



Analisis dan Desain Migrasi Arsitektur Sistem Monolitik ke Mikroservis di PT. MALINDO: Pendekatan Konseptual

Ego Oktafanda¹, Nofri Wandi Al-Hafiz², Abdul Latif³, Firman Santosa⁴

^{1,4}Universitas Rokania, Rokan Hulu, Riau, Indonesia.

²Universitas Islam Kuantan Singingi, Riau, Indonesia.

³Universitas Bina Sarana Informatika, Jakarta, Indonesia

Informasi Artikel

Riwayat artikel:

Diterima 04 08, 2025

Direvisi 15 April 2025

Diterima 16 Mei 2025

Kata kunci:

Monolit

Layanan mikro

Skalabilitas

Transisi

ABSTRAK

Studi ini menganalisis transisi dari arsitektur monolitik ke arsitektur microservices dalam lingkungan perusahaan, khususnya di PT.MALINDO. Meskipun arsitektur monolitik pernah efektif, kini arsitektur tersebut menghadirkan beberapa tantangan, termasuk kompleksitas pemeliharaan, skalabilitas yang terbatas, dan seringnya terjadi deadlock. Analisis ini menyoroti kebutuhan yang semakin meningkat akan sistem yang lebih adaptif dan fleksibel untuk merespons kebutuhan bisnis yang dinamis. Dalam konteks ini, arsitektur microservices diusulkan sebagai solusi untuk mengatasi keterbatasan pendekatan monolitik. Microservices memecah aplikasi menjadi layanan-layanan kecil dan independen, sehingga memungkinkan pengembangan, penyebaran, dan pemeliharaan yang lebih mudah dan efisien. Dengan menggunakan komponen seperti API Gateway dan broker pesan seperti Apache Kafka, komunikasi antar layanan dapat dikelola lebih efektif, sehingga memastikan peningkatan kinerja dan ketahanan sistem. Temuan menunjukkan bahwa transisi ke arsitektur microservices dapat secara signifikan meningkatkan kinerja, fleksibilitas, dan skalabilitas sistem.

Hipotesis yang dihasilkan menunjukkan bahwa penerapan microservices akan menawarkan manfaat jangka panjang bagi PT.MALINDO, khususnya dalam mengatasi kompleksitas sistem dan meningkatkan daya tanggap terhadap kebutuhan bisnis yang terus berkembang. Rekomendasi mencakup strategi implementasi dan praktik manajemen perubahan yang diperlukan untuk memastikan transisi yang lancar dan berisiko rendah.

Artikel ini merupakan artikel akses terbuka di bawah lisensi [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/).



Penulis Korespondensi:

Ego Oktafanda

Jurusan Ilmu Komputer Universitas
Rokania Riau, Indonesia

Email:

egooktafanda1097@gmail.com © Penulis
2025

1. Pendahuluan

Di era digital yang berkembang pesat saat ini, adaptasi teknologi telah menjadi faktor penting bagi keberlanjutan dan keberhasilan organisasi. Sistem monolitik, yang dulunya menjadi dasar utama pengembangan perangkat lunak, kini menghadapi berbagai tantangan yang tidak dapat lagi diabaikan. Penelitian terbaru oleh Hatma Suryotrisongko (2017) menekankan masalah ini dan menyoroti kebutuhan mendesak untuk mengeksplorasi arsitektur alternatif seperti microservices untuk mengatasi kompleksitas dan keterbatasan sistem monolitik. Sistem informasi perusahaan umumnya dibangun menggunakan pendekatan monolitik, di mana seluruh aplikasi dikemas menjadi satu unit besar. Akibatnya, setiap modifikasi pada satu bagian basis kode dapat berdampak pada seluruh aplikasi.

dampak signifikan pada komponen lain. Namun, pendekatan tradisional ini semakin bergeser ke arah model terdistribusi, di mana aplikasi dibagi menjadi modul-modul yang lebih kecil dan khusus (kohesi tinggi) yang beroperasi secara independen (kopling longgar). Layanan-layanan ini biasanya dikembangkan dan diintegrasikan menggunakan endpoint API (Application Programming Interface) [2]. PT.MALINDO, sebuah perusahaan yang bergerak di bidang logistik, saat ini mengandalkan sistem monolitik untuk operasional dan manajemen datanya. Salah satu masalah utama yang dihadapi dalam pengaturan ini adalah seringnya terjadi deadlock dan timeout, yang menghambat respons sistem secara keseluruhan. Seiring pertumbuhan sistem, kompleksitas koordinasi dan pemeliharaan juga meningkat, sehingga membatasi ketangkasan perusahaan dalam menanggapi perubahan pasar secara efisien. Lebih lanjut, kemampuan pemantauan yang tidak memadai di banyak sistem monolitik PT.MALINDO menimbulkan risiko serius, karena menyulitkan untuk mendeteksi dan menyelesaikan masalah dengan cepat. Selain itu, meningkatnya permintaan konsumsi data menambah kompleksitas lebih lanjut pada arsitektur monolitik PT.MALINDO, yang berdampak negatif pada kinerja dan efisiensi pemrosesan data. Ketergantungan pada teknologi tertentu dan skalabilitas yang terbatas juga merupakan masalah penting yang perlu diatasi. Studi ini bertujuan untuk mengeksplorasi solusi potensial terhadap tantangan-tantangan tersebut melalui penerapan arsitektur microservices. Dengan memahami secara menyeluruh keterbatasan sistem monolitik dan keunggulan yang ditawarkan oleh microservices, PT.MALINDO dapat mengambil keputusan yang tepat untuk meningkatkan adaptabilitas, skalabilitas, dan efisiensi operasionalnya. Diharapkan studi ini akan memberikan wawasan yang berharga dan berkontribusi secara signifikan terhadap pengembangan sistem informasi modern yang berkelanjutan di PT.MALINDO.

2. Tinjauan pustaka

Arsitektur Microservices adalah pendekatan pengembangan perangkat lunak yang menyusun aplikasi sebagai kumpulan layanan yang terisolasi dan dapat diimplementasikan secara independen. Pendekatan ini mengatasi banyak tantangan yang dihadapi oleh sistem monolitik tradisional dengan menawarkan peningkatan fleksibilitas, skalabilitas, dan ketahanan terhadap perubahan. Arsitektur microservices dicirikan oleh jaringan layanan kecil dan otonom yang beroperasi secara independen dan berkomunikasi menggunakan protokol ringan seperti HTTP (Hypertext Transfer Protocol). Setiap layanan berjalan dalam prosesnya sendiri, dibangun di sekitar kemampuan bisnis yang berbeda, dan dapat diterapkan secara independen[3]. Layanan-layanan ini berfungsi secara independen, memungkinkan pengembangan, penerapan, dan modifikasi tanpa memengaruhi layanan lain. Modularitas ini memungkinkan pengembang untuk fokus membangun layanan-layanan kecil yang terdefinisi dengan baik, sehingga meningkatkan skalabilitas dan adaptabilitas sistem secara keseluruhan. Mikroservis telah menjadi gaya arsitektur yang populer untuk aplikasi berbasis data karena kemampuannya untuk memecah aplikasi kompleks menjadi layanan yang lebih kecil dan otonom. Pemecahan ini memfasilitasi skalabilitas, isolasi yang kuat, dan spesialisasi sistem data yang disesuaikan dengan beban kerja dan format data tertentu [4].

2.1. Keunggulan Microservices a. Skalabilitas

Sistem dapat

diskalakan secara horizontal dengan menambah atau mengurangi jumlah instance layanan tertentu, sehingga memungkinkan sistem untuk menangani lonjakan permintaan tanpa memengaruhi bagian lain dari sistem [5].

b. Fleksibilitas Pengembangan

Setiap layanan dapat dikembangkan secara independen, memungkinkan tim pengembangan yang berbeda untuk bekerja secara bersamaan tanpa saling mengganggu. Hal ini mempercepat siklus pengembangan dan rilis produk [6].

c. Ketahanan

terhadap Perubahan

Isolasi layanan memungkinkan perubahan dilakukan pada satu layanan tanpa berdampak pada seluruh sistem. Hal ini mendukung adaptasi yang lebih cepat dan efisien terhadap kebutuhan bisnis yang terus

berkembang [7]. d. Pemeliharaan yang Lebih Mudah

Pemeliharaan dan debugging disederhanakan karena ruang lingkup dan fungsionalitas setiap layanan terbatas. Peningkatan atau perbaikan dapat dilakukan pada layanan individual tanpa mengganggu seluruh sistem [8].

2.2. Kekurangan Microservices a. Kompleksitas

Manajemen

Meskipun menawarkan fleksibilitas, mengelola sejumlah besar layanan terdistribusi menimbulkan kompleksitas.

Koordinasi komunikasi antar layanan, pemantauan, dan kontrol versi memerlukan infrastruktur manajemen yang matang dan dirancang dengan baik [9]. b.

Overhead Jaringan

Membagi aplikasi menjadi layanan-layanan terpisah meningkatkan kebutuhan akan komunikasi antar layanan melalui jaringan. Dalam lingkungan terdistribusi skala besar, hal ini dapat menyebabkan beban jaringan yang signifikan dan berdampak pada kinerja sistem secara keseluruhan [6]. c. Pemeliharaan Infrastruktur

Migrasi ke arsitektur *microservices* dapat meningkatkan kebutuhan pemeliharaan infrastruktur.

Mengelola basis data terdistribusi, pengaturan konfigurasi, dan penanganan kesalahan menjadi lebih kompleks dan membutuhkan keahlian teknis yang lebih besar

[10]. d. Konsistensi Transaksi Lintas Layanan

Menangani transaksi yang mencakup beberapa layanan bisa jadi rumit. Memastikan konsistensi data di seluruh layanan ini membutuhkan desain yang cermat dan strategi implementasi yang kuat [6]. e. Tantangan

Penanganan Kesalahan

Kegagalan pada satu layanan dapat berdampak pada bagian lain dari sistem. Menerapkan mekanisme penanganan dan pemulihan kesalahan yang efektif menjadi lebih menantang dan membutuhkan strategi yang matang [10]. f. Tidak

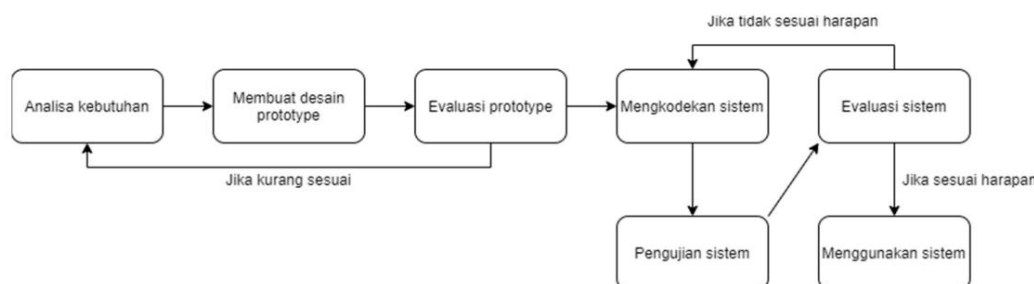
Cocok untuk Semua Aplikasi

Arsitektur *microservices* mungkin tidak ideal untuk semua jenis aplikasi. Aplikasi yang lebih kecil atau kurang kompleks mungkin tidak mendapatkan manfaat signifikan dari pendekatan ini, sementara biaya manajemen yang terkait dapat melebihi keuntungannya [6].

3. Metodologi Penelitian Dalam

pengembangan Arsitektur *Microservices*, studi ini mengadopsi model Siklus Hidup Pengembangan Perangkat Lunak (SDLC) sebagai metodologi penelitian utama. SDLC adalah proses terstruktur yang digunakan untuk membuat dan memodifikasi sistem perangkat lunak, yang mencakup beberapa fase termasuk perencanaan, analisis, desain, implementasi, pengujian, dan pemeliharaan. Dengan mengikuti model SDLC, pengembang dapat memastikan bahwa pengembangan sistem dilakukan secara terorganisir dan efisien, sekaligus memenuhi tujuan bisnis dan persyaratan pengguna. Untuk fase desain sistem, penulis merekomendasikan penggunaan metode *prototyping*, berdasarkan pengamatan bahwa perubahan yang sering terjadi selama pengembangan perangkat lunak dapat memengaruhi stabilitas sistem dan menyebabkan penundaan yang signifikan. Pendekatan *prototyping* dipilih karena melibatkan evaluasi prototipe sebelum tahap pengkodean sebenarnya. Dari pengamatan ini, penulis menekankan bahwa fase analisis harus diselesaikan secara menyeluruh sebelum melanjutkan ke implementasi. Keuntungan utama dari metode *prototyping* terletak pada tahap evaluasi ini, di mana kesesuaian desain dapat dinilai dan divalidasi oleh pemangku kepentingan. Jika ditemukan ketidaksesuaian atau penyesuaian yang diperlukan, analisis yang lebih terfokus dan mendalam dapat dilakukan sebelum memasuki fase pengkodean. Pendekatan ini diharapkan dapat mengurangi risiko perubahan yang tidak terduga selama pengembangan. Selain itu, penggunaan metode pembuatan prototipe menawarkan manfaat keterlibatan pengguna sejak dini, memungkinkan umpan balik dan kolaborasi pada tahap awal proses.

Oleh karena itu, penulis berpendapat bahwa pendekatan ini dapat mempercepat pengembangan dan meningkatkan efisiensi secara keseluruhan, sekaligus menyoroti pentingnya analisis menyeluruh sebelum fase pengkodean dimulai.



Gambar 1 Prototipe SDLC

4. Hasil dan Diskusi Analisis Persyaratan

Fungsional dilakukan untuk memahami kemampuan sistem yang dirancang dan bagaimana sistem tersebut dapat mengatasi potensi masalah. Ini melibatkan fase awal pengamatan dan elaborasi proses bisnis, yang memungkinkan pemilihan sistem untuk mengantisipasi ukuran aplikasi, volume pengguna, dan lalu lintas yang diharapkan. Dalam konteks ini, memilih arsitektur *microservices* dianggap sebagai keputusan yang tepat.

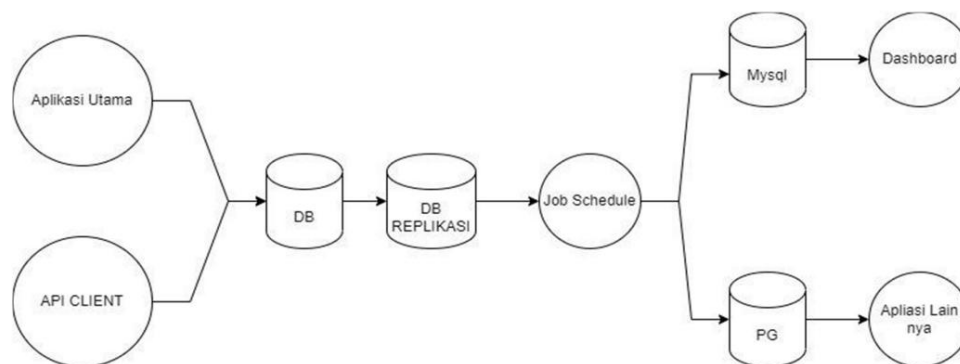
Arsitektur *microservices* memecah aplikasi menjadi layanan-layanan yang lebih kecil dan mudah dikelola, sehingga sistem lebih mudah dipahami dan dipelihara. Berbeda dengan pendekatan monolitik yang menghasilkan paket aplikasi yang besar dan kompleks, *microservices* menawarkan fleksibilitas yang lebih besar dalam pengembangan, pemeliharaan, dan penerapan.

Ketika ada kebutuhan untuk mengadopsi teknologi atau bahasa pemrograman baru, *microservices* memungkinkan proses yang lebih lancar.

Transisi lebih mudah dilakukan karena setiap layanan dapat dikembangkan dan dikelola secara independen. Selama fase pemeliharaan, tim khusus seringkali diperlukan untuk mengelola perubahan yang berasal dari sistem eksternal, seperti pembaruan sistem operasi atau pergeseran teknologi. Dalam aplikasi yang menggunakan arsitektur *microservices*, perubahan tersebut lebih mudah diimplementasikan karena ukurannya yang lebih kecil dan sifat modular dari layanan tersebut. Hal ini berbeda dengan sistem monolitik, di mana pembaruan tersebut mungkin memerlukan pembangunan ulang seluruh aplikasi untuk mengakomodasi teknologi atau bahasa pemrograman baru.

4.1. Arsitektur Sistem yang Ada Arsitektur

monolitik saat ini dikembangkan sebagai satu unit besar menggunakan kerangka kerja *Serenity .NET*. Server beroperasi pada sistem operasi berbasis *Windows*, dan semua data disimpan dalam satu basis data *SQL Server*. Berbagai komponen dan modul aplikasi berinteraksi langsung dengan basis data terpusat ini untuk melakukan operasi seperti menyimpan, memperbarui, dan mengambil data. Aplikasi utama berfungsi sebagai koordinator pusat, mengelola aktivitas sistem dan memastikan integritas data di semua modul. Modul lain—seperti manajemen pengguna, pemrosesan transaksi, dan pelaporan—juga terhubung langsung ke basis data yang sama, mengandalkan data yang tersimpan untuk mendukung tugas operasional. Hal ini menciptakan tingkat saling ketergantungan yang tinggi antar komponen, di mana perubahan di satu bagian sistem dapat berdampak luas pada bagian lainnya.



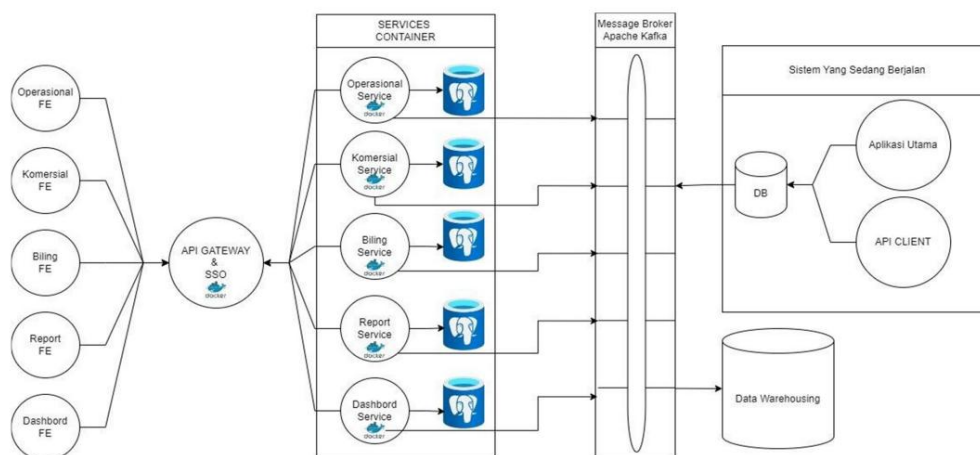
Gambar 2 Alur Sistem Saat Ini

Seperti yang ditunjukkan pada Gambar 2, sistem monolitik saat ini bergantung pada satu basis data pusat yang digunakan bersama oleh semua aplikasi. Baik aplikasi utama maupun klien API berinteraksi langsung dengan basis data utama (DB), yang kemudian mereplikasi data ke basis data sekunder (Replikasi DB). Penjadwal tugas mengelola proses replikasi untuk memastikan data di DB utama diperbarui secara berkala di DB yang direplikasi. Dari DB yang direplikasi, data didistribusikan ke dua basis data yang berbeda: *MySQL* dan *PostgreSQL (PG)*, masing-masing melayani kebutuhan aplikasi yang berbeda. *MySQL* digunakan untuk menyediakan data ke aplikasi dasbor, sementara *PostgreSQL* mendukung aplikasi tambahan lainnya. Arsitektur ini memberikan beban yang cukup besar pada basis data utama, karena semua permintaan data dan tugas replikasi melewati satu titik pusat. Pengaturan ini menyoroti keterbatasan signifikan dari arsitektur monolitik, terutama pada skala besar: ketergantungan pada satu basis data menciptakan hambatan kinerja, yang berdampak negatif pada efisiensi dan responsivitas sistem secara keseluruhan.

4.2. Arsitektur Sistem yang Diusulkan Arsitektur ini

menekankan desain modular dan terisolasi, terdiri dari empat lapisan berbeda yang mendukung pengembangan layanan independen. Dengan mengadopsi pendekatan *microservices*, sistem diharapkan lebih responsif terhadap perubahan, menawarkan skalabilitas yang lebih besar, dan mempercepat proses pengembangan. Perencanaan desain juga mendorong pertimbangan menyeluruh tentang bagaimana setiap *microservice* berinteraksi dalam ekosistem, memastikan integrasi yang lancar dan koheren di seluruh sistem. Penggunaan *API Gateway*, *service container*, dan *message broker (Apache Kafka)* memainkan peran kunci dalam memfasilitasi komunikasi dan aliran data yang efisien antar komponen. Dengan desain ini, arsitektur yang diusulkan bertujuan untuk memaksimalkan efisiensi, meningkatkan ketahanan sistem, dan memungkinkan adaptasi yang lebih cepat terhadap kebutuhan bisnis yang terus berkembang dan kemajuan teknologi di masa mendatang.

Arsitektur sistem yang diusulkan terstruktur menjadi beberapa lapisan, seperti yang diilustrasikan pada Gambar 3.



Gambar 3 Arsitektur Sistem yang Diusulkan

4.2.1. Lapisan Frontend

Pemisahan antara lapisan antarmuka pengguna (frontend) dan logika bisnis telah menjadi pendekatan arsitektur yang semakin banyak diadopsi dalam pengembangan sistem modern. Berbagai penelitian telah menunjukkan bahwa pemisahan ini tidak hanya meningkatkan efisiensi pengembangan perangkat lunak tetapi juga mendukung prinsip modularitas dan skalabilitas [11]. Dalam arsitektur berbasis microservices, frontend memainkan peran penting sebagai antarmuka visual untuk mengakses layanan backend terdistribusi, memungkinkan pengguna untuk berinteraksi dengan fungsionalitas yang kompleks melalui antarmuka yang disederhanakan dan terfokus. Dari sudut pandang teknis, mengisolasi lapisan frontend memungkinkan pengembang untuk menerapkan prinsip desain berpusat pada pengguna tanpa dibebani oleh kompleksitas aliran data backend. Pemisahan ini lebih lanjut didukung oleh penggunaan API Gateway dan mekanisme Single Sign-On (SSO), yang mengkonsolidasikan kontrol akses di berbagai layanan backend melalui titik masuk terpadu. Penelitian dalam manajemen kompleksitas sistem menunjukkan bahwa arsitektur berlapis ini tidak hanya menyederhanakan debugging dan pengujian tetapi juga meningkatkan ketangkasan pengembangan dengan memungkinkan tim untuk bekerja secara paralel di berbagai lapisan [12]. Selain itu, penggunaan frontend terdesentralisasi yang selaras dengan domain layanan seperti Operational FE, Commercial FE, dan Billing FE sangat cocok untuk sistem berskala besar yang terus berkembang, di mana lapisan presentasi dapat disesuaikan dengan logika bisnis spesifik dari setiap domain. Dengan demikian, frontend menjadi lebih dari sekadar antarmuka visual; ia berfungsi sebagai komponen strategis yang mendukung transformasi digital, mendorong praktik pengembangan yang efisien, dan memberikan pengalaman pengguna yang lebih intuitif.

4.2.2. Lapisan API Gateway dan Single Sign-On (SSO) Implementasi

Kong Gateway dan sistem Single Sign-On (SSO) memainkan peran penting dalam membangun arsitektur sistem yang aman, efisien, dan terintegrasi. Kedua komponen ini secara khusus digunakan untuk menyederhanakan interaksi pengguna di berbagai layanan sekaligus meningkatkan keamanan akses. Kong Gateway, yang dipilih sebagai solusi API gateway dalam studi ini, bertindak sebagai pengontrol pusat untuk lalu lintas komunikasi antar layanan. Fungsionalitasnya meluas melampaui sekadar perutean permintaan hingga mencakup agregasi data, manajemen beban, dan penerapan kebijakan otentikasi dan otorisasi yang konsisten di seluruh sistem. Sementara itu, integrasi sistem SSO dalam desain sistem memungkinkan pengguna untuk melakukan autentikasi sekali dan mendapatkan akses ke berbagai layanan tanpa perlu login berulang. Pendekatan ini secara signifikan meningkatkan pengalaman pengguna dengan mengurangi hambatan login, sekaligus memperkuat keamanan melalui manajemen kredensial terpusat. Hal ini meminimalkan risiko yang terkait dengan beberapa kali login dan memastikan konsistensi kebijakan di seluruh sistem. Akibatnya, sistem menjadi lebih aman dan ramah pengguna, dua atribut penting untuk pengembangan layanan digital yang kompleks dan terdistribusi. Secara keseluruhan, adopsi Kong Gateway dan SSO dalam lingkup penelitian ini menunjukkan dampak signifikan pada efisiensi manajemen layanan, peningkatan keamanan sistem, dan integrasi layanan yang lancar. Teknologi ini berkontribusi pada pembentukan kontrol akses terpusat, arsitektur sistem modular, dan alur kerja yang terstandarisasi.

Hal ini menegaskan bahwa pemilihan infrastruktur yang tepat pada lapisan integrasi dapat sangat memengaruhi kinerja, skalabilitas, dan keandalan sistem dalam konteks penelitian ini.

4.2.3. Lapisan Layanan

Lapisan layanan mewakili struktur arsitektur yang membagi aplikasi menjadi komponen-komponen yang independen namun saling terhubung, semuanya dihubungkan melalui API Gateway. Dalam pendekatan ini, satu aplikasi dipecah menjadi beberapa modul mandiri, yang masing-masing dikelola oleh tim atau divisi pengembangan yang terpisah. Layanan independen ini dirancang dengan tanggung jawab yang jelas dan mampu berfungsi secara otonom, tanpa memerlukan ketergantungan langsung pada modul lain. API Gateway memainkan peran penting dalam arsitektur ini, berfungsi sebagai titik masuk terpusat yang menangani permintaan dan respons di seluruh layanan terdistribusi. Ia bertindak sebagai antarmuka terpadu yang menyederhanakan dan merampingkan komunikasi antar berbagai layanan, memungkinkan pertukaran data yang efisien dan terkoordinasi dengan baik. Dengan mengelola lapisan integrasi melalui gateway standar, sistem mengurangi kompleksitas, meningkatkan efisiensi operasional, dan mempertahankan kontrol yang lebih baik atas interaksi antar layanan. Pengaturan arsitektur ini meningkatkan skalabilitas, fleksibilitas, dan pemeliharaan, karena setiap layanan dapat dikembangkan, diterapkan, dan dipelihara secara independen. Ini mendorong lingkungan adaptif yang memungkinkan tim pengembang untuk merespons dengan cepat dan efektif terhadap perubahan dalam konteks bisnis yang dinamis. Sementara lapisan pertama—lapisan frontend—biasanya mengikuti rekomendasi arsitektur tertentu, lapisan layanan mempertahankan fleksibilitas untuk mengadopsi paradigma Model-View-Controller (MVC) bila sesuai. Bahkan tanpa secara ketat mengikuti resep lapisan frontend, lapisan layanan masih dapat mendukung prinsip-prinsip MVC. Hal ini memungkinkan pemisahan yang jelas antara manajemen data (Model), logika presentasi (View), dan alur kontrol (Controller). Pada akhirnya, lapisan layanan menyediakan lingkungan terbuka di mana pengembang dapat menerapkan pola desain yang efektif seperti MVC berdasarkan kebutuhan dan karakteristik spesifik dari setiap layanan. Fleksibilitas ini mendukung pendekatan pengembangan yang disesuaikan tanpa bergantung pada lapisan frontend sebagai prasyarat.

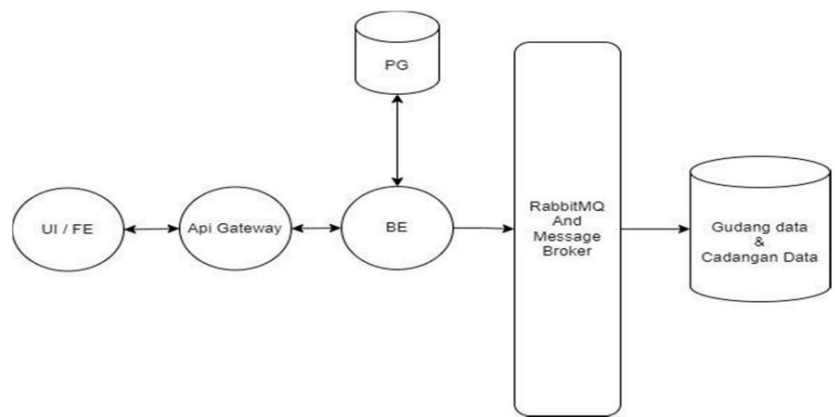
4.2.4. Broker Pesan Lapisan Apache

Kafka berperan sebagai perantara komunikasi penting dalam arsitektur microservices, memungkinkan interaksi yang efisien dan terstruktur di antara berbagai layanan dan sistem. Peran utamanya berpusat pada memfasilitasi komunikasi antar layanan—memungkinkan layanan yang berbeda seperti Layanan Operasional, Layanan Komersial, Layanan Penagihan, Layanan Pelaporan, dan Layanan Dasbor untuk bertukar pesan tanpa ketergantungan langsung satu sama lain. Kafka mendukung prinsip kopling longgar, di mana layanan dirancang untuk independen, sehingga memudahkan pengelolaan, penskalaan, dan evolusi sistem. Setiap layanan dapat menghasilkan atau mengonsumsi data melalui Kafka tanpa perlu mengetahui cara kerja internal layanan lain yang berkomunikasi dengannya. Pemisahan ini meningkatkan modularitas dan menyederhanakan pemeliharaan. Salah satu kemampuan Kafka yang paling ampuh adalah memungkinkan streaming data real-time. Ini memungkinkan aliran data yang lancar dari berbagai sumber layanan ke tujuan target, baik itu sistem operasional langsung atau penyimpanan jangka panjang seperti gudang data. Transmisi real-time ini memastikan bahwa wawasan dan tindakan berdasarkan data dapat terjadi dengan penundaan minimal, meningkatkan responsivitas dan efisiensi sistem. Kafka juga memberikan kontribusi signifikan terhadap skalabilitas dan ketahanan sistem. Ia dapat menangani volume data yang tinggi sambil memastikan pengiriman pesan yang andal, bahkan jika terjadi kegagalan layanan. Desain toleransi kesalahan ini mendukung perilaku sistem yang konsisten dan dapat diandalkan, yang sangat penting untuk arsitektur terdistribusi. Selain itu, Kafka membantu memisahkan produsen dan konsumen data, menghilangkan kebutuhan akan koneksi langsung antar layanan. Lapisan abstraksi ini tidak hanya menyederhanakan komunikasi tetapi juga membuat pengembangan dan penyelarasan layanan lebih fleksibel dan lincah. Sebagai pengelola antrian pesan, Kafka menjamin bahwa pesan dikirimkan secara teratur dan aman antar layanan. Ia mengatur aliran data, memastikan bahwa setiap pesan mencapai tujuan yang dimaksud dengan benar dan andal. Terakhir, Kafka memfasilitasi integrasi dengan sistem lama. Ia memungkinkan data dari aplikasi tradisional dan klien eksternal untuk dialihkan melalui Kafka ke layanan modern yang dikontainerisasi—tanpa perlu modifikasi besar pada infrastruktur yang ada. Ini memastikan interoperabilitas yang lancar antara komponen lama dan modern. Singkatnya, Apache Kafka adalah landasan arsitektur microservices. Hal ini memungkinkan komunikasi yang efisien, mempercepat pemrosesan data, dan meningkatkan modularitas, fleksibilitas, dan skalabilitas sistem.

4.2.5. Lapisan Data Warehouse

Meskipun dianggap sebagai lapisan opsional, Lapisan Data Warehouse memainkan peran penting dalam meningkatkan proses analisis data dan mendukung replikasi data. Lapisan ini berfungsi sebagai komponen strategis yang membantu meminimalkan risiko kehilangan data ketika terjadi gangguan layanan yang tidak terduga. Dengan mengintegrasikan Lapisan Data Warehouse, data dapat diakses dan dianalisis lebih efisien, memungkinkan pengambilan keputusan yang lebih

Selain itu, fungsionalitas replikasi data yang disediakan oleh lapisan ini berkontribusi pada keandalan sistem. Jika terjadi kegagalan layanan, data yang direplikasi tetap tersedia, memastikan kontinuitas dan integritas informasi penting. Oleh karena itu, meskipun tidak wajib, Lapisan Gudang Data menawarkan keunggulan strategis dalam mengelola layanan kompleks dengan mengoptimalkan alur kerja data dan menyediakan jaring pengaman untuk perlindungan data.



Gambar 4 Fragmen Arsitektur

Gambar 4 mengilustrasikan uraian sederhana dari arsitektur sistem yang diusulkan, menunjukkan alur proses yang dimulai dari komponen terkecil. Dalam diagram ini, Antarmuka Pengguna (UI) atau Front-End (FE) berinteraksi langsung dengan API Gateway, yang berfungsi sebagai titik masuk untuk mengelola permintaan masuk dan mengarahkannya ke Back-End (BE). Back-end berkomunikasi dengan basis data PostgreSQL (PG) untuk menangani operasi data inti. Secara paralel, RabbitMQ, yang bertindak sebagai broker pesan, mengelola komunikasi asinkron di berbagai komponen sistem. Hal ini memastikan bahwa sistem tetap terukur dan responsif, bahkan dalam kondisi beban tinggi. Data yang dihasilkan dan diproses melalui arsitektur ini kemudian disimpan di Data Warehouse. Penyimpanan terpusat ini mendukung analitik lebih lanjut dan bertindak sebagai mekanisme pencadangan yang andal, berkontribusi pada kekokohan dan efisiensi sistem secara keseluruhan.

4.3. Deskripsi Teknologi yang Digunakan

Deskripsi Tabel 1

Barang-barang	Keterangan	Total
Server OS	Linux Ubuntu	3
Manajemen kontainer	Kubernetes	3
Kontainer Isolasi	buruh pelabuhan	21
Melayani	Layanan aplikasi	8
Frontend	ReactJs (Typescript)	
Backend	NodeJs	
Basis data	PostgreSQL & MySQL	
Broker Pesan	Apache Kafka	1
Gerbang API	Gerbang Kong	
Pengembang Datawarehouse &	Postgresql	
Replikasi	2 frontend / 2 backend	4

Bagian ini menyajikan daftar teknologi yang diimplementasikan dalam studi ini, mencakup komponen inti yang mendukung arsitektur sistem berbasis microservices. Pemilihan teknologi ini dipandu oleh persyaratan sistem untuk skalabilitas, manajemen layanan yang efisien, serta keamanan dan keandalan dalam komunikasi antar komponen. Sistem ini dihosting pada tiga server Linux Ubuntu, yang menawarkan lingkungan yang stabil dan sangat kompatibel untuk alat pengembangan berbasis kontainer. Untuk orkestrasi kontainer,

Kubernetes digunakan, memungkinkan penyebaran dan pengelolaan layanan secara terpusat dan otomatis. Docker digunakan untuk mengisolasi layanan, dengan total 21 kontainer aktif, masing-masing mewakili berbagai komponen layanan mikro dalam sistem. Layanan inti diimplementasikan sebagai bagian dari lapisan layanan aplikasi, yang terdiri dari delapan unit, termasuk frontend, backend, dan kontainer layanan. Antarmuka frontend dikembangkan menggunakan ReactJS dengan TypeScript, sedangkan backend menggunakan NodeJS, mendukung kinerja tinggi dan pemrosesan asinkron. Lapisan basis data menggabungkan PostgreSQL dan MySQL untuk memungkinkan replikasi dan mendukung kebutuhan analitik data. Untuk memfasilitasi komunikasi asinkron antar layanan, Apache Kafka digunakan sebagai broker pesan. Kong Gateway menangani manajemen permintaan API dan terintegrasi dengan mulus dengan sistem Single Sign-On (SSO) untuk otentikasi terpadu. Untuk penyimpanan dan replikasi data jangka panjang, PostgreSQL berfungsi sebagai fondasi solusi gudang data. Tim pengembang terdiri dari dua pengembang frontend dan dua pengembang backend, sehingga total ada empat pengembang aktif yang bertanggung jawab untuk m

5. Hasil, Kesimpulan, dan Rekomendasi Hasil penelitian

yang dilakukan didasarkan pada proses desain, implementasi, dan evaluasi sistem arsitektur microservices pada studi kasus PT. MALINDO. Analisis dilakukan untuk menguji hipotesis yang telah ditetapkan sebelumnya, mengevaluasi dampak pada kinerja dan efisiensi sistem, serta menilai keberhasilan transformasi arsitektur dari struktur monolitik ke microservices. Kesimpulan diambil dari temuan yang diperoleh, dan rekomendasi diberikan sebagai panduan untuk langkah selanjutnya dalam pengembangan sistem.

5.1. Hipotesis Studi

ini didasarkan pada hipotesis bahwa penerapan arsitektur microservices dapat mengatasi keterbatasan sistem monolitik yang saat ini digunakan oleh PT.MALINDO, khususnya dalam hal kinerja, skalabilitas, dan fleksibilitas pengembangan. Hipotesis utama yang diuji meliputi: a. Bahwa dengan menerapkan kontainerisasi menggunakan Docker, setiap layanan dapat diisolasi dan dikonfigurasi dengan sumber daya khusus, sehingga mencegah terjadinya timeout sistem dan penurunan kinerja di seluruh layanan bahkan ketika dihosting pada server fisik yang sama. b. Bahwa penggunaan alat dan platform sumber terbuka termasuk ReactJS, NodeJS, Kong Gateway, Apache Kafka, PostgreSQL, dan Kubernetes—tidak hanya meningkatkan fleksibilitas dan modularitas tetapi juga menawarkan manfaat ekonomi karena ketersediaan pustaka, plugin, dan dukungan komunitas yang kuat. c. Bahwa pemisahan lapisan frontend, backend, dan logika layanan memungkinkan proses pengembangan dan pemeliharaan yang lebih terfokus dan terstruktur, yang pada gilirannya menyederhanakan upaya debugging dan penskalaan.

5.2. Temuan dan Kesimpulan

Berdasarkan desain arsitektur, hasil simulasi, dan literatur pendukung, penelitian ini menyimpulkan bahwa transisi ke arsitektur microservices menawarkan keuntungan substansial dibandingkan pendekatan monolitik, terutama dalam konteks kebutuhan operasional PT.MALINDO. Temuan utama meliputi: a. Dekomposisi layanan menjadi unit-unit independen berhasil mengurangi hambatan sebelumnya dan potensi kebuntuan yang ditemui dalam sistem monolitik.

b. Dengan penerapan dan orkestrasi berbasis kontainer, layanan individual kini dapat dikembangkan, diuji, dan diimplementasikan secara independen tanpa mengganggu stabilitas sistem secara keseluruhan.

c. Integrasi Kong Gateway memungkinkan manajemen permintaan API yang terpusat dan aman, sementara penyertaan sistem Single Sign-On (SSO) meningkatkan pengalaman pengguna dengan menyediakan kontrol akses yang lancar dan konsisten.

d. Penerapan Apache Kafka secara signifikan meningkatkan komunikasi layanan asinkron, khususnya dalam menangani data transaksional dan alur kerja replikasi.

Proses pengembangan mendapat manfaat dari tim-tim khusus yang berfokus pada bidang tertentu, sehingga memungkinkan pembagian tugas yang lebih efektif dan keselarasan yang lebih jelas dengan logika bisnis di seluruh layanan.

Secara keseluruhan, pendekatan microservices telah terbukti meningkatkan ketahanan sistem, efisiensi pengembangan, kemudahan integrasi, dan skalabilitas layanan, yang semuanya penting dalam mendukung tujuan transformasi digital organisasi.

5.3. Rekomendasi

Berdasarkan temuan penelitian ini, disarankan agar migrasi ke microservices dilakukan secara bertahap dan strategis, dengan mempertimbangkan infrastruktur yang ada dan kesiapan sumber daya internal. Saran strategis berikut diajukan: a. Melakukan evaluasi komprehensif terhadap infrastruktur yang ada dan menyelaraskannya dengan persyaratan microservices, khususnya dalam hal orkestrasi kontainer, perantara pesan, dan sinkronisasi data.

- b. Membentuk tim pengembangan khusus bidang dan berinvestasi dalam peningkatan keterampilan mereka dengan pelatihan teknis tentang alat-alat seperti Docker, Kubernetes, dan Kong Gateway.
- c. Menerapkan strategi DevSecOps untuk memastikan keamanan, otomatisasi, dan keberlanjutan jangka panjang di seluruh pipeline pengembangan dan penerapan.
- d. Mengadopsi kerangka kerja observabilitas yang kuat, termasuk pemantauan waktu nyata, pencatatan terpusat, dan pelacakan terdistribusi, untuk mendapatkan wawasan operasional yang lebih mendalam.
- e. Pastikan keterlibatan aktif para pemangku kepentingan dan dorong kolaborasi tim lintas fungsi, karena keberhasilan migrasi tidak hanya bergantung pada teknologi tetapi juga pada budaya organisasi dan manajemen perubahan.

Dengan pendekatan yang komprehensif dan konsisten, migrasi ke arsitektur microservices dapat menjadi fondasi yang kuat untuk keberlanjutan digital dan kesuksesan jangka panjang PT.MALINDO di lingkungan bisnis yang berkembang pesat saat ini.

Referensi

- [1] H. Suryotrisongko, "Arsitektur microservice untuk ketahanan sistem informasi," SISFO, vol. 6, tidak. 2, hal.6, 2017.
- [2] G. Munawar dan A. Hodijah, "Analisis model arsitektur microservice pada sistem informasi DPLK," Sinkron: Jurnal dan Penelitian Teknik Informatika, vol. 3, tidak. 1, hal.232–238, 2018.
- [3] MG de Almeida dan ED Canedo, "Autentikasi dan otorisasi dalam arsitektur microservices: Tinjauan literatur sistematis," Applied Sciences, vol. 12, no. 6, hlm. 3023, 2022.
- [4] R. Laigner, Y. Zhou, MAV Salles, Y. Liu, dan M. Kalinowski, "Manajemen data dalam layanan mikro: Kondisi praktik, tantangan, dan arah penelitian," arXiv preprint arXiv:2103.00170, 2021.
- [5] M. Kleppmann, Mendesain Aplikasi Intensif Data: Ide-ide Besar di Balik Aplikasi yang Andal, Terukur, dan Sistem yang Dapat Dipelihara, O'Reilly Media, 2017.
- [6] S. Newman, Membangun Layanan Mikro: Merancang Sistem yang Lebih Detail, edisi ke-2, O'Reilly Media, 2021.
- [7] M. Nygard, Release It!: Design and Deploy Production-Ready Software, edisi ke-2, Pragmatic Rak Buku, 2018.
- [8] O. Greevy, S. Ducasse, dan T. Girba, "Menganalisis evolusi perangkat lunak melalui tampilan fitur," Jurnal Pemeliharaan dan Evolusi Perangkat Lunak: Penelitian dan Praktik, vol. 18, no. 6, hlm. 425–456, 2005.
- [9] C. Richardson, Pola Layanan Mikro: Dengan Contoh dalam Java, Simon and Schuster, 2018.
- [10] VF Pacheco, Pola dan Praktik Terbaik Microservice: Jelajahi Pola Seperti CQRS dan Event Sourcing untuk Membuat Microservice yang Dapat Diukur, Dipelihara, dan Diuji, Packt Publishing, 2018.
- [11] M. Fowler, "Layanan mikro," MartinFowler.com, 2014. <https://martinfowler.com/articles/microservices.html> [On line]. Tersedia:
- [12] S. Newman, Membangun Layanan Mikro, O'Reilly Media, 2021.
- [13] Nofri Wandu Al-Hafiz, Helpi Nopriandi, dan Harianja, "Rancangan Alat Ukur Intensitas Curah Hujan "Instrumen yang Menggunakan Mikrokontroler Berbasis IoT", JTOS, vol. 7, no. 2, hlm. 202 - 211, Des. 2024.
- [14] NW Al-Hafiz dan H. Harianja, "Desain Kontrol Pemberian Makan Kucing Otomatis Berbasis Internet of Things Perangkat (IoT)", Mandiri, vol. 13, no. 1, hlm. 161–169, Juli 2024.
- [15] PutriD. dan Al-HafizN., "SISTEM INFORMASI SURAT KETERANGAN GANTI RUGI TANAH PADA KECAMATAN KUANTAN TENGAH MENGGUNAKAN WEBGIS", Biner : Jurnal Ilmiah Informatika dan Komputer, vol. 2, tidak. 2, hlm.112-121, Juli 2023.
- [16] H. Harianja, NW Al-Hafiz, dan J. Jasri, "Analisis Data Mahasiswa Teknik Informatika Universitas Islam Kuantan Singingi", JTOS, vol. 6, tidak. 1, hal. 23 - 30, Januari 2023.

-
- [17] Siregar, M., dan N. Al-Hafiz. "Perancangan Cloud Computer Untuk Mendukung Server Sistem Informasi Independen Universitas Islam Kuantan Singingi." *Jurnal Penelitian Sistem Informasi (JOSH)* 3.2 (2022)
- [18] A. Apri Denta, H. Nopriandi, dan E. Erlinda, "Perancangan Sistem Informasi dan Pencapaian Kinerja Kantor Tenaga Kerja Kabupaten Kuantan Singingi", *JTOS*, vol. 7, no. 1, hlm. 01 - 09, Nov. 2024.
- [19] A. AP Putra, E. Erlinda, dan M. Yusufahmi, "Sistem Penjualan di Bisnis Telepon Seluler Endocell Menggunakan Metode CRM (Customer Relationship Management) di Desa Kompe Berangin, Kecamatan Cerenti", *JTOS*, vol. 7, no. 1, hlm. 10 - 21, Nov. 2024.
- [20] D. Setiawan, F. Haswan, dan J. Jasri, "Desain Aplikasi Penjadwalan Bel Sekolah Berdasarkan Arduino Uno di MTs Babussalam Simandolak", *JTOS*, vol.7, no.1, pp.22 - 30, Juli 2024.
- [21] D. Juniarti, A. Aprizal, dan S. Chairani, "Sistem Informasi Analisis Tren Penyakit di Wilayah UPTD Kesehatan Cerenti", *JTOS*, vol. 7, tidak. 1, hlm. 31 - 43, Juni 2024.
- [22] F. Restuadi, H. Nopriandi, and A. Aprizal, "ANALISIS QOS JARINGAN INTERNET FAKULTAS TEKNIK UNIVERSITAS ISLAM KUANTAN SINGINGI MENGGUNAKAN WIRESHARK 4.0.3", *JTOS*, vol. 7, tidak. 1, hal. 44 - 54, Juni 2024.
- [23] J. Jasri dan NW Al-hafiz, "Perancangan Aplikasi Infaq Berbasis Mobile Markazul Quran Wassunnah Foundation (MQS) Kuantan Singingi", *J. Teknik Informatika CIT Medicom*, vol. 15, tidak. 5, hal.247–254, November 2023.
- [24] R. Nazli, A. Amrizal, H. Hendra, dan S. Syukriadi, "Pemodelan Desain User Interface Aplikasi E-Business untuk Memasarkan Produk Umkm di Kota Payakumbuh Menggunakan Pieces Framework", *JTOS*, vol. 7, tidak. 2, hlm. 55 - 66, November 2024.
- [25] Siti Saniah dan Mhd. Furqan, "Klasifikasi Penyakit Tanaman Padi Menggunakan Algoritma K-Nearest Neighbor Berdasarkan Ekstraksi Warna Nilai Saturasi Hue dan Fitur Matriks Ko-Kemunculan Tingkat Abu-abu", *JTOS*, vol. 7, no. 2, hlm. 212 - 223, Des. 2024.