Nama: Muhammad Syauqi Syafiq

NIM: 2211081019

Kelas: TRPL-3D

Resume: "***Why Breaking Up Your Monolith Can Kill Your Project: Mistakes you can't afford to make***"

Banyak perusahaan yang ingin meninggalkan sistem lama(legacy system) monolith, karena monolith tidak sustainable, dibebani juga dengan tantangan dalam maintenance dan perlahan rusak. Monolith dianggap system lama yang ketinggalan zaman.

Penggunaan microservice telah menjadi trend di industry saat ini. Banyak yang mengatakan "*Microservices are the future!"* atau "*Everyone is doing microservices these days*". Namun ada keputusan penting yang harus di pertimbangkan dengan baik sebelum memindahkan sistem lama monolith menjadi microservices, karena sangat kompleks dan penuh dengan potensi kegagalan.

Ada banyak keuntungan dalam menggunakan microservices, meskipun banyak keuntungan ketika menggunakan microservice, namun bukan solusi umum untuk menggunakannya. Kita harus memiliki plan dan pemahaman yang dalam mengenai seluk beluk sistem kita karena ketika gagal dalam mengatur sistem kita di microservice dapat menyebabkan masalah yang tidak terduga dan memungkinkan melumpuhkan project kita.

Berikut adalah 9 pertimbangan penting dalam menentukan apakah tetap menggunakan monolith atau migrasi ke microservice:

1. **Tidak Mempertimbangkan Kekurangan Microservice**
   Migrasi membuat kompleksitas baru, seperti ketergantungan pada komunikasi jaringan yang berisiko memengaruhi kinerja jika desain layanan buruk atau jaringan tidak andal. Selain itu, dibutuhkan mekanisme penanganan kegagalan seperti retry, load balancer, dan circuit breaker.
   Pengelolaan error juga lebih rumit karena log yang tersebar di berbagai layanan, Monitoring dan alerting menjadi lebih menantang karena setiap layanan perlu diawasi secara independen. CI/CD yang dulunya sederhana menjadi lebih kompleks karena layanan harus dibangun, diuji, dan dideploy secara terpisah.
   Dari sisi biaya, infrastruktur untuk mendukung microservices bisa jauh lebih mahal dibandingkan monolith.
   Jadi kita membutuhkan perencanaan yang matang dan pemahaman yang mendalam untuk memitigasi tantangan ini agar sukses dalam menerapkan microservices dalam system kita.

2. **Tidak Memiliki Alasan Yang Kuat Untuk Menggunakan Microservice**
   Pemilihan antara monolith dan microservices harus didasarkan pada kebutuhan proyek. Monolith unggul dalam kesederhanaan teknis, sedangkan microservices menawarkan skalabilitas, isolasi data, dan otonomi tim, tetapi membawa kompleksitas tambahan.

   Alasan valid untuk memilih microservices meliputi:

- **Deploy tanpa downtime** untuk pengembangan berkelanjutan.
- **Isolasi data** untuk memenuhi regulasi.
- **Otonomi tim** dalam pengelolaan layanan.

Tanpa alasan kuat, monolith atau modular monolith lebih cocok, terutama pada awal proyek. Struktur organisasi juga perlu diperhatikan agar selaras dengan arsitektur yang dipilih

3. **Tidak Mempertimbangkan Struktur Organisasi**

   Keberhasilan microservices sangat dipengaruhi oleh struktur organisasi. Dalam tim kecil (misalnya 5 orang), microservices seringkali tidak efisien karena menambah kompleksitas tanpa manfaat yang sebanding, seperti kebutuhan mengelola banyak repository bahkan untuk perbaikan kecil.

   Sebaliknya, dalam organisasi besar, seperti di sektor perbankan, microservices efektif karena memungkinkan tim bekerja secara mandiri tanpa koordinasi yang intens. Prinsip Conway's Law menegaskan bahwa desain sistem mencerminkan struktur komunikasi organisasi: tim kecil cenderung menghasilkan sistem terdistribusi, sementara tim besar condong ke monolith.

   Keselarasan antara arsitektur dan struktur organisasi sangat penting untuk memastikan efisiensi dan hasil yang optimal.

4. **Tidak Menghindari Monolith Yang Terdistribusi**

   Distributed monolith terjadi ketika layanan dalam microservices tetap saling terikat erat, sehingga kompleksitas meningkat tanpa manfaat penuh dari microservices, seperti skalabilitas dan toleransi kesalahan.

   Masalah utama distributed monolith:

   - Kompleksitas tinggi, manfaat minim: Ketergantungan antar layanan menghambat skalabilitas dan kemandirian.
   - Kesulitan deployment: Perubahan di satu layanan memengaruhi layanan lain, mempersulit deployment independen.
   - Tidak dapat skalabilitas independen: Beban layanan saling memengaruhi, menciptakan bottleneck.
   - Pengaruh pada toleransi kesalahan: Ketergantungan jaringan meningkatkan risiko kegagalan sistem.
   - Hambatan kerja tim: Batasan layanan yang tidak jelas menyulitkan tim bekerja secara mandiri.

   Pencegahan distributed monolith memerlukan perencanaan matang, prinsip desain seperti low coupling dan high cohesion, serta penerapan domain-driven design untuk mendefinisikan batasan layanan dengan jelas. Microservices yang dirancang baik memberikan modularitas, skalabilitas, dan kemandirian yang optimal.

5. **Tidak Mempertimbangkan Data Ketika Membagi Kode**

   Kesalahan umum dalam penerapan microservices adalah membagi kode tanpa memperhatikan interaksi data. Penggunaan database bersama oleh beberapa layanan dapat menciptakan distributed monolith secara tidak langsung, menghadirkan sejumlah tantangan:

- Kompleksitas perubahan database: Perubahan pada database memengaruhi banyak layanan, memerlukan koordinasi yang rumit.
- Kinerja: Hubungan antar data (join/foreign key) dapat memperlambat performa dan meningkatkan latensi.
- Integritas data: Relasi yang rumit dapat merusak konsistensi data, memerlukan solusi seperti soft delete atau penggunaan referensi nama.
- Transaksi terdistribusi: Menyinkronkan perubahan antar database mempersulit penerapan sifat ACID dan meningkatkan risiko deadlock.

Pendekatan alternatif seperti saga pattern atau restrukturisasi microservices dapat membantu menghindari kompleksitas ini. Perencanaan data yang matang memastikan arsitektur microservices tetap efisien dan sesuai dengan tujuan bisnis.

6. **Dekomposisi Tanpa Tujuan Yang Jelas**

Dekomposisi microservices tanpa visi yang jelas dapat menimbulkan masalah. Proses ini harus dilakukan secara strategis dan bertahap, dengan fokus pada nilai dan kemudahan ekstraksi.

Pendekatan dekomposisi yang efektif:

- Mulai dengan bagian sederhana: Ekstraksi bagian monolith yang terisolasi dan mudah dipisahkan.
- Gunakan grafik ketergantungan: Prioritaskan bagian dengan banyak ketergantungan keluar, yang menjadi pusat bagi layanan lain.
- Pendekatan bertahap: Bangun layanan secara bertahap dengan menangani ketergantungan secara hati-hati.

Dengan perencanaan dan evaluasi yang matang, dekomposisi dapat menghasilkan arsitektur microservices yang terstruktur dan efisien.

7. **Tidak Ada Peningkatan Migrasi**

Kesalahan besar dalam migrasi ke microservices adalah melakukan transformasi secara langsung dan menyeluruh ("big-bang rewrite"). Pendekatan ini cenderung memperbesar risiko masalah di produksi, mengingat semakin banyak layanan berarti semakin banyak potensi gangguan.

Pendekatan bertahap lebih bijak, dengan langkah-langkah berikut:

- Strangler Fig Pattern: Ekstraksi layanan secara bertahap dari monolith sambil mengarahkan lalu lintas melalui proxy. Layanan baru secara perlahan menggantikan fungsi monolith.
- Feature Toggle: Gunakan mekanisme untuk memilih antara monolith dan layanan baru selama transisi.
- Fokus pada pelanggan: Pastikan migrasi membawa manfaat nyata bagi pengguna tanpa mengganggu pengembangan fitur baru.

Dengan strategi ini, sistem dapat bertransisi ke arsitektur microservices secara aman, efektif, dan berkelanjutan.

8. **Tidak Mempertimbangkan Perjalanan Pelanggan**

Kesalahan umum dalam microservices adalah membagi layanan berdasarkan entitas, yang sering kali menyebabkan distributed monolith dengan masalah seperti: Ketergantungan antar layanan yang erat, Kesulitan dalam skalabilitas dan toleransi kesalahan.

Solusi yang lebih efektif adalah membagi layanan berdasarkan perjalanan pelanggan (customer journey). Pendekatan ini menyesuaikan layanan dengan skenario pengguna nyata, sehingga memberikan pengalaman yang lebih baik.

9. **Tidak Mempertimbangkan Gaya Arsitektur Lainnya**

   Kesalahan mendasar dalam arsitektur perangkat lunak adalah mengabaikan gaya arsitektur alternatif selain microservices. Seperti ungkapan lama "Nobody gets fired for buying IBM," keputusan sering kali didasarkan pada keamanan dan popularitas, bukan pada kecocokan dengan kebutuhan bisnis.

   Dengan membuka wawasan terhadap berbagai pilihan arsitektur, organisasi dapat membangun fondasi digital yang lebih tangguh dan berkelanjutan. Arsitektur yang tepat adalah yang mendukung tujuan bisnis dengan optimal, baik saat ini maupun di masa depan.

Kesimpulan

Dalam perjalanan dari monolith ke microservices, tidak ada jawaban universal. Setiap proyek memiliki kebutuhan dan preferensi unik. Memahami kekuatan masing-masing arsitektur dan mengutamakan pendekatan yang disesuaikan dengan tujuan klien serta keahlian tim.

Dalam dunia pengembangan perangkat lunak yang dinamis, keputusan arsitektur yang matang adalah kunci keberhasilan. Dengan pemahaman mendalam dan pendekatan yang hati-hati, proyek dapat berkembang baik dengan monolith maupun microservices, selama pilihan tersebut mendukung tuntutan unik proyek.