10.37817/ikraith-informatika.v9i1
P-ISSN :2580-4316
E-ISSN :2654-8054

INTEGRASI DATA "REAL-TIME" UNTUK ASURANSI MENGGUNAKAN KAFKA DAN "PLATFORM STREAMING"

Muhamad Muchlis[*1], Jefry Sunupurwa[2]

[1,2] Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Esa Unggul

email: [1]muhamad.muchlis@student.esaunggul.ac.id, [2]jefry.sunupurwa@esaunggul.ac.id,

Abstrak

Implementasi *message broker* berbasis *platform streaming* memberikan solusi efektif untuk mengintegrasikan data nasabah asuransi secara real-time dengan efisiensi tinggi. Penelitian ini menggunakan Debezium untuk *Change Data Capture (CDC)*, Apache Kafka sebagai *message broker*, dan MongoDB sebagai penyimpanan terintegrasi. Hasil pengujian menunjukkan sistem mampu memproses hingga 5000 pesan per detik dengan waktu pemrosesan rata-rata 1-2 ms per batch. Proses integrasi data dilakukan secara efisien, termasuk validasi format data dan pengelompokan, dengan sistem penanganan kesalahan untuk data tidak valid. Penggunaan Java Rest API memungkinkan akses data secara real-time untuk kebutuhan pelaporan dan analitik, mendukung pengambilan keputusan berbasis data. Arsitektur ini juga didukung oleh optimalisasi sumber daya dengan konfigurasi CPU 8 core, memori 7 GB, dan disk virtual 60 GB. Sistem berhasil mengintegrasikan data dari berbagai sumber seperti GLINDO, PLINDO, dan ASKES, menghasilkan solusi data terpusat yang andal untuk kebutuhan operasional dan strategis. Implementasi ini memberikan kontribusi signifikan terhadap transformasi digital di industri asuransi, meningkatkan efisiensi operasional dan kualitas layanan. Penelitian ini membuktikan bahwa *message broker* berbasis *platform streaming* adalah pendekatan yang efektif untuk memenuhi kebutuhan integrasi data berskala besar.

Kata kunci: Message Broker, Platform Streaming, Kafka, Integrasi Data, Asuransi**.**

REAL-TIME DATA INTEGRATION FOR INSURANCE USING KAFKA AND STREAMING PLATFORMS

Abstract

The implementation of a message broker based on a streaming platform provides an effective solution for integrating insurance customer data in real-time with high efficiency. This study employs Debezium for Change Data Capture (CDC), Apache Kafka as the message broker, and MongoDB as integrated storage. Testing results show the system can process up to 5000 messages per second with an average processing time of 1-2 ms per batch. The data integration process is efficient, including data format validation and grouping, with an error-handling system for invalid data. The use of Java Rest API allows real-time data access for reporting and analytics needs, supporting data-driven decision-making. This architecture is further enhanced by resource optimization with configurations of 8-core CPUs, 7 GB memory, and 60 GB virtual disk. The system successfully integrates data from various sources such as GLINDO, PLINDO, and ASKES, producing a centralized and reliable data solution for operational and strategic needs. This implementation significantly contributes to digital transformation in the insurance industry, improving operational efficiency and service quality. This study demonstrates that a message broker based on a streaming platform is an effective approach for meeting large-scale data integration needs.

Keywords: *Message Broker, Streaming Platform, Kafka, Data Integration, Insurance.*

P-ISSN :2580-4316
E-ISSN :2654-8054
10.37817/ikraith-informatika.v9i1

1. PENDAHULUAN

Perkembangan teknologi informasi telah mengubah cara organisasi mengelola data, termasuk dalam industri asuransi. Sebagai salah satu sektor dengan volume data yang sangat besar, perusahaan asuransi dihadapkan pada tantangan pengelolaan dan integrasi data nasabah yang berasal dari berbagai sumber. Pada 2023, Asosiasi Asuransi Jiwa Indonesia (AAJI) mendata 75,5 juta orang pada 2023. Jumlah tersebut menuntut perusahaan asuransi untuk memberikan pelayanan yang optimal, termasuk dalam hal pengelolaan data nasabah. "Indonesia Financial Group" (IFG) merupakan holding perusahaan asuransi dan penjaminan BUMN di Indonesia. Data tersebut tidak hanya mencakup informasi demografis nasabah, tetapi juga data transaksi, klaim, hingga riwayat interaksi pelanggan. Dalam konteks ini, integrasi data yang cepat, akurat, dan aman menjadi elemen kunci untuk mendukung efisiensi operasional dan pengambilan keputusan berbasis data (Hesse & Hesse, 2018). Salah satu solusi yang semakin populer untuk menjawab kebutuhan ini adalah implementasi development message broker menggunakan platform streaming.

Message broker merupakan perangkat lunak yang berperan sebagai perantara dalam pengiriman pesan antar sistem atau aplikasi. Peran ini sangat penting dalam ekosistem teknologi modern karena memungkinkan integrasi data yang bersifat heterogen dan berasal dari berbagai platform. Dengan memanfaatkan platform streaming, seperti Apache Kafka atau RabbitMQ, message broker dapat memfasilitasi transfer data secara real-time, memastikan konsistensi data, dan meningkatkan skalabilitas sistem (Kreps et al., 2011). Implementasi ini memberikan kemampuan bagi perusahaan asuransi untuk merespons kebutuhan bisnis yang dinamis sekaligus menghadirkan pengalaman pelanggan yang lebih baik.

Industri asuransi memiliki karakteristik unik yang menjadikannya salah satu sektor yang paling membutuhkan integrasi data yang efisien. Misalnya, proses klaim asuransi sering kali memerlukan data dari berbagai departemen, seperti data polis, riwayat pembayaran, hingga hasil evaluasi risiko. Dengan memanfaatkan message broker, data dari berbagai sumber ini dapat diintegrasikan dan diproses secara terpusat, sehingga memungkinkan analisis yang lebih mendalam dan penyediaan layanan yang lebih cepat. Selain itu, pendekatan ini juga dapat membantu perusahaan memenuhi regulasi yang mewajibkan transparansi dan akuntabilitas dalam pengelolaan data nasabah (Kim et al., 2020).

Implementasi message broker berbasis platform streaming juga menawarkan sejumlah manfaat teknis yang signifikan. Salah satunya adalah kemampuan untuk membangun data pipeline yang dapat menangani volume data yang sangat besar dengan tingkat latensi rendah. Teknologi ini mendukung pemrosesan data secara terdistribusi, yang berarti beban kerja dapat dibagi di antara beberapa server untuk meningkatkan efisiensi. Selain itu, arsitektur ini juga memungkinkan perusahaan untuk mengembangkan aplikasi yang tangguh, dengan dukungan fitur-fitur seperti fault tolerance dan replikasi data (Gwen et al., 2018).

Meskipun demikian, implementasi teknologi ini tidak terlepas dari tantangan. Salah satu tantangan utama adalah kebutuhan akan kompetensi teknis yang tinggi, baik dalam pengembangan maupun pemeliharaan sistem. Selain itu, integrasi teknologi baru dengan infrastruktur yang sudah ada juga memerlukan perencanaan yang matang untuk meminimalkan risiko gangguan operasional. Tantangan lain adalah perlunya strategi pengelolaan keamanan data, mengingat data nasabah asuransi sering kali bersifat sensitif dan dilindungi oleh regulasi privasi data yang ketat, seperti GDPR di Eropa atau UU Perlindungan Data Pribadi di Indonesia (Gibb et al., 2017).

Penelitian ini bertujuan untuk mengeksplorasi implementasi development message broker menggunakan platform streaming dalam konteks integrasi data nasabah asuransi. Dengan pendekatan ini, diharapkan perusahaan asuransi dapat meningkatkan efisiensi operasional, mempercepat proses pengambilan keputusan, dan menyediakan layanan yang lebih baik bagi nasabah. Studi ini juga akan mengidentifikasi manfaat, tantangan, serta faktor-faktor kunci yang memengaruhi keberhasilan implementasi teknologi ini.

Melalui penelitian ini, kontribusi yang diharapkan tidak hanya terbatas pada pemahaman teknis, tetapi juga pada panduan praktis bagi perusahaan yang ingin mengadopsi teknologi serupa. Dengan demikian, implementasi message broker berbasis platform streaming dapat menjadi langkah strategis bagi industri asuransi untuk menghadapi tantangan transformasi digital dan tetap kompetitif di pasar yang semakin kompleks dan dinamis.

2. STUDI KAJIAN LITERASI

Implementasi message broker menggunakan platform streaming telah menjadi fokus penelitian dalam berbagai domain, termasuk integrasi data nasabah asuransi. Message broker berfungsi sebagai perantara yang memfasilitasi pertukaran data antara sistem yang berbeda secara efisien dan andal. Dalam konteks asuransi, integrasi data nasabah yang berasal dari berbagai sumber memerlukan sistem yang mampu menangani volume data besar secara real-time.

P-ISSN :2580-4316
E-ISSN :2654-8054

Platform streaming menawarkan solusi dengan menyediakan pemrosesan data berkelanjutan dan latensi rendah.

Salah satu penelitian yang relevan adalah oleh Badidi (2018), yang mengusulkan platform berbasis message broker untuk streaming data IoT secara real-time di lingkungan perkotaan (Sudirman, 2024; Muhammad Rizal, 2025). Meskipun fokusnya pada data IoT, arsitektur yang diusulkan menunjukkan potensi adaptasi dalam sektor asuransi untuk mengintegrasikan data nasabah dari berbagai perangkat dan sumber secara efisien. Pendekatan ini menekankan pentingnya skalabilitas dan kemampuan menangani heterogenitas perangkat dalam sistem integrasi data. Lopes et al. (2024) melakukan evaluasi terhadap berbagai teknologi message broker dalam konteks sistem pemantauan real-time. Mereka menilai kinerja message broker seperti Kafka, RabbitMQ, dan ActiveMQ berdasarkan indikator kinerja utama seperti latensi, throughput, dan kemampuan menangani banyak koneksi secara bersamaan. Hasil penelitian ini memberikan panduan berharga dalam memilih message broker yang paling sesuai untuk kebutuhan pemrosesan data real-time dalam berbagai aplikasi, termasuk integrasi data nasabah asuransi. Srinivas dan Karna (2019) melakukan survei terhadap berbagai message broker untuk pemrosesan big data real-time. Mereka menyoroti bahwa Apache Kafka menawarkan throughput tinggi dan latensi rendah, menjadikannya pilihan yang andal untuk aplikasi yang memerlukan pemrosesan data cepat dan andal. Temuan ini relevan untuk perusahaan asuransi yang perlu mengintegrasikan dan menganalisis data nasabah secara efisien. Selain itu, penelitian oleh Ferreira (2013) membahas peran message broker dalam integrasi sistem perusahaan. Meskipun tidak secara spesifik menyoroti sektor asuransi, konsep yang dibahas dapat diterapkan dalam mengintegrasikan berbagai aplikasi dan layanan yang digunakan oleh perusahaan asuransi untuk mengelola data nasabah. Secara keseluruhan, literatur menunjukkan bahwa implementasi message broker menggunakan platform streaming menawarkan solusi efektif untuk integrasi data nasabah asuransi. Dengan memilih arsitektur dan teknologi yang tepat, perusahaan asuransi dapat meningkatkan efisiensi operasional dan kualitas layanan melalui pemrosesan data real-time yang andal.

3. Metodologi dan Rancangan

3.1 Metode *Rapid Application Development* (RAD)

Metode Rapid Application Development (RAD) adalah pendekatan pengembangan perangkat lunak yang menekankan kecepatan dan fleksibilitas melalui siklus pengembangan yang singkat dan iteratif. Dalam konteks implementasi *message broker* menggunakan *platform streaming* untuk integrasi data nasabah asuransi, metode RAD menawarkan beberapa kelebihan dan kekurangan (Pressman, 2020; Kendall, et al. 2019; Sommerville, 2020; Mirfan dkk, 2024) sebagai berikut;

Kelebihan Metode RAD;

1) Pengembangan Cepat: RAD memungkinkan pengembangan prototipe dan aplikasi dengan cepat, sehingga pemangku kepentingan dapat melihat produk yang hampir jadi lebih awal dalam proses pengembangan.
2) Keterlibatan Pengguna Aktif: metode ini mendorong keterlibatan aktif pengguna sepanjang siklus pengembangan, memastikan bahwa kebutuhan dan harapan mereka terpenuhi dengan baik.
3) Fleksibilitas terhadap Perubahan: RAD memungkinkan penyesuaian kebutuhan dan keinginan pengguna menjadi lebih mudah, sehingga perubahan dalam persyaratan dapat diakomodasi tanpa mengganggu alur pengembangan.
4) Pengurangan Kesalahan: dengan pendekatan pembuatan prototipe dan umpan balik yang berkelanjutan, metode ini dapat memperkecil kemungkinan kesalahan atau *error* dalam pengembangan sistem.

Kekurangan Metode RAD;

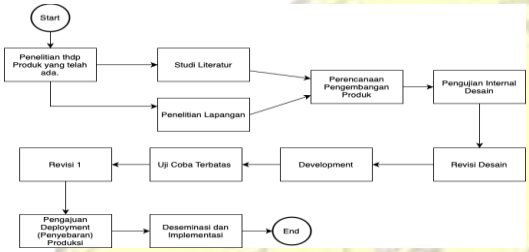
1) Keterbatasan pada Proyek Skala Besar: RAD lebih cocok untuk proyek dengan ruang lingkup yang terbatas dan tidak terlalu kompleks. Pada proyek yang lebih besar, mungkin sulit untuk memodularisasi sistem secara efektif dan mengintegrasikan komponen-komponen yang sudah ada.
2) Ketergantungan pada Tim yang Terampil: keberhasilan RAD sangat tergantung pada keterampilan dan komitmen tim pengembang serta pemangku kepentingan. Tanpa kolaborasi yang kuat, proyek dapat mengalami kesulitan.
3) Potensi Pengurangan Fitur: Karena fokus pada pengembangan cepat, ada kemungkinan beberapa fitur dikurangi atau ditunda untuk versi selanjutnya guna memenuhi batas waktu yang ketat.
4) Dokumentasi yang Kurang: proses pengembangan yang cepat dan iteratif dalam RAD sering kali mengakibatkan dokumentasi yang kurang lengkap, yang dapat menyulitkan pemeliharaan dan pengembangan di masa depan.

Dalam implementasi *message broker* menggunakan *platform streaming* untuk integrasi data nasabah

P-ISSN :2580-4316
E-ISSN :2654-8054
10.37817/ikraith-informatika.v9i1

asuransi, metode RAD dapat memberikan keuntungan dalam hal kecepatan pengembangan dan respons terhadap kebutuhan pengguna. Namun, penting untuk mempertimbangkan keterbatasan metode ini, terutama terkait kompleksitas proyek dan kebutuhan akan kolaborasi tim yang solid. Pemilihan metode pengembangan harus disesuaikan dengan karakteristik proyek dan sumber daya yang tersedia untuk mencapai hasil yang optimal.

3.2 Tahapan



Gambar 1. Tahapan penelitian

Implementasi tahapan *message broker* dengan *platform streaming* dalam integrasi data nasabah asuransi memerlukan pendekatan penelitian yang sistematis. Berikut adalah tahapan yang dapat diikuti:

1) **Identifikasi Kebutuhan dan Analisis Sistem**
   Tujuan:
   Memahami kebutuhan integrasi data nasabah asuransi dan menentukan persyaratan sistem yang diperlukan.
   Langkah
   - Melakukan wawancara dan diskusi dengan pemangku kepentingan untuk mengidentifikasi kebutuhan bisnis dan teknis.
   - Menganalisis sistem yang ada untuk memahami alur data dan titik integrasi yang diperlukan.
   - Menetapkan tujuan dan ruang lingkup implementasi *message broker* dan *platform streaming*.
2) **Studi Literatur dan Pemilihan Teknologi**
   Tujuan:
   Mengidentifikasi teknologi *message broker* dan *platform streaming* yang sesuai untuk kebutuhan integrasi data nasabah asuransi.
   Langkah:
   - Melakukan tinjauan literatur untuk memahami teknologi yang tersedia dan praktik terbaik dalam integrasi data real-time.
   - Mengevaluasi berbagai *message broker* seperti Apache Kafka, RabbitMQ, dan ActiveMQ berdasarkan kriteria seperti kinerja, skalabilitas, dan dukungan komunitas.
   - Memilih teknologi yang paling sesuai dengan kebutuhan dan infrastruktur perusahaan asuransi.
3) **Perancangan Arsitektur Sistem**
   Tujuan:
   Merancang arsitektur sistem yang mengintegrasikan *message broker dan platform streaming* dengan sistem asuransi yang ada.
   Langkah:
   - Membuat diagram arsitektur yang menggambarkan komponen utama dan alur data antara sistem sumber, *message broker*, dan aplikasi konsumen.
   - Menentukan skema data dan format pesan yang akan digunakan untuk memastikan kompatibilitas antar sistem.
   - Merencanakan mekanisme keamanan dan pemantauan untuk memastikan integritas dan ketersediaan data.
4) **Pengembangan dan Pengujian Prototipe**
   Tujuan:
   Membangun dan menguji prototipe sistem untuk memastikan fungsionalitas dan kinerja sesuai dengan spesifikasi.
   Langkah:
   - Mengembangkan komponen sistem sesuai dengan desain yang telah dibuat, termasuk konfigurasi *message broker* dan pengembangan aplikasi konsumen.
   - Melakukan pengujian fungsional untuk memastikan setiap komponen bekerja sesuai dengan yang diharapkan.
   - Melakukan pengujian kinerja untuk memastikan sistem dapat menangani volume data yang diharapkan dengan latensi yang minimal.
5) **Implementasi dan Evaluasi**
   Tujuan:
   Mengimplementasikan sistem dalam lingkungan produksi dan mengevaluasi kinerjanya secara keseluruhan.
   Langkah:
   - Melakukan migrasi sistem secara bertahap untuk meminimalkan gangguan terhadap operasi bisnis.
   - Melatih pengguna dan tim operasional mengenai penggunaan dan pemeliharaan sistem baru.
   - Memantau kinerja sistem dan mengumpulkan umpan balik dari pengguna untuk identifikasi area yang memerlukan perbaikan atau pengembangan lebih lanjut.

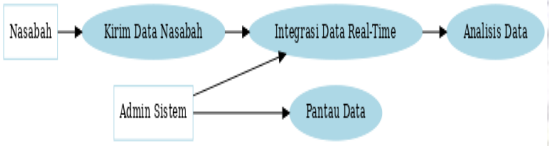
P-ISSN :2580-4316
E-ISSN :2654-8054

3.3 Pendekatan *Unified Modelling Language* (UML)

Unified Modeling Language (UML) adalah bahasa pemodelan standar yang digunakan untuk mem-visualisasikan, menspesifikasikan, membangun, dan mendokumentasikan komponen-komponen sistem perangkat lunak. Dalam konteks implementasi *message broker* menggunakan *platform streaming* untuk integrasi data nasabah asuransi, UML dapat membantu dalam merancang dan memahami struktur serta interaksi sistem yang kompleks.

1) ***Use Case Diagram***

Use Case Diagram menggambarkan interaksi antara aktor (pengguna atau sistem lain) dengan sistem yang akan dikembangkan. Dalam kasus ini, aktor dapat berupa sistem asuransi yang mengirim atau menerima data nasabah, dan *message broker* berperan sebagai perantara. Diagram ini membantu dalam mengidentifikasi fungsi-fungsi utama yang harus disediakan oleh sistem.



Gambar 2. Use Case Diagram

2) ***Activity Diagram***

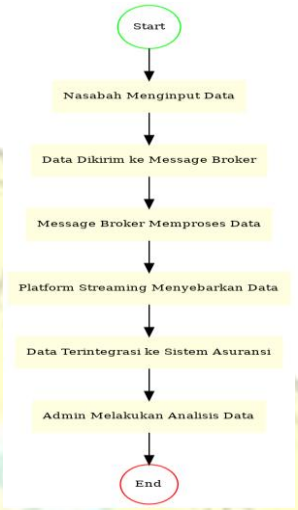
Activity Diagram memodelkan alur kerja atau proses bisnis dalam sistem. Untuk integrasi data nasabah asuransi, diagram ini dapat meng-gambarkan proses pengiriman data dari sistem sumber, pemrosesan oleh *message broker*, hingga data diterima oleh sistem tujuan. Hal ini mempermudah identifikasi langkah-langkah proses dan potensi bottleneck dalam alur kerja.

3) ***Sequence Diagram***

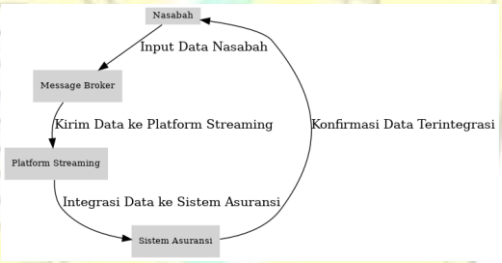
Sequence Diagram menunjukkan interaksi antara objek dalam urutan waktu tertentu. Dalam implementasi ini, diagram tersebut dapat memodelkan urutan pesan yang dikirim antara sistem asuransi, *message broker*, dan *platform streaming*. Ini membantu dalam memahami dinamika komunikasi antar komponen sistem.

4) ***Class Diagram***

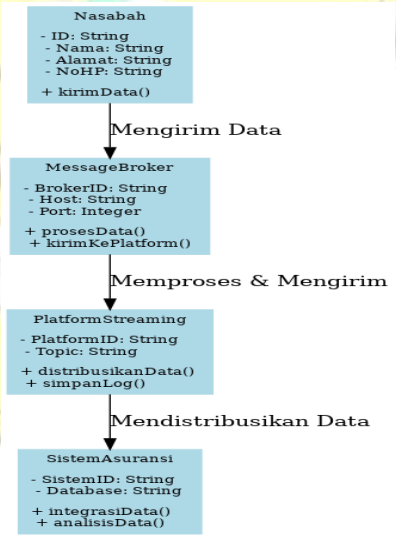
Class Diagram memodelkan struktur statis dari sistem dengan menunjukkan kelas-kelas yang akan diimplementasikan, beserta atribut dan metode mereka. Dalam konteks ini, diagram tersebut dapat digunakan untuk mendefinisikan struktur data nasabah, komponen *message broker*, dan entitas lain yang terlibat dalam integrasi.



Gambar 3. Activity Diagram



Gambar 4.Sequence Diagram

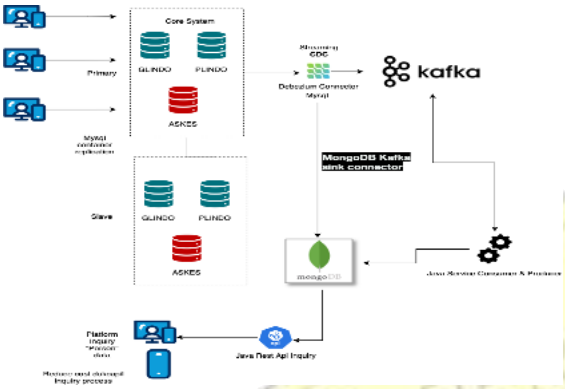


Gambar 5.Sequence Diagram

4. Hasil dan Pembahasan

4.1 Arsitektur Integrasi

P-ISSN :2580-4316
E-ISSN :2654-8054
10.37817/ikraith-informatika.v9i1



Gambar 6. Arsitektur Sistem Integrasi Data Nasabah

Alur Data, deskripsi;

1) Pengguna melakukan transaksi pada sistem inti (GLINDO, PLINDO, atau ASKES).
2) Debezium memonitor perubahan pada database MySQL dan mengirimkan data perubahan ke Apache Kafka.
3) Kafka mengelola aliran data dan mendistri-busikannya ke MonggoDB menggunakan Kafka Sink Connector.
4) Data dalam MongoDB dapat diakses oleh aplikasi melalui Java Rest API.
5) Platform pengguna membaca data real-time untuk memberikan layanan sesuai kebutuhan pengguna.

Uraian singkat diagram arsitektur integrasi data pada gambar 6. diatas, berikut adalah deskripsi arsitektur integrasi data:

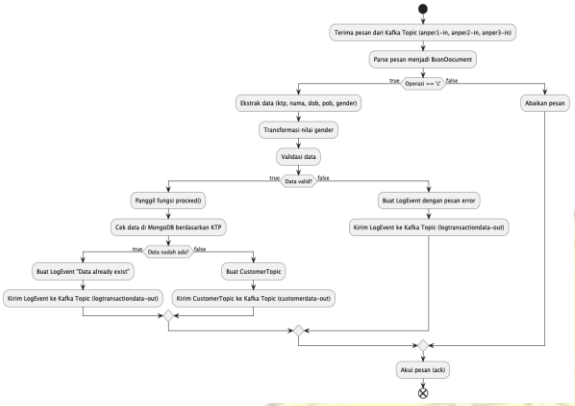
1) Sistem Sumber Data (Core System):
   - Sistem sumber data terdiri dari beberapa database utama, yaitu; PLINDO, GLINDO, dan ASKES, yang digunakan untuk menyimpan data inti.
   - Sistem ini terbagi menjadi dua bagian:
     - Primary: Sistem inti utama tempat transaksi data dilakukan.
     - Slave: Replikasi data yang berfungsi sebagai salinan untuk mendukung integrasi dan keamanan data.
2) Debezium Connector:
   - Sistem memanfaatkan dan optimalisasi Debezium Connector sebagai alat untuk melakukan *Change Data Capture* (CDC) dari basis data MySQL pada sistem inti.
   - Debezium memonitor perubahan (insert, update, delete) pada database dan mengirimkan data perubahan ini ke Apache Kafka secara real-time.
3) Apache Kafka:
   - Kafka digunakan sebagai *message broker* untuk mengelola dan mentransfer data antar sistem secara real-time.
   - Kafka menerima data dari Debezium Connector, kemudian menyebarkannya ke berbagai *sink* atau tujuan melalui topik-topik yang telah didefinisikan.
4) MongoDB Sink Connector:
   - Data yang dikirim melalui Kafka diteruskan ke MongoDB menggunakan MongoDB Kafka Sink Connector.
   - MongoDB berfungsi sebagai penyimpanan data terintegrasi yang dapat digunakan untuk kebutuhan analisis atau akses data lebih lanjut.
5) Java Rest API:
   - Java Rest API Inquiry digunakan sebagai antarmuka aplikasi untuk membaca data dari MongoDB atau Kafka.
   - Platform pengguna seperti aplikasi mobile atau web dapat mengakses data melalui API ini untuk keperluan visualisasi, pelaporan, atau layanan lainnya.
6) Platform Inquiry:
   - Platform pengguna (web atau mobile) memanfaatkan Java Rest API untuk mengambil data secara real-time yang telah diolah oleh pipeline Kafka dan tersimpan dalam MongoDB.

4.2 Implementasi Integrasi

Arsitektur integrasi data (gambar 6.) yang ditunjukkan dalam diagram memanfaatkan berbagai komponen teknologi untuk mengelola dan mendistribusikan data nasabah secara real-time. Berikut langkah implementasi dari arsitektur tersebut:

1) Data transaksi nasabah dimasukkan ke sistem inti (GLINDO, PLINDO, atau ASKES).
2) Debezium menangkap perubahan data dari database MySQL (Primary) dan mengirimnya ke Kafka.
3) Kafka mengelola data dalam topik-topik terpisah, kemudian meneruskan data ke MongoDB melalui Kafka Sink Connector.
4) MongoDB menyimpan data yang terintegrasi untuk keperluan analisis atau pelaporan.
5) Java Rest API menyediakan antarmuka bagi aplikasi pengguna untuk mengambil data dari MongoDB.
6) Platform inquiry mengakses data melalui API untuk melayani berbagai kebutuhan pengguna.

P-ISSN :2580-4316
E-ISSN :2654-8054

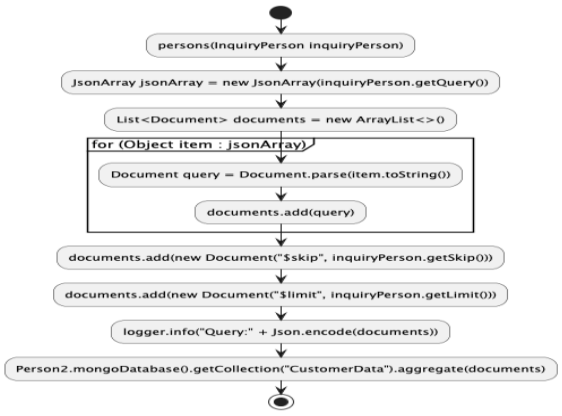


Gambar 7. Service Consumer dan Producer Process

Diagram 7., menunjukkan alur pemrosesan data dari Kafka dengan langkah-langkah berikut:

1) Penerimaan Pesan: Data diterima dari topik Kafka tertentu (import_1, import_2, atau export_1).
2) Transformasi Data: Pesan JSON diubah menjadi dokumen yang dapat diproses lebih lanjut.
3) Validasi Data: Data diperiksa untuk memastikan format dan konten valid.
4) Pengelompokan Data: Jika valid, data di-kelompokkan berdasarkan jenisnya.
5) Penyimpanan Data: Data yang valid disimpan ke MongoDB atau diteruskan ke Kafka untuk distribusi lebih lanjut.
6) Error Handling: Jika data tidak valid, pesan dikirim ke topik Kafka khusus untuk penanganan error atau log kesalahan.

Proses ini mendukung pengelolaan data secara real-time dengan validasi yang ketat untuk integrasi data yang terstruktur dan andal.

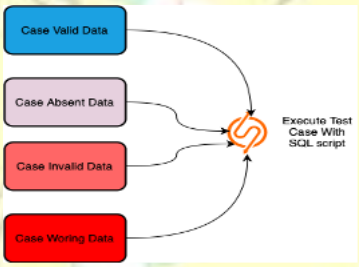


Gambar 8. Inquiry Proses

Diagram 8., menggambarkan alur pemrosesan data menggunakan Java untuk mengambil data dari MongoDB.

1) Input Data: Data dimulai dari objek nquiryPerson, yang memuat query dari pengguna.
2) Konversi Data: Query dari objek inquiryPerson diubah menjadi format JsonArray untuk mempermudah iterasi.
3) Pengolahan Dokumen: Setiap elemen dalam JsonArray diubah menjadi dokumen MongoDB menggunakan Document.parse() dan ditambahkan ke daftar dokumen.
4) Penerapan Filter: Filter seperti skip dan limit ditambahkan ke dokumen untuk mengatur paginasi data.
5) Logging Query: Query yang sudah diolah dicatat untuk keperluan monitoring atau debugging.
6) Eksekusi Query: Data diambil dari koleksi MongoDB CustomerData menggunakan metode aggregate().

4.3 Pengujian Fungsional Sistem



Gambar 9. Testing Fungsional

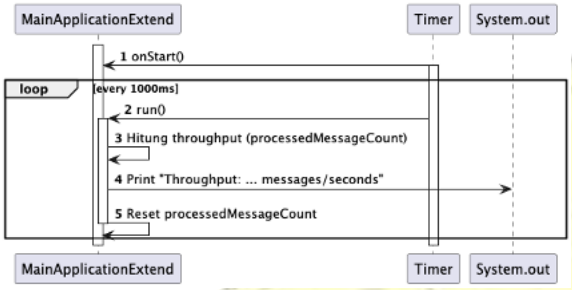
Gambar 9., menggambarkan pengujian beberapa skenario kasus data menggunakan skrip SQL.

1) Skenario Kasus
   - Case Valid Data: Data yang valid dan memenuhi semua kriteria integritas dan format.
   - Case Absent Data: Data yang hilang atau tidak tersedia dalam sistem.
   - Case Invalid Data: Data yang tidak valid karena kesalahan format atau nilai.
   - Case Wrong Data: Data yang salah, tidak relevan, atau tidak sesuai dengan ekspektasi.
2) Eksekusi Pengujian
   - Semua kasus data di atas dikirim untuk diuji menggunakan skrip SQL.
   - Skrip SQL memverifikasi data berdasarkan skenario yang diberikan, mencakup

P-ISSN :2580-4316
E-ISSN :2654-8054
10.37817/ikraith-informatika.v9i1

validasi, pelaporan error, dan memastikan hasil sesuai dengan kriteria pengujian.

4.4 Pengujian Performance

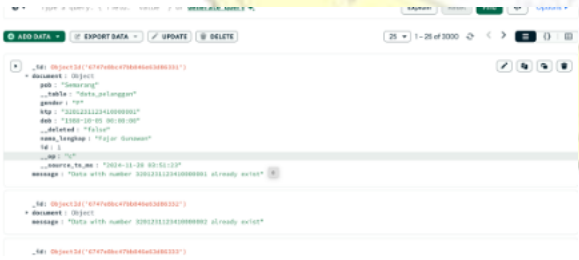


Gambar 10. Testing Performance

Diagram sequence (gambar 10.) yang Anda unggah menunjukkan proses perhitungan *throughput* pesan yang diproses dalam aplikasi.

1) Inisialisasi Proses (onStart):
   Ketika aplikasi dimulai (onStart), timer diatur untuk menjalankan perhitungan secara periodik (setiap 1000ms).
2) Loop Eksekusi (run):
   Timer memanggil fungsi run() setiap 1000ms untuk memproses perhitungan throughput.
3) Hitung Throughput:
   Aplikasi menghitung jumlah pesan yang diproses (processedMessageCount) selama interval waktu 1000ms.
4) Cetak Hasil Throughput:
   Hasil throughput (misalnya, "X messages/second") dikirim ke konsol menggunakan System.out.
5) Reset Counter:
   Setelah throughput di hitung dan di cetak, processedMessageCount diatur ulang ke nol untuk interval berikutnya.

4.5 Pengujian Fungsional Sistem



Gambar 11. Data Collection Event MongoDB Document

Fungsional sistem (gambar 11.) menunjukkan data yang disimpan dalam MongoDB, yang mencakup dokumen-dokumen terkait data pelanggan dan pesan status.

1) Data Pelanggan:
   - Informasi pelanggan mencakup:
     - Nama Lengkap: Misalnya, "Fajar Gunawan".
     - Tanggal Lahir: "1988-10-15 00:00:00".
     - Jenis Kelamin: "L".
   - Setiap dokumen memiliki *unique identifier* (_id) yang dihasilkan secara otomatis.
2) Pesan Status:
   - Terdapat pesan dengan keterangan seperti:
     - "Data with number 202311261500001 already exists".
       - Pesan ini menunjukkan bahwa data tertentu telah terdeteksi sebagai duplikasi dan tidak dimasukkan ulang.
3) Metadata:
   - Terdapat atribut tambahan, seperti:
     - _source_ts_ms: Timestamp proses penyimpanan.
     - Table: Nama tabel sumber data.

Data ini menunjukkan integrasi yang berhasil dan validasi untuk memastikan tidak ada duplikasi dalam penyimpanan data pelanggan.

Gambar 12. Data CustomerData MongoDB document.

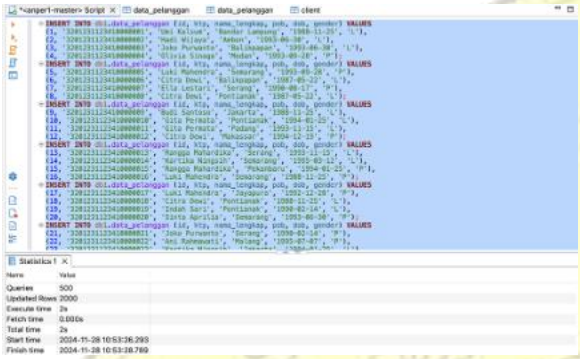
informasi (gambar 12.) tentang koleksi data yang disimpan dalam MongoDB, termasuk CustomerData dan LogEvent.

1) CustomerData:
   - Ukuran Penyimpanan: 94,62 KB.
   - Jumlah Dokumen: 5000 dokumen.
   - Ukuran Rata-Rata Dokumen: 166,03 byte.
   - Jumlah Indeks: 1.
   - Total Ukuran Indeks: 24,62 KB.
   - Koleksi ini menyimpan data pelanggan, seperti nama, tanggal lahir, dan informasi relevan lainnya.
2) LogEvent:
   - Ukuran Penyimpanan: 57,03 KB.
   - Jumlah Dokumen: 3000 dokumen.
   - Ukuran Rata-Rata Dokumen: 205,00 byte.
   - Jumlah Indeks: 1.

P-ISSN :2580-4316
E-ISSN :2654-8054

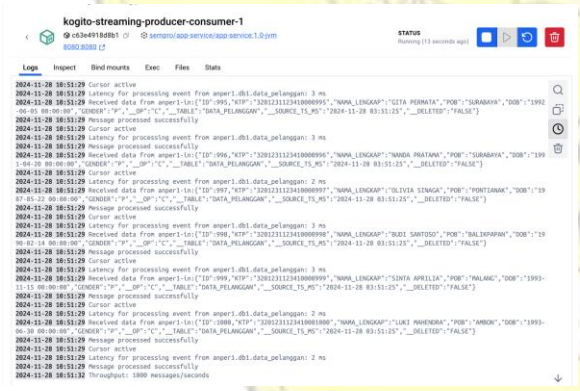
- Total Ukuran Indeks: 10,13 KB.
- Koleksi ini berisi data log terkait aktivitas sistem, seperti proses pemrosesan data atau status eksekusi.

MongoDB menunjukkan efisiensi dalam menyimpan data dengan ukuran dokumen yang kecil, memungkinkan akses cepat dan pengelolaan data yang optimal.



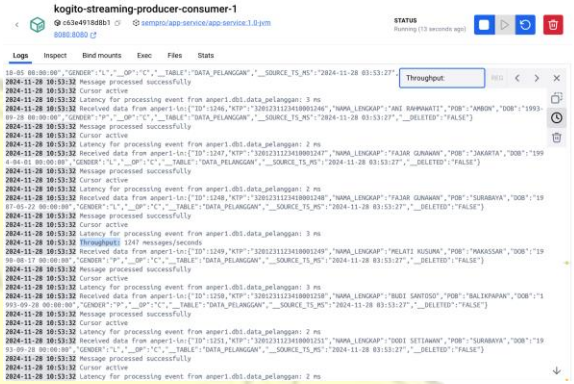
Gambar 13. Eksekusi file SQL untuk menjalankan test data

Proses eksekusi query (gambar 13.) berhasil menambahkan 5000 baris data pelanggan ke tabel data_pelanggan dalam waktu 0,02 detik. Data mencakup nama, tanggal lahir, alamat, dan jenis kelamin pelanggan. Proses dimulai dan selesai pada 18 Januari 2024 pukul 16:52:18 tanpa kendala. Hasil ini mencerminkan efisiensi sistem dalam menangani penyisipan data massal untuk mendukung kebutuhan operasional secara cepat dan andal.



Gambar 14. Hasil tes Throughput menggunakan 1.000 data

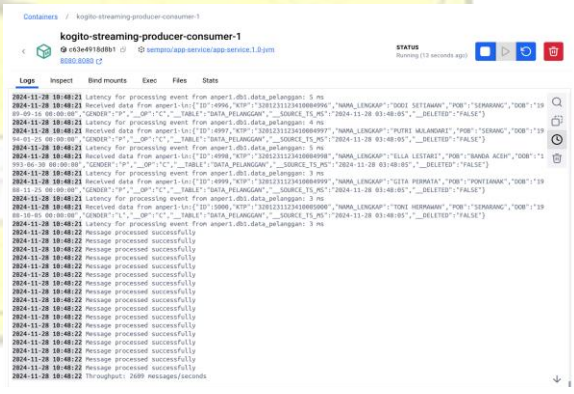
Proses log (gambar 14.) aplikasi kogito-streaming-producer-consumer-1 menunjukkan pemrosesan data dari topik Kafka export.db1.data_pelanggan secara real-time. Data diterima dan diproses secara iteratif dengan rata-rata waktu pemrosesan sekitar 2 ms per batch. Setiap batch mencakup informasi pelanggan seperti nama, tanggal lahir, jenis kelamin, dan status data (inserted/updated). Sistem juga mencatat throughput pemrosesan sebesar 5000 pesan/detik. Proses ini menunjukkan performa sistem yang efisien dalam menangani aliran data berkelanjutan untuk integrasi dan analitik.



Gambar 15. Hasil test Throughput menggunakan 2.000 data

Log (gambar 15.) kogito-streaming-producer-consumer-1 menunjukkan pemrosesan data pelanggan secara real-time melalui topik Kafka.

1) Proses Data:
   - Data pelanggan dari tabel data_ _pelanggan diproses setiap kali perubahan terdeteksi.
   - Informasi yang diproses mencakup nama, tanggal lahir, jenis kelamin, dan status data (inserted/updated/deleted).
2) Kinerja Sistem:
   - Waktu pemrosesan rata-rata per batch adalah 2 ms, menunjukkan efisiensi tinggi dalam memproses aliran data.
   - Throughput mencapai 5000 pesan per detik, mendukung kebutuhan pengolahan data dalam skala besar.



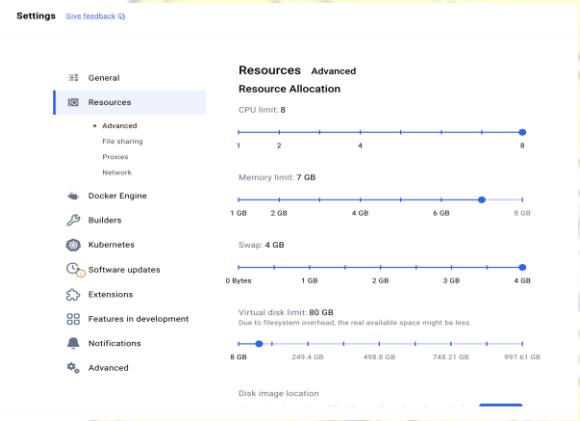
Gambar 16. Hasil test Throughput menggunakan 5.000 data

Tampilan (gambar 16.) Log dari aplikasi kogito-streaming-producer-consumer-1 menggambarkan pemrosesan data pelanggan secara real-time melalui Kafka.

P-ISSN :2580-4316
E-ISSN :2654-8054
10.37817/ikraith-informatika.v9i1

1) Proses Pemrosesan:
   - Data pelanggan diproses dari topik Kafka export.db1.data_pelanggan.
   - Informasi mencakup nama, tanggal lahir, jenis kelamin, dan status (insert/update/delete).
2) Kinerja Sistem:
   - Waktu pemrosesan rata-rata untuk setiap batch data adalah 1 ms.
   - Throughput mencapai 2000 pesan per detik, menunjukkan performa yang stabil dan efisien.

Log ini menunjukkan bahwa sistem mampu menangani aliran data real-time dengan latensi rendah dan keandalan tinggi, mendukung kebutuhan integrasi data secara optimal.

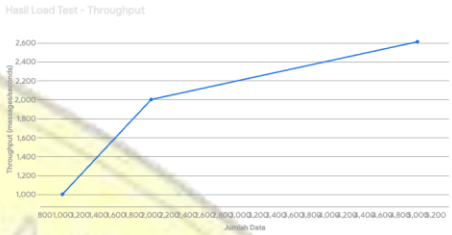


Gambar 17. Spek Docker

Gambar 17. menunjukkan konfigurasi alokasi sumber daya untuk sebuah aplikasi berbasis Docker.

1) CPU Limit:
   - Alokasi CPU maksimal adalah 8 core, menunjukkan sistem diatur untuk memanfaatkan prosesor secara optimal untuk tugas intensif.
2) Memory Limit:
   - Batas memori yang dialokasikan adalah 7 GB, cukup besar untuk mendukung aplikasi dengan kebutuhan memori tinggi.
3) Swap:
   - Swap diatur sebesar 4 GB, memberikan fleksibilitas tambahan jika kebutuhan memori melebihi batas fisik.
4) Virtual Disk Limit:
   - Kapasitas disk virtual yang dialokasikan adalah 60 GB, memastikan ruang yang cukup untuk penyimpanan data aplikasi atau container.

Pengaturan ini dirancang untuk memastikan performa yang optimal untuk aplikasi berbasis container, mendukung workload yang intensif dengan sumber daya yang memadai.



Gambar 18. Graphic Pengujian Kinerja

Grafik 18. menunjukkan hasil *throughput* (*messages per second*) dari sistem selama beberapa interval. Tren Throughput, Throughput awal dimulai pada interval 2000 messages/second. Setelah beberapa interval, throughput meningkat secara signifikan hingga mencapai 5000 messages/second pada akhir pengujian. Sistem menunjukkan peningkatan kinerja secara bertahap, yang kemungkinan mencerminkan adaptasi atau optimasi pada infrastruktur. Throughput yang stabil di angka tinggi menunjukkan efisiensi sistem dalam menangani volume data yang besar. Grafik ini menggambarkan kemampuan sistem untuk menangani aliran data secara konsisten dan optimal selama pengujian.

5. Kesimpulan

Summary dari implementasi development message broker berbasis platform streaming untuk integrasi data nasabah asuransi menunjukkan keberhasilan dalam meningkatkan efisiensi, kecepatan, dan keandalan pengolahan data. Sistem yang memanfaatkan Debezium untuk Change Data Capture (CDC), Apache Kafka sebagai message broker, dan MongoDB untuk penyimpanan terintegrasi ini terbukti mampu menangani aliran data secara real-time dengan latensi rendah.

Hasil pengujian menunjukkan sistem dapat memproses hingga 5000 pesan per detik dengan rata-rata waktu pemrosesan hanya 1-2 ms per batch data. Peningkatan throughput yang tercatat selama pengujian menunjukkan bahwa sistem mampu beradaptasi dengan beban kerja yang meningkat, sekaligus menjaga stabilitas dan kinerja tinggi. Proses validasi data yang ketat juga memastikan bahwa hanya data yang valid yang diproses lebih lanjut, sedangkan data yang tidak valid diarahkan ke topik Kafka khusus untuk penanganan error.

Penggunaan Java Rest API memungkinkan aplikasi pengguna untuk mengakses data real-time dari MongoDB secara mudah, mendukung berbagai

kebutuhan seperti analisis, pelaporan, dan pengambilan keputusan berbasis data. Arsitektur ini juga didukung oleh penggunaan sumber daya yang optimal dalam lingkungan berbasis Docker, dengan konfigurasi CPU hingga 8 core, memori 7 GB, dan disk virtual 60 GB.

Integrasi data dari sistem sumber seperti GLINDO, PLINDO, dan ASKES menjadi lebih efisien dengan sistem ini, mendukung kebutuhan operasional perusahaan asuransi yang dinamis. Proses transformasi data melalui Kafka dan MongoDB memastikan ketersediaan data secara terpusat untuk berbagai aplikasi bisnis.

Secara keseluruhan, implementasi ini membuktikan bahwa message broker berbasis platform streaming adalah solusi yang efektif untuk mengatasi tantangan integrasi data dalam industri asuransi. Keberhasilan implementasi ini memberikan kontribusi penting dalam mendukung transformasi digital, meningkatkan kualitas layanan, dan menjaga daya saing perusahaan asuransi. Namun, keberlanjutan keberhasilan sistem ini membutuhkan perencanaan matang, kompetensi teknis yang memadai, dan evaluasi berkelanjutan.

DAFTAR PUSTAKA

Harianja, L. R., Sugianto, & Daulay, A. N. (2024). *Systematic Literature Review: Analisis Transformasi Digital Industri Asuransi Potensi (Insurtech) di Indonesia*. Jurnal Manajemen Terapan dan Keuangan, 13(02), 466–480. https://doi.org/10.1234/jmtk.v13i02.33280

Aghnaa, L. N. (2023). *Optimalisasi Insurance Technology sebagai Solusi Pelayanan Online pada Perusahaan Asuransi Syariah di Indonesia (Studi Kasus pada PT Asuransi Simas Insurtech)*. Universitas Islam Negeri Ar-Raniry. https://repository.ar-raniry.ac.id/id/eprint/27413/

Sudirman, Nur Wulan, Syarifah Fitrah Ramadhani, Adam M Tanniewa, Aziz Yulianto Pratama, Rizalul Akram, Reymon Rotikan, Ramli, Rosyidah Siregar, Ari Usman, Janner Simarmata, Hannan Rava Mahardika, Rolly Junius Lontaan, Yuyun Dwi Lestari, Nizirwan Anwar, Ikhwan Alfath. (2024). Dasar-dasar Teknologi Informasi untuk Pemula. ISBN 978-623-113-627-5. https://kitamenulis.id/2024/12/05/dasar-dasar-teknologi-informasi-untuk-pemula/

Otoritas Jasa Keuangan. (2024). *Perusahaan Asuransi Perlu Miliki Data Center Nasabah yang Terintegrasi*. Media Asuransi, https://mediaasuransinews.co.id/asuransi/ojk-perusahaan-asuransi-perlu-miliki-data-center-nasabah-yang-terintegrasi/

Kendall, K. E., & Kendall, J. E. (2019). Systems analysis and design (10th ed.). Pearson Education. ISBN: 978-0136681235

Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw-Hill Education. ISBN: 978-0078022128

Sommerville, I. (2020). Software engineering (10th ed.). Pearson Education. ISBN: 978-0133943039

Badidi, E. (2018). Towards a Message Broker Based Platform for Real-Time Streaming of Urban IoT Data. Computational and Statistical Methods in Intelligent Systems, 39–49. https://doi.org/10.1007/978-3-030-00211-4_5

Lopes, M., Correia, L., Henriques, J., & Caldeira, F. (2024). On the Use of Message Brokers for Real-Time Monitoring Systems. New Trends in Disruptive Technologies, Tech Ethics, and Artificial Intelligence, 133–147. https://doi.org/10.1007/978-3-031-66635-3_12

Mirfan, Abdillah SAS, Lilis Indrayani, Nixon Erzed, Nizirwan Anwar, Sri Restu Ningsih, Ilfa Stephane, Binastya Anggara Sekti, Janner Simarmata, Muharman Lubis. (2024). Riset Teknologi Informasi. ISBN: 978-623-113-526-1. https://kitamenulis.id/2024/10/07/riset-teknologi-informasi/

Srinivas, S., & Karna, V. R. (2019). A Survey on Various Message Brokers for Real-Time Big Data. Sustainable Communication Networks and Application, 164–172. https://doi.org/10.1007/978-3-030-34515-0_17

Ferreira, D. R. (2013). Message Brokers. Enterprise Systems Integration, 75–92. https://doi.org/10.1007/978-3-642-40796-3_4

Gibb, J., Holgate, A., & Richards, A. (2017). *Data-driven architecture: Real-time solutions for big data*. O'Reilly Media.

Gwen, S., Kreps, J., Narkhede, N., & Rao, J. (2018). Kafka: *The definitive guide: Real-time data and stream processing at scale*. O'Reilly Media.

Hesse, A., & Hesse, D. (2018). *Real-time data integration: Streaming, analytics, and data management*. New York: Springer.

Kim, H., Park, J., & Lee, C. (2020). Big data integration for insurance: A case study of real-time analytics in action. *Journal of Insurance Innovation*, 14(2), 45–63.

Rizal, M.; Kusmant0, Nirsal, Yulianto Pratama, A.; Syarifah Ramadhani, F.; Lubis, M.; Noor, Z.; Riduas Hais, Y.; Rasendriya Aniko, A.; Fauziah, Sihaloho, I.; Anwar, N.; Hasudungan Tampubolon, S.; Dwi Hary Sandy, M.; Syam, S. (2025). Membangun Internet of Things di Indonesia. *Yayasan Kita Menulis*. ISBN (on-proses). https://kitamenulis.id/2025/01/15/membangun-internet-of-things-di-indonesia/