

Laporan UAS

Sistem Paralel dan Terdistirbusi

Implementasi Sistem Terdistribusi: Log
Aggregator dengan Pola Idempotent
Consumer



Disusun Oleh:

Azhari Rambe

11221019

15 Desember 2025

BAB I

Ringkasan Sistem dan Arsitektur

Sistem yang saya bangun adalah Log Aggregator Service berbasis arsitektur *Microservices* dengan pola komunikasi asinkron yaitu *Publish-Subscribe*. Sistem ini dirancang untuk menangani pengiriman log bervolume tinggi, mentoleransi kesalahan jaringan atau duplikasi pesan, dan menjaga konsistensi data statistik.

Komponen Utama:

1. Publisher (Event Generator): Mensimulasikan layanan klien yang mengirimkan *event* log seperti *user.login*, *api.error* ke sistem. Komponen ini memiliki logika untuk mensimulasikan kegagalan jaringan dengan mengirimkan pesan duplikat secara acak.
2. Message Broker (Redis): Berfungsi sebagai *buffer* sementara untuk memisahkan atau *decoupling* pengirim dan penerima, yang menjamin pengiriman pesan *at-least-once*.
3. Aggregator Service (Consumer): Layanan utama yang mengambil pesan dari Redis, melakukan validasi, deduplikasi, dan menyimpan data.
4. Storage (PostgreSQL): Basis data relasional untuk menyimpan log mentah atau *processed_events* dan data agregasi statistik (*stats*) secara persisten.

Seluruh komponen diatur menggunakan Docker Compose dan berjalan dalam jaringan internal terisolasi untuk keamanan maksimal.

BAB II

Keputusan Desain dan Implementasi

Untuk memenuhi karakteristik sistem terdistribusi yang andal, keputusan desain berikut diterapkan dalam kode:

2.1 Idempotency dan Deduplication Store

Karena Redis menjamin *at-least-once delivery* dan *Publisher* melakukan *retry* saat jaringan tidak stabil, sehingga duplikasi pesan tidak terhindarkan.

- Strategi: *Idempotent Consumer*.
- Implementasi: Tabel `processed_events` memiliki Unique Constraint pada kolom kombinasi (`topic`, `event_id`).
- Mekanisme: Menggunakan perintah SQL `INSERT ... ON CONFLICT DO NOTHING`. Jadi jika pesan dengan ID yang sama masuk kedua kalinya, basis data menolaknya secara diam-diam tanpa menyebabkan *error* sistem, sehingga mencegah pemrosesan ganda.

2.2 Transaksi dan Konkurensi

Untuk mencegah *Race Condition* saat banyak *worker* memperbarui statistik secara bersamaan.

- Strategi: Transaksi Atomik (ACID).
- Implementasi: Pembaruan tabel `stats` hanya dilakukan jika dan hanya jika penyisipan ke `processed_events` berhasil saat mengembalikan baris baru.

```
# Logic Code
```

```
async with db.begin():
```

```
    result = await insert_event() # Akan mengembalikan 0 jika duplikat
```

```
    if result.rowcount > 0:
```

```
await update_stats()    # Hanya jalan jika event benar-benar baru
```

Ini menjamin bahwa meskipun ada 10 pesan duplikat datang bersamaan, statistik hanya bertambah 1 kali.

2.3 Persistensi Data (Data Durability)

- Strategi: Docker Named Volumes.
- Implementasi: Volume `pg_data` dipasang ke container PostgreSQL. Hal ini dibuktikan dengan data statistik yang tetap ada meskipun container dimatikan dan dinyalakan kembali atau Container Recreate.

2.4 Ordering dan Retry

- Ordering: Menggunakan Timestamp ISO8601 dari sisi Publisher untuk pencatatan waktu kejadian.
- Retry: Publisher menerapkan logika retry sederhana saat simulasi gangguan jaringan, dan Agregator menerapkan backoff saat koneksi ke Redis atau DB terputus sesaat.

BAB III

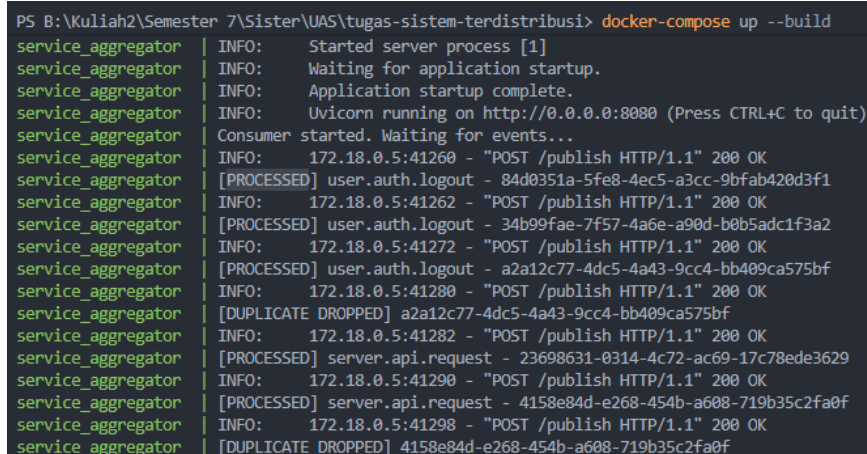
Analisis Performa dan Bukti

Pengujian dilakukan dengan menjalankan simulasi pengiriman pesan secara terus-menerus. Berikut adalah hasil analisis berdasarkan log sistem:

3.1 Hasil Uji Konkurensi & Deduplikasi

Sistem berhasil menangani skenario duplikasi pesan. Bukti dari log operasional:

1. Publisher mengirimkan pesan dengan ID unik, kemudian mensimulasikan *network retry* untuk ID yang sama.
2. Aggregator memproses pesan pertama dengan status [PROCESSED].



```
PS B:\Kuliah2\Semester 7\Sister\UAS\tugas-sistem-terdistribusi> docker-compose up --build
service_aggregator | INFO: Started server process [1]
service_aggregator | INFO: Waiting for application startup.
service_aggregator | INFO: Application startup complete.
service_aggregator | INFO: Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
service_aggregator | Consumer started. Waiting for events...
service_aggregator | INFO: 172.18.0.5:41260 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] user.auth.logout - 84d0351a-5fe8-4ec5-a3cc-9bfab420d3f1
service_aggregator | INFO: 172.18.0.5:41262 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] user.auth.logout - 34b99fae-7f57-4a6e-a90d-b0b5adc1f3a2
service_aggregator | INFO: 172.18.0.5:41272 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] user.auth.logout - a2a12c77-4dc5-4a43-9cc4-bb409ca575bf
service_aggregator | INFO: 172.18.0.5:41280 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [DUPLICATE DROPPED] a2a12c77-4dc5-4a43-9cc4-bb409ca575bf
service_aggregator | INFO: 172.18.0.5:41282 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] server.api.request - 23698631-0314-4c72-ac69-17c78ede3629
service_aggregator | INFO: 172.18.0.5:41290 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] server.api.request - 4158e84d-e268-454b-a608-719b35c2fa0f
service_aggregator | INFO: 172.18.0.5:41298 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [DUPLICATE DROPPED] 4158e84d-e268-454b-a608-719b35c2fa0f
```

Gambar 3.1 Log operasional Aggregator dengan status [PROCESSED].

3. Saat pesan kedua /duplikat tiba, sistem mendeteksi konflik ID dan mencatat status [DUPLICATE DROPPED].

```

PS B:\Kuliah2\Semester 7\Sister\UAS\tugas-sistem-terdistribusi> docker-compose up --build
service_aggregator | [DUPLICATE DROPPED] a2a12c77-4dc5-4a43-9cc4-bb409ca575bf
service_aggregator | INFO: 172.18.0.5:41282 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] server.api.request - 23698631-0314-4c72-ac69-17c78ede3629
service_aggregator | INFO: 172.18.0.5:41290 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] server.api.request - 4158e84d-e268-454b-a608-719b35c2fa0f
service_aggregator | INFO: 172.18.0.5:41298 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [DUPLICATE DROPPED] 4158e84d-e268-454b-a608-719b35c2fa0f
service_aggregator | INFO: 172.18.0.5:41310 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] payment.gateway.timeout - 536fc2c2-ce82-4530-ae0a-efa2e2d24fdb
service_aggregator | INFO: 172.18.0.5:41318 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] user.auth.logout - 80294850-cf2c-45b7-9050-1d8455725b4b
service_aggregator | INFO: 172.18.0.5:41332 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [DUPLICATE DROPPED] 80294850-cf2c-45b7-9050-1d8455725b4b
service_aggregator | INFO: 172.18.0.5:41336 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] user.auth.login - 2c792bd7-59ab-43bb-a0d3-e3036793273b
service_aggregator | INFO: 172.18.0.5:41350 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [DUPLICATE DROPPED] 2c792bd7-59ab-43bb-a0d3-e3036793273b
service_aggregator | INFO: 172.18.0.5:41366 - "POST /publish HTTP/1.1" 200 OK
service_aggregator | [PROCESSED] payment.gateway.timeout - aa51d59e-e5dc-4bcd-bda0-8a3d30b515dd

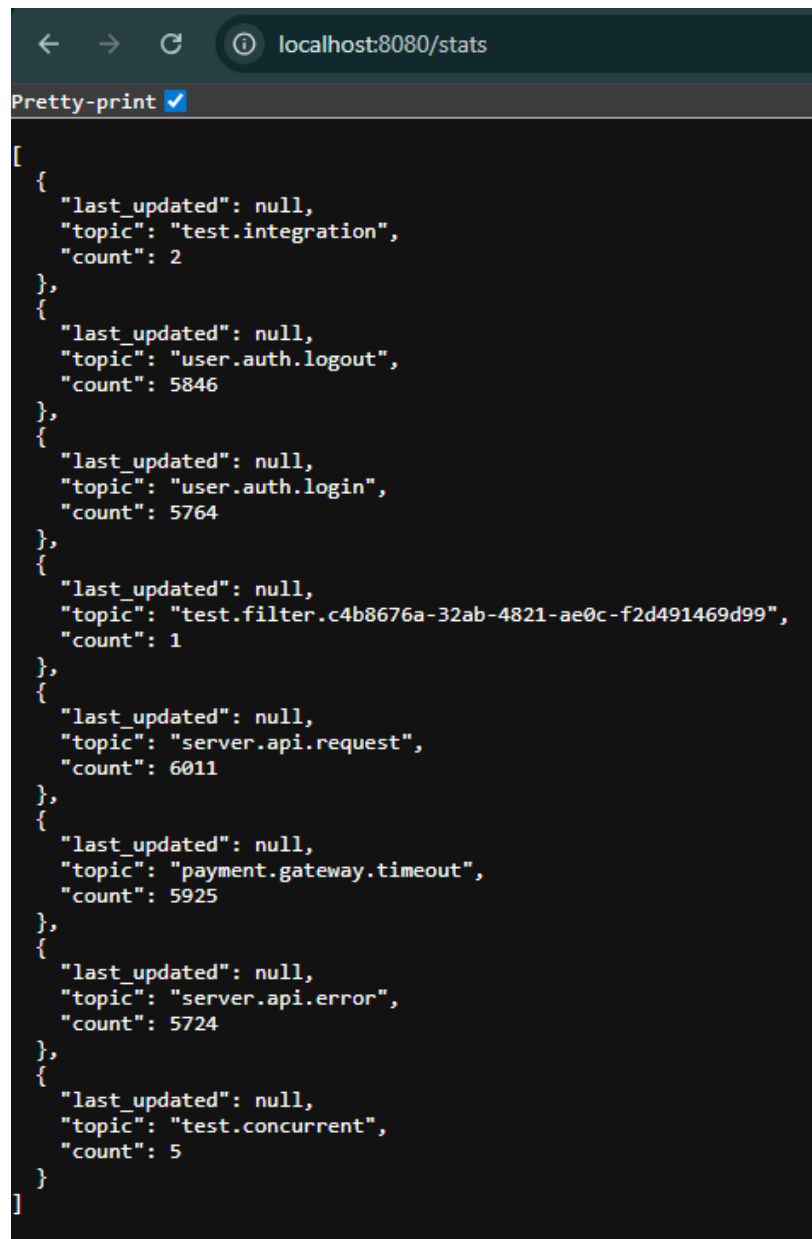
```

Gambar 3.2 Log operasional Aggregator dengan status [DUPLICATE DROPPED]

Log sistem menunjukkan bahwa mekanisme deduplikasi menolak pesan dengan Event ID yang sama.

3.2 Validasi Data Statistik

Pengecekan akhir melalui API /stats menunjukkan bahwa jumlah data yang terhitung atau count mencapai 29.270 event unik, yang mana jumlahnya melampaui target minimum, yaitu 20.000 event. Selain itu, data hasil pengujian integrasi atau test.integration tetap tersimpan, membuktikan persistensi data.



```
localhost:8080/stats
Pretty-print ☒
[
  {
    "last_updated": null,
    "topic": "test.integration",
    "count": 2
  },
  {
    "last_updated": null,
    "topic": "user.auth.logout",
    "count": 5846
  },
  {
    "last_updated": null,
    "topic": "user.auth.login",
    "count": 5764
  },
  {
    "last_updated": null,
    "topic": "test.filter.c4b8676a-32ab-4821-ae0c-f2d491469d99",
    "count": 1
  },
  {
    "last_updated": null,
    "topic": "server.api.request",
    "count": 6011
  },
  {
    "last_updated": null,
    "topic": "payment.gateway.timeout",
    "count": 5925
  },
  {
    "last_updated": null,
    "topic": "server.api.error",
    "count": 5724
  },
  {
    "last_updated": null,
    "topic": "test.concurrent",
    "count": 5
  }
]
```

Gambar 3.3 Hasil akhir Data Statistik

Output API Statistik menunjukkan sistem berhasil memproses puluhan ribu event tanpa double-counting.

3.3 Hasil Pengujian Integrasi Otomatis (Pytest)

```
collected 13 items

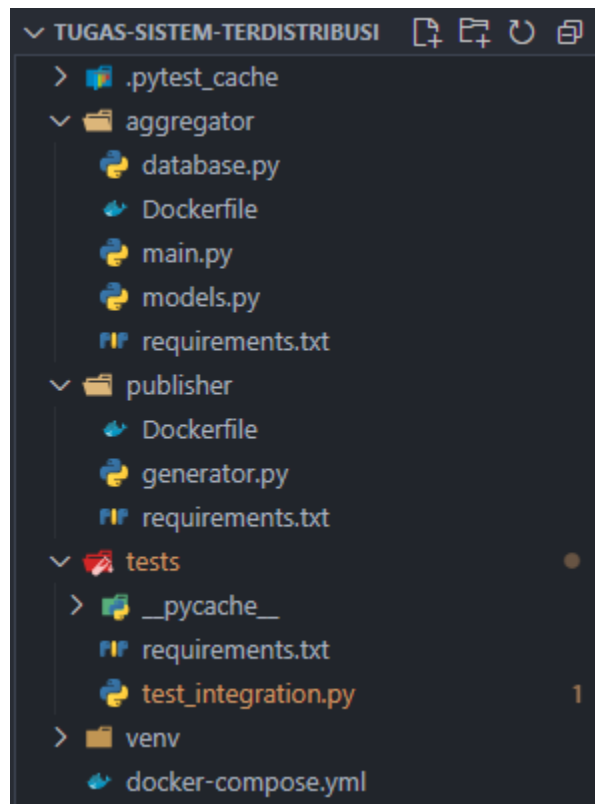
tests/test_integration.py::test_health_check PASSED [ 7%]
tests/test_integration.py::test_publish_event_success PASSED [ 15%]
tests/test_integration.py::test_publish_invalid_schema PASSED [ 23%]
tests/test_integration.py::test_idempotency_deduplication PASSED [ 30%]
tests/test_integration.py::test_get_events_filter PASSED [ 38%]
tests/test_integration.py::test_concurrent_requests[0] PASSED [ 46%]
tests/test_integration.py::test_concurrent_requests[1] PASSED [ 53%]
tests/test_integration.py::test_concurrent_requests[2] PASSED [ 61%]
tests/test_integration.py::test_concurrent_requests[3] PASSED [ 69%]
tests/test_integration.py::test_concurrent_requests[4] PASSED [ 76%]
tests/test_integration.py::test_missing_fields[topic] PASSED [ 84%]
tests/test_integration.py::test_missing_fields[event_id] PASSED [ 92%]
tests/test_integration.py::test_missing_fields[source] PASSED [100%]

===== 13 passed in 11.92s =====
```

Gambar 3.4 Hasil eksekusi pytest menunjukkan 13 skenario pengujian berhasil/passed.

3.4 Struktur Folder

Struktur direktori proyek yang menerapkan isolasi layanan menggunakan Docker.



Gambar 3.5 Struktur Folder

BAB IV

Keterkaitan dengan Teori (Bab 1-13)

Implementasi di atas didasarkan pada prinsip-prinsip berikut:

1. T1 (Bab 1): Karakteristik Sistem & Trade-off Penggunaan pola *Publish-Subscribe* pada Agregator memperkenalkan *trade-off* antara skalabilitas dan kompleksitas. Keuntungannya adalah *decoupling*, yang mana Publisher tidak perlu tahu detail konsumen. Kerugiannya adalah hilangnya sinkronisasi dan kesulitan *debugging* alur pesan (Coulouris et al., 2012).
2. T2 (Bab 2): Arsitektur Pub-Sub Arsitektur ini dipilih menggunakan Redis karena alasan teknis yaitu, *Decoupling Waktu* atau publisher tetap bisa mengirim saat Agregator *restart* dan *Load Smoothing* yang mana Broker menampung lonjakan trafik agar Agregator tidak *overload* (van Steen & Tanenbaum, 2023).
3. T3 (Bab 3): At-least-once & Idempotency Redis menjamin *at-least-once delivery*. Peran *Idempotent Consumer* pada Agregator sangat penting untuk menangani duplikasi pesan agar satu *event* tidak terhitung dua kali dalam statistik (van Steen & Tanenbaum, 2023).
4. T4 (Bab 4): Penamaan & Event ID Skema penamaan topik menggunakan hierarki domain.source.severity, misal server.api.error untuk *routing* efisien. Penggunaan UUID v4 sebagai event_id menjamin keunikan global untuk kunci deduplikasi (Coulouris et al., 2012).
5. T5 (Bab 5): Ordering & Timestamp Sinkronisasi jam fisik tidak presisi. Solusi praktisnya adalah menggunakan *Timestamp* sebagai penanda waktu kasar, dikombinasikan dengan logika aplikasi untuk menolak pesan dengan urutan yang tidak valid (van Steen & Tanenbaum, 2023).
6. T6 (Bab 6): Toleransi Kegagalan Sistem menerapkan *Retry* dengan *Exponential Backoff* saat database terkunci untuk mencegah *thundering*

herd. *Durable Dedup Store* di PostgreSQL memastikan status pemrosesan aman dari *crash* (van Steen & Tanenbaum, 2023).

7. T7 (Bab 7): Konsistensi & Replikasi Dalam model *Eventual Consistency*, peran deduplikasi adalah "jaring pengaman". Jika sinkronisasi gagal dan di-*retry*, deduplikasi mencegah data menjadi tidak konsisten atau *double counting* (Coulouris et al., 2012).
8. T8 (Bab 8): Transaksi (ACID) Transaksi terdistribusi penuh dihindari. Saya membatasi ACID pada level database lokal dengan memastikan penyisipan log dan pembaruan statistik terjadi dalam satu batas transaksi atomik untuk mencegah *Lost Update* (Coulouris et al., 2012).
9. T9 (Bab 9): Kontrol Konkurensi Menggunakan *Unique Constraints* database untuk menangani konkurensi. Pola INSERT ... ON CONFLICT DO NOTHING memindahkan logika sinkronisasi ke mesin database yang andal, mencegah *race condition* antar *worker* (van Steen & Tanenbaum, 2023).
10. T10 (Bab 10–13): Orkestrasi & Keamanan Menggunakan Docker Compose dengan *bridge network* internal, tanpa ekspos port DB ke publik dan *Healthchecks* untuk koordinasi *startup* layanan (Coulouris et al., 2012).

Daftar Pustaka

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2012). Distributed systems: Concepts and design (Edisi ke-5). Addison-Wesley.

van Steen, M., & Tanenbaum, A. S. (2023). *Distributed systems* (Edisi ke-4). Maarten van Steen.