Nama : Nabila Chairunnisa

NIM : 11221005

Kelas : SPT – A

LAPORAN IMPLEMENTASI UTS

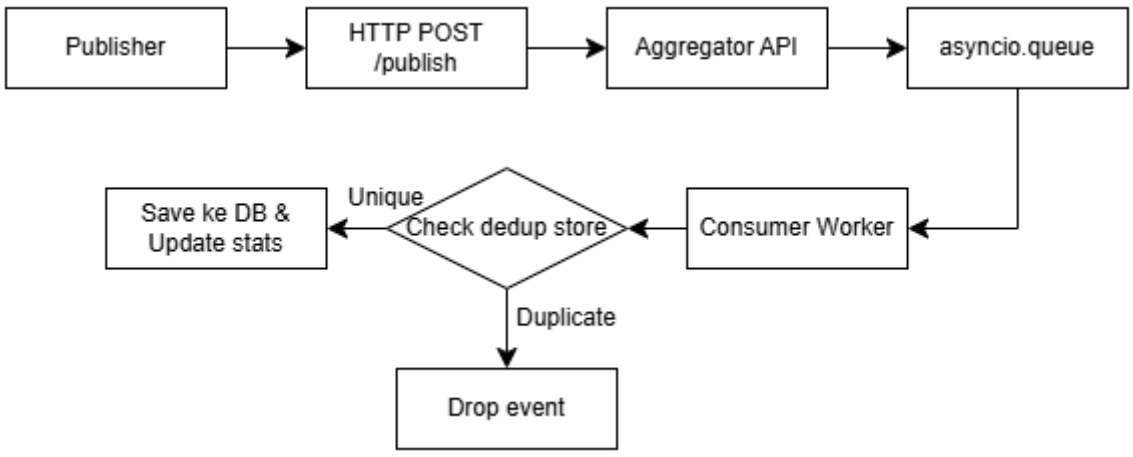
1. Ringkasan sistem dan arsitektur

   Sistem Pub-Sub Log Aggregator yang dikembangkan merupakan implementasi dari arsitektur event-driven dengan menerapkan pola producer–consumer menggunakan asynchronous queue untuk menangani aliran event dari sisi publisher. Dari sisi rancangan, sistem ini mengadopsi layered architecture yang memisahkan tanggung jawab secara jelas antara beberapa lapisan, yaitu API layer, queue management, consumer worker, dan persistence layer yang menggunakan SQLite sebagai basis data (Steen & Tanenbaum, 2023, Bab 2).

   Komponen publisher service beroperasi secara independen dan mengirimkan batch event melalui metode HTTP POST ke endpoint /publish pada aggregator service. Setiap event yang diterima akan melalui tahap validasi untuk memastikan kesesuaian schema dan timestamp, sebelum kemudian dimasukkan ke dalam in-memory asyncio.Queue sebagai buffer untuk menciptakan decoupling antara respons API dan proses event processing.

   Selanjutnya, consumer worker yang berjalan sebagai background task mengambil event dari antrean secara asynchronous, melakukan deduplikasi berdasarkan kombinasi (topic, event_id), dan menyimpan event yang terverifikasi unik ke dalam database SQLite yang dipersistenkan melalui Docker volume. Selain itu, sistem ini juga menerapkan middleware pattern (lihat Bab 2.2), di mana lapisan middleware berfungsi menyediakan antarmuka yang konsisten bagi aplikasi serta menyembunyikan kompleksitas distribusi data dan komunikasi antarkomponen.

   Secara keseluruhan, diagram arsitektur sistem seperti pada gambar di bawah ini, dengan setiap komponen memiliki peran dan tanggung jawab yang terdefinisi dengan jelas untuk memastikan integritas serta efisiensi dalam pemrosesan event.



2. Keputusan desain
   a. Idempotency dan Deduplication

      Salah satu keputusan desain fundamental adalah penerapan idempotency untuk memastikan bahwa event yang sama tidak diproses lebih dari sekali, sesuai dengan prinsip fault tolerance (Steen & Tanenbaum, 2023, Bab 8). Setiap event diidentifikasi secara unik melalui kombinasi (topic, event_id) yang disimpan sebagai composite primary key di tabel dedup_store pada database SQLite. Ketika consumer worker menerima event dari queue, ia melakukan pengecekan terlebih dahulu apakah pasangan (topic, event_id) sudah pernah diproses sebelumnya. Jika ditemukan duplikat, event akan didrop dan counter duplicate_dropped akan diinkremen, sedangkan jika event baru, maka akan dilakukan marking sebagai processed dan disimpan ke database. Pendekatan ini mengimplementasikan at-least-once delivery semantics dengan garantir bahwa meskipun publisher mengirim event yang sama berkali-kali (simulasi network retry atau failure), sistem hanya akan memproses event tersebut sekali saja. Mekanisme deduplication ini bersifat persistent, artinya bahkan setelah aggregator service restart, dedup store tetap mempertahankan record sehingga event duplikat yang datang setelah restart tetap dapat terdeteksi dan ditolak.

   b. Ordering dan Timestamp

      Sistem ini mengadopsi logical time melalui penggunaan timestamp ISO8601 yang diberikan oleh publisher pada setiap event (Steen & Tanenbaum, 2023, Bab 5). Setiap event memiliki field timestamp yang mencatat kapan event tersebut terjadi di sisi publisher, dan field received_at yang mencatat kapan event tersebut diterima dan diproses oleh aggregator dengan menggunakan datetime.now(timezone.utc). Meskipun sistem tidak menerapkan Lamport timestamps atau vector clocks untuk total ordering, penggunaan timestamp ISO8601 memungkinkan partial ordering berdasarkan waktu event creation. Event yang di-query melalui endpoint /events akan diurutkan berdasarkan received_at untuk memberikan gambaran kronologis yang akurat tentang urutan processing. Sistem tidak memaksakan strict causal ordering antar event karena event dari berbagai publisher dianggap independent, namun ordering per topic dapat dengan mudah diimplementasikan jika diperlukan.

   c. Retry dan Fault Tolerance

      Publisher service dirancang dengan retry mechanism untuk mengatasi situasi dimana aggregator belum ready atau mengalami temporary failure (Steen & Tanenbaum, 2023, Bab 8). Implementasi ini menggunakan exponential backoff dengan maximum retry attempts untuk memberikan jeda yang semakin lama antar retry, mengurangi beban pada aggregator yang sedang recovery. Dari sisi aggregator, penggunaan async queue sebagai buffer memberikan resilience terhadap spike dalam incoming requests. Jika API layer menerima burst traffic, event akan dimasukkan ke queue dan API dapat langsung memberikan response "accepted" tanpa menunggu processing selesai, sehingga menghindari timeout di sisi client. Consumer worker memproses event secara asynchronous dengan rate yang sustainable, dan jika terjadi exception saat processing, event tersebut akan di-log sebagai error dan counter duplicate_dropped akan diinkremen untuk mencegah infinite retry loop. Persistence melalui SQLite dengan Docker volume mapping memastikan bahwa data tidak hilang saat container restart. Ketika aggregator service restart, ia akan me-load kembali semua event yang sudah diproses dari database, sehingga dedup store dan event store tetap konsisten.

3. Analisis performa dan metrik

   Dari perspektif scalability, sistem ini menunjukkan karakteristik yang baik untuk size scalability dalam konteks single-instance deployment. Dengan menggunakan asyncio queue dan asynchronous processing, sistem dapat menangani ribuan event per detik tanpa blocking API layer. Testing dengan batch 1000 event menunjukkan response time API di bawah 1 detik, dengan total processing time (termasuk deduplication dan database writes) sekitar 12-15 detik. Metrik key performance yang dimonitor meliputi:

   a. Throughput, diukur dari stats.received per unit waktu, yang menunjukkan rate incoming events;
   b. Processing latency, waktu antara event masuk queue hingga selesai diproses, dipengaruhi oleh database I/O dan dedup check;
   c. Deduplication rate, ratio duplicate_dropped / received, yang pada simulasi dengan 20% duplikat menunjukkan akurasi 100% dalam mendeteksi duplikat;
   d. Queue depth, jumlah event yang menunggu di queue, yang tetap stabil bahkan saat burst traffic karena consumer worker efficiently draining the queue.

4. Keterkaitan dengan Bab 1-7

   a. Bab 1 (Introduction): Sistem ini menjawab goal dari distributed systems dalam hal resource sharing dimana multiple publishers dapat mengirim event ke shared aggregator service, dan distribution transparency dimana publisher tidak perlu tahu detail implementasi internal aggregator seperti queue management dan database storage. Desain mempertimbangkan scalability dimensions terutama size scalability melalui asynchronous processing, meskipun geographical dan administrative scalability memerlukan enhancements seperti geo-distributed deployment dan multi-tenancy support (Steen & Tanenbaum, 2023, Bab 1).
   b. Bab 2 (Architectures): Implementasi mengadopsi layered architecture dengan pemisahan jelas antara presentation layer (REST API), application layer (event validation dan queue management), dan data layer (SQLite persistence). Sistem juga menerapkan middleware organization dimana aggregator service bertindak sebagai middleware yang menyediakan abstraksi tingkat tinggi untuk event processing, menyembunyikan kompleksitas deduplication dan persistence dari publisher. Penggunaan simple client-server architecture terlihat jelas dalam komunikasi HTTP antara publisher (client) dan aggregator (server), dengan tambahan asynchronous processing untuk improved responsiveness (Steen & Tanenbaum, 2023, Bab 2).
   c. Bab 3 (Processes): Consumer worker diimplementasikan sebagai background task (implicitly covered in process models) yang berjalan di dalam event loop asyncio, demonstrating lightweight concurrency model. Penggunaan asyncio.Queue memungkinkan efficient coordination antara API handler (producer) dan consumer worker tanpa blocking, implementing producer-consumer pattern yang scalable (Steen & Tanenbaum, 2023, Bab 3).
   d. Bab 4 (Communication): Sistem menggunakan HTTP-based communication dengan RESTful API untuk request-reply pattern antara publisher dan aggregator. Komunikasi bersifat transient dimana publisher harus aktif untuk mengirim event, namun aggregator menyediakan persistent storage untuk event yang sudah diterima. Penggunaan async queue menciptakan asynchronous communication dimana API dapat langsung return response tanpa menunggu processing selesai, improving overall system throughput (Steen & Tanenbaum, 2023, Bab 4).

e. Bab 5 (Coordination): Deduplication mechanism mengimplementasikan bentuk distributed mutual exclusion dimana hanya satu instance dari setiap unique event (identified by topic + event_id) yang boleh diproses. Meskipun implementasi saat ini centralized (single aggregator), konsep ini dapat diperluas ke distributed setting dengan coordination protocol. Penggunaan timestamp dan ordering by received_at berhubungan dengan logical clocks meskipun tidak menggunakan Lamport timestamps, sistem tetap maintain happened-before relationship untuk events (Steen & Tanenbaum, 2023, Bab 5).
f. Bab 6 (Naming): Event addressing menggunakan structured naming melalui kombinasi topic (hierarchical identifier) dan event_id (flat identifier), memungkinkan efficient lookup dan filtering. Topic-based organization memfasilitasi attribute-based naming yang umum dalam publish-subscribe systems, dimana subscriber (dalam konteks ini, query via /events?topic=X) dapat mengakses event based on topic attribute (Steen & Tanenbaum, 2023, Bab 6).
g. Bab 7 (Consistency and Replication): Meskipun sistem saat ini single-instance, desain database dengan dual-table approach (dedup_store dan events) dan penggunaan composite primary key mengimplementasikan konsep data-centric consistency. Jika sistem di-replicate di masa depan, dedup store harus maintained consistently across replicas untuk prevent duplicate processing, yang akan require consistency protocols. Persistence mechanism dengan Docker volume memastikan durability yang merupakan bagian dari ACID properties dalam database systems (Steen & Tanenbaum, 2023, Bab 7).

Referensi:

Steen, M. V., & Tanenbaum, A. S. (2023). *Distributed Systems.* Maarten van Steen.