UTS

Sistem Paralel dan Terdistribusi

Pub-Sub Log Aggregator

Dosen Pengampu :

Riska Kurniyanto Abdullah, S.T., M.Kom.
198803082020121011

Disusun Oleh :

Ardi Dwi Saputra 11221026

24 Oktober 2025

A. Teori

1. Karakteristik Utama Sistem Terdistribusi dan Trade-off pada Desain Pub-Sub Log Aggregator

Sistem terdistribusi didefinisikan sebagai kumpulan komputer otonom yang terhubung melalui jaringan dan dilengkapi perangkat lunak yang memungkinkan mereka berkoordinasi untuk menyajikan tampilan sistem tunggal yang terintegrasi (Tanenbaum & Van Steen, 2023, Bab 1). Karakteristik fundamental yang membedakannya dari sistem terpusat meliputi: (1) concurrency of components, di mana beberapa proses berjalan secara simultan; (2) lack of a global clock, yang mempersulit sinkronisasi dan pengurutan event secara global; serta (3) independent failures, di mana kegagalan satu komponen tidak serta-merta menghentikan komponen lain (Coulouris dkk., 2012, Bab 1).

Dalam konteks desain Pub-Sub Log Aggregator, karakteristik ini memunculkan trade-off yang krusial. Tujuan utamanya adalah mencapai scalability (kemampuan menangani peningkatan beban log) dan fault tolerance. Namun, hal ini sering kali berbenturan dengan consistency. Sesuai CAP theorem, sistem harus memilih antara konsistensi kuat dan ketersediaan tinggi (high availability) saat terjadi partisi jaringan (Tanenbaum & Van Steen, 2023, Bab 7). Log aggregator umumnya memprioritaskan availability dan partition tolerance, sehingga mengadopsi model eventual consistency. Trade-off lainnya adalah antara throughput dan latency. Untuk menjamin reliability (misalnya, dengan at-least-once delivery dan durable storage), sistem mungkin mengorbankan latency karena adanya overhead I/O disk dan pemrosesan retry.

2. Perbandingan Arsitektur Client-Server dan Publish-Subscribe

Arsitektur client-server merupakan model komunikasi fundamental di mana proses dibagi menjadi dua peran: server (penyedia layanan) dan client (peminta layanan). Komunikasi ini bersifat request-response, sering kali sinkron, dan menciptakan keterikatan (coupling) yang erat antara client dan server (Tanenbaum & Van Steen, 2023, Bab 2). Client harus mengetahui lokasi atau identitas server, dan server dapat menjadi bottleneck jika menerima terlalu banyak permintaan simultan.

Sebaliknya, arsitektur publish-subscribe (Pub-Sub) adalah model perpesanan asinkron yang memisahkan publisher (pengirim pesan) dan subscriber (penerima pesan) melalui entitas perantara, seperti message broker (Coulouris dkk., 2012, Bab 6). Model ini memberikan decoupling dalam tiga dimensi: spatial (komponen tidak perlu saling mengetahui), temporal (komponen tidak harus aktif bersamaan), dan synchronization (publisher tidak diblokir saat mengirim pesan). Untuk log aggregator, Pub-Sub lebih superior karena sifatnya

yang event-driven. Arsitektur ini mendukung scalable event dissemination ke banyak consumer secara paralel dan meningkatkan resilience. Broker dapat berfungsi sebagai buffer untuk menangani burst load log tanpa membebani publisher. Meskipun Pub-Sub menambah kompleksitas dalam hal ordering dan delivery semantics, fleksibilitas dan skalabilitasnya sangat esensial untuk sistem agregasi log skala besar.

3. **At-least-once vs Exactly-once Delivery Semantics dan Pentingnya Idempotent Consumer**

   Delivery semantics mendefinisikan jaminan yang diberikan oleh sistem perpesanan. At-least-once delivery menjamin bahwa setiap pesan akan terkirim ke consumer setidaknya satu kali. Meskipun andal dalam mencegah kehilangan data, mekanisme retry yang agresif untuk mengatasi kegagalan jaringan atau consumer crash dapat mengakibatkan pengiriman pesan duplikat (Coulouris dkk., 2012, Bab 6). Di sisi lain, exactly-once delivery adalah jaminan ideal di mana setiap pesan dijamin terkirim dan diproses tepat satu kali. Namun, dalam praktiknya, exactly-once sangat sulit dan mahal untuk diimplementasikan dalam sistem terdistribusi asinkron, karena seringkali memerlukan transaksi terdistribusi (seperti two-phase commit) yang kompleks dan berdampak signifikan pada performance (Tanenbaum & Van Steen, 2023, Bab 8).

   Oleh karena itu, pendekatan yang lebih pragmatis adalah mengimplementasikan at-least-once delivery pada lapisan infrastruktur, dan membebankan tanggung jawab penanganan duplikat ke aplikasi melalui idempotent consumer. Operasi idempotent adalah operasi yang dapat dieksekusi berkali-kali tanpa mengubah hasil setelah eksekusi pertama. Dalam konteks presence of retries, consumer yang idempotent (misalnya, dengan melacak event_id yang sudah diproses) menjadi krusial. Ini memungkinkan sistem mencapai efek pemrosesan exactly-once secara efektif, mencegah side effects ganda (seperti data ganda di database), tanpa mengorbankan ketersediaan dan throughput sistem.

4. **Skema Penamaan untuk Topic dan Event_ID serta Dampaknya terhadap Deduplication**

   Dalam sistem terdistribusi, skema penamaan (naming scheme) sangat penting untuk identifikasi entitas secara unik dan tidak ambigu (Tanenbaum & Van Steen, 2023, Bab 4). Pada log aggregator, dua entitas utama adalah topic dan event_id. Topic berfungsi sebagai saluran logis untuk mengkategorikan event. Skema penamaan topic sebaiknya bersifat hierarkis (misalnya, app.service.component.level) untuk memungkinkan subscriber memfilter event secara fleksibel (misalnya, berlangganan ke app.service.component.*).

Sementara itu, event_id harus unik secara global dan collision-resistant untuk berfungsi sebagai kunci utama dalam deduplication. Skema yang robust, seperti Universally Unique Identifier (UUID) versi 4 (berbasis acak) atau versi 7 (berbasis waktu), sangat ideal karena dapat dihasilkan oleh publisher secara independen tanpa memerlukan koordinasi terpusat, sehingga menghindari contention. Alternatif lain adalah hash dari konten event yang dikombinasikan dengan timestamp dan source_id. Dampak skema event_id yang kuat terhadap deduplication sangat signifikan. Skema ini menjadi dasar bagi idempotent consumer. Dedup store (penyimpanan deduplikasi) dapat menggunakan event_id ini sebagai kunci untuk lookup yang efisien. Saat event baru tiba, consumer memeriksa event_id tersebut di dedup store; jika sudah ada, event tersebut diabaikan, sehingga mencegah pemrosesan ganda (Coulouris dkk., 2012, Bab 6).

5. Ordering: Kapan Total Ordering Tidak Diperlukan dan Pendekatan Praktis

Total ordering (pengurutan total) menjamin bahwa semua proses dalam sistem terdistribusi mengamati semua event dalam urutan global yang identik. Implementasinya memerlukan algoritma sinkronisasi yang kompleks, seperti konsensus atau atomic broadcast, yang dapat menjadi bottleneck performa dan membatasi scalability (Tanenbaum & Van Steen, 2023, Bab 5). Untuk banyak aplikasi, termasuk log aggregator, total ordering tidak diperlukan. Analisis log sering kali hanya membutuhkan causal ordering (urutan sebab-akibat) atau partial ordering (urutan per-sumber), misalnya, memastikan bahwa event dari satu publisher spesifik diproses sesuai urutan pengirimannya.

Pendekatan praktis yang umum adalah menggunakan kombinasi event timestamp berpresisi tinggi (misalnya, ISO8601 dengan milidetik atau nanodetik) yang dihasilkan oleh publisher, ditambah dengan monotonic counter (pencacah monoton) atau nomor urut (sequence number) per publisher (Coulouris dkk., 2012, Bab 5). Kombinasi (publisher_id, timestamp, counter) memungkinkan consumer untuk mengurutkan event dari sumber yang sama secara andal. Batasan utama dari pendekatan ini adalah clock skew (perbedaan waktu) antar node publisher. Perbedaan jam dapat menyebabkan event yang terjadi lebih dulu (menurut waktu nyata) memiliki timestamp yang lebih akhir daripada event yang terjadi belakangan di node lain. Namun, untuk kebutuhan agregasi log, kompromi ini umumnya dapat diterima demi throughput dan availability yang lebih tinggi.

6. Failure Modes dan Strategi Mitigasi

Sistem log aggregator rentan terhadap berbagai failure modes yang umum di sistem terdistribusi (Tanenbaum & Van Steen, 2023, Bab 8). Mode yang paling relevan meliputi: (1) Crash failures, di mana publisher, broker, atau consumer berhenti beroperasi; (2) Omission failures, di mana pesan hilang dalam transmisi jaringan (baik dari publisher ke broker maupun broker ke consumer); dan (3)

Timing failures, di mana pesan tiba dengan kelambatan (latency) yang signifikan atau out-of-order. Kegagalan ini bermanifestasi sebagai kehilangan data, duplikasi data (akibat retry setelah consumer crash), atau out-of-order processing.

Strategi mitigasi berfokus pada fault tolerance (Coulouris dkk., 2012, Bab 8). Untuk mencegah kehilangan data akibat omission atau crash, broker harus menggunakan durable queue (antrean persisten) yang menulis pesan ke disk. Untuk menangani duplikasi, sistem mengandalkan strategi at-least-once delivery yang dikombinasikan dengan idempotent consumer yang didukung oleh durable dedup store. Durable dedup store (misalnya, basis data terindeks) memastikan bahwa status event yang telah diproses tetap bertahan bahkan jika consumer mengalami crash dan restart. Selain itu, mekanisme retry pada consumer harus diimplementasikan dengan exponential backoff untuk menghindari pembebanan berlebih pada sistem (thundering herd) saat terjadi kegagalan sementara.

7. Eventual Consistency dan Peran Idempotency + Deduplication

Eventual consistency (konsistensi pada akhirnya) adalah model konsistensi yang menjamin bahwa jika tidak ada pembaruan baru yang dilakukan, semua replika data dalam sistem terdistribusi pada akhirnya akan converge (konvergen) ke nilai yang sama (Tanenbaum & Van Steen, 2023, Bab 7). Ini adalah konsistensi model yang lebih lemah dibandingkan strong consistency, namun lebih disukai dalam sistem yang memprioritaskan availability dan scalability, seperti log aggregator. Dalam aggregator, event mungkin tiba out-of-order atau duplikat karena sifat asinkron Pub-Sub dan retries.

Idempotency dan deduplication adalah mekanisme kunci untuk mencapai eventual consistency dalam konteks ini. Idempotency adalah properti consumer yang memastikan bahwa pemrosesan ulang event yang sama (karena at-least-once retries) tidak akan menghasilkan side effects tambahan atau mengubah status akhir sistem (Coulouris dkk., 2012, Bab 7). Deduplication, yang diimplementasikan melalui durable dedup store yang melacak event_id, adalah cara praktis untuk menegakkan idempotency. Kombinasi ini menjamin bahwa meskipun event mungkin diterima berkali-kali dan pada waktu yang berbeda oleh consumer yang berbeda (jika ada replikasi), hanya satu salinan unik dari setiap event yang akan diproses dan disimpan secara permanen. Dengan demikian, database atau data lake tujuan akhir akan eventually konsisten, mencerminkan himpunan event yang benar, bebas dari duplikasi.

8. Metrik Evaluasi Sistem dan Keterkaitannya dengan Keputusan Desain

Evaluasi performa sistem log aggregator harus didasarkan pada metrik kuantitatif yang relevan. Metrik utama meliputi:

- Throughput: Didefinisikan sebagai jumlah event unik yang berhasil diproses (termasuk ingestion, deduplication, dan persistensi) per satuan waktu (misalnya, events per second). Metrik ini mengukur scalability dan efisiensi pemrosesan sistem (Tanenbaum & Van Steen, 2023, Bab 1).
- End-to-end Latency: Waktu rata-rata yang dibutuhkan sejak event dipublikasikan oleh publisher hingga event tersebut tersedia untuk dianalisis di penyimpanan akhir. Latency dapat dibagi lagi menjadi ingestion latency dan processing latency.
- Duplicate Rate: Persentase event duplikat yang diterima oleh consumer dibandingkan dengan jumlah total event yang diterima. Metrik ini penting untuk mengukur overhead yang disebabkan oleh at-least-once delivery dan efektivitas dedup store.

Metrik ini terkait erat dengan keputusan desain (Bab 1–7). Pemilihan arsitektur Pub-Sub (Bab 2) dirancang untuk memaksimalkan throughput dengan memungkinkan pemrosesan paralel oleh banyak consumer. Keputusan untuk menggunakan at-least-once delivery (Bab 3) demi reliability (Bab 6, 8) secara inheren akan meningkatkan duplicate rate yang harus ditangani. Implementasi durable dedup store (Bab 6) untuk idempotency (Bab 7) sangat penting untuk integritas data, namun dapat meningkatkan end-to-end latency (karena I/O disk) dibandingkan dengan in-memory store (Coulouris dkk., 2012, Bab 1). Dengan demikian, selalu ada trade-off antara reliability, latency, dan throughput yang harus diseimbangkan.

B. Implementasi

1. Build image dan menjalankan container

a. Build image dengan perintah:
`docker build -t uts-aggregator .`

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$ docker build -t uts-aggregator .
[+] Building 60.5s (12/12) FINISHED                                                                                  docker:default
 => [internal] load build definition from Dockerfile                                                                          0.4s
 => => transferring dockerfile: 409B                                                                                          0.3s
 => [internal] load metadata for docker.io/library/python:3.11-slim                                                           8.5s
 => [auth] library/python:pull token for registry-1.docker.io                                                                 0.0s
 => [internal] load .dockerignore                                                                                             0.1s
 => => transferring context: 2B                                                                                               0.0s
 => [1/6] FROM docker.io/library/python:3.11-slim@sha256:1738c75ae61595d2a9a5301d60a9a2f61abe7017005b3ccb660103d2476c6946     9.8s
 => => resolve docker.io/library/python:3.11-slim@sha256:1738c75ae61595d2a9a5301d60a9a2f61abe7017005b3ccb660103d2476c6946     0.1s
 => => sha256:19fb8589da0207a0e7d3baa0c1b71a67136b1ad06c4b2e65cc771664592e6d9e 249B / 249B                                    0.6s
 => => sha256:e73850a50582f63498f7551a987cc493e848413fcae176379acff9144341f77f 14.36MB / 14.36MB                              7.6s
 => => sha256:a9ffe18d7fdb9bb2f5b878fdc08887ef2d9644c86f5d4e07cc2e80b783fbea04 1.29MB / 1.29MB                                 3.3s
 => => extracting sha256:a9ffe18d7fdb9bb2f5b878fdc08887ef2d9644c86f5d4e07cc2e80b783fbea04                                      0.5s
 => => extracting sha256:e73850a50582f63498f7551a987cc493e848413fcae176379acff9144341f77f                                     1.8s
 => => extracting sha256:19fb8589da0207a0e7d3baa0c1b71a67136b1ad06c4b2e65cc771664592e6d9e                                     0.1s
 => [internal] load build context                                                                                             0.6s
 => => transferring context: 22.20kB                                                                                          0.5s
 => [2/6] WORKDIR /app                                                                                                        2.1s
 => [3/6] RUN adduser --disabled-password --gecos '' appuser && chown -R appuser:appuser /app                                 1.7s
 => [4/6] COPY requirements.txt ./                                                                                            0.1s
 => [5/6] RUN python -m pip install --no-cache-dir -r requirements.txt                                                       26.5s
 => [6/6] COPY src/ ./src/                                                                                                    1.5s
 => exporting to image                                                                                                        8.5s
 => => exporting layers                                                                                                       6.8s
 => => exporting manifest sha256:8e8d8a32bbaab8f8e26c0b90be8a33edb30caf7729f8dc2ed8474d2f83d8031a                             0.0s
 => => exporting config sha256:590bafcda171d2bbee3220b82faf91b6f83173dc01b5f61bf1b710ee7f05f879                               0.0s
 => => exporting attestation manifest sha256:15551e728d53ba143b54643c0e0d17ffefb04bf3a1f89287adcd3b29090d04a2                 0.0s
 => => exporting manifest list sha256:6ac1199442ced61057b76a3049d79b4694a874c5c3b785f8a08cf78816071 0b7                       0.0s
 => => naming to docker.io/library/uts-aggregator:latest                                                                      0.0s
 => => unpacking to docker.io/library/uts-aggregator:latest                                                                   1.4s
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$
```

b. Menjalankan container. Gunakan --name agar mudah di-stop dan di-start ulang. Gunakan -d (detached) agar terminal Anda bebas untuk menjalankan curl. Perintahnya adalah:
`docker run -d -p 8080:8080 --name uts-aggregator uts-aggregator`

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$ docker run -d -p 8080:8080 --name uts-aggregator uts-aggregator
ac0de4f939ee15f708082692a84c7de76c57d5d507143886a4358225eaa2e3b4
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$
```

c. Tampilkan bahwa *container* berjalan dengan perintah:
`docker ps`

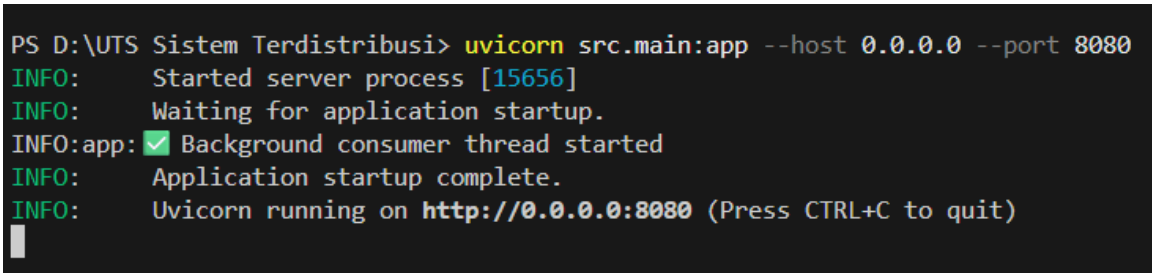
```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS          PORTS                                       NAMES
ac0de4f939ee   uts-aggregator  "uvicorn src.main:ap…"   36 seconds ago   Up 35 seconds   0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp   uts-aggregator
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$
```

2. Idempotency dan Deduplication

Aktifkan server dengan perintah:

`uvicorn src.main:app --host 0.0.0.0 --port 8080`

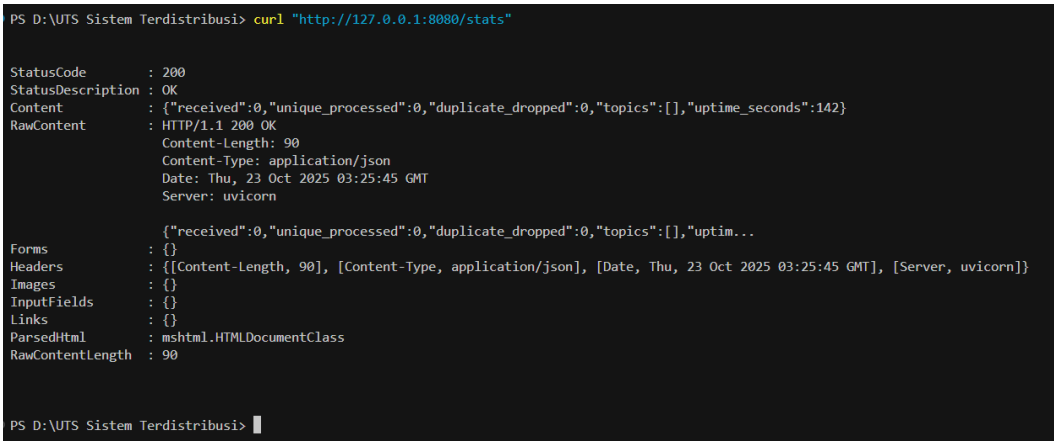
Layanan akan aktif di http://0.0.0.0:8080.



a. Cek Kondisi Awal (GET /stats)

Jalankan curl untuk melihat statistik awal dengan perintah:

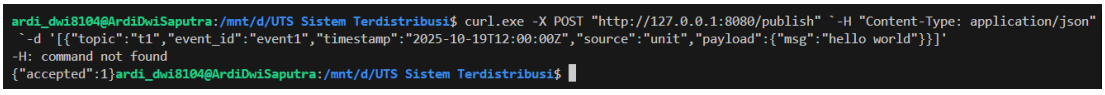
`curl "http://127.0.0.1:8080/stats"`



b. Kirim Event Pertama

Siapkan perintah curl (POST) Anda. Gunakan payload JSON valid yang dibungkus single quote. Perintahnya:

```
curl.exe -X POST "http://127.0.0.1:8080/publish" `-H "Content-Type: application/json" `-d '[{"topic":"t1","event_id":"event1","timestamp":"2025-10-19T12:00:00Z","source":"unit","payload":{"msg":"hello world"}}]'
```



Jalankan curl untuk melihat statistik lagi dengan perintah:

`curl "http://127.0.0.1:8080/stats"`

```
PS D:\UTS Sistem Terdistribusi> curl "http://127.0.0.1:8080/stats"


StatusCode        : 200
StatusDescription : OK
Content           : {"received":1,"unique_processed":1,"duplicate_dropped":0,"topics":["t1"],"uptime_seconds":384}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 94
                    Content-Type: application/json
                    Date: Thu, 23 Oct 2025 10:11:52 GMT
                    Server: uvicorn

                    {"received":1,"unique_processed":1,"duplicate_dropped":0,"topics":["t1"],"u...
Forms             : {}
Headers           : {[Content-Length, 94], [Content-Type, application/json], [Date, Thu, 23 Oct 2025 10:11:52 GMT], [Server, uvicorn]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 94



PS D:\UTS Sistem Terdistribusi>
```

c. Kirim Event Duplikat (Simulasi At-Least-Once)

Siapkan perintah curl (POST) Anda. Gunakan payload JSON valid yang dibungkus single quote. Perintahnya:

```
curl.exe -X POST "http://127.0.0.1:8080/publish" `-H "Content-Type: application/json" `-d '[{"topic":"t1","event_id":"e1","timestamp":"2025-10-19T12:00:00Z","source":"unit","payload":{"msg":"hello world"}}]'
```

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$ curl.exe -X POST "http://127.0.0.1:8080/publish" `-H "Content-Type: application/json"
 `-d '[{"topic":"t1","event_id":"event1","timestamp":"2025-10-19T12:00:00Z","source":"unit","payload":{"msg":"hello world"}}]'
-H: command not found
{"accepted":1}ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$
```

Jalankan curl untuk melihat statistik lagi dengan perintah:

```
curl "http://127.0.0.1:8080/stats"
```

```
PS D:\UTS Sistem Terdistribusi> curl "http://127.0.0.1:8080/stats"


StatusCode        : 200
StatusDescription : OK
Content           : {"received":2,"unique_processed":1,"duplicate_dropped":1,"topics":["t1"],"uptime_seconds":579}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 94
                    Content-Type: application/json
                    Date: Thu, 23 Oct 2025 10:15:08 GMT
                    Server: uvicorn

                    {"received":2,"unique_processed":1,"duplicate_dropped":1,"topics":["t1"],"u...
Forms             : {}
Headers           : {[Content-Length, 94], [Content-Type, application/json], [Date, Thu, 23 Oct 2025 10:15:08 GMT], [Server, uvicorn]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 94



PS D:\UTS Sistem Terdistribusi>
```

3. Restart container dan tunjukkan dedup store persisten mencegah reprocessing

a. Matikan container uts-aggregator dengan perintah:

```
docker container stop uts-aggregator
```

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$ docker container stop uts-aggregator
uts-aggregator
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$
```

b. Nyalakan container uts-aggregator dengan perintah:
`docker container start uts-aggregator`

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$ docker container start uts-aggregator
uts-aggregator
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$
```

c. Jalankan curl untuk cek /stats tanpa mengirim apa-apa dengan perintah:
`curl "http://127.0.0.1:8080/stats"`

```
PS D:\UTS Sistem Terdistribusi> curl "http://127.0.0.1:8080/stats"


StatusCode        : 200
StatusDescription : OK
Content           : {"received":2,"unique_processed":1,"duplicate_dropped":1,"topics":["t1"],"uptime_seconds":579}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 94
                    Content-Type: application/json
                    Date: Thu, 23 Oct 2025 10:15:08 GMT
                    Server: uvicorn

                    {"received":2,"unique_processed":1,"duplicate_dropped":1,"topics":["t1"],"u...
Forms             : {}
Headers           : {[Content-Length, 94], [Content-Type, application/json], [Date, Thu, 23 Oct 2025 10:15:08 GMT], [Server, uvicorn]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 94


PS D:\UTS Sistem Terdistribusi>
```

d. Kirim Event Duplikat setelah restart. Perintahnya:
`curl.exe -X POST "http://127.0.0.1:8080/publish" `-H "Content-Type: application/json" `-d '[{"topic":"t1","event_id":"e1","timestamp":"2025-10-19T12:00:00Z","source":"unit","payload":{"msg":"hello world"}}]'`

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$ curl.exe -X POST "http://127.0.0.1:8080/publish" `-H "Content-Type: application/json"
 `-d '[{"topic":"t1","event_id":"e1","timestamp":"2025-10-19T12:00:00Z","source":"unit","payload":{"msg":"hello world"}}]'
-H: command not found
{"accepted":1}ardi_dwi8104@ArdiDwiSaputra:/mnt/d/UTS Sistem Terdistribusi$
```

d. Jalankan curl untuk cek /stats dengan perintah:
`curl "http://127.0.0.1:8080/stats"`

```
PS D:\UTS Sistem Terdistribusi> curl "http://127.0.0.1:8080/stats"


StatusCode        : 200
StatusDescription : OK
Content           : {"received":3,"unique_processed":0,"duplicate_dropped":0,"topics":["t1"],"uptime_seconds":93}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 93
                    Content-Type: application/json
                    Date: Thu, 23 Oct 2025 05:06:55 GMT
                    Server: uvicorn

                    {"received":3,"unique_processed":0,"duplicate_dropped":0,"topics":["t1"],"u...
Forms             : {}
Headers           : {[Content-Length, 93], [Content-Type, application/json], [Date, Thu, 23 Oct 2025 05:06:55 GMT], [Server, uvicorn]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 93


PS D:\UTS Sistem Terdistribusi>
```

Daftar Pustaka

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

Tanenbaum, A. S., & van Steen, M. (2023). Distributed systems (4th ed., Version 4.01). Maarten van Steen.