Makalah Seminar SeNTIK 2017 – STMIK JAKARTA STI&K

26 Juli 2017

Perancangan Sistem Integrasi Pembuatan Buku Polis Elektronis menggunakan *Microservices*

Faisal Reza

Universitas Gunadarma

Jl. Margonda Raya no. 100

faisal.reza.160590@gmail.com

ABSTRAK

Microservices saat ini sedang menjadi tren dikalangan pengembang. Dari sisi perancangan tetntunya berbeda dengan bagaimana cara merancang sebuah aplikasi monolitik dirancang. Aplikasi berbasis *microservices* dirancang per proses yang berjalan didalamnya, mirip seperti merancang sebuah fungsi. *Microservices* sangat cocok untuk dunia bisnis yang seringkali menuntut perubahan. Dalam tulisan diterangkan bagaimana sebuah aplikasi yang mengintegrasikan sistem asuransi dengan korespondensi sistem. Terdapat 5 *microservices* untuk mendukung keselurahan proses. Dengan menggunakan *microservices* sebagai sebuah konsep pengembangan aplikasi maka tidak lagi ada perbedaan pengembang *back end* atau *middleware* karena *microservices* dapat menjadi keduanya.

Kata Kunci : *microservices*, asuransi, integrasi

Pendahuluan

Teknologi semakin berkembang untuk mendukung suatu kebutuhan, baik publik atau pribadi. Pada suatu perusahaan yang sudah berjalan puluhan tahun dan masih menggunakan teknologi generasi awal *client-server*. Pada zamannya teknologi *client-server,* lebih berfokus pada performa dan keamanan, namun teknologi terus berkembang dan kebutuhan semakin bervariasi.

Pada perusahaan asuransi buku polis merupakan sebuah kontrak kepada nasabah, buku polis digunakan sebagai pedoman tanggung jawab perusahaan terhadap nasabahnya. Beberapa tahun yang lalu seorang nasabah perlu menunggu hingga 1-2 bulan sampai menerima buku polis karena buku polis hanya ada dalam bentuk cetak. Sehingga terkadang muncul asumsi bahwa nasabah tidak dapat mengklaim resiko karena mereka belum memiliki buku polis. Berdasarkan hal tersebut, maka, semakin cepat sebuah buku polis diterima oleh nasabah, maka nasabah merasa aman terhadap perlindungan resiko.

Kesenjangan teknologi antara sistem inti dengan sistem pendukung harus segera diatasi untuk merealisasikan pemenuhan kebutuhan nasabah terhadap buku polis. Sistem korespondensi diketahui dapat menerima permintaan dengan menggunakan protokol *message broker*. Sistem inti pun dapat mengeluarkan data menggunakan *message broker*. Namun data dari sistem inti tidak serta merta dapat digunakan oleh sistem korespondensi.

Microservices merupakan sebuah konsep yang sedang digemari pengembang aplikasi. Namun, *microservices* lebih sering diimplementasikan menjadi sebuah *webservices* berbasis REST API yang tentunya tidak dapat secara langsung mengintegrasikan kedua sistem.

Karya Terkait

Saat ini, teknologi ilmu komputer tidak hanya harus menangani perkembangan terbaru. Namun, harus dapat mengatasi perbedaan teknologi yang digunakan dari mulai yang tradisional hingga teknologi terkini. Tren terbaru seperti *Internet of Things*, *cloud computing* dan *microservices* menjadi pilihan bagi para pengembang. Dalam [1] penulis menyampaikan sebuah model komputasi terdistribusi dan *middleware*nya untuk menangani *High Performance Computing* (HPC). Mereka memperkirakan bahwa cara yang mereka gunakan dapat menjamin efisiensi dan dapat diperbesar sesuai kebutuhan sedangkan model terdisitribusi dapat menjamin HPC dengan baik.

Sistem inti asuransi memiliki keterbatasan dalam melakukan integrasi, pilihan yang tersedia adalah menghasilkan sebuah *file* atau menggunakan *message broker*, dalam [2] penulis mempresentasikan sebuah solusi untuk desain komunikasi berbasis *message*. Dengan

digunakannya metode yang mereka tulis dapat dijamin sebuah sistem yang andal, ringan dan independen dari *platform*. Desain yang mereka buat menutup celah antara solusi berorientasi *message* kelas berat dengan banyak aplikasi sederhana yang membutuhkan komunikasi satu arah.

Kebutuhan akan sistem yang andal, mampu berkembang dan dapat digunakan ulang menjadi kunci pemilihan arsitektur *microservices*. Seperti yang diungkapkan penulis pada artikel [5], kenyatannya, bukan tentang mengikuti apa yang paling modern dan sedang tren, yang lebih penting adalah untuk melakukan pengecekan apakah aplikasi dapat berfungsi dengan sempurna dan efisien menggunakan arsitektur terpilih.

Dalam penelitiannya [3], disampaikan sebuah desain *middleware* berbasis *message* untuk *cluster* yang menydiakan efektifitas dan efisiensi untuk mengatasi permasalahan. Mereka juga menambahkan mekanisme *dynamic load balancing* berdasarkan faktor yang terkena imbas dari penggunaan *message broker*.

Metode Penelitian

Dalam proses perancangan ini, penulis berencana menggunakan metode *agile* [6] sebagai metode pengembangan aplikasi, dimana proses perancangan dan pengembangan dilakukan dengan informasi yang seadanya dan dibentuk sebagai sebuah *story* dan dijalankan dalam *sprint* yang singkat. Lalu berdasarkan *story* yang telah didapatkan pengembangan dijalankan dan menghasilkan sebuah *microservices* [4].

Analisis *Output* Sistem Inti Asuransi

Output dari sistem inti asuransi yang sudah terdefinisi dikirim ke *message broker*. *Microservices* membaca dan menggunakannya sebagai *input* untuk korespondensi sistem. *Message* yang dikirimkan oleh sistem inti asuransi merupakan pemecahan dari informasi polis yang terpecah menjadi beberapa baris terpisah dalam satu *file* teks. Setiap produk memiliki jumlah baris yang berbeda-beda namun beberapa baris untuk masing-masing memiliki kode baris yang sama (kode baris umum). Dalam sebuah *message* setiap informasi dipisahkan menggunakan delimeter tabulasi. Untuk mengaitkan antara 1 *message* dengan *message* yang lain digunakan informasi pada *field* ke 2 setiap *message*, yaitu nomor Surat Pengajuan Asuransi Jiwa(SPAJ). Karena penggunaan *message broker* dan sistem pembaca *message broker* bisa ditambah sesuai kebutuhan sehingga pesan dari *message broker* tidak akan bisa diurutkan dimana pada sistem terdahulu *message* dikeluarkan dari sistem inti asuransi dengan urutan tertentu. Sebuah *message* tambahan yang disebut *COMPLETED message* dibutuhkan untuk mengurutkan kembali data yang terlah tersimpan.

Analisis *Input* Korespondensi Sistem

Korespondensi sistem adalah produk yang digunakan pada suatu perusahaan untuk menghasilkan dokumen elektronis yang digunakan untuk informasi dari perusahaan ke nasabah.

Input untuk korespondensi sistem terdiri dari informasi polis berupa *file* teks dan dokumen pendukung dalam bentuk *file* pdf. Didalam *file* teks terdiri dari beberapa baris yang memiliki kegunaan masing-masing pada template yang telah dibuat untuk setiap produk.

Analisis Kebutuhan *Microservices*

Diketahui bahwa *Sistem inti asuransi* tidak dapat langsung berkomunikasi dengan korespondensi sistem. *Microservices* yang bersifat sebagai *middleware* dipisahkan menjadi 5 fungsi utama, yaitu jembatan untuk mengambil *message* dari *message broker*, proses integrasi pembuatan buku polis, jembatan ke *document management system* dimana lokasi *file* pendukung berada, layanan untuk menentukan versi dokumen yang digunakan dalam pencetakan dan jembatan ke korespondensi sistem.

Sebagai sistem yang mengambil setiap *message* dari *message broker* diperlukan *library* seperti *Java Message Services*(JMS). *Message* yang masuk satu persatu tidak dapat langsung digunakan untuk membuat buku polis di korespondensi sistem maka diperlukan *database* untuk menyimpan *message* sementara menunggu *message* telah diterima semuanya oleh *middleware*. Selain itu, terdapat *message COMPLETED* yang menandakan bahwa semua *message* terlah terkirim dan *middleware* dapat mentriger untuk menjalankan proses integrasi pembuatan buku polis di *microservices*.

Sebagaimana kebutuhan dari *microservices* sebagai jembatan, dimana perlu menyimpan sementara. Untuk menyimpannya diperlukan sebuah layanan untuk menyimpan *message*-

message tersebut. Selain itu diperlukan layanan untuk menjalankan proses integrasi yaitu melengkapi kebutuhan *input* korespondensi sistem. Dalam proses integrasi dilakukan beberapa kegiatan diantaranya, menyusun uratan baris *message*, menambahkan baris yang berisikan informasi versi dokumen yang dibutuhkan dan menambahkan baris yang berisikan nama *file* dari setiap dokumen pendukung. *Microservices* juga mengirimkan *file* ke lokasi dimana korespondensi sistem dapat membaca dokumen pendukung melalui *Secure File Transfer Protocol*(SFTP).

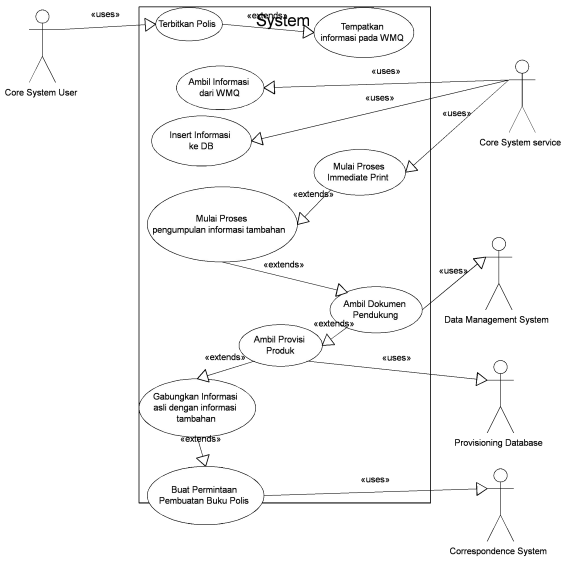
Layanan provisi dokumen memiliki *table* untuk menyimpan versi. Layanan ini dipublikasi sebagai sebuah REST API menggunakan format JSON untuk permintaan dan responnya. Dengan begitu lebih mudah bagi *microservices* untuk mengkonsumsi informasinya karena dapat dipetakan sebagai sebuah objek dibahasa pemrograman seperti Java.

Selanjutanya terdapat layanan untuk menghubungkan proses integrasi ke *Document Management System*(DMS). Layanan ini juga dipublikasi sebagai REST API menggunakan format JSON. Setiap dokumen pendukung memiliki folder dan metadata yang digunakan untuk memfilter informasi yang dicari. Layanan yang dibuat perlu didefinisikan secara umum agar pengguna layanan dapat leluasa dalam mencari dokumen yang dibutuhkan dengan kombinasi berupa nama folder dan pencarian berdasarkan metadata yang dibutuhkan.

Terakhir ada *microservices* untuk menghubungkan ke korespondensi sistem. Dipublikasi menggunakan REST API, sehingga dapat dipanggil menggunakan protokol http ketika dibutuhkan. Layanan ini menkonversi setiap panggilan dari *client* ke format yang dipahami oleh korespondensi sistem. Format tersebut akan dikonsumsi dan didistribusikan ke variabel-variabel yang sudah didefinisikan dikorespondensi sistem. Komunikasi dari *microservices* ke korespondensi sistem menggunakan protocol *message broker*, yang bersifat *asynchronous*.

Use Case Diagram

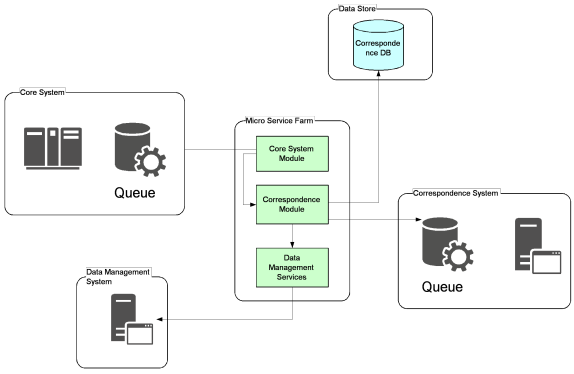
Berdasarkan kebutuhan yang telah didefinisikan dapat dibuat *use case diagram* sebagai berikut :



Gambar 1 Use Case Diagaram

Arsitektur Aplikasi

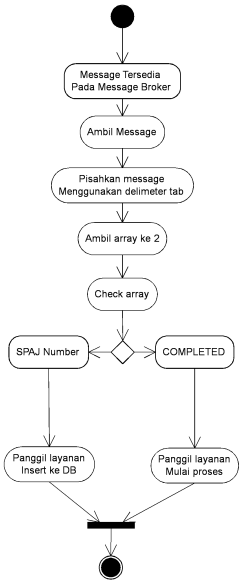
Sistem yang dibuat mengintegrasikan beberapa sistem, agar lebih mudah dipahami berikut adalah arsitektur aplikasi yang direncanakan untuk diimplementasikan :



Gambar 2 Arsitektur Aplikasi

Process Konsumsi *Message* Dari *Sistem inti asuransi*

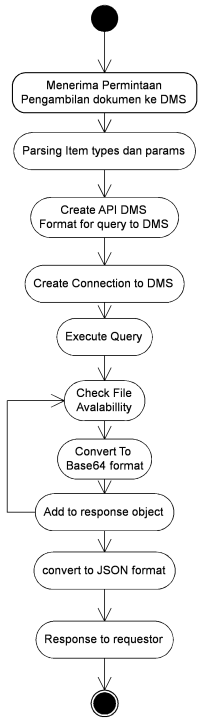
Message broker merupakan sebuah *environment* yang pasif dimana, setiap *message* hanya akan diantrikan dalam sebuah q*ueue*. *Queue* harus dikonsumsi oleh sebuah sistem agar *message*nya bisa diproses. Dalam *sistem inti asuransi* service, proses didefinisikan sebagai berikut :



Gambar 3 Flow Konsumsi *Message* dari *Sistem inti asuransi*

Proses *Document Management System* Service

Korespondensi sistem diketahui membutuhkan *input* berupa dokumen yang disisipkan dalam buku polis elektronis. DMS memiliki *Application Programing Interface*(API) untuk bisa berintegrasi. Dengan menggunakan API tersebut sebuah service integrasi ke DMS dibuat. Karena DMS tidak hanya dipakai oleh sistem integrasi ini, desain REST API untuk service ini harus bersifat umum, oleh karena itu disiapkan dua buah *parameter* dalam JSON yaitu, itemTypes dan params. Item types mendefinisikan folder lokasi filter yang dikemas dalam bentuk array string sedangkan params dikemas dalam bentuk array object berupa map key value. Kedua *parameter* itu mendukung berbagai kebutuhan pengguna untuk mencari dokumen dalam DMS dan meresponsenya dalam format JSON dimana *file* akan dikonversi dari *file* menjadi *string base64*.



Gambar 4 Flow Pengambilan Dokumen Pendukung

Proses Provisi Dokumen Service

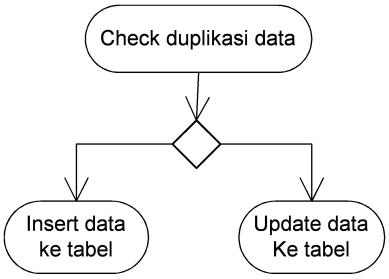
Provisi dokumen bisa dilakukan dengan mengetahui kapan tanggal polis terbentuk dalam *sistem inti asuransi* dan disesuaikan dengan produk yang dipilih. Service ini memiliki sebuah *string* koneksi ke *database* dan memaping sebuah *table* didalamnya. Proses ini memiliki 2 *parameter input* untuk melakukan *query* ke *database*.

Proses Layanan Integrasi Korespondensi

Korespondensi sistem merupakan produk yang dibuat sebuah *vendor* untuk melakukan pembuatan dokumen elektronis. Dengan *template* yang sudah dibuat dan data yang telah dipetakan sebelumnya, proses ini bisa berjalan dengan baik. Sedangkan proses *trigger* dari *outbond environment* sendiri dimungkinkan dengan cara yang sudah ditentukan oleh sistem itu sendiri. Korespondensi sistem dapat menerima *request* melalui beberapa cara yaitu SOAP API dan *message broker*. *Message broker* dipilih karena dinilai dapat menstabilkan permintaan yang masuk ke dalam sistem dengan cara mengantrikannya dalam sebuah *queue* dan hanya dapat memprosesnya paling banyak 20 *request* dalam 1 waktu. Format *message* yang dikirim dikemas dalam bentuk XML.

Proses Pencetakan Polis Elektronis

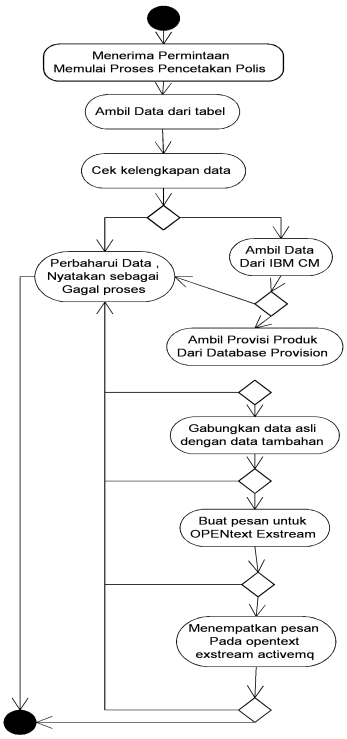
Seperti diketahui bahwa proses bisa berjalan dengan adanya *record* dalam *database*. Dimana *record-record* tersebut adalah *message* yang dikirim dari *sistem inti asuransi* ke *message broker* dan dilempar ke service *insert* to policy *table*. Ketika sudah lengkap proses dilanjutkan dengan proses penysunan dan melengkapi informasi polis. Berikut proses dalam service penyimpanan *message* :



Gambar 5 Proses Penyimpanan *Message*

Setiap *request* akan dicek SPAJ numbernya sudah pernah di*insert* atau belum di*table*, jika sudah maka proses menjadi update sedangkan jika belum ada maka proses menjadi *insert*.

Berikut adalah proses dalam proses penysunan dan melengkapi data polis sebelum dikirim ke korespondensi sistem :



Gambar 6 Proses Integrasi Pencetakan Polis

Perancangan *Database*

Dalam proses penyusunan dan melengkapi kebutuhan *input* korespondensi sistem, diperlukan penyimpanan sementara yang menyimpan *message* dari *sistem inti asuransi* selama proses berlangsung. Selain itu juga diperlukan *log table* untuk mencatat kejadian yang berlangsung ketika proses berjalan. *Table* untuk mencatat lokasi hasil pembuatan polis juga dibutuhkan untuk melayani permintaan sistem lain untuk mengambil buku polis. Terakhir adalah *table* provisi yang mencatat versi setiap dokumen dari seluruh produk.

Perencanaan Implementasi

Setiap proses dan layanan dikembangkan dengan konsep *microservices*. Bahasa yang digunakan adalah *java* dengan framework . Hal ini dikarenakan *springboot* dapat berdiri sendiri dalam sebuah *server*. *Springboot* sudah memiliki *tomcat* atau *jetty* didalamnya sehingga tidak perlu mempersiapkan *environment* yang tepat untuk aplikasi.

Simpulan

Dalam jurnal diterangkan banyak proses yang dipecah menjadi *microservices*. Penggunaan *microservices* sebagai suatu konsep pengembangan perangkat lunak membantu projek untuk bisa lebih fleksibel terhadap kebutuhan bisnis. Apabila ada perubahan pada suatu *resource*s pun, proses inti tidak perlu terganggu asalkan format untuk berkomunikasi tidak berubah pada *resource* yang baru. Selain sebagai proses inti *microservices* disini juga berperan sebagai *middleware*, dengan begitu *skill set* bagi para pencari kerja semakin terbuka lebar karena dengan menggunakan *microservices* tidak diperlukan kemampuan tambahan untuk membuat sebuah layer *middleware* dalam sistem. Dengan dibuatnya sistem sesuai rancangan maka proses pembuatan buku polis terutama polis elektronis dapat dilakukan dalam hitungan menit, sehingga apabila terjadi resiko, nasabah dapat melakukan klaim sesuai yang tercantum dalam buku polis dengan bukti yang sah dan diakui oleh perusahaan asuransi. Bagi perusahaan asuransi, desain ini bisa membagi beban kerja sistem dimana sebelumnya meningkat pada satu waktu karena proses *batch job* yang jalan sekali setiap hari.

Daftar Pustaka

[1] Benchara*, F. Z., Youssfi, M., Bouattane, O., & Ouajji, H., " A New Efficient Distributed *Computing Middleware* based on *Cloud* Micro-*Services* for HPC", 2016 5th International Conference on Multimedia *Computing* and *System*s (ICMCS), 2016.

[2] Chen, G., Du, Y., Qin, P., & Zhang, L., "Research of JMS based *Message* Oriented *Middleware* for *Cluster*", International Conference on Computational and Information Sciences , pp. 1628-1631, 2013.

[3] Ivaki, N., Laranjeiro, N., & Araujo, F., "Towards Designing Reliable Messaging Patterns", IEEE 15th International Symposium on Network *Computing* and *Application*s, pp. 204-207, 2016 .

[4] Langer, A. M, Guide to Software Development : Designing and Managing the Life Cycle, Springer, New York, 2016.

[5] Salah, T., Zemerly, M. J., Yeun, C. Y., AI-Qutayri, M., & AI-Hammadi, Y., " The Evolution of Distributed *System*s Towards *Microservices* Architecture", The 11th International Conference for Internet Techno*log*y and Secured Transactions 318-325, 2016.

[6] Schmidt, C, Agile Software Development Team : The Impact of Agile Development on Team Performance, Springer, Mannheim, 2016.