menunjukkan kemampuan untuk menerima event log secara asinkron, melakukan deduplikasi secara efektif menggunakan database SQLite yang persisten, dan menangani duplikasi event yang disimulasikan. Pengujian skala dan unit tests yang dilakukan mengonfirmasi bahwa sistem bekerja sesuai rancangan, terutama dalam aspek idempotency dan toleransi crash melalui pemulihan state setelah restart.

Implementasi ini memberikan pengalaman praktis dalam menerapkan konsep-konsep fundamental sistem terdistribusi yang dipelajari dari buku utama, mencakup arsitektur, komunikasi, penamaan, toleransi kegagalan, dan konsistensi. Meskipun terdapat trade-off performa terkait penggunaan SQLite untuk deduplikasi pada beban tinggi, proyek ini berhasil membangun fondasi log aggregator yang andal dan idempotent dalam lingkungan Docker. Selain itu, sistem ini dapat menjadi dasar bagi pengembangan lebih lanjut dengan integrasi message broker seperti Kafka atau Redis Stream guna meningkatkan skalabilitas dan durabilitas.

Daftar Pustaka

- Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2012). *Distributed systems: Concepts and design* (5th ed.). Addison-Wesley.
https://api.pageplace.de/preview/DT0400.9781447930174_A24570107/preview-9781447930174_A24570107.pdf
- Tanenbaum, A. S., & Van Steen, M. (2023). *Distributed systems* (4th ed., Version 4.01). Maarten van Steen. <https://www.distributed-systems.net/index.php/books/ds4/>

Lampiran

Link Youtube: <https://youtu.be/Emw6gazfT4k?si=GRph4NJm5XabcM39>

Link Github: <https://github.com/zayfitri/log-aggregator-py>