PENGEMBANGAN ARSITEKTUR MICROSERVICE DI PT. HATSONSURYA ELECTRIC UNTUK PENINGKATAN SKALABILITAS DAN KEMAMPUAN BERADAPTASI LAYANAN

Gafri Putra Aliffansah, Tubagus Mohammad Akhriza dan Dwi Safiroh Utsalina

STMIK Pradnya Paramita
Jl. Laksda Adi Sucipto No.249A, Pandanwangi, Blimbing, Malang, Jawa Timur
gafri.putra@gmail.com, akhriza@stimata.ac.id

ABSTRAK

Untuk saat ini web service (backend service) masih menggunakan konsep monolithic architecture, yang artinya semua fitur ada di dalam 1 projek. Sehingga mengalami beberapa kesulitan seperti ketika melakukan pembaharuan sistem, keseluruhan aplikasi harus dimatikan terlebih dahulu untuk memperbarui kodenya, sehingga terjadilah down time secara keseluruhan aplikasi dan karena semua fitur dalam 1 projek yang sama, maka bahasa pemrograman dan framework haruslah sama, sehingga terjadilah keterbatasan dalam mengimplementasikan berbagai teknologi yang berbeda didalam satu aplikasi. Monolithic architecture merupakan pendekatan pengembangan aplikasi di mana komponen atau layanan dari fungsi besar disimpan dalam bundel atau folder. Monolithic architecture mempunyai beberapa keuntungan - keuntungan seperti kemudahan proses deployment, tidak perlu memelihara banyak service atau aplikasi. Salah satu yang sukses mendapatkan keuntungan dari monolitik yaitu aplikasi ACM News. Microservices architecture merupakan pendekatan pengembangan aplikasi di mana aplikasi besar dibangun sebagai rangkaian layanan kecil modular dan setiap layanan berdiri sebagai layanan mandiri. Microservices pada umumnya diimplementasi di level Enterprise. Contoh seperti MGDIS SA – perusahaan software editing yang menggunakan arsitektur microservices – dan Hazmat Environmental – aplikasi pelaporan pengiriman barang kimia – merupakan aplikasi yang menerapkan arsitektur microservices dan berhasil memperoleh keuntungan yang didapatkan dari microservices architecture seperti penggunaan kembali fungsi, pengembangan aplikasi, dan kemudahan skalabilitas dari setiap komponen [4]. Berdasarkan keuntungan kedua arsitektur di atas, pada penelitian kali ini penulis akan menganalisis kemudahan skalabilitas dan penerapan teknologi yang berbeda disetiap service atau aplikasi di PT. HATSONSURYA ELECTRIC, sehingga masalah down time secara keseluruhan sistem dapat diminimalisir, dan dapat dengan mudah mengimplementasikan berbagai teknologi di setiap service/aplikasinya.

Kata Kunci: *Arsitektur Microservice, Arsitektur Monolitik, Skalabilitas.*

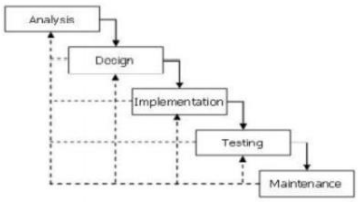
PENDAHULUAN

Di era teknologi dan informasi saat ini, hampir setiap aspek aktivitas di bidang apapun ditentukan oleh teknologi dan kualitas informasi yang kita terima dan hasilkan. Dalam hal ini komputer memberikan informasi yang ada dan memegang peranan yang sangat penting sebagai alat penunjang kinerja perusahaan. Perkembangan teknologi juga terjadi pada sisi arsitektur atau sistem desain dalam pengembangan aplikasi, yaitu arsitektur microservice. Arsitektur microservice semakin populer, dan pengadopsi melihat kesuksesan besar. Laporan "Microservices Adoption in 2020", berdasarkan jajak pendapat 1.500 insinyur perangkat lunak, sistem dan arsitek teknis, insinyur dan pembuat keputusan, menyatakan bahwa lebih dari tiga perempat (77 persen) bisnis kini telah mengadopsi layanan mikro. Dari mereka yang mengadopsi arsitektur microservice, hampir semua (92 persen) melaporkan tingkat keberhasilan yang tinggi. Lebih jauh lagi, sebagian besar bisnis (29 persen) bertaruh besar pada teknologi, tampak memindahkan sebagian besar sistem mereka ke layanan-layanan mikro (menjadi bagian-bagian kecil) [4]. Jika dalam arsitektur monolithic aplikasi terdapat dalam satu repository project yang besar, ketika perubahan pada satu bagian dari kode program memiliki pengaruh yang signifikan pada kode program lainnya. Sedangkan arsitektur microservice ini memiliki konsep aplikasi yang dibagi menjadi bagian-bagian

kecil yang memiliki fungsi tertentu dan tidak bergantung pada komponen program lain, konsep tersebut bertujuan agar sistem yang dibangun bisa menangani kegagalan total jika terdapat satu aplikasi yang bermasalah. Membagi aplikasi ke dalam layanan mikro memiliki keunggulan teknis yang berbeda, seperti fleksibilitas yang lebih besar, pemantauan dan pengelolaan yang lebih mudah, dan peningkatan kinerja, terutama untuk toko online dengan lalu trafik yang tinggi. Arsitektur microservice merupakan alternatif arsitektur yang lebih terukur dan lebih fleksibel [1]. Pada arsitektur microservice, sistem aplikasi dirancang untuk terdistribusi dan menyediakan layanan secara lebih spesifik. Fitur yang besar akan dipecah menjadi beberapa fitur kecil yang disusun dalam satu servis, dimana setiap servis memiliki ruang lingkup tersendiri. Dengan pendekatan ini, suatu sistem aplikasi akan terdiri dari beberapa service yang saling berkomunikasi dan dapat dikelola, didistribusikan secara independen, hal ini akan lebih memudahkan sistem untuk beradaptasi terhadap perubahan kebutuhan. Berdasarkan penelitian sebelumnya, arsitektur microservice sebagai sebuah arsitektur sistem yang terdistribusi telah menunjukkan peningkatan kualitas pada aspek resiliensi dengan studi kasus sistem penjualan, evaluasi model dilakukan melalui pembuatan proof of concept dari modifikasi arsitektur software yang terdistribusi berbasis microservice [3]. Kualitas ketahanan ini ditunjukkan ketika beberapa node layanan mengalami gangguan dan sistem dapat melanjutkan dapat berlanjut. Proses migrasi dari sistem berbasis monolitik ke sistem berbasis microservice dilakukan dalam 6 tahap. (Microservices Migration Patterns). Penelitian [5] telah mengembangkan metode migrasi dari arsitektur monolitik ke arsitektur microservice melalui 6 (enam) tahapan, dimana studi kasusnya adalah sistem perbankan yang mengelola 3,5 juta data nasabah dan hampir 2 juta transaksi per harinya. Identifikasi microservice telah berhasil dilakukan dengan menganalisis keterhubungan pada setiap subsistem (SS), fungsi bisnis (F,B), dan tabel database(D).

METODE PENELITIAN

Dalam membangun Arsitektur Microservice metode penelitian yang digunakan adalah model SDLC (Software Development Life Cycle). System Development Life Cycle (SDLC) adalah proses pembuatan dan pengubahan sistem serta model dan metodologi yang digunakan untuk mengembangkan sebuah sistem. SDLC juga merupakan pola yang diambil untuk mengembangkan sistem perangkat lunak, yang terdiri dari tahap – tahap : rencana (planning), analisis (analysis), desain (design), implementasi (implementation), uji coba (testing) dan pengelolaan (maintenance).



Gambar 1. *Metode penelitian*

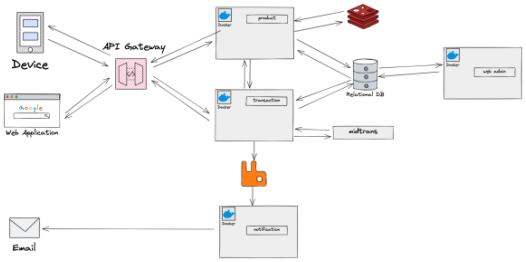
Tahap perencanaan (planning) dimulai dengan menentukan persyaratan sistem yang berlaku untuk perangkat lunak. Hal ini sangat penting mengingat perangkat lunak harus dapat berinteraksi dengan elemen lain seperti perangkat keras, database, dsb [4]. Fase ini sering disebut sebagai project definition. Pada fase ini, peneliti mengamati konsep arsitektur monolithic dan microservices. Pengamatan ini berfokus pada masalah yang terjadi ketika proses transaksi terjadi sangat banyak dalam waktu bersamaan. Fase analisis (analysis) mengintensifkan proses menemukan kebutuhan dan berfokus pada perangkat lunak. Untuk mengetahui jenis program apa yang harus dibuat, insinyur perangkat lunak perlu memahami domain informasi perangkat lunak. Berdasarkan pengamatan mereka, peneliti menganalisis bahwa ketika sebuah toko online menerima pesanan dalam jumlah besar, beban pada aplikasi tinggi. Kebutuhan akan fungsionalitas perangkat lunak untuk memenuhi kendala yang dihadapi pelanggan saat melakukan transaksi. Pada fase desain (design) proses ini digunakan untuk

mengubah kebutuhan fase diatas menjadi representasi ke dalam bentuk "blueprint" aplikasi sebelum coding dimulai. Desain harus dapat mengimplementasikan kebutuhan yang telah disebutkan pada tahap sebelumnya. Seperti 2 sebelumnya, maka proses ini juga harus didokumentasikan sebagai konfigurasi dari aplikasi. Pada fase implementasi (implementation) untuk dapat dimengerti oleh komputer, maka desain tadi harus diubah bentuknya menjadi bentuk yang dapat dimengerti oleh komputer, yaitu ke dalam bahasa pemrograman melalui proses penulisan bahasa pemrograman. Tahap ini merupakan implementasi dari tahap desain yang secara teknis nantinya dikerjakan oleh programmer [6]. Pada fase ini, peneliti membangun sebuah aplikasi berdasarkan desain "blueprint" yang telah dibuat. Pengembangan aplikasi ini dilakukan dari awal hingga aplikasi siap dijalankan. Pada fase uji coba (testing) semua fungsi-fungsi aplikasi harus dilakukan uji coba secara menyeluruh, agar software bebas dari error, dan hasilnya harus benar-benar sesuai dengan kebutuhan yang sudah didefinisikan sebelumnya. Setelah proses pembangunan aplikasi selesai, peneliti melakukan pengujian pada tahap ini. Aplikasi diuji menggunakan

HASIL DAN PEMBAHASAN

1. Analisis Kebutuhan
   Analisis kebutuhan fungsional dilakukan untuk memberikan gambaran tentang kemampuan sistem yang dirancang sehingga dapat mengatasi masalah yang mungkin dihadapi :
   - Setelah meneliti, mengamati, dan mengelaborasi proses bisnis di awal proses aplikasi sehingga Anda dapat memprediksi ukuran aplikasi, jumlah pengguna, dan jumlah trafik pengunjung, pemilik aplikasi harus memulai dengan arsitektur microservice.
   - Menggunakan arsitektur microservices membuat aplikasi lebih mudah dipahami karena layanan yang dibuat relatif kecil. Sebaliknya, arsitektur monolitik menghasilkan paket aplikasi yang sangat besar dan dapat membingungkan programmer untuk membaca alur aplikasi rumit.
   - Lebih mudah untuk dikembangkan, dipelihara, dan deploy. Aplikasi disimpan relatif kecil, membuat pengembangan lebih mudah, dan tidak perlu khawatir tentang mengubah satu fitur yang mempengaruhi yang lain, dan mereka dikelola secara otomatis, menyederhanakan proses deployment.
   - Lebih mudah untuk mengganti teknologi sesuai kebutuhan, jika pada awal pembuatan menggunakan bahasa pemrograman C# ketika kita ingin melakukan pengembangan teknologi dan ingin berganti bahasa pemrograman maka itu menjadi lebih mudah dan effort yang dibutuhkan modul bernama locust dengan bahasa pemrograman python untuk membandingkan kecepatan trafik data pada arsitektur monolitik dan arsitektur microservice.

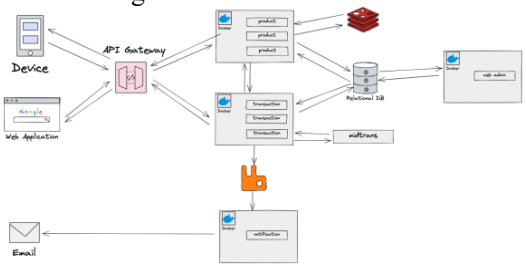
   Pada fase pemeliharaan (maintenance) tim diperlukan ketika adanya perubahan dari eksternal toko online seperti ketika ada pergantian sistem operasi, teknologi, atau perangkat lainnya. Peneliti belum sampai pada tahap ini, sehingga tahap ini belum terlaksana. tidak terlalu besar karena aplikasi yang dibuat relatif kecil, jika menggunakan arsitektur monolitik lalu ingin merubah bahasa pemrograman maka harus seluruh sistem dibangun ulang dengan teknologi atau bahasa pemrograman tersebut.



Gambar 2. *Penerapan perbedaan teknologi pada Arsitektur Microservice.*

- Mudah discale sesuai kebutuhan, pada Arsitektur Microservice jika kita ingin melakukan scaling pada fitur penjualan maka hanya service fitur penjualan yang dilakukan scaling, jadi tidak perlu seluruh fitur dilakukan scaling.

2. Perancangan Arsitektur



Gambar 4. *Arsitektur Microservice dengan skalabilitas*

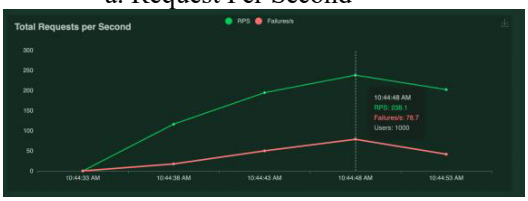
Pada gambar arsitektur atau sistem desain pada gambar 4 terlihat rancangan arsitektur microservice yang akan diimplementasikan, terlihat pada gambar aplikasi dibagi menjadi beberapa aplikasi kecil yaitu, Admin Service, Product Service, Transaction Service, Notification Service, sedangkan untuk teknologi pendukung pada Arsitektur Microservice terdapat API Gateway dan Message broker. Berikut adalah penjelasan masing – masing service dan teknologi pendukung yang digunakan :

- Admin Service Berfungsi untuk dashboard admin yang mengatur produk dan transaksi yang masuk. Service ini menggunakan bahasa pemrograman PHP dan Laravel sebagai frameworknya.
- Product Service Berfungsi untuk menghandle data produk yang akan ditampilkan ke user seperti judul, deskripsi, harga, gambar, dan detail produk lainnya. Service ini menggunakan bahasa pemrograman go dengan gin gonic sebagai frameworknya
- Transaction Service Pada service ini berfungsi untuk menghandle data transaksi yang dilakukan user ketika proses checkout.. Service ini menggunakan bahasa pemrograman go dengan fiber sebagai frameworknya. Service ini terintegrasi dengan pihak ke-3 yaitu midtrans sebagai payment gatewaynya. Dan juga melakukan publish message ke message broker (RabbitMQ) untuk keperluan notifikasi agar prosesnya dilakukan dengan asynchronous.
- Notification Service Setelah data masuk melalui transaction service, notification service akan menerima message berupa data transaksi dari message broker (RabbitMQ) yang selanjutnya akan dilakukan pengiriman email tentang detail transaksi dan detail pembayarannya. Service ini menggunakan bahasa pemrograman go dengan integrasi smtp.
- API Gateway Berfungsi sebagai gerbang antara client dengan internal service, service ini berfungsi untuk routing, logging, rate limiter, authentication, dan load balancer. Service ini menggunakan teknologi KONG Api Gateway.
- Message Broker Message broker digunakan sebagai jembatan komunikasi antara transaction service dengan notification service, message broker menggunakan istilah producer dan consumer dimana transaction service sebagai produser dan notification service sebagai consumer. Pada implementasinya saat user melakukan transaksi order maka transaction service akan mengirimkan data ke message broker dan message broker akan meneruskannya ke notification

service. Keuntungan menggunakan message broker adalah tidak ketergantungan service antara transaction service dengan notification service, jadi ketika notification service mengalami masalah atau servernya mati maka transaction service bisa tetap melakukan transaksi. Lalu bagaimana dengan notifikasi yang seharusnya dikirim setelah melakukan pemesanan tetapi notification service mati, pada saat notification service mati maka data – data pesan yang akan dikirimkan ke notification service akan di simpan di message broker, lalu jika notification service sudah diperbaiki dan menyala kembali maka pesan – pesan yang sebelumnya berada di message broker akan diteruskan ke notification service [2]. Service ini menggunakan teknologi RabbitMQ / Apache Kafka. 3. Performance Test Pada tahap pengujian yang dilakukan yaitu masing – masing arsitektur akan dilakukan performance test dengan 10000 data menggunakan modul Locust (Python). Pada pengujian akan dilakukan oleh 100 user, kemudian diberlakukan jeda akses antar user dengan total jeda 100 detik, karena user yang mengakses sebanyak 100 orang dan total jeda 100 detik maka jeda akses antar user adalah 1 detik (100 detik / 100 user), lalu diterapkan seberapa banyak user akan mengakses yaitu 100 kali. Dapat disimpulkan bahwa pada pengujian ini akan dilakukan oleh 100 user dengan jeda masing – masing user selama 1 detik dan masing – masing user melakukan akses sebanyak 100 kali pada akhirnya akan dilakukan pengujian sebanyak 10000 http request. Pada pengujian tersebut dibuat skenario saat user ingin melihat data produk sebanyak 100 data. Adapun pengujian service terbagi menjadi 3 jenis, yaitu. monolitik, microservice tanpa skalabilitas, dan microservice dengan skalabilitas (3 duplicate node disetiap servicenya).

i. Monolitik

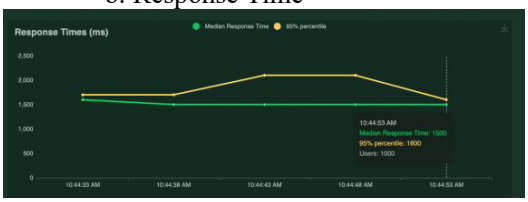
a. Request Per Second



Gambar 5.1 *Hasil benchmark request per second untuk arsitektur monolitik*

Dari hasil pengujian Request Per Second (RPS) untuk arsitektur Monolitik, di dapatkanlah hasil tertingginya yaitu 238.1 request per detik.

b. Response Time



Gambar 5.2 *Hasil benchmark response time untuk arsitektur monolitik*

Dari hasil pengujian Response Time untuk arsitektur Monolitik, di dapatkanlah hasil response time tercepatnya adalah 1600 ms (95% request yang berhasil).

ii. Microservice tanpa skalabilitas

a. Request Per Second

Gambar 5.3 *Hasil benchmark request per second untuk arsitektur microservice*

Dari hasil pengujian Request Per Second (RPS) untuk arsitektur microservice tanpa skalabilitas, di

dapatkanlah hasil tertingginya yaitu 351.5 request per detik.

b. Response Time

Gambar 5.4 *Hasil benchmark response time untuk arsitektur microservice*

Dari hasil pengujian Response Time untuk arsitektur Monolitik, di dapatkanlah hasil response time tercepatnya adalah 1100 ms (95% request yang berhasil).

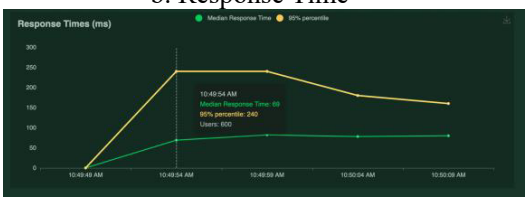
iii. Microservice dengan skalabilitas

a. Request Per Second

Gambar 5.5 *Hasil benchmark request per second untuk arsitektur microservice dengan skalabilitas*

Dari hasil pengujian Request Per Second (RPS) untuk arsitektur Microservice tanpa skalabilitas, di dapatkanlah hasil tertingginya yaitu 2404.9 request per detik.

b. Response Time



Gambar 5.6 *Hasil benchmark response time untuk arsitektur microservice dengan skalabilitas*

Dari hasil pengujian Response Time untuk arsitektur monolitik, di dapatkanlah hasil response time tercepatnya adalah 240 ms (95% request yang berhasil).

PENUTUP

Dari proses analisis, perancangan, implementasi dan pengujian yang dilakukan pada bab-bab sebelumnya, maka dapat diambil kesimpulan sebagai berikut:

1. Konsep arsitektur microservice menjadi salah cara untuk mengatasi trafik yang besar secara bersamaan, selain itu juga lebih mudah untuk mengembangkan fitur karena sudah dipecah menjadi bagian bagian kecil.
2. Memanfaatkan docker container sebagai wadah untuk proses deploy, karena lebih mudah untuk melakukan proses deployment dan skalabilitas.
3. Telah dihasilkan perbandingan antara penerapan arsitektur microservice dengan arsitektur monolitik dengan merujuk pada hasil dari pengujian. Pada hasil pengujian didapati arsitektur microservice memiliki hasil performance test yang lebih baik dibanding arsitektur monolitik.

No	Service	Request Per Second	Response Time (ms)
1	Monolithic	238.1	1600
2	Microservice	351.5	1100
3	Microservice With Scale	2404.9	240

4. Berhasil mengimplementasikan berbeda bahasa pemrograman dan framework di setiap servicenya. Pada implementasinya arsitektur microservice digunakan ketika sudah mendapat trafik yang sangat besar dan berimbas pada performa aplikasi yang menjadi lambat, server berulang kali terjadi downtime karena tidak kuat menangani trafik yang besar, semakin

bertambahnya fitur – fitur sehingga struktur aplikasi menjadi sangat besar, dengan semakin bertambah fitur bertambah juga programmer yang menghandle aplikasi tersebut sehingga sering terjadi konflik dalam pengerjaannya.

Untuk pengembangan sistem selanjutnya, dapat diberikan saran-saran sebagai berikut :

1. Perlunya Unit & Integration Test, agar ketika mengembangakan fitur baru, selalu memastikan fitur yang sudah ada masih sesuai dengan semestinya.
2. Perlunya benchmark test, untuk melihat performa disetiap servicenya.
3. Perlunya proses CI/CD dalam proses deployment, agar menghemat banyak waktu dan resource untuk proses testing dan deployment.
4. Perlu dikembangkan lagi untuk mengintegrasikan Docker Container dengan Kubernetes, agar proses scale up/down tiap servicenya berjalan dengan otomatis.

DAFTAR PUSTAKA

[1] A. S., Rosa dan Shalahuddin, M, 2013, “Rekayasa Perangkat Lunak Terstruktur Dan Berorientasi Objek", Informatika, Bandung.

[2] C. Okutan and N. K. Cicekli, 2010, "A monolithic approach to automated composition of semantic web services with the Event Calculus," Knowledge-Based Systems, vol. 23, no. 5, pp. 440-454.

[3] Gata, Windu dan Gata, Grace, 2013, “Sukses Membangun Aplikasi Penjualan dengan Java”, Elex Media Komputindo, Jakarta.

[4] Newman, S, 2015, “Building Microservices”, O’Reilly Media, Inc.

[5] Priyadarshini, S.(2017). Microservices Architecture. International Research Journal of Computer Science (IRJCS), Issuwe 04, Volume 4.

[6] Teguh Wahyono, 2004 : 14, “Sistem Informasi (Konsep Dasar, Analisis Desain dan Implementasi)”.