Jurnal Infosecure Vol.6 No.2, Oktober 2025
p-ISSN: 2087-1023
MTI-UNLA

Arsitektur *MicroServices* dalam Integrasi Sistem untuk Peningkatan Layanan Asuransi Jasa Raharja

Arifin Hidayat[1], Hadi Prasetyo Utomo[2], Yucki Prihadi[3]

Program Studi Magister Teknik Informatika, Pascasarjana, Universitas Langlangbuana, Bandung, Indonesia

[1]bjulietoscar@gmail.com

[2]hadi@informatika.unla.ac.id

[3]yuckiprihadi@gmail.com

Abstrak Penelitian ini membahas penerapan arsitektur *microServices* dalam integrasi sistem untuk meningkatkan layanan asuransi di PT. Jasa Raharja. Dalam era digital yang semakin berkembang, terdapat kebutuhan mendesak untuk memperbaiki dan mempercepat layanan kepada pelanggan, terutama dalam sektor asuransi yang sangat kompetitif. Penelitian ini bertujuan untuk menganalisis bagaimana arsitektur *microServices* dapat diimplementasikan untuk mengatasi tantangan integrasi dan meningkatkan efisiensi operasional, serta kualitas layanan yang diberikan. Metodologi yang digunakan dalam penelitian ini meliputi pendekatan studi kasus, yang mencakup analisis sistem yang ada di PT. Jasa Raharja, identifikasi masalah yang dihadapi, dan eksplorasi solusi berbasis *microServices*. Melalui pengumpulan data kualitatif dan kuantitatif, serta wawancara dengan stakeholders, penelitian ini menemukan bahwa arsitektur *microServices* menawarkan fleksibilitas yang lebih besar, pengembangan yang lebih cepat, dan pemeliharaan yang lebih mudah dibandingkan dengan arsitektur monolitik tradisional. Hasil penelitian menunjukkan bahwa penerapan arsitektur *microServices* dapat mengurangi waktu respons layanan dan meningkatkan interaksi antar sistem, yang pada gilirannya berdampak positif pada kepuasan pelanggan. Dengan demikian, tesis ini merekomendasikan implementasi arsitektur *microServices* sebagai strategi inovatif untuk meningkatkan layanan di PT. Jasa Raharja dengan harapan memberikan kontribusi signifikan terhadap pengembangan industri asuransi di Indonesia.

Kata kunci Arsitektur *MicroServices*, Integrasi Sistem, Layanan Asuransi, PT. Jasa Raharja, Inovasi Teknologi.

I. PENDAHULUAN

PT. Jasa Raharja, sebagai penyedia layanan asuransi sosial yang berperan penting dalam perlindungan masyarakat terhadap risiko kecelakaan, harus mampu mengadaptasi teknologi informasi untuk meningkatkan kualitas layanan. Sistem informasi yang ada di PT. Jasa Raharja saat ini sering kali terintegrasi dengan cara yang konvensional dan monolitik, mengakibatkan beberapa kendala. Di antara kendala tersebut adalah keterbatasan dalam skalabilitas, kesulitan dalam penerapan fitur baru, serta permasalahan dalam pemeliharaan dan pengembangan sistem yang berkelanjutan.

Arsitektur *microService*s memungkinkan pengembangan aplikasi yang lebih modular, di mana setiap bagian dari sistem berfungsi secara independen, dan lebih mudah untuk dikembangkan serta dipelihara. Hal ini juga memberikan kemudahan untuk mengintegrasikan berbagai sistem yang ada, sehingga kolaborasi antara layanan dapat dilakukan dengan lebih efektif [1].

Dijelaskan bahwa kecepatan dan efisiensi adalah manfaat kunci dari penerapan *microService*s. Dragoni berargumen bahwa dengan memanfaatkan arsitektur *microService*s, PT. Jasa Raharja dapat mengintegrasikan dan mengotomatisasi proses bisnis yang sebelumnya berbelit-belit [2].

Dalam pandangan [3] dijelaskan bahwa arsitektur mikroservis memberikan kelebihan dalam mengisolasi kesalahan, yang berimplikasi langsung pada keandalan sistem. Dalam arsitektur monolitik, kegagalan di satu bagian dari aplikasi dapat mengakibatkan keseluruhan sistem gagal, menyebabkan downtime yang merugikan bagi bisnis. Sebaliknya, pada arsitektur mikroservis, komponen-komponen dapat gagal secara independen tanpa membahayakan aplikasi secara keseluruhan.

Arsitektur *microService*s adalah sistem pengembangan perangkat lunak yang dibangun dari susunan beberapa *Service* kecil yang berkomunikasi melalui HTTP dengan mekanisme sederhana [4]. Arsitektur *microService*s membuat pengembangan sebuah aplikasi menjadi lebih cepat, lebih mudah, dan dapat diperbaharui secara terpisah. Beberapa kelebihan *MicroService*s adalah pembuatan kode aplikasi yang lebih sedikit dan mandiri, perangkat lunak yang mudah pemeliharaannya, mudah diperluas, dan memudahkan pengembang menggunakan setiap bahasa pemrograman dan framework [5].

Keterkaitan tiga institusi dalam integrasi sistem untuk meningkatkan layanan penanganan pembayaran asuransi meliputi PT. Jasa Raharja, institusi kepolisian, dan rumah sakit memiliki ekosistem yang saling terkait dalam penanganan kasus kecelakaan. Ketika suatu kecelakaan terjadi, kepolisian bertugas untuk mengidentifikasi lokasi kejadian, menyelidiki penyebab, dan mengumpulkan data serta bukti yang diperlukan untuk proses hukum. Selanjutnya, data ini penting bagi PT. Jasa Raharja untuk memproses klaim asuransi yang diajukan oleh korban atau pihak yang terlibat.

II. METODE

1. Objek Penelitian

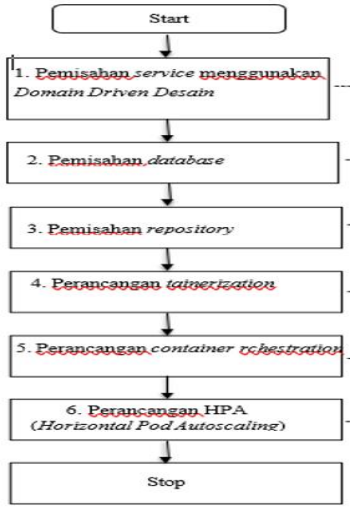
Fungsi PT. Jasa Raharja (Persero) ini berorientasi pada perintah Undang-undang No. 33 Tahun 1964 dan undang-undang No. 34 than 1964 tentang iuran dan sumbangan wajib untuk di pupuk dan dihimpun dan selanjutnya disalurkan kembali kepada masyarakat yang mengalami kecelakaan, sebagai Asuransi Jasa Raharja. Asuransi Jasa Raharja adalah perlindungan dan jaminan negara kepada rakyatnya yang mengalami kecelakaan, sedang obyeknya adalah manusia dan asuransi ini memberikan jaminan terhadap kerugian yang disebabkan oleh kecelakaan. Kerugian yang timbul dari kecelakaan dapat berupa meninggal, cacat sementara, cacat tetap, biaya pengobatan dan perawatan di rumah sakit.

2. Metode Penelitian

Penelitian rekayasa (engineering) adalah suatu kegiatan perancangan (design) yang tidak rutin, sehingga di dalamnya terdapat kontribusi baru, baik dalam bentuk proses maupun produk/prototipe [6]. Pada penelitian rekayasa, pembahasan kegiatan perancangan di dalamnya melibatkan hal-hal yang relatif baru, apabila kegiatan perancangan tersebut mengacu pada standar atau kode rancang bangun. Dalam Penelitian ini dilakukan dengan melakukan 6 tahapan dari mulai pemisahan monolith menjadi *microService* sampai pada akhirnya merancang arsitektur micre*Service* pada PT. Jasa Raharja yang memadukan kinerja dengan mitra yaitu institusi rumah sakit dan Kepolisian. Berikut tahapan-tahapan proses perancangan perubahan dari monolith ke *microService* [7].

3. Tahapan Penelitian

Dalam tahap ini menggambarkan serangkaian tahapan penelitian atau cara memecahkan masalah dalam proses penelitian ini seperti dalam gambar di bawah ini.



Gambar. 1 Tahapan Penelitian

1. Pemisahan *Service* Menggunakan *Domain Driven Design* (DDD)
   Untuk memisahkan setiap *Service* salah satunya dengan menggunakan pendekatan DDD. Untuk mendefinisikan DDD, pertama-tama harus menetapkan apa yang kita maksud dengan domain dalam konteks aplikasi [8]. Definisi kamus umum tentang domain adalah lingkungan pengetahuan atau aktivitas. Menelusuri sedikit dari itu, domain di ranah rekayasa perangkat lunak biasanya mengacu pada area subjek di mana aplikasi tersebut dimaksudkan untuk diterapkan [9].
2. Pemisahan *Database* Setiap *Service*
   Beberapa pattern *microService* ada yang menggunakan *database* secara berbagi dan ada juga yang terpisah berdasarkan layanannya. Untuk meningkatkan fleksibilitas baiknya setiap *database* khusus untuk menyimpan data dari sebuah *Service*. *Database* terpisah memiliki keuntungan setiap *database* dapat berdiri sendiri, mengurangi ketergantungan pada *database* lain, jika pun ada komunikasi data dilakukan antar *Service* melalui protokol *REST* atau RPC.
3. Pemisahan *Repository Source Code*
   MicroService juga dapat menyelesaikan masalah pada organisasi yang besar. Sebuah tim yang mempunyai jumlah yang besar akan kesulitan mengembangkan aplikasi dengan hanya sebuah repository untuk menyimpan kode. Setiap *Service* yang telah dipecah sebaiknya disimpan dengan repository yang berbeda antara satu dengan yang lainnya. Tujuan pemisahaan repository untuk memudahkan pengembangan aplikasi dengan tidak tercampurnya kode dan tidak saling tergantung secara keseluruhan. Repository yang umum dipakai adalah Github, Gitlab, *Bitbucket*. Repository dapat berupa disimpan sebagai project private ataupun publik.

4. Perancangan *Docker Container*
   Salah satu best practice pada arsitektur *microService* adalah penggunaan kontainer. Penggunaan kontainer sangat mempermudah ketika sebuah aplikasi dalam pengembangan dengan tidak ada isu lagi tentang di komputer lokal jalan, di server production aplikasi tidak jalan karena perbedaan environment [10].
5. Perancangan *Container Orchestration Kubernetes*
   Melakukan persiapan server master yang berfungsi sebagai node master, master di sini berfungsi untuk melakukan deploy kontainer ke dalam node worker Kubernetes. Persiapan selanjutnya adalah melakukan setup pada server node yang berfungsi sebagai *node worker*. Sebuah node dapat berupa VM atau server fisik tergantung dari kluster-nya [11]. Setiap node berisi beberapa object seperti pod, *Service*, volume, *namespace* dan lainnya yang diatur oleh komponen-komponen yang dimiliki oleh master node.
6. Perancangan *Horizontal Pod Autoscaler* (HPA) *Frame work* dengan Kubernetes
   Salah satu manfaat utama yang didapat dari Kubernetes adalah orkestrasi tingkat tinggi pada banyak kontainer. Secara khusus, kemampuan untuk secara otomatis scaling dan mengatur beban kerja yang dikerahkan pada sebuah server berjalan dapat menghilangkan kesulitan mengatur banyak kontainer pada sebuah aplikasi. Secara teori, kita dapat menentukan beberapa parameter yang mendorong aktivitas scaling dengan mendefinisikan matrik, lalu dengan otomatis Kubernetes melakukan pekerjaannya sebagai container orchestration manager yang dapat menambah atau mengurai kontainer berdasarkan metrik yang telah dicapai [12]. Jika tidak memiliki otomatisasi ini, maka seorang devops atau bagian infrastruktur harus menjalankan replika secara manual dan ini bisa sebanyak puluhan bahkan ratusan kali tergantung banyaknya beban server terpenuhi.

III. HASIL DAN PEMBAHASAN

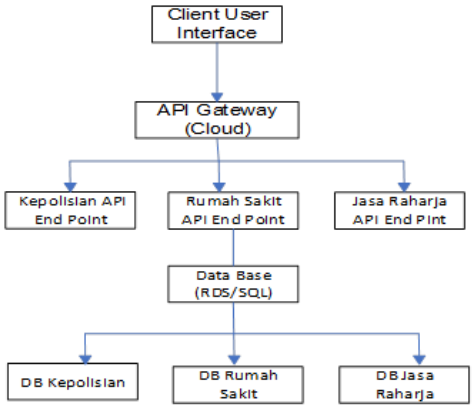
3.1 Analisis Integrasi Sistem di PT. Jasa Raharja

Demi meningkatkan efektivitas dan efisiensi layanan yang diberikan oleh Asuransi Jasa Raharja melalui integrasi sistem dengan institusi Kepolisian dan rumah sakit, berikut adalah serangkaian rekomendasi yang dapat diadopsi:

1. Pembangunan *Platform* Terintegrasi. Integrasi sistem di bangun menggunakan sebuah platform yang mengakomodir semua variable yang akan di integrasikan ke dalam sistem PT. Jasa Raharja dengan institusi terkait kepolisian dan rumah sakit agar lebih efektif fan efisiensi dalam komunikasi data.
2. Peningkatan Kapasitas Teknologi Informasi. Dalam meningkatkan kapasitas teknologi informasi perlu adanya beberapa penyempurnaan dalam sarana dan prasarana IT.
3. Standarisasi Proses dan Protokol. Dalam implementasi standarisasi proses dan protokol perlu adanya beberapa dokumen pelengkap.
4. Feedback dan Monitoring Berkelanjutan. Dalam penyempurnaan integrasi sistem perlu adanya mekanisme feedback dan monitoring dari pengguna.
5. Kolaborasi dan Komunikasi yang Efektif. Dalam mendukung integrasi sistem perlu adanya kolaborasi dan komunikasi antara institusi terkait dengan PT. Jasa Raharja.
6. Evaluasi dan Penyesuaian Sistem. Untuk mengevaluasi dan penyesuaian integrasi sistem.

3.2 Implementasi Integrasi Sistem di PT. Jasa Raharja

Implementasi integrasi sistem di PT. Jasa Raharja dalam meningkatkan efektivitas dan efisiensi layanan yang diberikan oleh Asuransi Jasa Raharja melalui integrasi sistem dengan institusi terkait kepolisian dan rumah sakit, yang tergambar seperti gambar di bawah ini.



Gambar. 2 Diagram Infrastruktur *Cloud* Integrasi Sistem PT. Jasa Raharja

Penjelasan Diagram:

1. Client *User* Interface: Memberikan layanan penggunaan sistem dari masing-masing institusi.
2. API *Gate Way* (Cloud) mengelola API yang akan mengintegrasikan data antar ketiga institusi. Fasilitas ini akan menawarkan keamanan, pemantauan, dan pembatasan tingkat akses.
3. Kepolisian API *End Point Application* Program Interface yang berisi code memberikan data kecelakaan.
4. Rumah Sakit API *End Point Point Application* Program Interface yang berisi code memberikan data Layanan.
5. Jasa Raharja API *End Point Point Application* Program Interface yang berisi code memberikan data Klaim.

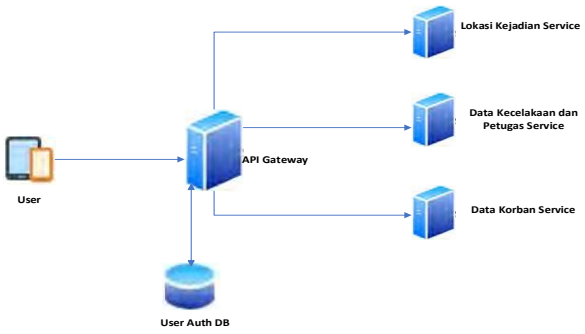
6. Data Base (RDS/SQL) Untuk penyimpanan data tidak terstruktur, Anda dapat memanfaatkan Amazon S3 atau *Google Cloud Storage.*
7. DB Kepolisian Penyimpanan Data Kecelakaan.
8. BD Rumah Sakit Penyimpanan Data Perawatan.
9. DB Jasa Raharja Penyimpanan Data Klaim.

3.3 Perancangan Arsitektur *MicroServices* PT. Jasa Raharja

PT. Jasa Raharja adalah lembaga yang memiliki tanggung jawab penting dalam menyediakan layanan perlindungan asuransi bagi masyarakat, khususnya dalam konteks kecelakaan lalu lintas dan transportasi. Dengan meningkatnya permintaan akan layanan yang cepat, akurat, dan responsif, serta kompleksitas sistem yang terus berkembang, PT. Jasa Raharja memerlukan transformasi digital untuk menghadapi tantangan ini. Salah satu pendekatan yang dipilih untuk mencapai hal ini adalah perancangan arsitektur *microServices*.

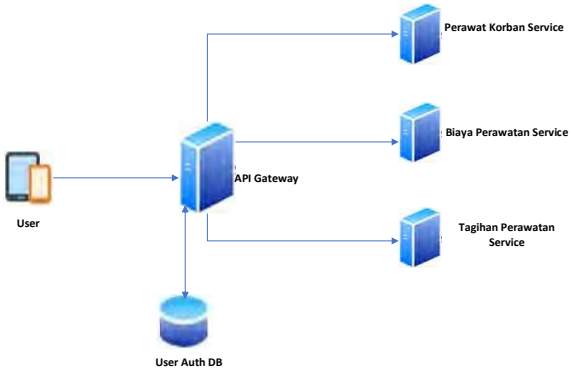
Perancangan arsitektur *microServices* di PT. Jasa Raharja merupakan langkah strategis untuk meningkatkan kualitas layanan dan responsivitas terhadap kebutuhan masyarakat. Dengan memanfaatkan pendekatan ini, PT. Jasa Raharja berpotensi untuk menjadi lembaga asuransi yang lebih inovatif, adaptif, dan efisien dalam menghadapi tantangan industri asuransi yang terus berubah.

1. Pemisahan *Service* Menggunakan *Domain Driven Design* (DDD)
   Pemisahan *Service* menggunakan DDD dilakukan untuk memecah *Service* sesuai dengan domain yang sama, *Service* dengan domain yang sama akan mempermudah proses bisnis sebuah fitur, karena sebuah domain hanya akan fokus mengembangkan fitur pada domain tersebut dan tidak berhubungan dengan domain lain.
   Mendefinisikan entitas kemudian melakukan pengecekan untuk fungsi agregat apakah ada antara *Service* yang terlibat. Aggregate dalam DDD merupakan sekelompok objek domain yang dapat diperlakukan satu unit. Dalam aggregate, dibuat satu kelas root yang memiliki identitas global untuk semua kelas dalam agregate tersebut, sedangkan kelas lainnya memiliki identitas lokal. Jika kita akan melakukan transaksi di kelas eksternal, kita hanya diizinkan untuk akses reference ke kelas root saja, tidak langsung ke reference kelas child.
   Mendefinisikan entities dilakukan berdasarkan domain yang sudah dipisah berdasarkan pada Gambar. 3. Kemudian adalah mendefinisikan *Service* yang juga berdasarkan pengelompokan domain. Terlihat *Service* yang akan dirancang adalah berjumlah 11 *Service* dengan ditambahkan API *Gateway* yang bertugas mengarahkan request data berdasarkan masing-masing *Service*.



Gambar. 3 Pemisahan *Service* Berdasarkan Domain pada Institusi Kepolisian

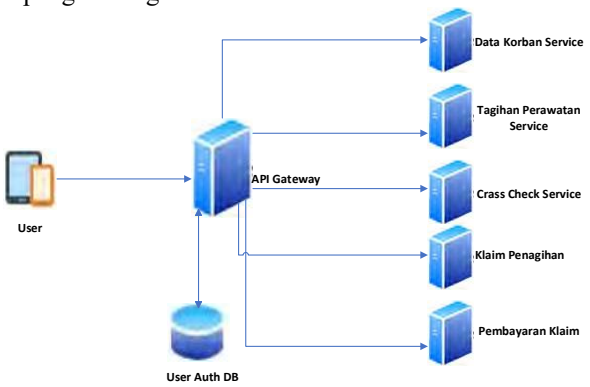
Gambar. 3 menunjukkan arsitektur pemisahan *Service* berdasarkan domain pada institusi kepolisian. Dalam arsitektur ini, pengguna (*user*) mengakses sistem melalui API *Gateway* yang menjadi pintu utama untuk menghubungkan berbagai layanan. API *Gateway* berfungsi sebagai pengelola permintaan yang kemudian diteruskan ke *Service* yang relevan, seperti Lokasi Kejadian *Service*, Data Kecelakaan dan Petugas *Service*, serta Data Korban *Service*. Untuk proses otentikasi dan otorisasi pengguna, sistem terhubung dengan *User* Auth DB yang menyimpan informasi identitas pengguna. Pemisahan *Service* berdasarkan domain ini memungkinkan pengelolaan data yang lebih terstruktur, skalabilitas yang lebih baik, serta mempermudah pengembangan dan pemeliharaan sistem secara modular.



Gambar. 4 Pemisahan *Service* Berdasarkan Domain pada Institusi Rumah Sakit

Gambar. 4 menggambarkan arsitektur pemisahan *Service* berdasarkan domain pada institusi rumah sakit. Pengguna (*user*) mengakses sistem melalui API *Gateway*, yang berfungsi sebagai titik masuk utama untuk semua permintaan layanan. API *Gateway* kemudian meneruskan permintaan ke layanan-layanan yang telah dipisahkan berdasarkan fungsinya, yaitu

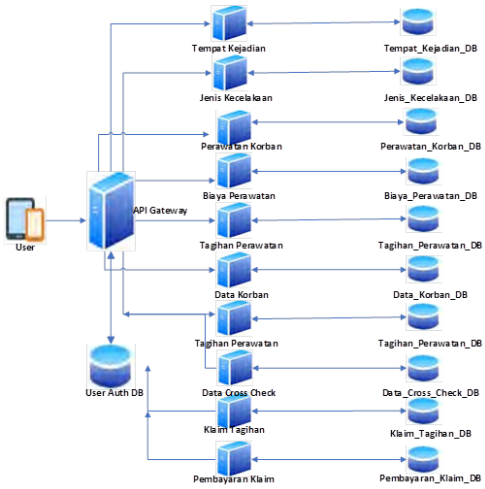
Perawat Korban *Service* untuk informasi terkait tenaga medis yang menangani pasien, Biaya Perawatan *Service* untuk pengelolaan data biaya pengobatan, serta Tagihan Perawatan *Service* untuk pengelolaan penagihan pasien. Sistem ini juga terhubung dengan *User* Auth DB untuk keperluan otentikasi pengguna. Pemisahan ini mendukung arsitektur *microServices* yang memberikan keuntungan dalam hal fleksibilitas, skalabilitas, dan efisiensi pengelolaan data serta pengembangan sistem rumah sakit.



Gambar. 5 Pemisahan *Service* Berdasarkan Domain pada PT. Jasa Raharja

Gambar. 5 menampilkan arsitektur pemisahan *Service* berdasarkan domain pada PT. Jasa Raharja. Dalam sistem ini, pengguna (*user*) mengakses layanan melalui API *Gateway* yang menjadi penghubung utama ke berbagai layanan internal perusahaan. Setiap layanan memiliki tanggung jawab domain spesifik, seperti Data Korban *Service* untuk menyimpan informasi korban, Tagihan Perawatan *Service* untuk mengelola data tagihan rumah sakit, *Cross Check Service* untuk melakukan verifikasi data, Klaim Pengajuan untuk menangani permohonan klaim, serta Pembayaran Klaim untuk memproses pencairan dana klaim. Semua proses diawali dengan otentikasi pengguna melalui *User* Auth DB. Pendekatan ini mencerminkan penerapan arsitektur *microServices* yang memudahkan pengelolaan sistem, meningkatkan efisiensi layanan, serta memisahkan tanggung jawab antar unit secara jelas dalam proses klaim dan penanganan kecelakaan.

2. Pemisahan *Database* Setiap *Service*
   Kondisi *database* saat ini adalah hanya menggunakan satu *database* yang digunakan untuk menyimpan seluruh data dari aplikasi. Pemisahan *database* dilakukan untuk memisahkan ketergantungan data dan memecah alur data sesuai tugas masing-masing *Service*. Pada Gambar. 6 telah dilakukan pengelompokan antara domain, aksi, aktor dan nama tabel yang selanjutnya tabel-tabel *database* tersebut akan dibuatkan *database* baru berdasarkan domain masing-masing.



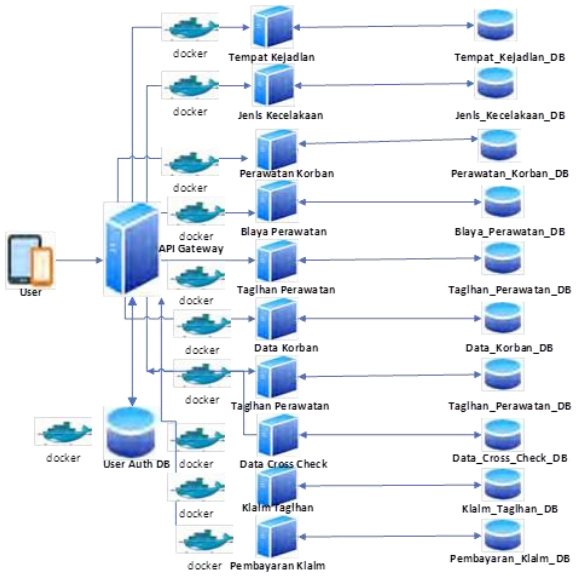
Gambar. 6 Pemisahan *Database*

3. Pemisahan *Repository Source Code*
   Pada tim pengembang aplikasi yang mempunyai jumlah anggota yang banyak akan kesulitan untuk melakukan koordinasi pada implementasi menulis kode program. Setiap kode dalam sebuah repository merupakan suatu kesatuan, jika terdapat error pada satu tempat maka akan berakibat error pada seluruh aplikasi. Pada tahapan ini akan dilakukan pemisahan repository yang sebelumnya mono repository menjadi multi repository. Adapun pemisahan didasarkan kepada domain yang sebelumnya telah dipisah. Mono-repo adalah pola kontrol sumber di mana semua kode disimpan dalam satu repository. Mudah untuk memberikan keseluruhan aplikasi kepada para pengembang dengan hanya sekali melakukan *clone* pada *version control system*.

4. Perancangan *Docker Container*
   Pada tahap perancangan *Docker* harus diinisialisasi pada semua *Service* yang dibuat. Penggunaan *Docker* dapat mempermudah dan mempercepat setup aplikasi pada sebuah server. Dahulu untuk membuat sebuah aplikasi berbasis web dibutuhkan beberapa konfigurasi yang dilakukan manual misal meng-install dahulu web server bisa *Apache* untuk *Linux* atau *Internet Information Services* (IIS) untuk *Windows*. Setelah *web server* siap baru selanjutnya membuat konfigurasi dari sisi aplikasi dan *database* yang harus dilakukan manual. Perbedaan *environment* seperti versi *database*, versi bahasa pemrograman, versi library yang dipakai sering membuat aplikasi error. Bayangkan jika terdapat kebutuhan untuk melakukan scaling dengan menambah

server, sebanyak itu pula harus melakukan konfigurasi manual.

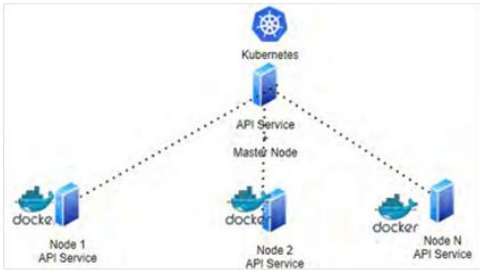
Penggunaan *Docker* akan meminimalkan setup yang dahulu dijalankan manual. *Docker* juga pasti menjalankan file konfigurasi yang sama Ketika aplikasi akan dilakukan *scaling*. Sebanyak apapun kebutuhan penambahan *server*, ketika sudah membungkus aplikasi atau *Service* menggunakan *Docker* akan lebih mudah dilakukan, karena tinggal menjalankan sebuah perintah *Docker* yang kemudian *Docker* akan mengeksekusi apa saja yang sudah ada di file konfigurasi *Docker-compose*.



Gambar. 7 Penggunaan *Docker* Container Pada Semua *Service*

5. Perancangan *Container Orchestration* Menggunakan *Kubernetes*

Untuk merancang arsitektur menggunakan *container orchestration* dibutuhkan sebuah master node dan minimal satu *worker* node. Untuk spesifikasi hardware yang digunakan sebaiknya memory minimal 2gb dan CPU memiliki 2 CPU.



Gambar. 8 Topologi Perancangan *Master Node* dan *Worker Node*

6. Perancangan *Horizontal Pod Autoscaler* (HPA) *Kubernetes*

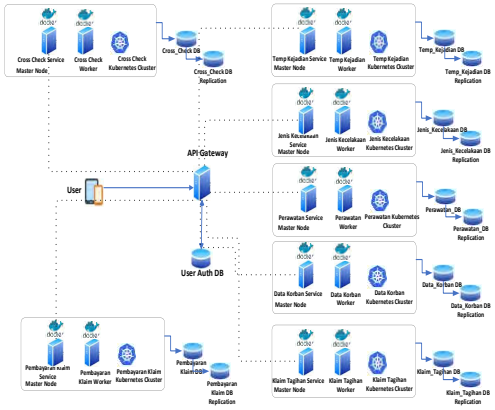
Jumlah pengguna akses aplikasi memang tidak bisa diprediksi secara tepat. Terkadang pemakaian aplikasi bisa sibuk dengan banyaknya pengguna, namun juga jumlah pengguna aplikasi terkadang sepi. Jika jumlah pengguna aplikasi sepi, aplikasi akan berjalan dengan lancar, namun Ketika jumlah pengguna meningkat drastis akan mengakibatkan konsumsi penggunaan memory atau cpu tinggi, maka akan ada kemungkinan performa aplikasi akan turun dan sulit diakses. Pada saat itulah scaling pada sisi infrastruktur diperlukan untuk menambah kapasitas server.

Scaling secara vertical mempunyai batasan untuk menambahkan kemampuan hardware. Misal sebuah server memiliki 4 slot RAM, saat ini diisi dengan 1 slot 8gb dan ketika dibutuhkan untuk proses *scaling* masing-masing diisi 8gb, maka scaling maksimal pada server tersebut adalah 4 x 8gb = 32gb.

Pada penggunaan scaling secara horizontal, proses *scaling* tidak memiliki batasan pada *hardware* sampai *server* yang dimiliki digunakan semua. Beban server dapat terbagi dengan penambahan *server* secara *horizontal.*

7. Rekomendasi Usulan Arsitektur *MicroService* PT. Jasa Raharja

Rekomendasi susulan berupa perubahan dari arsitektur monolitic ke *microService* harus dilakukan untuk mengantisipasi tantangan kebutuhan yang semakin meningkat baik dari jumlah pengguna maupun fitur aplikasi. Berdasarkan pengujian autoscaling dengan menggunakan container orchestration telah terbukti dapat menambah kemampuan aplikasi dalam menangani jumlah pengguna yang banyak. Kemudahan scaling yang dilakukan secara otomatis menjadi nilai lebih dalam rekomendasi usulan arsitektur di bawah ini.



Gambar. 9 Usulan Akhir Arsitektur *MicroService* PT. Jasa Raharja Menggunakan Container Orchestration

Berikut penjelasan dari usulan akhir arsitektur *microService* menggunakan *Container Orchestration Kubernetes* berdasarkan nomor pada Gambar. 9:

1. API *Gateway* bertugas untuk memecah aliran request yang berasal dari *frontend*,. Misal ada sebuah request dengan *wildcard /polling* maka api *Gateway* akan mengarahkan request tersebut kepada *Service* polling.
2. API *Gateway* akan melakukan pengecekan terhadap akses *user*. Jika berhasil *login* maka akan diteruskan ke request selanjutnya.
3. API *Gateway* meneruskan request menggunakan protokol HTTP REST API dengan fungsi-fungsi yang dipakai dapat berupa GET digunakan untuk mendapatkan data, POST untuk memasukan data, PUT untuk melakukan mengubah data, *DELETE* untuk melakukan penghapusan data.
4. Rekomendasi pada setiap domain harus memiliki sebuah klaster *Kubernetes* tersendiri, karena beban setiap domain pastilah berbeda.
5. Klaster *Kubernetes* berisikan *master node* dan *worker node*, file konfigurasi HPA tersimpan pada *master node*.
6. Penempatan konfigurasi *Docker container* terdapat di setiap *worker*.
7. *Database* yang digunakan hanya memiliki satu *server*. *Database* yang digunakan meliputi proses *read & write.*
8. Jika ke depan ada kebutuhan untuk memisahkan proses *read & write* dapat digunakan topologi *database slave master replication*, yaitu memisahkan antara proses *write* yang disimpan pada *database* master dan read yang disimpan pada *database slave*. DB *Replication* juga bisa berfungsi sebagai *Database* Backup dari *database* master atau utama.

IV. SIMPULAN DAN SARAN

4.1 Simpulan

Tesis ini telah berhasil menganalisis penerapan arsitektur *microService*s dalam integrasi sistem untuk peningkatan layanan di PT. Jasa Raharja. Berdasarkan studi kasus yang dilakukan, dapat disimpulkan bahwa:

1. Dengan konsep integrasi sistem dalam penelitian maka penulis dapat menggambarkan integrasi sistem berbagai layanan publik dalam meningkatkan klaim asuransi Jasa Raharja.
2. Arsitektur *microService*s memungkinkan pengembangan sistem secara independent dan dapat meningkatkan interoperabilitas, skalabilitas dan fleksibilitas dalam pertukaran data antara berbagai layanan publik di PT. Jasa Raharja yang sesuai dengan hasil penelitian yang telah dibahas.
3. Kesimpulan akhir manfaat bagi Perusahaan bukan hanya dapat meningkatkan efisiensi operasional, tetapi juga memberikan solusi jangka panjang untuk pengembangan layanan asuransi yang lebih responsif dan berbasis pelanggan. Penelitian ini merekomendasikan agar PT. Jasa Raharja melanjutkan proses transformasi ke *microService*s, dengan perhatian khusus pada pelatihan tim teknis dan penyediaan infrastruktur yang mendukung untuk mengoptimalkan implementasi sistem tersebut.

4.2 Saran

Berdasarkan temuan dan analisis yang dilakukan dalam tesis ini mengenai penerapan arsitektur *microService*s di PT. Jasa Raharja, berikut adalah beberapa saran yang dapat dijadikan pertimbangan untuk implementasi dan pengembangan lebih lanjut:

1. Disarankan agar PT. Jasa Raharja menginvestasikan dalam pelatihan bagi tim pengembang dan IT terkait arsitektur *microService*s. Pemahaman yang mendalam tentang prinsip-prinsip dan praktik terbaik dalam pengembangan *microService*s, termasuk alat dan teknologi yang relevan, akan sangat krusial untuk kesuksesan implementasi.
2. Dalam merancang *microService*s, penting untuk mempertimbangkan kebutuhan pelanggan secara menyeluruh. Penerapan pendekatan desain yang berfokus pada pengalaman pengguna (*user* experience) dapat membantu dalam menciptakan layanan yang tidak hanya efisien tetapi juga intuitif dan mudah diakses oleh pengguna akhir.
3. Penelitian ini dapat dikembangkan untuk kasus-kasus yang lain yang berhubungan dengan penggunaan *MicroService*.

REFERENSI

[1] Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

[2] Dragoni, N., et al. (2017). "*Microservices: Challenges and opportunities for a service-oriented architecture*". *Future Generation Computer Systems*, 109, 19-24.

[3] Chen, L., & Zhao, Z. (2019). "*Research on Microservices Architecture in the Insurance Industry". Proceedings of the 2019 IEEE International Conference on Software Engineering and Service Science* (ICSESS), 1-5.

[4] A.Qalam, 2022 Ilmiah Keagamaan dan Kemasyarakatan, and Z. Riko Virgiawan, "PERANCANGAN ARSITEKTUR BACKEND MICROSERVICE PADA *STARTUP CAMPAIGN.COM*," Jurnal Ilmiah Keagamaan dan Kemasyarakatan, vol. 16, no. 1, 2022.

[5] Afreen, S., & Yousuf, A. (2020). "*Analyzing the Impact of Microservices on Service Integration in Insurance Industry". Journal of Software Engineering and Applications,* 13(5), 218-227.

[6] Kurniawan, D., & Saraswati, I. (2021). "*Prototyping Method for Software Development: A Comparative Study". Journal of Computer Science and Technology*, 21(3), 1-10. doi:10.3844/jcsst.2021.1.10.

[7] Khodabocus, H., & Kalb, G. (2020). "*The Role of Prototyping in Engineering Project Development: A Systematic Review". Advances in Engineering Software, 146,* 102747. doi:10.1016/j.advengsoft.2020.102747.

[8] Raluca, M., & Patriţiu, L. (2021). "*Adopting Domain-Driven Design for Microservices: Challenges and Best Practices". Journal of Systems and Software,* 170, 110754. doi:10.1016/j.jss.2020.110754.

[9] Evans, E., & Ward, D. (2020). *Domain-Driven Design Distilled* (3rd ed.). Addison-Wesley.

[10] Morabito, R., & Palmisano, M. (2020). *"Containerization: The Future of Application Deployment and Management". Journal of Computer Networks and Communications,* Volume 2020, Article ID 9780821. doi:10.1155/2020/9780821.

[11] Wiggins, J., & Rakhmatullin, S. (2021). "*Advanced Kubernetes: Design, Build, and Manage Scalable Applications". Journal of Cloud Computing: Advances, Systems and Applications*, 10(1), 1-15. doi:10.1186/s13677-021-00225-w.

[12] Bhosale, A., & Deshmukh, Y. (2020). "*Dynamic Scaling of Kubernetes Deployments with Horizontal Pod Autoscaling". International Journal of Software Engineering & Applications* (IJSEA), 11(3), 29-45. doi:10.5121/ijsea.2020.11303.