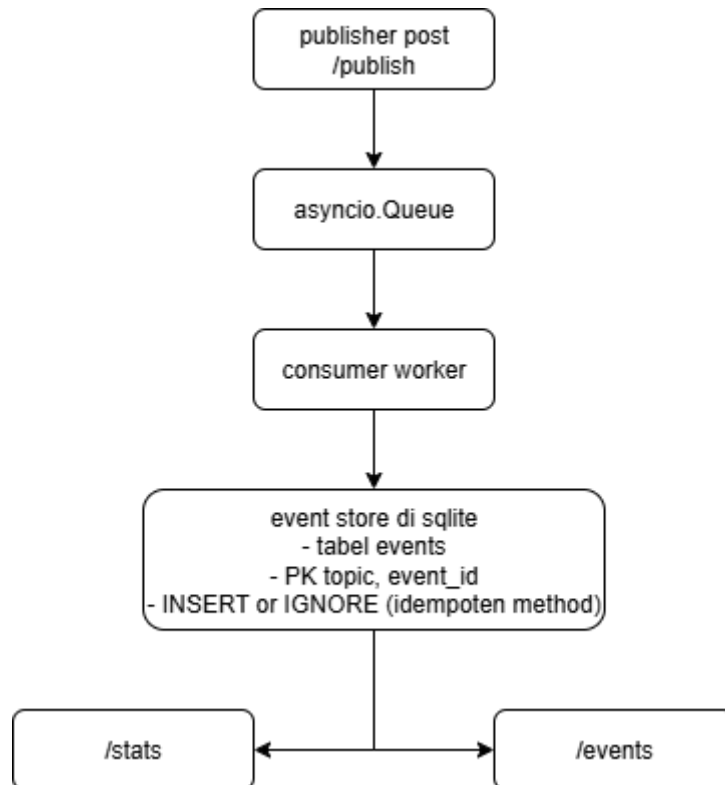


Nama : Erizki Fadli  
NIM : 11221014  
Kelas : SisTer B

## UTS

### Ringkasan sistem dan arsitektur



publisher (aggregator): menerima request, validasi, dan enqueue.

asyncio.Queue: sistem dibuat buffer in-memory terlebih dahulu agar dalam kasus penerimaan request banyak, tidak terhambat I/O disk.

Consumer worker: mendapatkan event dari queue lalu melakukan dedup dan persist ke SQLite.

SQLite EventStore (persisten): data yang sudah melalui queue disimpan disini secara permanen sehingga tahan restart server. Endpoint stats dan event membaca data disini.

/stats dan /events: membaca data event yang telah tersimpan di database.

### Keputusan desain

#### 1. Idempotency pada consumer

Idempotency pada sisi consumer dipilih karena ini adalah salah satu metode untuk mencapai *exactly-once processing* diatas sistem *at-least-once*. Dengan PK topic dan event\_id, serta method INSERT OR IGNORE di database, data duplikasi dari publisher tidak akan diproses ulang

#### 2. Dedup store pada SQLite

Menggunakan PK gabungan topic dan event\_id, event yang sama dari publish tidak diproses ulang. SQLite dipilih karena bersifat embedded, dan atomic via PK serta memiliki performa baik untuk menangani beban baca-tulis 5000 event dengan 20% diantaranya data duplikat.

### 3. Total Ordering

Sistem tidak mengimplementasikan total ordering dikarenakan perannya sudah diambil alih oleh properti processed\_at yang tercantum pada setiap data publish yang masuk dan properti ini dapat digunakan untuk mengurutkan data dengan konsisten.

### 4. Retry

Untuk mencegah masalah retry, sistem mengimplementasikan idempotent consumer dan dedup store persisten dimana jika terdapat request duplikat yang masuk, hanya request pertama yang akan diproses dan dalam keadaan crash atau restart server, event yang pertama yang sudah selesai diproses akan disimpan dan diingat agar tidak memproses event duplikat kedua.

## Analisis Performa dan Metrik

Pada program terdapat lima unit test yang menguji fungsionalitas dan performa sistem yakni dedup, persistence, schema, get stats dan events, dan stress test dengan 5000 data dan 20% duplikat yang mana semuanya berhasil dilakukan dalam waktu 0.49 detik.

```
2025-10-24 01:01:58,958 INFO [DUP] drop load/e-997
2025-10-24 01:01:58,958 INFO [DUP] drop load/e-998
2025-10-24 01:01:58,958 INFO [DUP] drop load/e-999
2025-10-24 01:01:58,959 INFO HTTP Request: GET http://testserver/stats "HTTP/1.1 200 OK"
2025-10-24 01:01:58,960 INFO HTTP Request: GET http://testserver/stats "HTTP/1.1 200 OK"

INFO aggregator:main.py:39 [DUP] drop load/e-995
INFO aggregator:main.py:39 [DUP] drop load/e-996
INFO aggregator:main.py:39 [DUP] drop load/e-997
INFO aggregator:main.py:39 [DUP] drop load/e-998
INFO aggregator:main.py:39 [DUP] drop load/e-999
INFO httpx:_client.py:1038 HTTP Request: GET http://testserver/stats "HTTP/1.1 200 OK"
INFO httpx:_client.py:1038 HTTP Request: GET http://testserver/stats "HTTP/1.1 200 OK"

===== short test summary info =====
PASSED tests/test_dedup.py::test_dedup_only_once_processed
PASSED tests/test_persistence.py::test_event_and_dedup_persist_across_reopen
PASSED tests/test_schema.py::test_invalid_schema_rejected
PASSED tests/test_stats_events.py::test_events_and_stats_consistent
PASSED tests/test_stress.py::test_stress_batch_5000_with_duplicates
===== 5 passed in 0.49s =====
```

## Keterkaitan Sistem Dengan Bab 1-7 Buku Utama

### T1

Sistem terdistribusi dicirikan oleh heterogenitas, skalabilitas, desentralisasi kontrol, dan adanya partial failures. Tujuan desain klasik meliputi accessibility, openness, distribution transparency, dan scalability, namun pencapaiannya melibatkan trade-off berupa transparansi penuh yang sering berakibat kompleksitas tinggi. Pada log-aggregator Pub-Sub, keputusan penting adalah memisahkan jalur penerimaan (HTTP + enqueue) dari jalur pemrosesan (consumer), memilih penyimpanan persisten (SQLite) untuk idempoten/dedup, dan menerima sifat at-least-once dari pengiriman. Transparansi lokasi dimana publisher tidak perlu tahu letak consumer/DB, mengorbankan total ordering global demi throughput tinggi. Prinsip

“jangan anggap jaringan andal/aman/seragam” juga relevan: karena duplikasi dan retries tak terhindarkan, konsumen harus idempoten. Intinya, agregator menukar sebagian consumption ordering dengan availability dan kesederhanaan operasi. (van Steen & Tanenbaum, 2023). Rujukan bab: Bab 1 (Introduction).

## **T2**

Model client–server cocok bila ada pasangan peran jelas (klien dan server) dan kebutuhan kontrol/koordinasi terpusat. Publish–subscribe memisahkan produsen dari konsumen melalui topik, memungkinkan fan-out dan loose coupling. Untuk log-aggregator, Pub-Sub lebih tepat karena: (1) decoupling in space/time: publisher mengirim ke endpoint/tema tanpa mengetahui subscriber; (2) elastisitas: mudah menambah konsumen/pekerja; (3) dukungan multicast/message-oriented middleware (MOM) untuk pengiriman ke banyak pihak. Kelemahannya berupa urutan global sulit dan semantik pengiriman bervariasi yang ditangani dengan idempoten dan dedup di sisi konsumen. Arsitektur Bab 2 juga menekankan pola middleware wrappers/interceptors dan pemosisian RPC/MOM; aggregator ini memosisikan HTTP ingress - queue - workers sebagai middleware tipis dengan penyimpanan persisten di bawahnya. Hasilnya: scaling baca/tulis yang baik tanpa ketergantungan kuat antar komponen. (van Steen & Tanenbaum, 2023). Rujukan bab: Bab 2 (Architectures)

## **T3**

Semantik pengiriman umum: at-most-once (tidak pernah ganda, bisa hilang), at-least-once (tidak hilang dengan retries, bisa ganda), dan exactly-once (ideal, sulit/mahal). Dalam praktik Pub-Sub melalui HTTP/MOM, retries untuk ketahanan membuat duplicates sangat mungkin, sehingga konsumen idempoten menjadi krusial. Strategi yang dianjurkan adalah menjadikan side-effects idempoten yakni menggunakan key (topic, event\_id) dan operasi INSERT OR IGNORE (atau upsert) agar duplikasi tidak memicu pemrosesan ulang. Dengan begitu, sistem mewujudkan efek exactly-once processing di atas jaringan at-least-once delivery. Bab 4 menempatkan ini dalam kerangka model komunikasi (RPC, MOM, persistent messaging, multicast) serta konsekuensi pilihan semantik untuk keandalan dan performa. Pada aggregator, kita menerima reorder/duplicates di jaringan, lalu close over di storage idempoten—membiarkan pengangkutan fokus pada pengiriman, dan aplikasi fokus pada dedup. (van Steen & Tanenbaum, 2023). Rujukan bab: Bab 4 (Communication: RPC, MOM, persistent messaging, semantik).

## **T4**

Penamaan yang baik memisahkan nama (identitas logis) dari alamat (titik akses) dan menuntut properti uniqueness serta location-independence. Untuk aggregator, skema praktis adalah: topic sebagai namespace hierarkis (mis. app/env/component) dan event\_id sebagai pengenal tahan tabrakan (UUID v4/v7 atau hash kriptografis dari source, ts, dan payload). Kombinasi (topic, event\_id) menjadi identifier murni (tidak melekat pada alamat konsumen) yang stabil lintas retries dan re-routing, sehingga dedup cukup mengecek kunci tersebut. Dampaknya: (1) collision sangat kecil sehingga false duplicate minimal; (2) idempoten

sederhana di level storage; (3) mobilitas produsen/konsumen tidak mempengaruhi resolusi identitas event. Bab penamaan menekankan beda names/identifiers/addresses dan teknik resolution—persis selaras dengan kebutuhan aggregator untuk binding logis tanpa ketergantungan lokasi. (van Steen & Tanenbaum, 2023). Rujukan bab: Bab 5 (Naming: names, identifiers, addresses).

## **T5**

Total ordering sering tidak perlu untuk agregasi log: tujuan utama adalah akurasi konten (unik, tidak ganda) dan timeliness, bukan urutan global lintas topik. Biaya total order (mis. konsensus) akan mengurangi throughput dan menambah latensi. Alternatif praktis: (1) per-topic FIFO: jalankan satu worker per topik agar urutan konsumsi sesuai enqueue; (2) ordering berbasis event time: ORDER BY timestamp dan tiebreaker event\_id; (3) clock-aided ordering: gunakan monotonic counter lokal atau Lamport clock/vector clock untuk mendeteksi causality bila relevan. Batasannya: event time dapat skewed; per-topic FIFO tidak berlaku lintas topik; logical clocks memberi happens-before parsial, bukan real-time order. Bab 6 (koordinasi/sinkronisasi) membahas absennya jam global, sinkronisasi, dan cara memperoleh ordering yang memadai konteks, kita dapat memanfaatkan prinsip itu untuk memilih ordering minimal yang cukup, menjaga sistem tetap cepat dan sederhana. (van Steen & Tanenbaum, 2023). Rujukan bab: Bab 6 (Coordination/Synchronization: clocks & ordering).

## **T6**

Tiga failure modes utama: (1) duplikasi akibat retries/multicast; (2) out-of-order karena beberapa jalur/pekerja; (3) crash pada publisher/aggregator. Mitigasinya: exponential backoff dan jitter (mengurangi thundering herd), idempotency di consumer (kunci (topic, event\_id)), dan dedup store yang durable (SQLite/WAL) agar reprocess tidak terjadi setelah restart. Di jalur komunikasi, pilih semantik at-least-once dan persistent messaging bila perlu; di jalur aplikasi, minimalisir side-effect non-idempoten. Untuk crash sebelum commit, kita menerima kehilangan item yang masih di in-memory queue, penanganan produksinya adalah antrian durable atau outbox transaksional. Bab 1 mengingatkan “fallacies” soal jaringan; Bab 4 membahas semantik komunikasi dan persistent messaging; Bab 7 mengaitkan dampak kegagalan terhadap konsistensi/repikasi. Kombinasi retry-aware, idempoten, dan dedup durable memberikan safety net praktis dengan kompleksitas moderat. (van Steen & Tanenbaum, 2023). Rujukan bab: Bab 1, Bab 4, Bab 7.

## **T7**

Pada aggregator, eventual consistency berarti setelah beberapa waktu tanpa pembaruan/kegagalan baru, semua pembaca akan melihat data event unik yang sama di penyimpanan persisten, meski sementara terdapat lag atau reorder. Idempotency memastikan efek pemrosesan konvergen dan duplikasi tidak mengubah state, hanya meningkatkan metrik

duplicate-dropped. Dedup menjamin data final tidak berisi ganda sehingga replica/readers menyatu pada fixpoint yang identik. Jika ada replikasi baca atau caching, invalidasi/penyegaran periodik ditambah pembacaan berbasis processed\_at membawa klien ke keadaan konsisten tanpa menuntut strong consistency. Bab 7 di buku menguraikan spektrum konsistensi dan menempatkan eventual consistency sebagai kompromi yang sangat berguna ketika availability dan kinerja prioritas, selaras dengan pola Pub-Sub yang berurusan dengan retries dan out-of-order. Dengan demikian, idempoten dan dedup adalah “mesin konvergensi” yang mendorong sistem menuju konsistensi akhirnya dengan biaya koordinasi minimal. (van Steen & Tanenbaum, 2023). Rujukan bab: Bab 7 (Consistency & Replication).

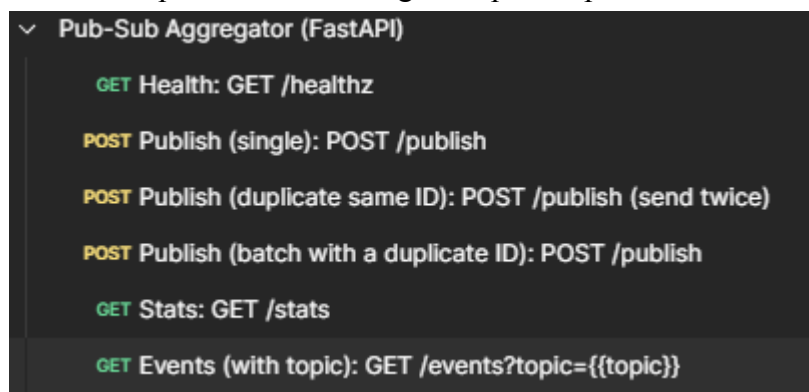
## T8

Metrik evaluasi kunci: throughput (event/detik diproses unik), latency end-to-end (publish ke persist), duplicate rate (dupe/total), backlog/queue depth, dan freshness (lag pembacaan /events). Design levers yang mempengaruhi metrik ini terpetakan ke: Bab 1 (tujuan & fallacies) mendikte target skalabilitas; Bab 2 (arsitektur) menentukan fan-out/fan-in dan elastisitas; Bab 3 (proses/pekerja) mengatur paralelisme; Bab 4 (komunikasi) mempengaruhi semantik retries/batching; Bab 5 (penamaan) mempengaruhi biaya dedup (kualitas identifier); Bab 6 (koordinasi) mempengaruhi ordering vs latensi; Bab 7 (konsistensi) mendikte kapan kita puas dengan eventual. Secara praktis: tambahkan workers hingga DB (penulis tunggal) menjadi bottleneck; gunakan batching/transaction untuk throughput; ukur p99 latency agar tidak tertipu rata-rata; dan pantau duplicate rate—bila tinggi, optimalkan generator event\_id (UUID v7/sha-hash) atau pre-filter (Bloom). Strategi ini menjaga keseimbangan availability, performa, dan keakuratan hasil, tepat seperti prinsip Bab 1–7. (van Steen & Tanenbaum, 2023). Rujukan bab: Bab 1–7.

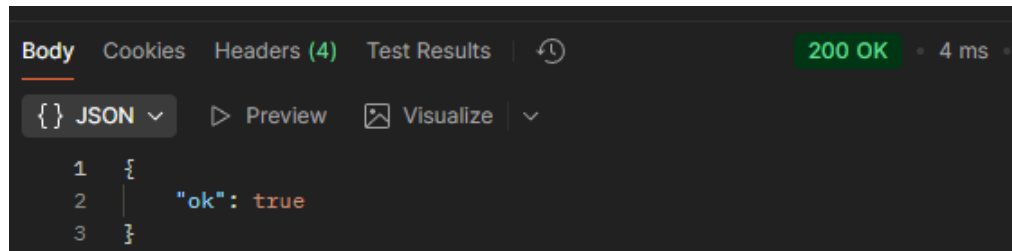
## Bagian Implementasi

### a. Model event & API

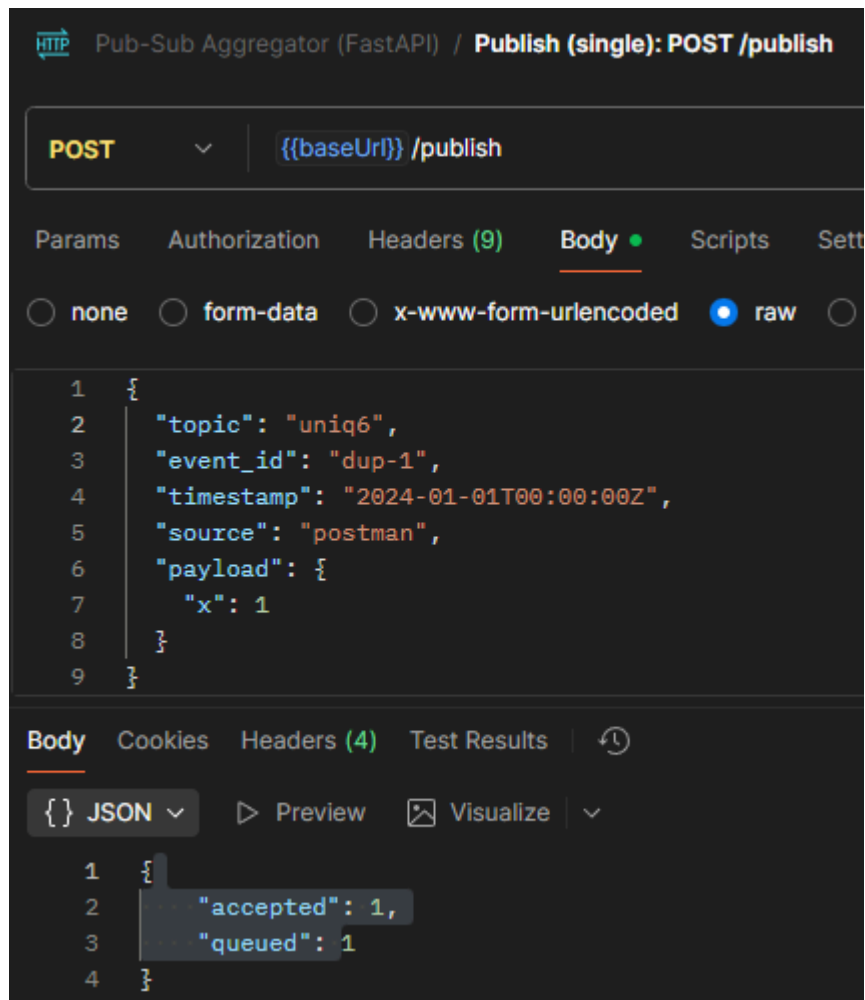
Sistem diimplementasikan dengan empat endpoint berikut:



- /healthz: berfungsi untuk mengecek koneksi ke server apakah berhasil atau tidak. Jika koneksi berhasil, maka akan merespon seperti gambar dibawah



- /publish: berfungsi untuk mengirimkan data untuk diantrekan/enqueue in memory.

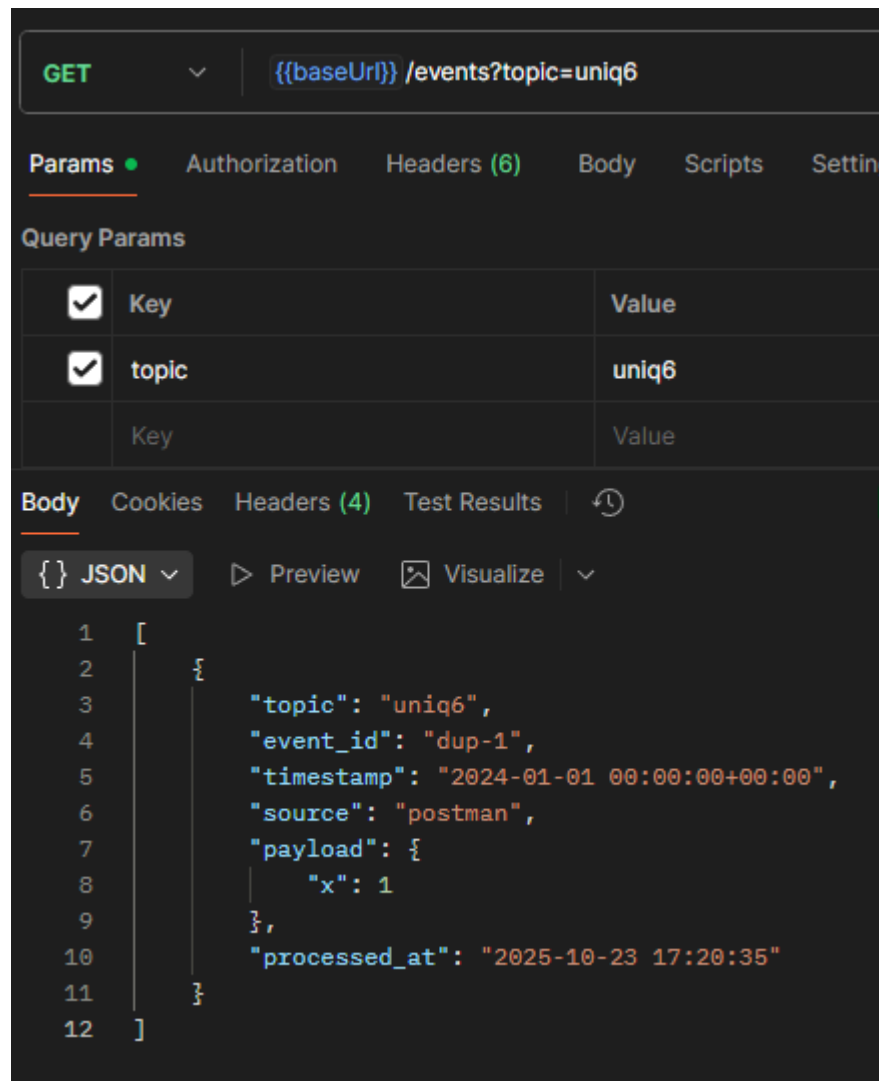


- /stats: berfungsi untuk menampilkan statistik data yang diterima, disimpan, dan dibuang karena duplikat.

The screenshot shows a web browser interface for a 'Pub-Sub Aggregator (FastAPI)' application. The address bar displays the URL 'Stats: GET /stats'. The 'GET' method is selected, and the URL is shown as '{{baseUrl}}/stats'. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Scripts', and 'Settings'. The 'Body' tab is active, showing a JSON response. The JSON data includes statistics such as 'received', 'unique\_processed', 'duplicate\_dropped', 'topics' (a list of topic names), 'uptime\_seconds', 'queue\_depth', 'workers', and 'db\_path'.

```
1  {
2    "received": 5761,
3    "unique_processed": 14403,
4    "duplicate_dropped": 960,
5    "topics": [
6      "kano1",
7      "kano2",
8      "kano3",
9      "kano4",
10     "uniq4",
11     "uniq5",
12     "uniq6"
13   ],
14   "uptime_seconds": 234.348,
15   "queue_depth": 0,
16   "workers": 2,
17   "db_path": "/data/store.db"
18 }
```

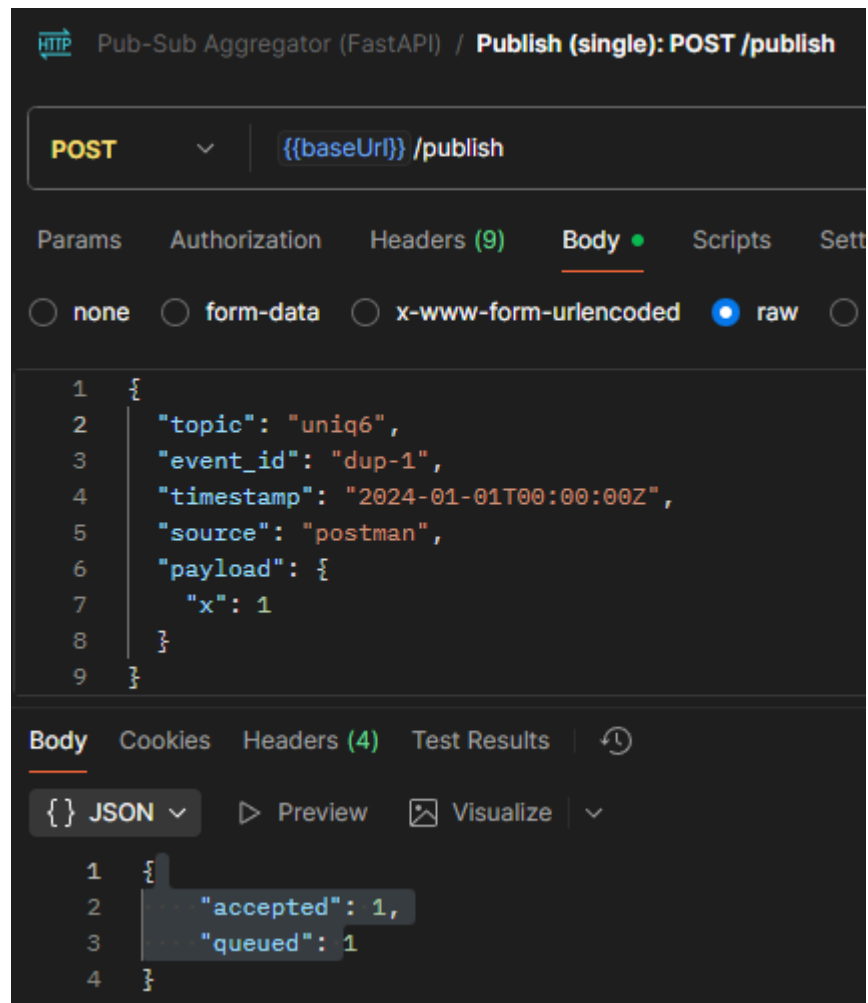
- /events: berfungsi untuk menampilkan seluruh event yang disimpan jika tanpa parameter topic, dan detail sebuah topic jika menggunakan parameter nama topic.



## b. Idempotency & deduplication

Dedup store menggunakan basis data SQLite yang mana data tersimpan secara permanen sehingga tidak terpengaruh restart server. Idempotency diimplementasikan dengan pengecekan PK `topic` dan `event_id`. Jika ada yang keduanya sama persis dengan yang sudah disimpan, maka akan dibuang.





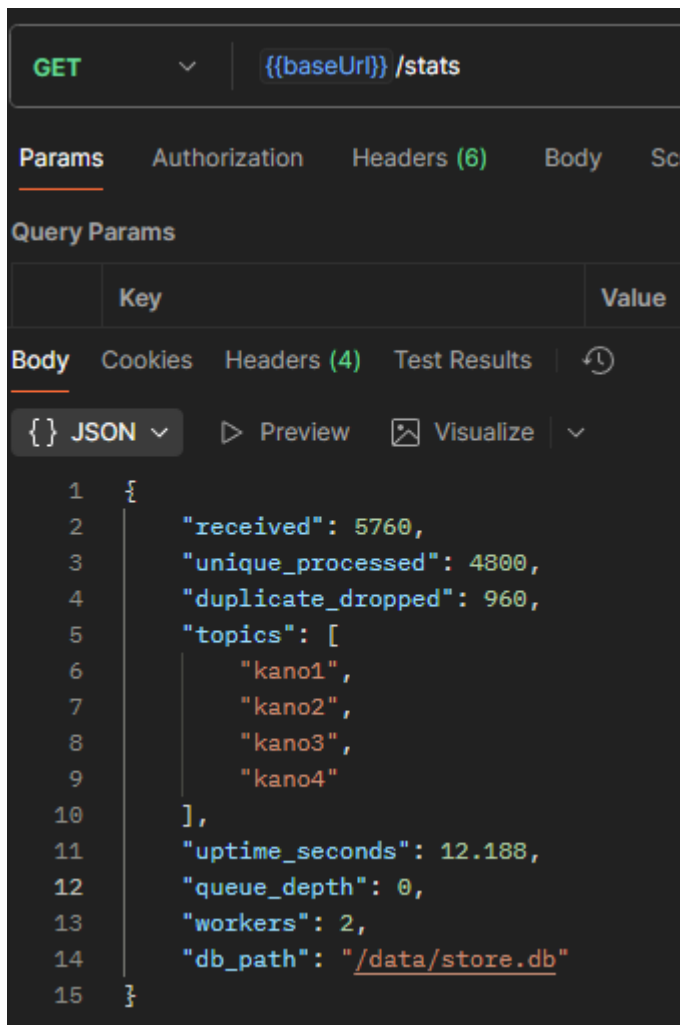
Disini kita coba memasukkan data lagi yang sudah disimpan pada poin a tadi tanpa perubahan.

The screenshot shows a REST client interface with a GET request to `{{baseUrl}}/stats`. The response is a JSON object with the following structure:

```
1  {
2    "received": 5762,
3    "unique_processed": 14403,
4    "duplicate_dropped": 961,
5    "topics": [
6      "kano1",
7      "kano2",
8      "kano3",
9      "kano4",
10     "uniq4",
11     "uniq5",
12     "uniq6"
13   ],
14   "uptime_seconds": 603.958,
15   "queue_depth": 0,
16   "workers": 2,
17   "db_path": "/data/store.db"
18 }
```

Hasilnya received bertambah satu namun unique\_processed dan topics tidak bertambah dan duplicate\_dropped bertambah yang berarti data yang dipublish tadi merupakan duplikat.

### c. Reliability dan ordering

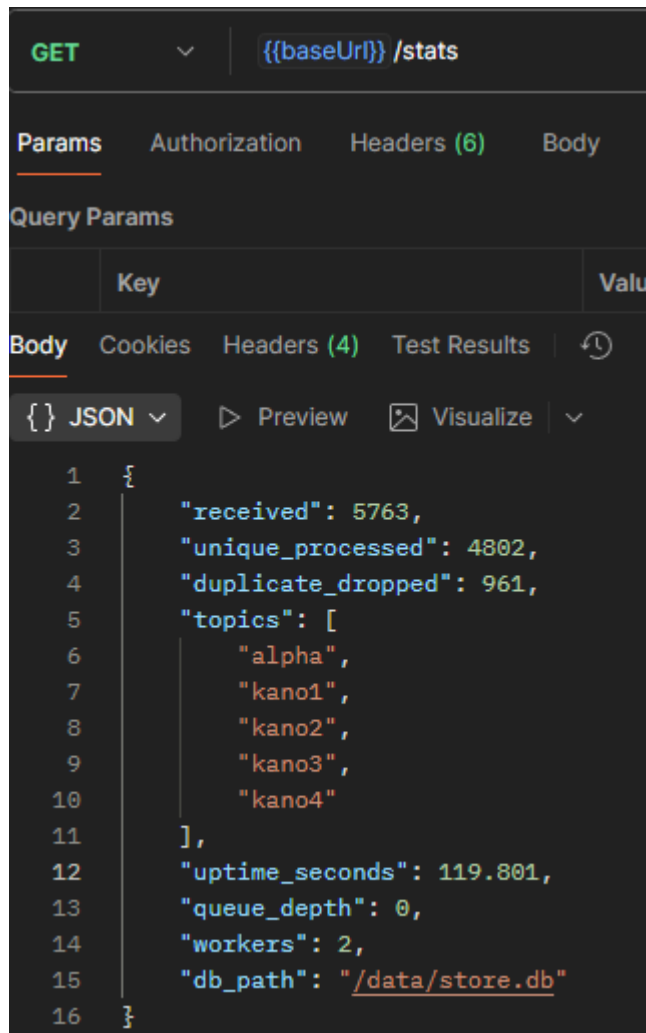


Stats awal sebelum at-least-once-delivery.

```
[
  {
    "topic": "alpha",
    "event_id": "b-1",
    "timestamp": "2024-01-01T00:00:00Z",
    "source": "postman",
    "payload": {
      "n": 1
    }
  },
  {
    "topic": "alpha",
    "event_id": "b-2",
    "timestamp": "2024-01-01T00:00:00Z",
    "source": "postman",
    "payload": {
      "n": 2
    }
  },
  {
    "topic": "alpha",
    "event_id": "b-2",
```

```
[{"timestamp": "2024-01-01T00:00:00Z",
  "source": "postman",
  "payload": {
    "n": 2,
    "dup": true
  }
}]
```

Payload simulasi



Hasilnya hanya dua data dari tiga data dalam payload yang disimpan.

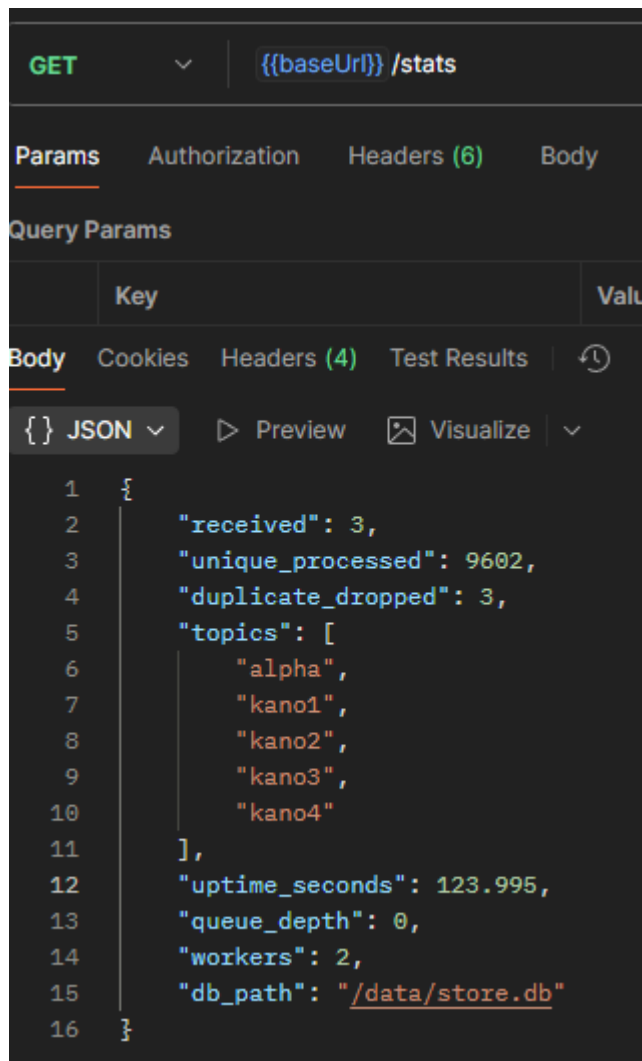
```
(.venv) PS F:\KULIAH\ITK\E-learning\semester 7\sister\aggregator> docker compose restart
[+] Restarting 2/2
✓ Container aggregator-aggregator-1 Started
✓ Container aggregator-publisher-1 Started
```

Kemudian saya coba untuk restart container

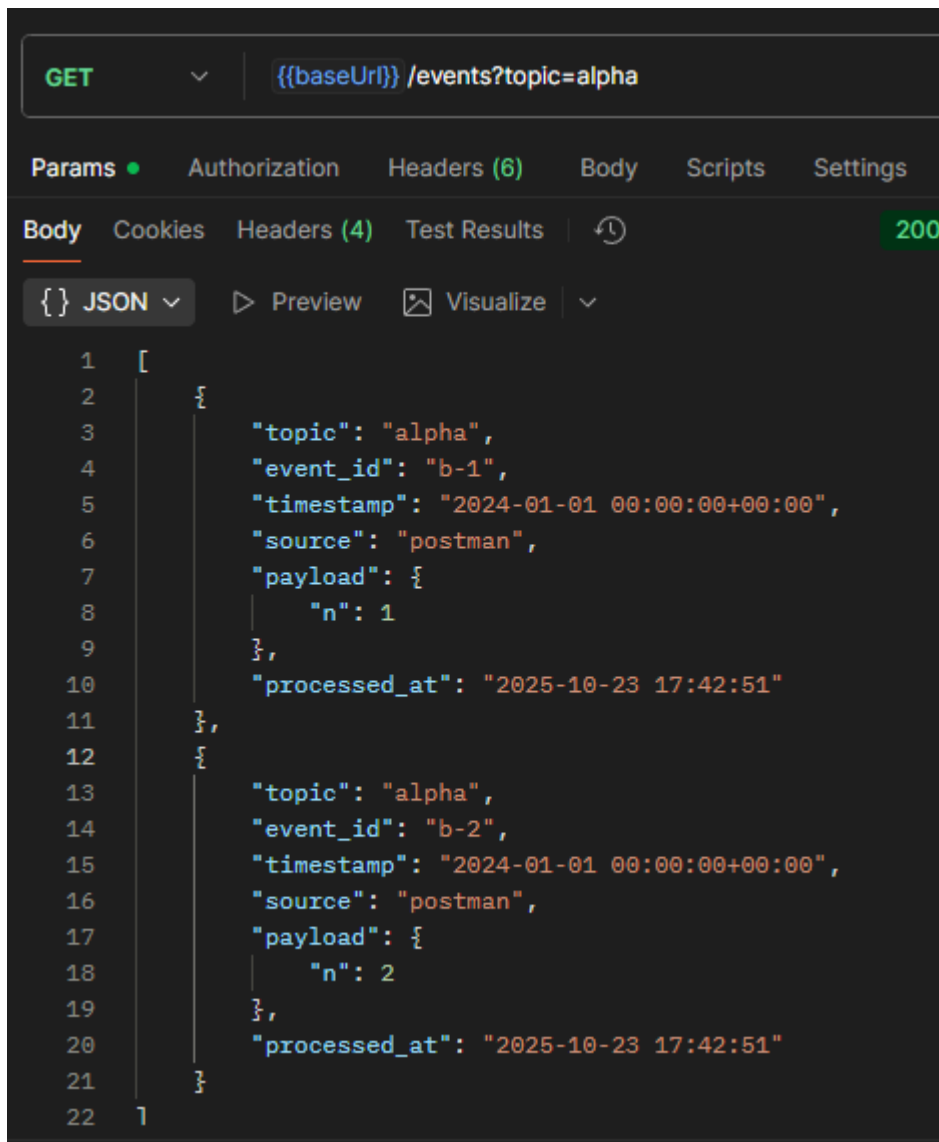
The screenshot shows a REST client interface with a GET request to `{{baseUrl}}/stats`. The response is a JSON object with the following structure:

```
1  {
2    "received": 0,
3    "unique_processed": 9602,
4    "duplicate_dropped": 0,
5    "topics": [
6      "alpha",
7      "kano1",
8      "kano2",
9      "kano3",
10     "kano4"
11   ],
12   "uptime_seconds": 3.577,
13   "queue_depth": 0,
14   "workers": 2,
15   "db_path": "/data/store.db"
16 }
```

Saat container restart, stats received dan duplicate\_dropped akan reset namun data topics yang tersimpan, masih ada.



Payload publish diatas dijalankan kembali, ketiga data topic ditolak karena topic yang dikirim sudah tersimpan sebelum container restart.



The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** `{{baseUrl}}/events?topic=alpha`
- Headers:** 6 headers are listed.
- Body:** The response is displayed in JSON format.
- Status:** 200
- JSON Response:**

```
1  [  
2    {  
3      "topic": "alpha",  
4      "event_id": "b-1",  
5      "timestamp": "2024-01-01 00:00:00+00:00",  
6      "source": "postman",  
7      "payload": {  
8        "n": 1  
9      },  
10     "processed_at": "2025-10-23 17:42:51"  
11   },  
12   {  
13     "topic": "alpha",  
14     "event_id": "b-2",  
15     "timestamp": "2024-01-01 00:00:00+00:00",  
16     "source": "postman",  
17     "payload": {  
18       "n": 2  
19     },  
20     "processed_at": "2025-10-23 17:42:51"  
21   }  
22 ]
```

Pada sistem ini, tidak dibutuhkan dikarenakan perannya sudah diambil alih oleh properti `processed_at` yang tercantum pada setiap data publish yang masuk dan properti ini dapat digunakan untuk mengurutkan data dengan konsisten.

#### d. Performa minimum

```
you, 2 days ago | 1 author (you)
1 import time
2
3 def test_stress_batch_5000_with_duplicates(client):
4     uniq = 4000
5     dup = 1000
6     base = [{
7         "topic": "load",
8         "event_id": f"e-{i}",
9         "timestamp": "2024-01-01T00:00:00Z",
10        "source": "test",
11        "payload": {"i": i}
12    } for i in range(uniq)]
13    batch = base + base[:dup]
14    client.post("/publish", json=batch)
15
16    t0 = time.time()
17    deadline = 10.0
18    while time.time() - t0 < deadline:
19        s = client.get("/stats").json()
20        if s["unique_processed"] >= uniq:
21            break
22        time.sleep(0.02)
23
24    s = client.get("/stats").json()
25    assert s["unique_processed"] >= uniq
26    assert s["duplicate_dropped"] >= dup
27
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL GITLENS

```
(.venv) PS F:\KULIAH\ITK\E-learning\semester 7\sister\aggregator> python -m pytest tests/test_stress.py
>>
===== test session starts =====
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.6.0
rootdir: F:\KULIAH\ITK\E-learning\semester 7\sister\aggregator
configfile: pytest.ini
plugins: anyio-4.6.2.post1, asyncio-0.23.8
asyncio: mode=Mode.STRICT
collected 1 item

tests\test_stress.py .

===== 1 passed in 0.38s =====
(.venv) PS F:\KULIAH\ITK\E-learning\semester 7\sister\aggregator> 
```

Stress testing dilakukan menggunakan 5000 data dengan 1000 data duplikat dan berhasil diselesaikan dalam waktu 0.38 detik.

#### e. docker



```
Dockerfile > ...
You, 2 days ago | 1 author (You)
1 FROM python:3.11-slim
2
3 ENV PYTHONDONTWRITEBYTECODE=1 \
4     PYTHONUNBUFFERED=1
5
6 WORKDIR /app
7
8 COPY requirements.txt .
9 RUN pip install --no-cache-dir -r requirements.txt
10
11 RUN adduser --disabled-password --gecos '' appuser && chown -R appuser:app
12
13 COPY src/ ./src/
14 COPY tests/ ./tests/
15
16 EXPOSE 8080
17
18 CMD ["python", "-m", "src.main"]
19

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL GITLENS

PS F:\KULIAH\ITK\E-learning\semester 7\sister\aggregator> & "F:\KULIAH\ITK\E-learning\se
(.venv) PS F:\KULIAH\ITK\E-learning\semester 7\sister\aggregator> docker compose restart
[+] Restarting 2/2
✓ Container aggregator-publisher-1 Started
✓ Container aggregator-aggregator-1 Started
```

Sistem memiliki dua container terpisah untuk aggregator dan publisher.

## f. Docker Compose

```
services:
  aggregator:
    build: .
    user: "0:0"
    command: [ "sh", "-lc", "mkdir -p /data && python -m src.main" ]
    environment:
      - DB_PATH=/data/store.db
      - CONSUMER_WORKERS=2
    volumes:
      - dedup_data:/data
    ports:
      - "8080:8080"
    restart: unless-stopped

    healthcheck:
      test: ["CMD", "python", "-c", "import urllib.request,json,sys;
sys.exit(0) if
json.loads(urllib.request.urlopen('http://127.0.0.1:8080/healthz').re
ad().decode()).get('ok') else sys.exit(1)"]
      interval: 5s
      timeout: 2s
      retries: 15
      start_period: 5s
```

```

publisher:
  build: .
  command: ["python", "-m", "src.publisher"]
  environment:
    - AGGREGATOR_URL=http://aggregator:8080
    - PUBLISH_COUNT=6000
    - DUP_RATIO=0.20
    - BATCH_SIZE=500
    - TOPICS=kano1,kano2,kano3,kano4

  depends_on:
    aggregator:
      condition: service_healthy

  restart: on-failure

volumes:
  dedup_data:

```

Sistem dapat di build menggunakan docker compose dimana terdapat perintah

- Environment: mengatur target direktori untuk database dan jumlah worker.
- Volume: untuk membuat volume yang akan dijadikan penyimpanan DB.
- Port: diatur ke 8080:8080 untuk komunikasi dengan host.
- Healthcheck: untuk mengetes koneksi ke container.
- Publisher: berfungsi untuk stress test dengan 6000 data dengan 20% data duplikat untuk pengecekan performa.

#### g. Unit test

Pada program terdapat lima unit test yang menguji fungsionalitas dan performa sistem yakni:

- test\_dedup.py: berfungsi untuk menguji kemampuan memproses only-once sistem.
- test\_persistence.py: berfungsi untuk menguji kemampuan persistence data sistem dengan cara mengirim data duplikat
- test\_schema.py: berfungsi untuk menguji validasi json /publish.
- test\_stats\_events.py: berfungsi untuk menguji fungsionalitas endpoint /stats dan /publish.
- test\_stress.py: berfungsi untuk menguji responsivitas sistem dengan 5000 data dengan 1000 data duplikat.

Kelima unit test berhasil dijalankan dengan hasil sebagai berikut:

```

2025-10-24 01:01:58,958 INFO [DUP] drop load/e-997
2025-10-24 01:01:58,958 INFO [DUP] drop load/e-998
2025-10-24 01:01:58,958 INFO [DUP] drop load/e-999
2025-10-24 01:01:58,959 INFO HTTP Request: GET http://testserver/stats "HTTP/1.1 200 OK"
2025-10-24 01:01:58,960 INFO HTTP Request: GET http://testserver/stats "HTTP/1.1 200 OK"

```

```
INFO aggregator:main.py:39 [DUP] drop load/e-995
INFO aggregator:main.py:39 [DUP] drop load/e-996
INFO aggregator:main.py:39 [DUP] drop load/e-997
INFO aggregator:main.py:39 [DUP] drop load/e-998
INFO aggregator:main.py:39 [DUP] drop load/e-999
INFO httpx:_client.py:1038 HTTP Request: GET http://testserver/stats "HTTP/1.1 200 OK"
INFO httpx:_client.py:1038 HTTP Request: GET http://testserver/stats "HTTP/1.1 200 OK"
===== short test summary info =====
PASSED tests/test_dedup.py::test_dedup_only_once_processed
PASSED tests/test_persistence.py::test_event_and_dedup_persist_across_reopen
PASSED tests/test_schema.py::test_invalid_schema_rejected
PASSED tests/test_stats_events.py::test_events_and_stats_consistent
PASSED tests/test_stress.py::test_stress_batch_5000_with_duplicates
===== 5 passed in 0.49s =====
```

**Link video demo**

<https://youtu.be/l6pNOFgUCPY>

## Daftar Pustaka

van Steen, M., & Tanenbaum, A. S. (2023). *Distributed systems* (Edisi ke-4, Versi 01).

distributed-systems.net. [https://www.distributed-systems.net/index.php/books/ds4/  
\(distributed-systems.net\)](https://www.distributed-systems.net/index.php/books/ds4/distributed-systems.net)