Ujian Akhir Semester

Sistem Paralel dan Terdistribusi A

Pub-Sub Log Aggregator

Disusun Oleh :

Muhammad Fachrudy Al-Ghifari 11231054

16 Desember 2025

RINGKASAN SISTEM DAN ARSITEKTUR

Sistem yang dibangun merupakan Pub–Sub Aggregator berbasis arsitektur multi-service yang dijalankan menggunakan Docker Compose. Sistem terdiri dari empat komponen utama, yaitu publisher, aggregator, broker, dan storage. Publisher bertugas mengirim event ke sistem, aggregator menyediakan API dan worker untuk memproses event, Redis digunakan sebagai broker antrian, dan PostgreSQL berperan sebagai penyimpanan persisten.

Aggregator menerima event melalui endpoint HTTP dan tidak langsung menuliskannya ke database, melainkan memasukkannya ke broker untuk diproses secara asynchronous oleh beberapa worker. Pendekatan ini memungkinkan sistem menangani lonjakan event tanpa membebani database secara langsung. PostgreSQL menyimpan event yang telah diproses dengan mekanisme deduplikasi berbasis constraint unik. Seluruh komponen berjalan dalam jaringan lokal Docker tanpa dependensi eksternal.

KEPUTUSAN DESAIN

Idempotency dan Deduplication Store

Sistem menggunakan at-least-once delivery, sehingga event yang sama dapat diproses lebih dari satu kali. Untuk menjaga correctness, consumer dirancang idempotent. Idempotency dicapai dengan menyimpan event ke PostgreSQL menggunakan unique constraint pada (topic, event_id). Jika event duplikat diproses ulang, database akan menolak insert tersebut tanpa menghasilkan efek samping. PostgreSQL berfungsi sebagai deduplication store yang durable, sehingga dedup tetap berlaku walaupun container di-restart.

Transaksi dan Konkurensi

Setiap pemrosesan event dilakukan dalam transaksi database untuk menjamin atomicity dan consistency. Isolation level yang digunakan adalah READ COMMITTED, karena konflik ditangani oleh constraint unik, bukan oleh locking manual. Multi-worker dapat berjalan secara paralel tanpa race condition karena database menangani konflik secara atomik melalui indeks unik.

Ordering Event

Sistem tidak menjamin ordering global. Timestamp disertakan sebagai metadata untuk observasi dan query, tetapi tidak digunakan untuk menentukan urutan pemrosesan. Keputusan ini diambil untuk menghindari overhead koordinasi total order yang mahal. Ordering lemah dianggap cukup untuk use case log aggregation.

Retry dan Failure Handling

Jika worker gagal atau container crash, event dapat diproses ulang dari broker. Pendekatan ini menerima duplikasi sebagai trade-off dan mengandalkan idempotency untuk menjaga konsistensi. Strategi ini sesuai dengan prinsip fault tolerance pada sistem terdistribusi.

ANALISIS PERFORMA DAN METRIK SISTEM

Analisis performa sistem dilakukan untuk mengevaluasi perilaku pemrosesan event, deduplikasi, dan konsistensi hasil pada arsitektur publish-subscribe yang dibangun. Pengujian dilakukan dengan memberikan beban terkontrol berupa pengiriman event dalam jumlah besar, kemudian mengamati metrik sistem hingga mencapai kondisi stabil. Metrik diperoleh langsung dari endpoint observability sistem, yaitu GET /stats, serta status antrian pada broker Redis.

Kondisi Awal Sistem

Sebelum beban diberikan, sistem berada pada kondisi awal tanpa event yang diproses. Hal ini diverifikasi dengan memanggil endpoint /stats, yang menunjukkan bahwa belum ada event diterima maupun diproses.

Kode: curl http://localhost:8080/stats

```
PS D:\Projects\Sister\uas> curl http://localhost:8080/stats
{"received":0,"unique_processed":0,"duplicate_dropped":0,"workers":4,"uptime_sec":148}
```

Pemberian Beban dan Pemrosesan Asynchronous

Beban diberikan menggunakan service publisher dengan mengirim 20.000 event ke endpoint aggregator. Publisher dikonfigurasi untuk menyertakan event duplikat guna menguji mekanisme idempotency dan deduplication.

Kode: docker compose run --rm publisher

```
PS D:\Projects\Sister\uas> docker compose run --rm publisher
 [+] Creating 3/3
  ✓ Container uas-storage-1     Running
  ✓ Container uas-broker-1      Running
  ✓ Container uas-aggregator-1  Running
 [+] Running 2/2
  ✓ Container uas-broker-1   Healthy
  ✓ Container uas-storage-1  Healthy
 [+] Running 1/1
  ! publisher Warning pull access denied for uts-publisher, repository does not exist or may require 'docker login'
 [+] Building 2.4s (14/14) FINISHED
  => [internal] load local bake definitions
  => => reading from stdin 971B
  => [internal] load build definition from Dockerfile
  => => transferring dockerfile: 252B
  => [internal] load metadata for docker.io/library/python:3.11-slim
  => [auth] library/python:pull token for registry-1.docker.io
  => [internal] load .dockerignore
  => => transferring context: 2B
  => [1/6] FROM docker.io/library/python:3.11-slim@sha256:158caf0e080e2cd74ef2879ed3c4e697792ee65251c8208b7afb56683c32ea6c
  => => resolve docker.io/library/python:3.11-slim@sha256:158caf0e080e2cd74ef2879ed3c4e697792ee65251c8208b7afb56683c32ea6c
  => [internal] load build context
  => => transferring context: 69B
  => CACHED [2/6] RUN useradd -m appuser
  => CACHED [3/6] WORKDIR /app
  => CACHED [4/6] COPY requirements.txt .
  => CACHED [5/6] RUN pip install --no-cache-dir -r requirements.txt
  => CACHED [6/6] COPY publisher.py .
  => exporting to image
  => => exporting layers
  => => exporting manifest sha256:066d71c8fb0ae457d761be5a4d213d3605c50b0f5755f3084402065f870d8ed5
  => => exporting config sha256:cb4b0382e23c382ae4d2304379cf866be20f83a4ec42c7f69094715f3aa1da2b
  => => exporting attestation manifest sha256:64c9b925c1862659eb39e20f338007b27ecbf89a3aa50b4aa7cc235099a63a47
  => => exporting manifest list sha256:687fd823cd7f66202df2736d6da0212af0d3e95066df731458ac414d4fbca11b
  => => naming to docker.io/library/uts-publisher:latest
  => => unpacking to docker.io/library/uts-publisher:latest
  => resolving provenance for metadata file
sent=20000 time_sec=0.49 eps=41098.45
```

Observasi Konkurensi dan Antrian Event

Selama pemrosesan berlangsung, sistem menunjukkan perilaku asynchronous dan concurrent. Hal ini dibuktikan dengan memeriksa panjang antrian Redis yang masih berisi event.

Kode: docker compose exec broker redis-cli LLEN events_queue

```
PS D:\Projects\Sister\uas> docker compose exec broker redis-cli LLEN events_queue
(integer) 16426
```

Output menunjukkan nilai lebih dari nol, menandakan masih terdapat event yang menunggu diproses. Pada saat yang sama, log aggregator memperlihatkan beberapa worker aktif memproses event secara paralel.

Kode: docker compose logs -f aggregator

```
aggregator-1  | INFO:worker:worker=w1 topic=stats event_id=12c266ac-57bb-45ea-aa80-8ddd4cced5c5 status=processed
aggregator-1  | INFO:worker:worker=w4 topic=billing event_id=0ac09ba3-9db8-4e99-ad8c-71ea70228753 status=processed
aggregator-1  | INFO:worker:worker=w3 topic=stats event_id=886d7dab-41e6-4fcc-847a-653804b160eb status=processed
aggregator-1  | INFO:worker:worker=w1 topic=billing event_id=c6dd68d6-693c-4781-b136-d879cfea8de0 status=processed
aggregator-1  | INFO:worker:worker=w4 topic=billing event_id=34454180-5314-408f-b582-22c3db0a09b3 status=processed
aggregator-1  | INFO:worker:worker=w3 topic=stats event_id=6c7f001d-07ac-4d78-a27c-cb53f20f2852 status=processed
aggregator-1  | INFO:worker:worker=w2 topic=stats event_id=6acd0cca-c3c9-4bd6-a1c0-df18cf421f04 status=processed
aggregator-1  | INFO:worker:worker=w1 topic=stats event_id=2d13f4fd-9e67-4b2c-864f-647201bd7ee8 status=processed
aggregator-1  | INFO:worker:worker=w3 topic=upload event_id=12686b9a-71a3-4879-9e5f-b2b5ad58cf0f status=processed
```

Log worker menunjukkan beberapa event sedang diproses secara bersamaan oleh worker yang berbeda, yang menandakan bahwa mekanisme pemrosesan paralel (konkurensi) pada aggregator berjalan dengan baik.

Kondisi Akhir dan Konsistensi Hasil

Setelah seluruh event diproses, antrian Redis menjadi kosong. Hal ini diverifikasi dengan perintah yang sama hingga menghasilkan nilai nol.

Kode: docker compose exec broker redis-cli LLEN events_queue

```
PS D:\Projects\Sister\uas> docker compose exec broker redis-cli LLEN events_queue
(integer) 0
```

Selanjutnya, endpoint /stats dipanggil kembali untuk memperoleh metrik akhir sistem.

Kode: curl http://localhost:8080/stats

```
PS D:\Projects\Sister\uas> curl http://localhost:8080/stats
{"received":20000,"unique_processed":18279,"duplicate_dropped":1721,"workers":4,"uptime_sec":528}
```

Hasil akhir menunjukkan bahwa jumlah event yang diterima (received) sama dengan jumlah event unik yang diproses (unique_processed) ditambah event duplikat yang ditolak (duplicate_dropped). Invariant ini membuktikan bahwa mekanisme idempotency dan deduplication bekerja dengan benar meskipun sistem memproses event secara paralel.

KETERKAITAN DENGAN BAB 1-13

T1 - Bab 1

Sistem terdistribusi adalah kumpulan komponen independen yang berjalan pada node berbeda dan berkomunikasi melalui jaringan untuk mencapai tujuan bersama. Ciri utamanya meliputi konkurensi, tidak adanya clock global, serta kemungkinan partial failure, yaitu sebagian komponen dapat gagal sementara komponen lain tetap berjalan. Pada rancangan Pub-Sub aggregator, karakteristik ini tampak dari pemisahan publisher, Redis broker, FastAPI aggregator, dan PostgreSQL storage dalam container terpisah. Trade-off desainnya adalah kompleksitas koordinasi yang lebih tinggi dibanding sistem terpusat, terutama terkait observasi, penanganan kegagalan, dan konsistensi data. Namun, keuntungan utamanya adalah skalabilitas dan elastisitas publisher tidak bergantung pada kecepatan consumer karena event dapat ditampung di broker. Konsekuensinya, sistem harus menerima duplikasi pesan dan ketidakpastian urutan global. Karena itu, correctness dijaga melalui idempotent consumer dan deduplication di database. Dalam demo, konsistensi dibuktikan lewat invariant pada /stats setelah queue habis. (Coulouris et al., 2012; van Steen & Tanenbaum, 2023)

T2 - Bab 2

Arsitektur publish-subscribe lebih tepat daripada client-server ketika beban berupa event burst, pemrosesan dapat dilakukan asynchronous, dan sistem perlu loose coupling. Pada client-server, client harus mengetahui alamat server dan sering menunggu respons sinkron, sehingga throughput dibatasi oleh kapasitas server dan latensi round-trip. Pada publish-subscribe, publisher mengirim event ke broker tanpa mengetahui consumer dan tanpa bergantung pada ketersediaannya. Dalam sistem aggregator, ini memungkinkan publisher mengirim 20.000 event tanpa menunggu operasi database, karena pekerjaan berat dipindahkan ke worker. Selain itu, pola ini memudahkan scaling horizontal worker dapat ditambah untuk menaikkan throughput tanpa mengubah publisher. Trade-offnya adalah semantik pengiriman dan ordering yang lebih lemah, sehingga sistem harus mengelola duplikasi dan reordering secara eksplisit. Karena itu, publish-subscribe cocok untuk event-driven processing seperti log aggregation, sedangkan client-server cocok untuk request-response dengan konsistensi kuat. Broker juga memudahkan menambah consumer baru tanpa mengubah publisher. (Coulouris et al., 2012; van Steen & Tanenbaum, 2023)

T3 - Bab 3

At-least-once delivery menjamin pesan akan diproses minimal satu kali, tetapi memungkinkan duplikasi exactly-once berupaya menjamin pemrosesan tepat satu kali, namun sulit dan mahal karena memerlukan koordinasi kuat, logging, serta protokol commit yang kompleks. Dalam praktik, banyak sistem memilih at-least-once karena lebih sederhana dan lebih tahan terhadap kegagalan jaringan atau crash consumer. Rancangan aggregator menggunakan Redis queue dengan asumsi at-least-once jika worker gagal setelah mengambil pesan, pesan dapat diproses ulang. Agar tetap benar, consumer harus idempotent, yaitu memproses event yang sama berulang kali tanpa menggandakan efek. Implementasi idempotency dilakukan di database melalui unique constraint pada (topic, event_id) dan insert yang aman konflik (upsert/ON CONFLICT). Dengan ini, event duplikat tidak menambah baris baru, melainkan terhitung sebagai duplicate_dropped dan tercatat di audit log. Pendekatan ini memindahkan kompleksitas dari messaging layer ke storage layer yang memiliki operasi atomik dan konsisten. Karena itu, duplikasi terukur melalui duplicate_dropped dan audit log. (Coulouris et al., 2012; van Steen & Tanenbaum, 2023)

T4 - Bab 4

Skema penamaan yang baik memudahkan routing, audit, dan deduplication dalam sistem terdistribusi. Pada rancangan ini, topic berfungsi sebagai namespace logis yang mengelompokkan event menurut domain (misalnya auth, payment, atau stats-test). Event_id menjadi identifier unik untuk tiap event dan harus collision-resistant tanpa koordinasi pusat. Karena itu, publisher menghasilkan event_id menggunakan UUID, yang probabilitas tabrakannya sangat kecil dan dapat dibuat secara terdesentralisasi. Kunci dedup disusun sebagai pasangan (topic, event_id), sehingga duplikasi didefinisikan secara kontekstual per topic, bukan global ini mencegah konflik lintas domain. Pendekatan ini sejalan dengan prinsip penamaan yang tidak mengikat lokasi, sehingga event tetap dapat ditelusuri meski komponen berpindah atau direstart. Di database, unique constraint pada pasangan tersebut menjadikan dedup durable (tahan restart) dan race-free saat multi-worker berjalan. Pengujian memakai UUID agar test repeatable walau database persisten sepenuhnya. (Coulouris et al., 2012; van Steen & Tanenbaum, 2023)

T5 - Bab 5

Ordering global sulit dicapai karena tidak ada clock global dan jaringan dapat menunda atau mengubah urutan kedatangan pesan. Karena itu, sistem sering memakai ordering praktis berbasis timestamp dan, bila perlu, counter monotonik per publisher. Dalam rancangan aggregator, timestamp disertakan sebagai metadata untuk observability dan query, bukan sebagai jaminan bahwa event diproses dalam urutan waktu. Worker mengambil pesan dari queue secara paralel sehingga event dengan timestamp lebih kecil dapat selesai setelah event yang lebih besar. Dampaknya, aplikasi tidak boleh mengasumsikan "urutan proses = urutan waktu", kecuali menambahkan mekanisme ordering tambahan (misalnya partitioning per key, single consumer per stream, atau log terurut). Keputusan ini adalah trade-off sistem menghindari biaya koordinasi total order demi throughput yang tinggi dan latensi rendah. Untuk kebutuhan log aggregation, ordering ketat sering tidak wajib, karena fokusnya pada ketercapaian pemrosesan dan hasil akhir yang konsisten. Jika diperlukan, aplikasi klien harus menangani reordering pada level bisnis. (Coulouris et al., 2012; van Steen & Tanenbaum, 2023)

T6 - Bab 6

Untuk Bind Mount, kita bisa gunakan short sytax dan long syntax. Untuk short syntax, kita bisa gunakan nilai SOURCE:TARGET:MODE, dimana SOURCE adalah lokasi di host, dan TARGET adalah lokasi di container. MODE adalah mode bind mount, ro untul readonly, rw untuk read write (default). SOURCE bisa menggunakan relative path dengan diawali . (titik), atau absolute path

T7 - Bab 7

Failure modes umum pada Pub-Sub meliputi duplikasi pesan, kehilangan koneksi sementara, crash worker saat memproses, dan restart container. Rancangan aggregator memitigasi ini dengan beberapa keputusan delivery bersifat at-least-once, sehingga sistem lebih memilih risiko duplikasi daripada kehilangan pesan. Redis bertindak sebagai buffer saat database lambat, event tetap tersimpan di queue dan diproses kemudian. Untuk crash recovery, container dapat direstart tanpa menghapus data karena dedup store berada di PostgreSQL dengan volume. Jika worker memproses ulang event yang sama, operasi insert akan "aman" karena unique constraint menolak duplikasi, sehingga efek samping tidak berlipat. Selain itu, audit logging membantu mendiagnosis kegagalan, misalnya membedakan processed vs duplicate dan alasan konflik. Mitigasi praktis lain adalah retry/backoff di level worker (jika dipakai) agar tidak membanjiri database saat transient failure. Dengan cara ini, sistem memulihkan diri tanpa transaksi terdistribusi atau protokol recovery yang kompleks. Healthcheck Compose membantu memastikan storage dan broker siap sebelum aggregator start. (Coulouris et al., 2012; van Steen & Tanenbaum, 2023)

T8 - Bab 8

Desain transaksi pada sistem ini berfokus pada ACID di storage tunggal (PostgreSQL), bukan transaksi terdistribusi. Setiap pemrosesan pesan dilakukan dalam transaksi database agar operasi audit, update counter, dan insert event unik terjadi secara atomik. Isolation level READ COMMITTED memadai karena konflik utama diselesaikan oleh unique constraint transaksi tidak perlu mengunci tabel secara luas, sehingga throughput tetap tinggi. Strategi menghindari lost-update adalah menghindari read-modify-write yang rapuh, dan menggantinya dengan operasi update inkremental yang dieksekusi langsung oleh database. Dengan transaksi, sistem mendapatkan atomicity (tidak ada state "setengah jadi") dan durability (hasil tetap ada walau container crash). Karena hanya satu DB yang menjadi titik commit, sistem tidak membutuhkan two-phase commit ini mengurangi kompleksitas dan meningkatkan performa. Praktiknya, transaksi di worker menjadi "batas konsistensi" yang jelas event diproses sukses atau dianggap gagal dan dapat diproses ulang, tanpa menghasilkan data ganda. Ini mengurangi kebutuhan locking manual dan menjaga latensi tetap stabil di beban tinggi. (Coulouris et al., 2012; van Steen & Tanenbaum, 2023)

T9 - Bab 9

Kontrol konkurensi pada sistem ini memanfaatkan mekanisme database untuk mencegah race condition antar worker. Beberapa worker dapat mengambil event yang sama atau event yang berbeda secara bersamaan, sehingga diperlukan aturan yang memastikan tidak ada event unik tersimpan ganda. Unique constraint pada (topic, event_id) menyediakan "locking" implisit pada level indeks dua transaksi yang mencoba memasukkan kunci yang sama akan menghasilkan satu sukses dan satu konflik secara atomik. Pola ini dipadukan dengan upsert/ON CONFLICT DO NOTHING sehingga penanganan konflik tidak perlu locking eksplisit atau koordinasi antar worker. Ini merupakan idempotent write pattern operasi tulis aman diulang karena efeknya tidak berlipat. Dampaknya, sistem bebas race untuk kasus duplikasi, dan lebih mudah dipelihara dibanding pendekatan manual seperti mutex global atau distributed lock. Selain itu, pencatatan audit dan counter duplicate_dropped memberi bukti empiris bahwa konflik terjadi dan ditangani dengan benar. Dengan kontrol konkurensi berbasis constraint, sistem tetap konsisten sekalipun throughput tinggi dan worker paralel bertambah. (Coulouris et al., 2012; van Steen & Tanenbaum, 2023)

T10 - Bab 10-13

Orkestrasi dilakukan dengan Docker Compose yang mendefinisikan service aggregator, broker, storage, dan publisher dalam satu jaringan lokal. Keamanan jaringan lokal dijaga dengan tidak mengekspos port Redis dan PostgreSQL ke host hanya aggregator membuka port 8080 untuk API, sehingga dependensi internal tidak dapat diakses dari luar secara langsung. Persistensi data dijamin oleh volume Docker pada PostgreSQL, sehingga data tetap ada saat container aggregator dihapus dan dibuat ulang, dan dedup store tetap durable. Observability disediakan melalui logging worker serta metrik ringan di GET /stats yang menampilkan jumlah received, unique_processed, dan duplicate_dropped kombinasi ini memudahkan validasi invariant dan troubleshooting. Selain itu, healthcheck pada broker/storage membantu startup yang deterministik, mengurangi kegagalan “connection refused” saat boot. Dengan desain ini, sistem tidak bergantung pada layanan eksternal semua komponen berjalan lokal dalam Compose network. Dokumentasi yang baik melengkapi implementasi dengan langkah build/run, cara menjalankan pytest, serta demonstrasi persistensi volume dan crash recovery. Dokumentasi README merangkum perintah run, test, serta bukti persistensi volume. (Coulouris et al., 2012; van Steen & Tanenbaum, 2023)

DAFTAR PUSTAKA

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

van Steen, M., & Tanenbaum, A. S. (2023). *Distributed systems* (4th ed.). self-published (distributed-systems.net).

LAMPIRAN

Link GitHub: https://github.com/Kaijeman/pub-sub-log-aggregator-uas

Link Video YouTube: https://www.youtube.com/watch?v=Xi5UWjmE5Sg