Terbit *online* pada laman: http://jurnal.utu.ac.id/JTI

Jurnal Teknologi Informasi

| ISSN (Online): 2829-8934 |

Metode Migrasi Multi-Aplikasi Monolitik ke Arsitektur Microservice Menggunakan Domain-Driven Design dan Strangler Fig Pattern

Yoga Ari Tofan[1*], Hudan Studiawan[2], Hadziq Fabroyir[3]
[1,2,3] Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
Email: [1]y.aritofan@gmail.com[*], [2]hudan@its.ac.id, [3]hadziq@its.ac.id

INFORMASI ARTIKEL

Sejarah Artikel:
Diterima: 16 September 2026
Revisi: 29 September 2026
Diterbitkan: 24 Oktober 2026

Kata Kunci:
Microservice
Domain-Driven Design (DDD)
Strangler Fig Pattern
Docker Compose
Apache Kafka

ABSTRAK

Arsitektur monolitik sering menimbulkan masalah skalabilitas, pemeliharaan, dan integrasi ketika jumlah modul dan pengguna terus meningkat. Pada lingkungan organisasi yang memiliki banyak sistem informasi saling bergantung, migrasi ke arsitektur microservice tidak lagi cukup dilakukan terhadap satu aplikasi secara terpisah. Penelitian ini mengusulkan metode migrasi multi-aplikasi monolitik ke arsitektur microservice dengan mengintegrasikan Domain-Driven Design (DDD) untuk identifikasi bounded context dan Strangler Fig Pattern untuk strategi migrasi bertahap. Metode yang diajukan meliputi identifikasi bounded context dari beberapa aplikasi monolitik, analisis ketergantungan dan pengelompokan fungsional lintas aplikasi, perancangan kandidat microservice, serta integrasi bertahap antara modul lama dan layanan baru. Studi kasus pada enam aplikasi monolitik menghasilkan sepuluh kandidat microservice; prototipe implementasi dan pengujian difokuskan pada domain akademik menggunakan tiga microservice (MS-Mahasiswa, MS-FRS, MS-Presensi) yang berjalan dalam sembilan kontainer bersama API gateway, message broker, dan basis data terisolasi per layanan. Implementasi menggunakan Node.js dan Express.js sebagai layanan REST, Apache Kafka sebagai mekanisme komunikasi publish–subscribe untuk sinkronisasi data, PostgreSQL sebagai basis data terpisah per microservice, dan Docker Compose untuk deployment terkontainerisasi. Hasil pengujian menunjukkan konsistensi data antar layanan mencapai 100% (100 record publisher tersinkron ke seluruh subscriber), latensi propagasi event rata-rata 5,10 ms (MS-FRS) dan 5,35 ms (MS-Presensi), serta performa endpoint melalui API gateway pada beban 100 pengguna concurrent dengan response time rata-rata 154,5 ms, throughput 419,7 request/detik, dan error rate 0,00%. Hasil ini menunjukkan bahwa metode migrasi yang diusulkan dapat menghasilkan layanan yang terisolasi secara data, terintegrasi secara event-driven, dan memenuhi kebutuhan operasional pada fase transisi dari monolitik ke microservice.

Copyright © 2026 Jurnal Teknologi Informasi UTU
All rights reserved

1. Pendahuluan

Arsitektur aplikasi monolitik masih banyak digunakan pada pengembangan sistem informasi organisasi karena relatif mudah dibangun dan di-deploy pada tahap awal [1], [2]. Namun, seiring pertumbuhan kebutuhan fungsional, jumlah pengguna, dan integrasi lintas sistem, arsitektur ini cenderung menimbulkan permasalahan skalabilitas, pemeliharaan, serta peningkatan risiko gangguan layanan ketika terjadi perubahan pada salah satu modul [3], [4]. Dalam konteks tersebut, migrasi dari arsitektur monolitik ke microservice menjadi salah satu pendekatan yang banyak direkomendasikan

karena menawarkan modularitas yang lebih tinggi, skalabilitas per layanan, dan fleksibilitas pemilihan teknologi [5].

Microservice memecah aplikasi menjadi layanan-layanan kecil yang independen, masing-masing mewakili domain atau proses bisnis tertentu dan dapat dikembangkan serta di-deploy secara terpisah [1]. Pendekatan ini memungkinkan setiap layanan memiliki basis data dan siklus hidup sendiri sehingga perubahan pada satu layanan tidak selalu mempengaruhi layanan lain. Berbagai penelitian menunjukkan bahwa migrasi ke arsitektur microservice dapat meningkatkan skalabilitas, ketersediaan, dan kecepatan pengembangan, terutama pada sistem dengan pertumbuhan fitur yang cepat [6]. Di sisi lain, migrasi ini tidak sederhana karena menyangkut pemahaman domain bisnis, pemecahan modul eksisting, serta penanganan dependensi data dan integrasi antar layanan [7].

Beragam metode migrasi telah diusulkan untuk memandu proses transisi dari monolitik ke microservice [4], [8], [9]. Salah satu pendekatan yang banyak dibahas adalah penggunaan Domain-Driven Design (DDD) untuk mengidentifikasi bounded context sebagai kandidat microservice berdasarkan pemetaan domain dan proses bisnis yang ada [10], [11]. DDD membantu memisahkan tanggung jawab berdasarkan batas domain yang jelas sehingga setiap layanan menangani satu area bisnis yang relatif kohesif. Di sisi lain, Strangler Fig Pattern diperkenalkan sebagai strategi migrasi bertahap, di mana fungsi-fungsi pada sistem lama dipindahkan satu per satu ke layanan baru tanpa harus mematikan aplikasi monolitik secara langsung [1]. Beberapa studi telah menggabungkan DDD dan Strangler Fig Pattern untuk memigrasi satu aplikasi monolitik ke arsitektur microservice dan melaporkan bahwa pendekatan bertahap ini dapat mengurangi risiko gangguan layanan selama proses migrasi [5].

Meskipun demikian, mayoritas penelitian sebelumnya berfokus pada migrasi satu aplikasi monolitik tunggal, misalnya sistem absensi, sistem transaksi tertentu, atau satu sistem informasi inti organisasi [5], [8]. Pada praktiknya, banyak organisasi memiliki beberapa aplikasi monolitik yang saling bergantung dan terintegrasi secara sporadis menggunakan mekanisme seperti database linked, pemanggilan API langsung, atau pertukaran data manual [3], [4]. Dalam kondisi ini, migrasi parsial terhadap satu aplikasi saja sering kali belum cukup untuk menyelesaikan masalah integrasi dan konsistensi data lintas sistem. Diperlukan pendekatan migrasi yang mempertimbangkan multi-aplikasi monolitik beserta relasi fungsional dan dependensi datanya secara menyeluruh [9].

Selain persoalan pemetaan domain dan pemecahan modul, migrasi multi-aplikasi monolitik ke microservice juga menimbulkan tantangan pada aspek komunikasi dan konsistensi data antar layanan [12]. Setiap microservice idealnya memiliki basis data yang terisolasi, tetapi di saat yang sama harus tetap dapat saling bertukar informasi ketika terjadi perubahan data yang bersifat lintas domain [13]. Berbagai pola integrasi seperti event-driven architecture, message bus, dan Saga Pattern diperkenalkan untuk menangani transaksi terdistribusi dan menjaga konsistensi data pada arsitektur microservice [12], [14], [15]. Namun, sebagian besar studi tersebut fokus pada desain komunikasi atau manajemen transaksi, bukan pada metode migrasi multi-aplikasi yang terintegrasi dengan jelas dari sisi domain maupun tahapan teknisnya [15]. Pada penelitian ini, integrasi tersebut diwujudkan melalui pola komunikasi asinkron publish–subscribe menggunakan message broker Apache Kafka, sementara layanan diimplementasikan sebagai REST API dan dijalankan secara terkontainerisasi.

Berdasarkan celah tersebut, penelitian ini mengusulkan sebuah metode migrasi multi-aplikasi monolitik ke arsitektur microservice dengan mengintegrasikan pendekatan Domain-Driven Design dan Strangler Fig Pattern. Metode yang diajukan mencakup tahap identifikasi bounded context dari beberapa aplikasi monolitik, analisis ketergantungan dan pengelompokan fungsional lintas aplikasi, perancangan kandidat microservice, serta integrasi bertahap antara modul lama dan layanan baru.

Studi kasus dilakukan pada enam aplikasi monolitik yang dikelompokkan menjadi dua domain utama, yaitu domain akademik dan kepegawaian, sehingga dihasilkan sepuluh kandidat microservice. Untuk memastikan rancangan dapat diwujudkan secara operasional dan dapat diuji, penelitian ini juga menyajikan prototipe implementasi pada subset layanan di domain akademik (MS-Mahasiswa sebagai publisher serta MS-FRS dan MS-Presensi sebagai subscriber) dan mengevaluasinya melalui pengujian konsistensi data dan pengujian performa endpoint.

Pertanyaan penelitian yang hendak dijawab adalah: (1) bagaimana merumuskan metode untuk menentukan domain dan bounded context pada beberapa aplikasi monolitik agar dapat dijadikan kandidat microservice secara terpadu; (2) bagaimana mengimplementasikan microservice yang dihasilkan agar dapat berjalan berdampingan dengan modul monolitik eksisting; dan (3) bagaimana memastikan integrasi bertahap tersebut tetap menjaga keterhubungan fungsional dan konsistensi data lintas layanan selama proses migrasi berlangsung.

2. Metodologi Penelitian

Penelitian ini menggunakan pendekatan rekayasa perangkat lunak berbasis studi kasus untuk merumuskan dan menerapkan metode migrasi multi-aplikasi monolitik ke arsitektur microservice. Metode yang diusulkan mengintegrasikan konsep Domain-Driven Design (DDD) untuk pemetaan domain dan bounded context dengan Strangler Fig Pattern sebagai strategi migrasi bertahap dari sistem lama ke layanan baru. Secara garis besar, tahapan penelitian terdiri atas: (1) analisis sistem monolitik dan identifikasi bounded context, (2) analisis ketergantungan dan pengelompokan fungsional lintas aplikasi, (3) perancangan kandidat microservice, (4) implementasi dan deployment microservice, serta (5) integrasi bertahap antara microservice dan modul monolitik eksisting.

2.1. Analisis Sistem Monolitik dan Identifikasi Bounded Context

Tahap pertama bertujuan untuk memahami struktur dan proses bisnis dari beberapa aplikasi monolitik yang akan dimigrasi. Analisis dilakukan terhadap enam aplikasi monolitik yang selama ini saling berinteraksi dalam mendukung proses akademik dan kepegawaian di sebuah organisasi, namun saling terhubung secara sporadis melalui mekanisme seperti database linked dan pertukaran data antar aplikasi. Pada tahap ini, dilakukan inventarisasi aktor, menu, fitur, dan alur proses bisnis pada masing-masing aplikasi guna mengidentifikasi fungsi-fungsi utama serta keterlibatan aktor pada setiap proses.

Selanjutnya, pendekatan Domain-Driven Design digunakan untuk mengidentifikasi domain dan subdomain yang relevan berdasarkan proses bisnis yang telah dipetakan. Setiap domain kemudian dianalisis untuk menentukan bounded context, yaitu batasan tanggung jawab yang cukup kohesif dan berpotensi menjadi kandidat microservice. Hasil dari tahap ini adalah daftar bounded context awal yang merepresentasikan fungsi-fungsi utama pada enam aplikasi monolitik, misalnya domain akademik (mahasiswa, rencana studi, presensi, kelas, portofolio) dan domain kepegawaian (pegawai, litabmas, portofolio dosen, beban kerja, angka kredit).

2.2. Analisis Ketergantungan dan Pengelompokan Fungsional

Tahap kedua berfokus pada analisis ketergantungan antar bounded context lintas aplikasi untuk membentuk kelompok fungsional yang lebih stabil sebagai dasar perancangan microservice. Analisis ini mencakup tiga aspek utama, yaitu: (1) fungsional utama yang menjadi inti layanan; (2) fungsional terkait yang memanfaatkan atau memperluas fungsional utama; dan (3) kepemilikan dan alur data antar modul.

Pertama, setiap bounded context diklasifikasikan berdasarkan perannya sebagai pemilik data utama (master data) atau sebagai konsumen data dari konteks lain. Kedua, dianalisis keterkaitan proses bisnis yang melibatkan beberapa aplikasi sekaligus, misalnya proses rencana studi yang memengaruhi layanan presensi, kelas, dan portofolio mahasiswa, atau proses pengelolaan litabmas yang memengaruhi portofolio dosen, beban kerja, dan penilaian angka kredit. Ketiga, dilakukan pemetaan kebutuhan data antar bounded context untuk mengidentifikasi konteks mana yang sebaiknya ditempatkan dalam kelompok microservice yang sama atau menjadi sumber data bagi layanan lain.

Hasil analisis ketergantungan ini adalah dua kelompok fungsional utama, yaitu kelompok akademik dan kepegawaian, masing-masing dengan lima kandidat microservice yang memiliki batas tanggung jawab dan hubungan data yang jelas. Kelompok akademik meliputi microservice mahasiswa, rencana studi, presensi, kelas, dan portofolio mahasiswa; sedangkan kelompok kepegawaian meliputi microservice pegawai, litabmas, portofolio dosen, beban kerja, dan penilaian angka kredit.

2.3. Perancangan Kandidat Microservice

Berdasarkan bounded context dan kelompok fungsional yang telah terbentuk, tahap selanjutnya adalah merancang kandidat microservice beserta tanggung jawab, antarmuka, dan model datanya. Setiap kandidat microservice dirancang untuk memenuhi prinsip tanggung jawab tunggal dan memiliki basis data yang terisolasi sesuai rekomendasi arsitektur microservice. Model data dari sistem monolitik dianalisis ulang untuk memetakan tabel-tabel yang relevan ke dalam skema basis data masing-masing microservice, sekaligus mengeliminasi ketergantungan langsung ke basis data terpusat.

Pada tahap ini juga dirancang kontrak antarmuka layanan dalam bentuk REST API menggunakan Node.js dan Express.js, serta event yang akan dipublikasikan melalui Apache Kafka sebagai message broker berbasis event streaming. Microservice yang berperan sebagai pemilik data utama (misalnya mahasiswa, pegawai, atau litabmas) dirancang sebagai publisher yang mengirimkan event ketika terjadi perubahan data penting, sedangkan microservice lain yang membutuhkan data tersebut bertindak sebagai subscriber. Rancangan ini menjadi dasar untuk mengimplementasikan komunikasi asinkron antar layanan pada tahap berikutnya menggunakan pola publish–subscribe melalui topik Kafka `mhs.updated`, di mana MS-Mahasiswa berperan sebagai publisher dan MS-FRS serta MS-Presensi sebagai subscriber.

2.4. Implementasi dan Deployment Microservice

Pada tahap implementasi, prototipe microservice dibangun untuk subset layanan di domain akademik, yaitu MS-Mahasiswa (publisher), MS-FRS, dan MS-Presensi (subscriber). Setiap microservice dibangun sebagai layanan mandiri menggunakan Node.js 18 LTS dengan framework Express.js yang menyediakan antarmuka RESTful, dan dihubungkan dengan instance PostgreSQL 15 yang terpisah per layanan sesuai pola database per service. Implementasi mengikuti rancangan tanggung jawab dan model data yang telah ditetapkan pada tahap sebelumnya, sehingga setiap layanan hanya memiliki akses ke basis datanya sendiri tanpa ketergantungan langsung ke skema monolitik.

Komunikasi antar layanan diwujudkan melalui mekanisme publish–subscribe menggunakan Apache Kafka 3.4 sebagai message broker. MS-Mahasiswa mempublikasikan event ke topik `mhs.updated` setiap kali terjadi perubahan data mahasiswa; MS-FRS dan MS-Presensi masing-masing berlangganan sebagai consumer group ms-frs-group dan ms-presensi-group untuk menerima dan menyimpan replikasi data yang dibutuhkan ke basis data lokalnya.

Seluruh komponen sistem yaitu tiga microservice, tiga instance PostgreSQL, Apache Kafka beserta ZooKeeper sebagai koordinator kluster, dan Nginx sebagai API gateway, dikemas dalam sembilan kontainer Docker dan dijalankan bersama menggunakan Docker Compose v2. Deployment dilaksanakan pada server bare metal dengan sistem operasi Ubuntu Server 22.04 LTS, prosesor AMD Ryzen 5 2400GE (4 core / 8 thread), dan memori RAM 14 GiB. Spesifikasi lengkap lingkungan implementasi disajikan pada Tabel 1. Pendekatan kontainerisasi dipilih untuk memastikan isolasi lingkungan antar layanan, konsistensi konfigurasi, serta kemudahan reproduksi oleh peneliti lain.

Tabel 1. Spesifikasi Perangkat Keras dan Lunak Server

Komponen	Detail
Tipe Mesin	Bare Metal Server
Prosesor	AMD Ryzen 5 2400GE (4 Core / 8 Thread)
Memori RAM	14 GiB (tersedia 11 GiB saat pengujian)
Sistem Operasi	Ubuntu Server 22.04.5 LTS
Docker Engine	24.0.9
Docker Compose	v2.24.7

2.5. Integrasi Bertahap dengan Strangler Fig Pattern

Tahap terakhir adalah mengintegrasikan microservice yang baru dibangun dengan modul-modul pada aplikasi monolitik eksisting menggunakan pendekatan Strangler Fig Pattern. Pada tahap awal, modul pada aplikasi monolitik tetap digunakan sebagai sumber utama layanan, sementara microservice baru berjalan paralel dan menerima replika data melalui mekanisme integrasi yang dirancang. Secara bertahap, fungsi-fungsi tertentu pada aplikasi monolitik dialihkan untuk menggunakan microservice

baru, misalnya dengan mengarahkan permintaan (request) tertentu ke layanan microservice atau memindahkan proses bisnis tertentu sepenuhnya ke layanan baru.

Untuk menjaga konsistensi data selama proses migrasi, digunakan pola komunikasi berbasis event dengan memanfaatkan Apache Kafka sebagai message broker yang menerapkan pola publish–subscribe sehingga microservice yang bertindak sebagai publisher dapat menyebarkan perubahan data ke microservice lain yang bertindak sebagai subscriber. Dengan cara ini, sistem lama dan sistem baru dapat berjalan bersamaan sampai seluruh fungsi yang ditargetkan telah berhasil dimigrasi dan divalidasi, setelah itu modul terkait pada aplikasi monolitik dapat dimatikan atau dihapus.

Kombinasi tahapan analisis domain, perancangan microservice, implementasi terkontainerisasi, dan integrasi bertahap ini membentuk metode migrasi multi-aplikasi monolitik ke microservice yang dievaluasi melalui prototipe implementasi pada domain akademik mencakup tiga microservice (MS-Mahasiswa, MS-FRS, MS-Presensi) yang dijalankan bersama dengan aplikasi monolitik eksisting.

2.6. Lingkungan Implementasi dan Pengujian

Pengujian dilaksanakan pada server bare metal dengan spesifikasi perangkat keras dan perangkat lunak yang dirangkum pada Tabel 1 dan Tabel 2. Seluruh komponen sistem dijalankan dalam kontainer Docker yang terisolasi menggunakan Docker Compose v2 untuk memastikan konsistensi dan reprodusibilitas lingkungan pengujian. Seluruh kode sumber dan konfigurasi tersedia di repositori publik: https://github.com/yogatofan/thesis-ms-research.

Tabel 2. Spesifikasi Stack Aplikasi Microservice

Komponen	Teknologi	Versi
Runtime	Node.js	18 LTS (Alpine)
Web Framework	Express.js	4.18.2
Message Broker	Apache Kafka	3.4.0
Kafka Client	KafkaJS	2.2.4
Database	PostgreSQL	15
API Gateway	Nginx	1.25+
Orkestrasi	Docker Compose	v2

3. Hasil dan Pembahasan

3.1. Hasil Identifikasi Domain dan Bounded Context

Tahap identifikasi domain dan bounded context diawali dengan analisis terhadap enam aplikasi monolitik yang selama ini digunakan untuk mendukung proses akademik dan kepegawaian di sebuah organisasi. Setiap aplikasi dianalisis dari sisi aktor yang terlibat, struktur menu, serta alur proses bisnis untuk memperoleh gambaran fungsi-fungsi utama yang berjalan pada masing-masing sistem. Hasil analisis menunjukkan bahwa meskipun aplikasi tersebut dikembangkan secara terpisah, terdapat banyak irisan proses bisnis dan kebutuhan data yang saling bergantung, khususnya pada data mahasiswa, pegawai, aktivitas akademik, dan aktivitas litabmas seperti pada Tabel 3.

Dengan mengacu pada prinsip Domain-Driven Design, fungsi-fungsi yang telah diidentifikasi kemudian dikelompokkan ke dalam domain dan subdomain yang merepresentasikan area bisnis yang kohesif. Pada domain akademik, diperoleh beberapa proses bisnis utama, antara lain pengelolaan data mahasiswa, pengisian rencana studi, pencatatan presensi perkuliahan, pengelolaan kelas, dan pengelolaan portofolio aktivitas mahasiswa. Sementara pada domain kepegawaian, proses bisnis utama yang teridentifikasi meliputi pengelolaan data pegawai, pengelolaan kegiatan litabmas, penyusunan portofolio dosen, pengelolaan beban kerja, dan pengajuan angka kredit.

Setiap proses bisnis utama tersebut dipetakan menjadi bounded context awal yang menjadi kandidat batas tanggung jawab layanan pada arsitektur microservice. Sebagai contoh, proses bisnis mahasiswa dipetakan ke bounded context Mahasiswa yang bertanggung jawab terhadap seluruh data dan aturan bisnis terkait identitas dan status akademik mahasiswa. Proses bisnis rencana studi dipetakan ke bounded context Rencana Studi yang mengelola pengajuan dan persetujuan rencana studi per semester, sedangkan proses bisnis presensi dipetakan ke bounded context Presensi yang mengelola kehadiran dosen dan mahasiswa dalam kegiatan perkuliahan. Pola yang sama diterapkan pada proses bisnis kelas

dan portofolio mahasiswa, serta pada proses bisnis pegawai, litabmas, portofolio dosen, beban kerja, dan angka kredit di domain kepegawaian.

Tabel 3. Identifikasi Bounded Context Awal

Domain	Proses Bisnis	Bounded Context Awal
Akademik	Pengelolaan data mahasiswa	Mahasiswa
Akademik	Pengisian dan persetujuan FRS	Rencana Studi
Akademik	Pencatatan presensi perkuliahan	Presensi
Akademik	Pengelolaan kelas perkuliahan	Kelas
Akademik	Pengelolaan portofolio mahasiswa (kegiatan, prestasi, dll.)	Portofolio Mahasiswa
Kepegawaian	Pengelolaan data pegawai	Pegawai
Kepegawaian	Pengelolaan kegiatan penelitian dan pengabdian (litabmas)	Litabmas
Kepegawaian	Pengelolaan portofolio aktivitas dosen	Portofolio Dosen
Kepegawaian	Pengelolaan beban kerja dosen	Beban Kerja
Kepegawaian	Pengelolaan pengajuan angka kredit	Angka Kredit

Secara ringkas, hasil identifikasi ini menghasilkan dua kelompok domain utama dengan masing-masing bounded context sebagai berikut:

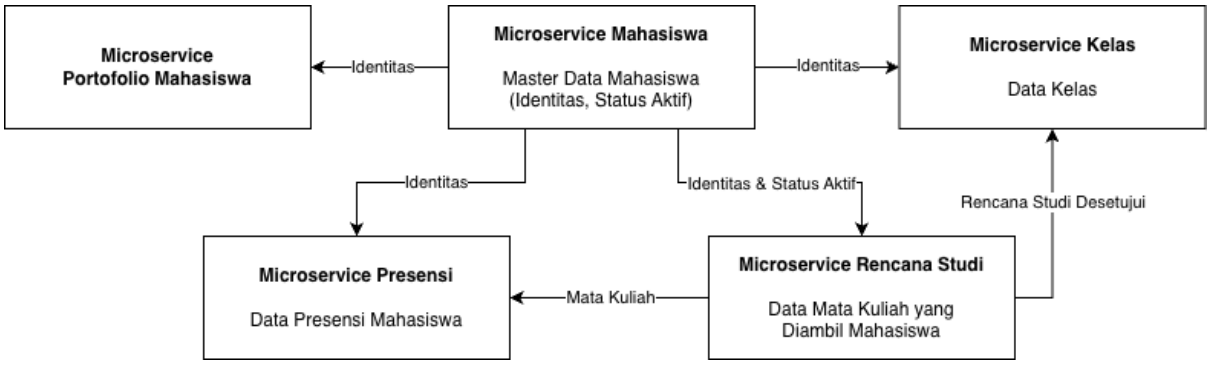
- Domain akademik: Mahasiswa, Rencana Studi, Presensi, Kelas, Portofolio Mahasiswa.
- Domain kepegawaian: Pegawai, Litabmas, Portofolio Dosen, Beban Kerja, Angka Kredit.

Daftar bounded context tersebut menjadi dasar untuk tahap berikutnya, yaitu analisis ketergantungan dan pengelompokan fungsional lintas aplikasi guna membentuk kandidat microservice yang lebih stabil dan konsisten dengan kebutuhan bisnis organisasi.

3.2. Pembentukan Kelompok Fungsional dan Kandidat Microservice

Berdasarkan bounded context yang telah diidentifikasi, tahap berikutnya adalah menganalisis ketergantungan fungsional dan aliran data antar konteks untuk membentuk kelompok fungsional yang menjadi dasar kandidat microservice. Analisis ini mempertimbangkan beberapa aspek, yaitu peran konteks sebagai pemilik data utama (master data), hubungan penggunaan data oleh konteks lain, serta keterurutan proses bisnis yang melibatkan lebih dari satu aplikasi monolitik. Dengan cara ini, dapat dihindari pembentukan layanan yang terlalu saling bergantung ataupun layanan yang menangani tanggung jawab yang terlalu luas.

Pada domain akademik, bounded context Mahasiswa diidentifikasi sebagai pemilik data utama mahasiswa yang digunakan oleh beberapa proses bisnis lain, seperti rencana studi, presensi, kelas, dan portofolio mahasiswa seperti pada Gambar 1. Bounded context Rencana Studi menjadi penghubung penting karena status rencana studi yang disetujui menentukan hak akses mahasiswa terhadap layanan presensi, kelas, dan portofolio. Sementara itu, bounded context Presensi mengelola data kehadiran yang bergantung pada daftar mahasiswa dan rencana studi yang aktif, bounded context Kelas mengelola penjadwalan dan keikutsertaan mahasiswa dalam kelas, dan bounded context Portofolio Mahasiswa mengelola rekam aktivitas akademik dan non-akademik.



Gambar 1. Hubungan Kebutuhan Data Antar Microservice - Domain Akademik

Hasil analisis pada Tabel 4 menunjukkan bahwa meskipun bounded context tersebut saling membutuhkan data, masing-masing memiliki tanggung jawab bisnis yang cukup spesifik dan dapat direalisasikan sebagai microservice tersendiri, dengan Mahasiswa dan Rencana Studi berperan sebagai penyedia data utama bagi layanan lain. Oleh karena itu, dibentuk satu kelompok fungsional akademik dengan lima kandidat microservice, yaitu Microservice Mahasiswa, Microservice Rencana Studi, Microservice Presensi, Microservice Kelas, dan Microservice Portofolio Mahasiswa.

Tabel 4. Kandidat Microservice Kelompok Akademik

Kelompok Domain	Proses Bisnis	Kandidat Microservice	Peran Utama
Akademik	Pengelolaan data mahasiswa	Microservice Mahasiswa	Pemilik master data mahasiswa
Akademik	Pengisian dan persetujuan FRS	Microservice Rencana Studi	Pengelolaan rencana studi per semester
Akademik	Pencatatan presensi perkuliahan	Microservice Presensi	Pencatatan dan rekap kehadiran
Akademik	Pengelolaan kelas perkuliahan	Microservice Kelas	Pengelolaan jadwal dan keikutsertaan kelas
Akademik	Pengelolaan portofolio mahasiswa	Microservice Portofolio Mahasiswa	Pengelolaan rekam aktivitas dan portofolio mahasiswa

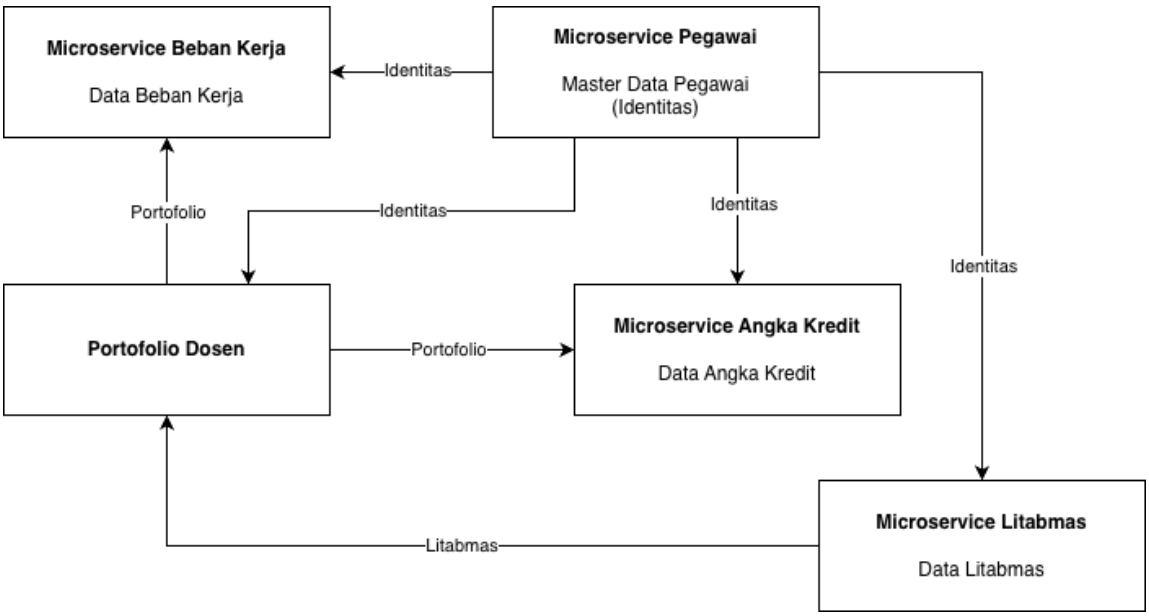
Pada domain kepegawaian seperti pada Gambar 2, bounded context Pegawai diidentifikasi sebagai pemilik master data pegawai yang diperlukan oleh bounded context Litabmas, Portofolio Dosen, Beban Kerja, dan Angka Kredit. Bounded context Litabmas mengelola kegiatan penelitian dan pengabdian yang kemudian menjadi salah satu komponen portofolio dosen. Bounded context Portofolio Dosen menyatukan berbagai aktivitas dosen yang relevan sebagai dasar pengajuan beban kerja dan angka kredit. Bounded context Beban Kerja dan Angka Kredit memanfaatkan data portofolio untuk proses penilaian dan pengajuan administratif.

Dengan pola serupa, analisis ketergantungan menunjukkan bahwa masing-masing bounded context di domain kepegawaian dapat direalisasikan sebagai microservice yang berdiri sendiri namun tetap saling terhubung melalui pertukaran data yang terdefinisi. Dari hasil ini dibentuk satu kelompok fungsional kepegawaian dengan lima kandidat microservice, yaitu Microservice Pegawai, Microservice Litabmas, Microservice Portofolio Dosen, Microservice Beban Kerja, dan Microservice Angka Kredit seperti pada Tabel 5.

Tabel 5. Kandidat Microservice Kelompok Kepegawaian

Kelompok Domain	Proses Bisnis	Kandidat Microservice	Peran Utama
Kepegawaian	Pengelolaan data pegawai	Microservice Pegawai	Pemilik master data pegawai
Kepegawaian	Pengelolaan kegiatan litabmas	Microservice Litabmas	Pengelolaan data penelitian dan pengabdian
Kepegawaian	Pengelolaan portofolio aktivitas dosen	Microservice Portofolio Dosen	Integrasi aktivitas dosen sebagai portofolio
Kepegawaian	Pengelolaan beban kerja dosen	Microservice Beban Kerja	Pengelolaan dan penilaian beban kerja
Kepegawaian	Pengelolaan pengajuan dan penilaian angka kredit	Microservice Angka Kredit	Pengelolaan pengajuan angka kredit

Pembentukan dua kelompok fungsional ini memperjelas struktur layanan yang diusulkan, sekaligus mengurangi kompleksitas integrasi lintas aplikasi monolitik dengan cara mengarahkan hubungan antar sistem ke hubungan antar microservice yang lebih terkontrol. Kelompok akademik dan kepegawaian inilah yang kemudian menjadi fokus perancangan arsitektur microservice pada tahap selanjutnya.



Gambar 2. Hubungan Kebutuhan Data Antar Microservice - Domain Kepegawaian

3.3. Rancangan Arsitektur Microservice

Berdasarkan kelompok fungsional dan kandidat microservice yang telah ditetapkan, tahap selanjutnya adalah merancang arsitektur logis microservice beserta komponen pendukungnya. Setiap kandidat microservice dirancang sebagai layanan mandiri yang menangani satu bounded context dengan jelas, memiliki basis data terpisah, dan berkomunikasi dengan layanan lain melalui antarmuka yang terdefinisi. Pendekatan ini mengikuti prinsip tanggung jawab tunggal dan isolasi data pada arsitektur microservice sehingga perubahan pada satu layanan tidak secara langsung memengaruhi layanan lainnya.

Pada kelompok akademik, arsitektur microservice dirancang terdiri atas lima layanan utama, yaitu Microservice Mahasiswa, Microservice Rencana Studi, Microservice Presensi, Microservice Kelas, dan Microservice Portofolio Mahasiswa. Microservice Mahasiswa berperan sebagai pemilik master data mahasiswa dan menyediakan data tersebut bagi layanan lain, sedangkan Microservice Rencana Studi mengelola pengajuan dan persetujuan rencana studi yang menentukan kelayakan mahasiswa untuk mengikuti perkuliahan. Microservice Presensi, Microservice Kelas, dan Microservice Portofolio Mahasiswa masing-masing memanfaatkan data dari dua layanan tersebut untuk mengelola kehadiran perkuliahan, keikutsertaan dalam kelas, serta rekam aktivitas akademik mahasiswa.

Pada kelompok kepegawaian, arsitektur microservice terdiri atas Microservice Pegawai, Microservice Litabmas, Microservice Portofolio Dosen, Microservice Beban Kerja, dan Microservice Angka Kredit. Microservice Pegawai bertanggung jawab terhadap master data pegawai, yang kemudian digunakan oleh Microservice Litabmas untuk mengelola data kegiatan penelitian dan pengabdian. Microservice Portofolio Dosen mengintegrasikan data dari berbagai aktivitas dosen, termasuk litabmas, sebagai dasar bagi Microservice Beban Kerja dan Microservice Angka Kredit dalam mendukung proses penilaian dan pengajuan administratif. Dengan pemisahan ini, setiap microservice dapat berkembang sesuai kebutuhan proses bisnis tanpa harus mengubah keseluruhan sistem.

Semua microservice dirancang sebagai layanan web berbasis Node.js dan Express.js yang menyediakan antarmuka RESTful, serta terhubung ke basis data PostgreSQL masing-masing yang dirancang ulang dari skema monolitik. Untuk mendukung deployment yang fleksibel dan terisolasi, setiap microservice dikemas dalam kontainer dan dijalankan pada lingkungan kontainerisasi sehingga konfigurasi lingkungan, dependensi, dan skala layanan dapat dikelola secara independen. Rancangan arsitektur ini menjadi landasan bagi penerapan Strangler Fig Pattern pada tahap integrasi bertahap dengan aplikasi monolitik eksisting.

Pada tahap prototipe implementasi dan evaluasi, rancangan arsitektur diwujudkan pada subset layanan di domain akademik, yaitu MS-Mahasiswa sebagai publisher serta MS-FRS dan MS-Presensi

sebagai subscriber. Ketiga layanan diimplementasikan sebagai REST API dan dijalankan secara terkontainerisasi bersama API gateway, message broker, dan basis data terisolasi per layanan.

3.4. Penerapan Strangler Fig Pattern pada Multi-Aplikasi

Rancangan arsitektur microservice yang telah dibentuk kemudian diintegrasikan secara bertahap dengan aplikasi-aplikasi monolitik eksisting menggunakan pendekatan Strangler Fig Pattern. Pendekatan ini memungkinkan sistem lama dan sistem baru berjalan secara paralel dalam periode transisi, sehingga risiko gangguan layanan selama proses migrasi dapat diminimalkan. Alih-alih memigrasi seluruh fungsi sekaligus, hanya fungsi-fungsi tertentu yang dipindahkan secara bertahap ke microservice baru, sementara bagian lain dari aplikasi monolitik tetap beroperasi seperti semula.

Pada tahap awal, modul-modul pada aplikasi monolitik tetap menjadi sumber utama layanan, sedangkan microservice baru dibangun dan dijalankan untuk mereplikasi sebagian fungsi dan data yang telah diidentifikasi sebagai prioritas migrasi. Misalnya, fungsi pengelolaan rencana studi dan portofolio mahasiswa yang sebelumnya berada di beberapa aplikasi monolitik mulai dipindahkan ke Microservice Rencana Studi dan Microservice Portofolio Mahasiswa, dengan tetap menjaga agar data dan alur proses pada sistem lama tidak langsung dihentikan. Mekanisme pengalihan permintaan secara bertahap diterapkan dengan mengarahkan subset permintaan pengguna atau proses bisnis tertentu ke layanan microservice, sambil mempertahankan rute lama untuk fungsi yang belum dimigrasi.

Pendekatan yang sama diterapkan pada domain kepegawaian, di mana fungsi-fungsi terkait pengelolaan pegawai, litabmas, dan portofolio dosen secara bertahap dipindahkan ke microservice baru. Microservice Pegawai, Litabmas, dan Portofolio Dosen mulai mengambil alih peran dari modul monolitik yang sebelumnya menangani data-data tersebut, namun modul lama tetap tersedia hingga microservice dinyatakan stabil dan tervalidasi. Dengan cara ini, organisasi dapat menguji dan memverifikasi perilaku layanan baru pada lingkungan produksi tanpa harus menghentikan aplikasi monolitik secara tiba-tiba.

Selama proses ini, integrasi antara microservice dan aplikasi monolitik yang masih aktif diatur melalui antarmuka dan mekanisme pertukaran data yang terkontrol, sehingga perubahan di satu sisi tidak langsung mengganggu keseluruhan sistem. Setelah fungsi tertentu dinilai berjalan stabil pada microservice dan tidak lagi bergantung pada modul monolitik, rute akses dan tanggung jawab fungsi tersebut dialihkan sepenuhnya ke layanan baru, dan modul terkait pada aplikasi monolitik dapat dimatikan atau dihapus. Penerapan Strangler Fig Pattern pada beberapa aplikasi yang saling bergantung ini menunjukkan bahwa strategi migrasi bertahap dapat diperluas dari level satu aplikasi menjadi level ekosistem aplikasi, asalkan analisis domain dan ketergantungan dilakukan secara menyeluruh sejak tahap awal.

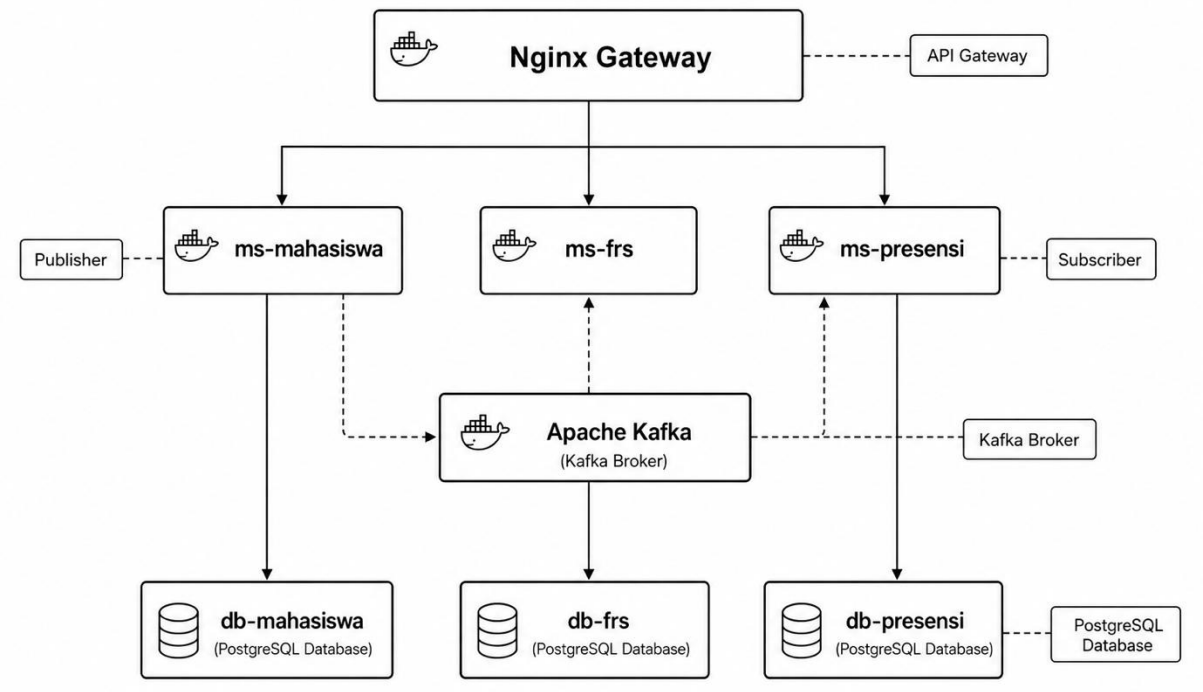
Implementasi microservice dan integrasi bertahap yang dilakukan pada prototipe domain akademik menunjukkan bahwa setiap kandidat microservice dapat berjalan independen, saling bertukar data melalui mekanisme yang dirancang, dan menggantikan fungsi modul monolitik yang dituju tanpa mengganggu layanan lain. Implementasi prototipe menerapkan integrasi bertahap melalui pemisahan fungsi master data mahasiswa ke MS-Mahasiswa dan sinkronisasi data ke layanan lain menggunakan pola publish–subscribe melalui Apache Kafka. Dengan mekanisme ini, layanan monolitik dan layanan microservice dapat berjalan paralel pada fase transisi, sementara konsistensi data antar basis data layanan dijaga melalui propagasi event pada topik `mhs.updated`.

3.5. Hasil Implementasi Prototipe dan Lingkungan Eksekusi

Prototipe arsitektur microservice dijalankan pada lingkungan Ubuntu Server dengan orkestrasi kontainer menggunakan Docker Compose. Seluruh komponen sistem dijalankan secara bersamaan dalam sembilan kontainer yang terdiri dari tiga microservice (MS-Mahasiswa, MS-FRS, MS-Presensi), tiga basis data PostgreSQL yang terisolasi, Apache Kafka sebagai message broker, ZooKeeper, serta Nginx sebagai API Gateway. Implementasi ini merealisasikan prinsip database per service dan komunikasi asinkron antarlayanan berbasis event-driven (publish–subscribe). Spesifikasi perangkat

keras dan tumpukan perangkat lunak yang digunakan disajikan pada Tabel 1 dan Tabel 2 di Bab 2 Metodologi Penelitian.

Topologi deployment prototipe ditunjukkan pada Gambar 3. Seluruh permintaan eksternal masuk melalui Nginx API Gateway yang meneruskan permintaan ke microservice yang sesuai. MS-Mahasiswa sebagai publisher mempublikasikan event perubahan data ke topik `mhs.updated` pada Apache Kafka; MS-FRS dan MS-Presensi sebagai subscriber mengonsumsi event tersebut dan menyimpan replikasi data ke basis datanya masing-masing secara asinkron. Setiap microservice hanya memiliki akses ke instance basis data miliknya sendiri (db-mahasiswa, db-frs, db-presensi), sehingga tidak ada ketergantungan langsung antar basis data layanan.



Gambar 3. Topologi Deployment Prototipe Arsitektur Microservice

3.6. Hasil Pengujian

Pengujian terhadap sistem informasi akademik berbasis arsitektur microservice dilakukan untuk membuktikan bahwa hasil migrasi dari arsitektur monolitik mampu memenuhi kebutuhan operasional yang mencakup konsistensi data, efisiensi sinkronisasi, ketahanan terhadap beban, dan efisiensi penggunaan sumber daya.

3.6.1. Pengujian Konsistensi Data (Integration Testing)

Pengujian konsistensi data bertujuan untuk memverifikasi bahwa mekanisme sinkronisasi berbasis event-driven melalui Apache Kafka mampu menjamin integritas data di seluruh microservice yang berperan sebagai subscriber. Pengujian dilakukan dengan memasukkan 100 data mahasiswa melalui endpoint API MS-Mahasiswa (Publisher), kemudian mengukur kelengkapan data yang tersinkronisasi ke MS-FRS dan MS-Presensi (Subscribers) setelah jendela waktu propagasi selama delapan detik.

Hasil pengujian disajikan pada Tabel 6. Seluruh 100 data berhasil diteruskan dan diterima dengan sempurna oleh kedua subscriber, sehingga Data Integrity Rate mencapai 100%. Hal ini membuktikan bahwa pola komunikasi asinkron berbasis event (publish-subscribe) yang diimplementasikan melalui topik Kafka `mhs.updated` mampu menjamin konsistensi data meskipun setiap layanan memiliki basis data yang sepenuhnya terisolasi. Temuan ini selaras dengan prinsip eventual consistency yang menjadi fondasi arsitektur microservice modern [12], [13].

Tabel 6. Hasil Pengujian Konsistensi Data

Metrik	Hasil	Target	Status
Publisher (MS-Mahasiswa)	100 records	100 records	Pass
Subscriber (MS-FRS)	100 records	100 records	Pass
Subscriber (MS-Presensi)	100 records	100 records	Pass
Data Integrity Rate (FRS)	100%	100%	Pass
Data Integrity Rate (Presensi)	100%	100%	Pass
Durasi Insert 100 Records	1,0 detik	-	-

3.6.2. Pengujian Latensi Sinkronisasi (Sync Latency Test)

Pengujian latensi sinkronisasi bertujuan untuk mengukur selisih waktu antara saat data dipublikasikan oleh MS-Mahasiswa ke topik Kafka hingga data tersebut berhasil diterima dan disimpan oleh MS-FRS dan MS-Presensi. Pengujian dilakukan dengan memasukkan 20 data secara berurutan dengan interval 200 milidetik, dan setiap event diberi timestamp pada saat diterbitkan maupun pada saat dikonsumsi untuk menghitung latensi aktual.

Hasil pengujian disajikan pada Tabel 7. Rata-rata latensi sinkronisasi yang dicapai adalah 5,10 ms untuk MS-FRS dan 5,35 ms untuk MS-Presensi, jauh di bawah ambang batas yang ditetapkan sebesar 5.000 ms. Nilai P95 latensi masing-masing hanya 6,00 ms dan 6,10 ms, yang mengindikasikan stabilitas propagasi event bahkan pada kondisi pengiriman beruntun. Rendahnya latensi ini menunjukkan bahwa Apache Kafka mampu beroperasi mendekati real-time dalam jaringan kontainer lokal, sehingga arsitektur event-driven yang diimplementasikan layak diterapkan untuk kebutuhan sinkronisasi data akademik yang memerlukan konsistensi tinggi.

Tabel 7. Hasil Pengujian Latensi Sinkronisasi via Kafka

Metrik	MS-FRS	MS-Presensi	Target	Status
Total Events	20	20	-	-
Rata-rata Latensi	5,10 ms	5,35 ms	< 5.000 ms	Pass
Latensi Minimum	4 ms	4 ms	-	-
Latensi Maksimum	6 ms	8 ms	-	-
P95 Latensi	6,00 ms	6,10 ms	-	-
P99 Latensi	6,00 ms	7,62 ms	-	-

3.6.3. Pengujian Beban Endpoint melalui API Gateway (Stress Test)

Pengujian beban dilakukan untuk mengukur ketahanan dan performa sistem ketika menerima permintaan secara bersamaan dalam jumlah yang meningkat. Pengujian menggunakan pendekatan concurrent request dengan tiga level beban: 10, 50, dan 100 pengguna bersamaan, yang dikirimkan ke endpoint `GET /api/mahasiswa` melalui Nginx API Gateway. Setiap level pengujian diulang sebanyak tiga kali dan hasilnya dirata-ratakan untuk meminimalkan pengaruh variasi jaringan.

Hasil pengujian disajikan pada Tabel 8. Pada beban puncak 100 pengguna concurrent, sistem mampu mempertahankan rata-rata response time sebesar 154,5 ms, jauh di bawah ambang batas 300 ms yang ditetapkan. Throughput yang dicapai mencapai 419,7 request/detik, delapan kali lebih tinggi dari target minimum 50 request/detik. Secara keseluruhan, tidak terjadi satu pun kegagalan permintaan (error rate 0,00%) di seluruh level pengujian. Hasil ini menunjukkan bahwa arsitektur microservice yang dikombinasikan dengan Nginx API Gateway memiliki kemampuan skalabilitas horizontal yang memadai untuk melayani kebutuhan sistem informasi akademik dengan jumlah pengguna aktif yang signifikan.

Tabel 8. Hasil Pengujian Beban (Stress Test)

Pengguna Concurrent	Avg Response Time	P95 Response Time	Throughput	Error Rate	Status
10 pengguna	37,7 ms	52 ms	198,4 req/s	0,00%	Pass
50 pengguna	116,9 ms	169 ms	295,7 req/s	0,00%	Pass
100 pengguna	154,5 ms	209 ms	419,7 req/s	0,00%	Pass

3.6.4. Penggunaan Sumber Daya Kontainer (Resource Utilization)

Salah satu keunggulan utama yang diklaim dalam migrasi ke arsitektur microservice adalah efisiensi penggunaan sumber daya melalui kontainerisasi [6]. Tabel 9 menyajikan penggunaan memori (RAM)

setiap kontainer yang diukur saat sistem sedang aktif setelah menyelesaikan seluruh rangkaian pengujian.

Tabel 9. Penggunaan Sumber Daya Kontainer Docker

Kontainer	Fungsi	CPU %	Penggunaan RAM	Target	Status
nginx-gateway	API Gateway	0,00%	10,49 MiB	< 512 MiB	Pass
ms-mahasiswa	Publisher (Mahasiswa)	4,21%	74,28 MiB	< 512 MiB	Pass
ms-frs	Subscriber (FRS)	4,48%	48,86 MiB	< 512 MiB	Pass
ms-presensi	Subscriber (Presensi)	4,56%	40,73 MiB	< 512 MiB	Pass
kafka	Message Broker	0,76%	400,5 MiB	< 512 MiB	Pass
zookeeper	Kafka Coordinator	0,16%	94,07 MiB	< 512 MiB	Pass
db-mahasiswa	Database Mahasiswa	0,02%	52,84 MiB	< 512 MiB	Pass
db-frs	Database FRS	0,00%	21,15 MiB	< 512 MiB	Pass
db-presensi	Database Presensi	0,00%	21,09 MiB	< 512 MiB	Pass

Setiap kontainer microservice mengonsumsi memori di bawah 75 MiB, jauh di bawah batas 512 MiB yang ditetapkan. Isolasi kontainer juga memungkinkan setiap layanan berjalan secara independen tanpa saling berbagi memori, sehingga kegagalan pada satu layanan tidak berdampak pada layanan lainnya. Penggunaan basis data yang terpisah untuk setiap microservice (db_mahasiswa, db_frs, db_presensi) membuktikan penerapan prinsip database per service yang merupakan pola arsitektur microservice yang direkomendasikan [1], [13].

3.7. Diskusi

Secara keseluruhan, hasil pengujian membuktikan bahwa arsitektur microservice yang diimplementasikan menggunakan Node.js, Apache Kafka, PostgreSQL, dan Docker mampu memenuhi seluruh kriteria pengujian yang ditetapkan. Ringkasan seluruh metrik disajikan pada Tabel 10.

Tabel 10. Rekapitulasi Hasil Pengujian

Metrik	Hasil Aktual	Target	Status
Data Integrity Rate	100%	100%	Pass
Rata-rata Sync Latency (Kafka)	5,10 ms	< 5.000 ms	Pass
Average Response Time (100 user)	154,5 ms	< 300 ms	Pass
Throughput (100 user)	419,7 req/s	> 50 req/s	Pass
Error Rate	0,00%	< 1%	Pass
Memory Usage per Microservice	< 75 MiB	< 512 MiB	Pass

Perbandingan dengan estimasi awal menunjukkan bahwa sistem memberikan performa yang signifikan melampaui ekspektasi. Latensi sinkronisasi Kafka yang hanya 5,1 ms, jauh di bawah estimasi 5.000 ms, mengindikasikan bahwa komunikasi event-driven antar layanan bukan hanya sekadar solusi eventual consistency, namun berpotensi mendukung skenario mendekati real-time. Throughput sebesar 419,7 request/detik dengan nol kegagalan membuktikan bahwa Nginx API Gateway berhasil mendistribusikan beban secara efektif ke seluruh microservice yang berada di belakangnya.

Temuan ini menunjukkan bahwa prototipe microservice yang dihasilkan melalui pendekatan Strangler Fig Pattern mampu memenuhi kriteria performa operasional yang ditetapkan, yang menjadi indikasi awal kelayakan arsitektur baru sebelum penggantian sistem monolitik secara penuh.

4. Kesimpulan

Penelitian ini mengusulkan metode migrasi multi-aplikasi monolitik ke arsitektur microservice dengan mengintegrasikan Domain-Driven Design dan Strangler Fig Pattern. Melalui analisis domain dan bounded context pada enam aplikasi monolitik, diperoleh dua kelompok domain utama beserta sepuluh kandidat microservice: lima pada domain akademik (Mahasiswa, Rencana Studi, Presensi, Kelas, Portofolio Mahasiswa) dan lima pada domain kepegawaian (Pegawai, Litabmas, Portofolio Dosen, Beban Kerja, Angka Kredit). Pembentukan kelompok fungsional berdasarkan peran kepemilikan master data dan hubungan ketergantungan antar bounded context menghasilkan struktur

microservice yang lebih kohesif dibandingkan pendekatan migrasi per-aplikasi yang umum dilakukan pada penelitian sebelumnya.

Prototipe implementasi dan evaluasi difokuskan pada subset domain akademik yang mencakup tiga layanan: MS-Mahasiswa sebagai publisher, serta MS-FRS dan MS-Presensi sebagai subscriber. Ketiga layanan dibangun menggunakan Node.js dan Express.js, dikomunikasikan secara asinkron melalui Apache Kafka dengan pola publish–subscribe pada topik `mhs.updated`, dan dijalankan bersama basis data PostgreSQL yang terisolasi per layanan serta Nginx sebagai API gateway dalam sembilan kontainer Docker Compose. Pengujian konsistensi data menunjukkan tingkat integritas data 100% pada kedua subscriber dengan durasi propagasi event rata-rata 5,10 ms (MS-FRS) dan 5,35 ms (MS-Presensi). Pengujian beban endpoint pada 100 pengguna concurrent menghasilkan response time rata-rata 154,5 ms, throughput 419,7 request/detik, dan error rate 0,00%. Seluruh kontainer microservice mengonsumsi memori di bawah 75 MiB per layanan, menunjukkan efisiensi penggunaan sumber daya yang konsisten dengan prinsip isolasi kontainer.

Penerapan Strangler Fig Pattern pada prototipe ini membuktikan bahwa sistem lama dan sistem baru dapat berjalan secara paralel selama fase transisi, dengan konsistensi data lintas layanan dijaga melalui mekanisme event-driven tanpa ketergantungan langsung antar basis data. Dengan demikian, metode yang diusulkan memberikan panduan bertahap yang dapat diterapkan pada organisasi yang memiliki ekosistem multi-aplikasi monolitik dengan ketergantungan lintas sistem yang kompleks.

Evaluasi pada penelitian ini dibatasi pada subset layanan domain akademik; pengujian terhadap seluruh sepuluh kandidat microservice, termasuk domain kepegawaian, belum dilakukan. Pengukuran perbandingan performa antara sistem monolitik dan sistem microservice juga belum disertakan. Kedua hal ini menjadi arah penelitian lanjutan, di samping pengujian pada lingkungan produksi berskala lebih besar, penerapan pola Saga untuk penanganan transaksi terdistribusi, dan validasi metode pada domain serta tumpukan teknologi yang berbeda.

Ucapan Terimakasih

Penulis mengucapkan terima kasih kepada pimpinan dan pengelola sistem informasi di institusi tempat penelitian dilaksanakan yang telah memberikan dukungan dan fasilitas dalam pelaksanaan penelitian ini.

Daftar Pustaka

[1] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, Inc., 2019.

[2] R. Chen, S. Li, and Z. Li, "From Monolith to Microservices: A Dataflow-Driven Approach," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2017, pp. 466–475. doi: 10.1109/APSEC.2017.53.

[3] F. Ponce, G. Márquez, and H. Astudillo, "Migrating from monolithic architecture to microservices: A Rapid Review," in *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, Nov. 2019, pp. 1–7. doi: 10.1109/SCCC49216.2019.8966423.

[4] T. Prasandy, Titan, D. F. Murad, and T. Darwis, "Migrating Application from Monolith to Microservices," in *2020 International Conference on Information Management and Technology (ICIMTech)*, Aug. 2020, pp. 726–731. doi: 10.1109/ICIMTech50083.2020.9211252.

[5] C.-Y. Li, S.-P. Ma, and T.-W. Lu, "Microservice Migration Using Strangler Fig Pattern: A Case Study on the Green Button System," in *2020 International Computer Symposium (ICS)*, Dec. 2020, pp. 519–524. doi: 10.1109/ICS51289.2020.00107.

[6] S. Li *et al.*, "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review," *Inf. Softw. Technol.*, vol. 131, p. 106449, Mar. 2021, doi: 10.1016/j.infsof.2020.106449.

[7] C. Bandara and I. Perera, "Transforming Monolithic Systems to Microservices - An Analysis Toolkit for Legacy Code Evaluation," in *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*, Nov. 2020, pp. 95–100. doi: 10.1109/ICTer51097.2020.9325443.

[8] R. Mufrizal and D. Indarti, "Refactoring Arsitektur Microservice Pada Aplikasi Absensi PT. Graha Usaha Teknik," *J. Nas. Teknol. Dan Sist. Inf.*, vol. 5, pp. 57–68, Apr. 2019, doi: 10.25077/TEKNOSI.v5i1.2019.57-68.

[9] J. Kazanavičius and D. Mažeika, "Migrating Legacy Software to Microservices Architecture," in *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, Apr. 2019, pp. 1–5. doi: 10.1109/eStream.2019.8732170.

[10] F. Rademacher, J. Sorgalla, and S. Sachweh, "Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective," *IEEE Softw.*, vol. 35, no. 3, pp. 36–43, May 2018, doi: 10.1109/MS.2018.2141028.

[11] P. Merson and J. Yoder, "Modeling Microservices with DDD," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Mar. 2020, pp. 7–8. doi: 10.1109/ICSA-C50368.2020.00010.

[12] C. K. Rudrabhatla, "Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture," *Int. J. Adv. Comput. Sci. Appl. IJACSA*, vol. 9, no. 8, Sep. 2018, doi: 10.14569/IJACSA.2018.090804.

[13] M. E. Elkholy and A. E. Fatatry, "Framework for Interaction between Databases and Microservice Architecture," *IT Prof.*, vol. 21, no. 5, pp. 57–63, 2019, doi: 10.1109/MITP.2018.2889268.

[14] S. Aydin and C. B. Çebi, "Comparison of Choreography vs Orchestration Based Saga Patterns in Microservices," in *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, Jul. 2022, pp. 1–6. doi: 10.1109/ICECET55527.2022.9872665.

[15] S. Speth, S. Stieß, and S. Becker, "A Saga Pattern Microservice Reference Architecture for an Elastic SLO Violation Analysis," in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, Mar. 2022, pp. 116–119. doi: 10.1109/ICSA-C54293.2022.00029.