Mengurai monolit menjadi layanan mikro

AWS Panduan Preskriptif

AWS Panduan Preskriptif: Mengurai monolit menjadi layanan mikro

Table of Contents

Mengurai monolit menjadi layanan mikro

Tabby Ward dan Dmitry Gulin, Amazon Web Services ()AWS

April 2023 (riwayat dokumen)

Migrasi ke Amazon Web Services (AWS) Cloud memiliki banyak keuntungan, termasuk kelincahan teknis dan bisnis, peluang pendapatan baru, dan pengurangan biaya. Untuk mendapatkan manfaat penuh dari keuntungan ini, Anda harus terus memodernisasi perangkat lunak organisasi Anda dengan memfaktorkan ulang aplikasi monolitik Anda ke dalam layanan mikro. Proses ini terdiri dari tiga langkah utama:

- Mengurai monolit menjadi layanan mikro — Gunakan pola dekomposisi yang disediakan oleh panduan ini untuk memecah aplikasi monolitik menjadi layanan mikro.
- Integrasikan layanan mikro — Integrasikan layanan mikro yang baru dibuat ke dalam arsitektur layanan mikro dengan menggunakan AWS layanan tanpa server.
- Aktifkan persistensi data untuk layanan mikro — Promosikan persistensi polyglot di antara layanan mikro Anda dengan mendesentralisasi penyimpanan data.

Modernisasi biasanya melibatkan dua jenis proyek:

- Proyek Brownfield melibatkan pengembangan dan penyebaran sistem perangkat lunak baru dalam konteks sistem yang ada atau lama.
- Proyek Greenfield melibatkan pembuatan sistem dari awal untuk lingkungan yang sama sekali baru, tanpa melibatkan kode lama.

Untuk proyek brownfield, salah satu langkah pertama dalam perjalanan modernisasi aplikasi Anda adalah menguraikan monolit dalam portofolio Anda menjadi layanan mikro.

Sebagian besar aplikasi dimulai sebagai monolit yang dirancang untuk kasus penggunaan bisnis tertentu. Jika arsitektur monolit tidak menerapkan desain modular, monolit dapat tetap menjadi pilihan yang valid untuk aplikasi yang tidak memiliki tanggung jawab yang jelas dalam batas-batas pengetahuan domain yang mapan. Karakteristik utama monolit sebagai satu unit penyebaran juga dapat membantu mengurangi kekurangan desain, seperti kopling ketat atau kurangnya struktur internal.

Meskipun monolit dapat menjadi pilihan yang valid untuk beberapa kasus penggunaan, biasanya tidak cocok untuk aplikasi modern. Struktur internal monolit yang tidak terdefinisi dengan baik dapat menyulitkan pemeliharaan kode, yang menciptakan kurva pembelajaran yang curam bagi pengembang baru dan menyebabkan biaya dukungan tambahan. Kopling tinggi dan kohesi rendah dapat secara signifikan meningkatkan waktu yang diperlukan untuk menambahkan fitur baru, dan Anda mungkin tidak dapat menskalakan komponen individual berdasarkan pola lalu lintas. Monolit juga membutuhkan banyak tim untuk berkoordinasi untuk satu rilis besar, yang meningkatkan beban kolaborasi dan transfer pengetahuan. Akhirnya, Anda dapat menemukan bahwa menambahkan fitur baru atau membangun pengalaman pengguna baru menjadi sulit ketika bisnis atau basis pengguna Anda tumbuh.

Untuk menghindari hal ini, Anda dapat menggunakan pola dekomposisi untuk memecah aplikasi monolitik, mengubahnya menjadi beberapa layanan mikro, dan memigrasikannya ke arsitektur layanan mikro. Arsitektur microservices menyusun aplikasi sebagai serangkaian layanan yang digabungkan secara longgar. Layanan mikro dirancang untuk mempercepat pengembangan perangkat lunak dengan memungkinkan proses pengiriman berkelanjutan dan penyebaran berkelanjutan (CI/CD).

Sebelum Anda memulai proses dekomposisi, Anda harus mengevaluasi monolit mana yang akan terurai. Pastikan untuk menyertakan monolit yang memiliki masalah keandalan atau kinerja, atau sertakan beberapa komponen dalam arsitektur yang digabungkan dengan erat. Kami juga menyarankan agar Anda memahami sepenuhnya kasus penggunaan bisnis untuk monolit, teknologinya, dan ketergantungannya dengan aplikasi lain.

Panduan ini untuk pemilik aplikasi, pemilik bisnis, arsitek, prospek teknis, dan manajer proyek. Ini membahas enam pola cloud-native berikut yang digunakan untuk menguraikan monolit, dan menjelaskan kelebihan dan kekurangan masing-masing:

- Terurai oleh kemampuan bisnis
- Terurai oleh subdomain
- Terurai oleh transaksi
- Layanan per pola tim
- Pola ara pencekik
- Cabang dengan pola abstraksi

Panduan ini merupakan bagian dari seri konten yang mencakup pendekatan modernisasi aplikasi yang direkomendasikan oleh. AWS Serial ini juga mencakup:

- Strategi untuk memodernisasi aplikasi di Cloud AWS
- Pendekatan bertahap untuk memodernisasi aplikasi di Cloud AWS
- Mengevaluasi kesiapan modernisasi untuk aplikasi di Cloud AWS
- Mengintegrasikan layanan mikro dengan menggunakan layanan tanpa server AWS

Hasil bisnis yang ditargetkan

Anda harus mengharapkan hasil berikut setelah Anda menguraikan monolit Anda menjadi layanan mikro:

- Transisi efisien aplikasi monolitik Anda menjadi arsitektur layanan mikro.
- Penyesuaian cepat terhadap permintaan bisnis yang berfluktuasi tanpa mengganggu aktivitas inti, seperti skalabilitas tinggi, peningkatan ketahanan, pengiriman berkelanjutan, dan isolasi kegagalan.
- Inovasi yang lebih cepat, karena setiap layanan mikro dapat diuji dan digunakan secara individual.

Pola untuk membusuk monolit

Setelah Anda memutuskan untuk menguraikan monolit dalam portofolio aplikasi Anda, Anda harus memilih pola dekomposisi yang sesuai dan memasukkannya ke dalam organisasi Anda.

> **Note**
>
> Anda dapat menggunakan beberapa pola untuk menguraikan monolit. Misalnya, Anda dapat menguraikan monolit berdasarkan kemampuan bisnis dan kemudian menggunakan pola subdomain untuk memecahnya lebih banyak.

Topik

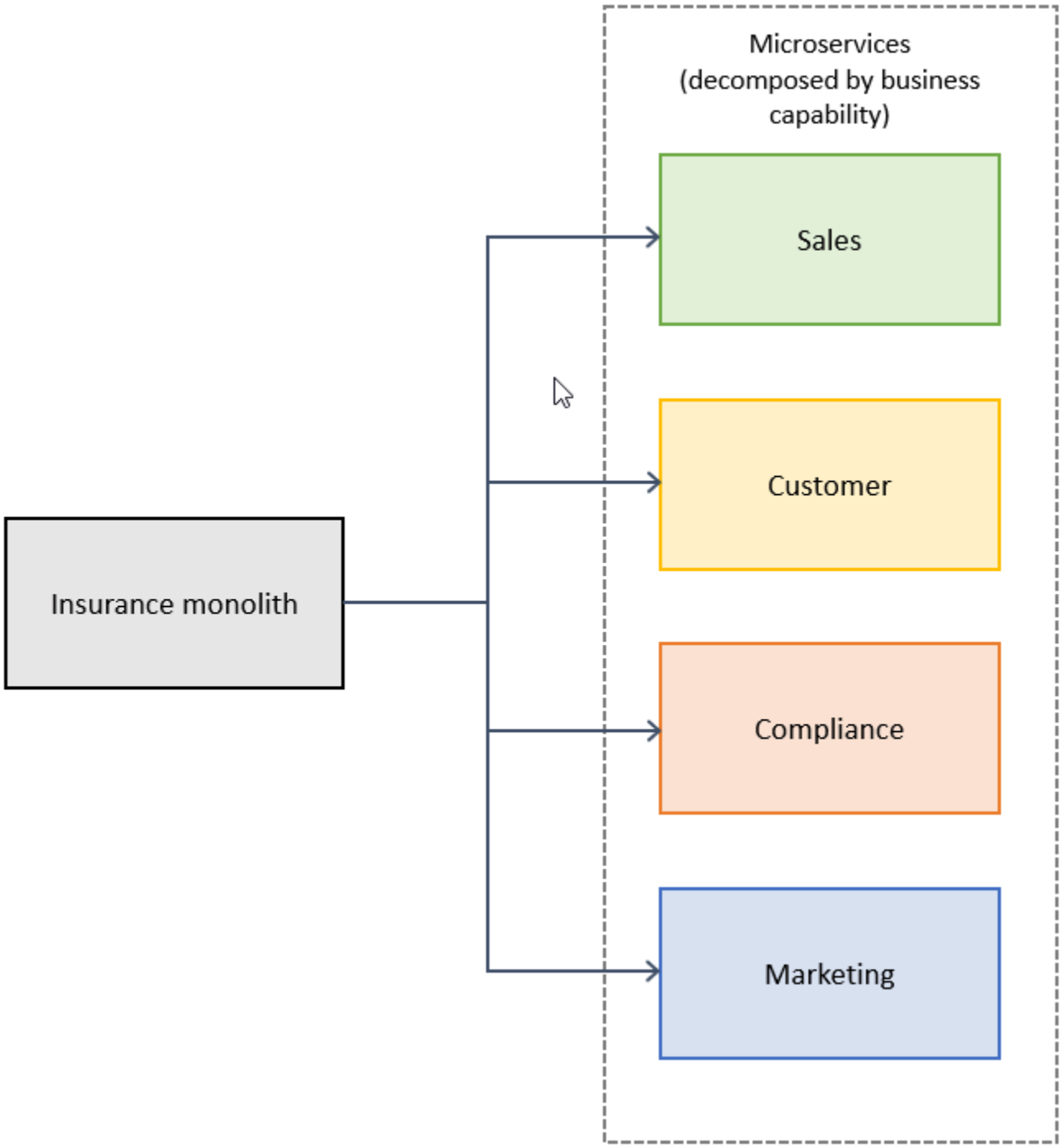
Terurai oleh kemampuan bisnis

Anda dapat menggunakan proses bisnis atau kemampuan organisasi Anda untuk menguraikan monolit. Kemampuan bisnis adalah apa yang dilakukan bisnis untuk menghasilkan nilai (misalnya, penjualan, layanan pelanggan, atau pemasaran). Biasanya, sebuah organisasi memiliki beberapa kemampuan bisnis dan ini bervariasi menurut sektor atau industri. Gunakan pola ini jika tim Anda memiliki wawasan yang cukup tentang unit bisnis organisasi Anda dan Anda memiliki pakar materi pelajaran (UKM) untuk setiap unit bisnis. Tabel berikut menjelaskan kelebihan dan kekurangan menggunakan pola ini.

Keuntungan	Kekurangan
• Menghasilkan arsitektur microservices yang stabil jika kemampuan bisnisnya relatif stabil.	• Desain aplikasi digabungkan erat dengan model bisnis.

Keuntungan	Kekurangan
• Tim pengembangan bersifat lintas fungsi dan terorganisir dalam memberikan nilai bisnis alih-alih fitur teknis. • Layanan digabungkan secara longgar.	• Membutuhkan pemahaman mendalam tentang bisnis secara keseluruhan, karena sulit untuk mengidentifikasi kemampuan dan layanan bisnis.

Dalam diagram berikut, monolit asuransi diuraikan menjadi empat layanan mikro berdasarkan kemampuan bisnis.



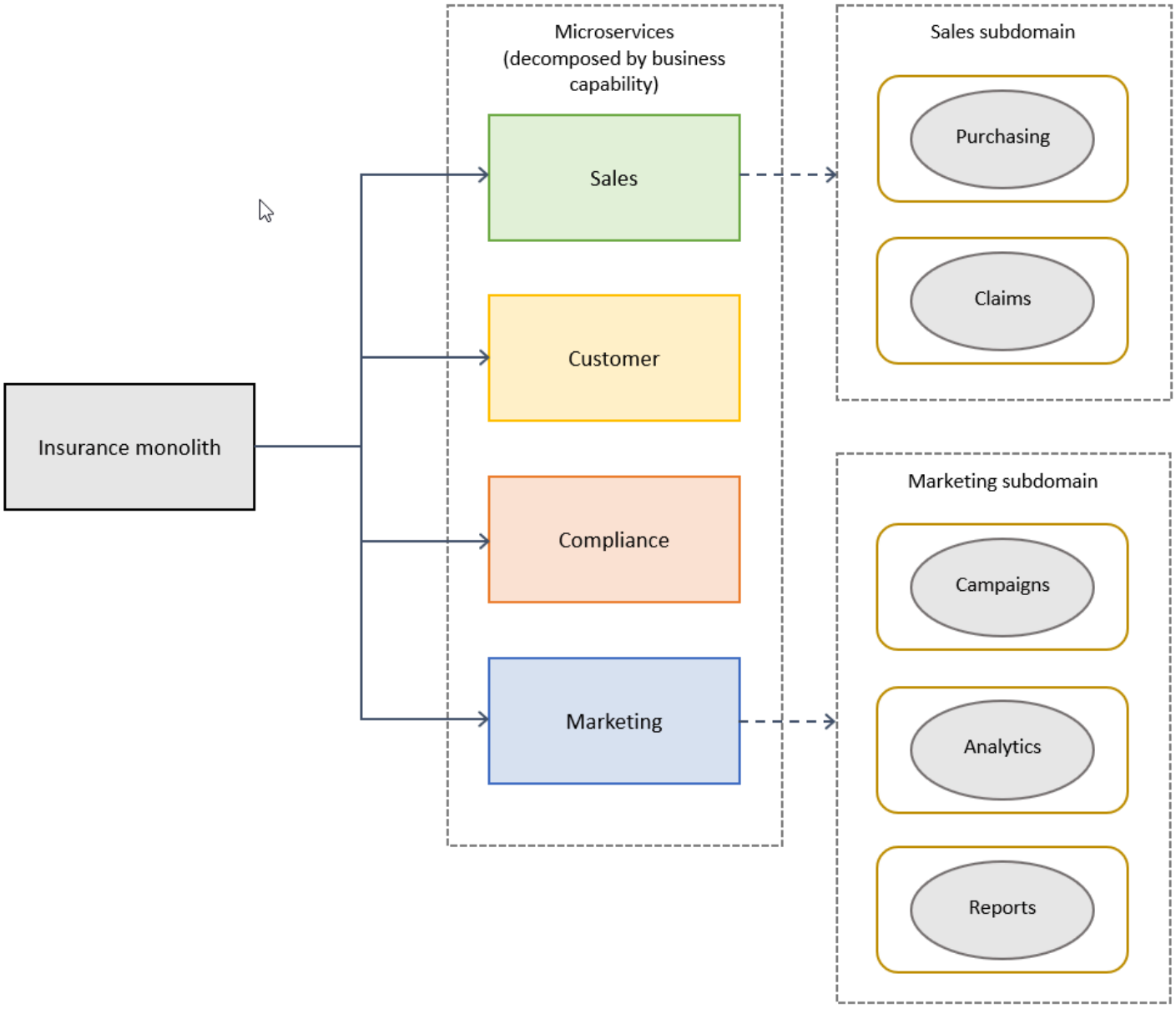
Terurai oleh subdomain

Pola ini menggunakan subdomain desain berbasis domain (DDD) untuk menguraikan monolit. Pendekatan ini memecah model domain organisasi menjadi subdomain terpisah yang diberi label sebagai inti (pembeda utama untuk bisnis), pendukung (mungkin terkait dengan bisnis tetapi bukan

pembeda), atau generik (tidak spesifik bisnis). Pola ini sesuai untuk sistem monolitik yang ada yang memiliki batas yang jelas antara modul terkait subdomain. Ini berarti Anda dapat menguraikan monolit dengan mengemas ulang modul yang ada sebagai layanan mikro tetapi tanpa menulis ulang kode yang ada secara signifikan. Setiap subdomain memiliki model, dan ruang lingkup model itu disebut konteks terbatas. Layanan mikro dikembangkan di sekitar konteks terbatas ini. Tabel berikut menjelaskan kelebihan dan kekurangan menggunakan pola ini.

Keuntungan	Kekurangan
• Arsitektur yang digabungkan secara longgar memberikan skalabilitas, ketahanan, pemeliharaan, ekstensibilitas, transparansi lokasi, independensi protokol, dan kemandirian waktu. • Sistem menjadi lebih terukur dan dapat diprediksi.	• Dapat membuat terlalu banyak layanan mikro, yang membuat penemuan dan integrasi layanan menjadi sulit. • Subdomain bisnis sulit diidentifikasi karena memerlukan pemahaman mendalam tentang bisnis secara keseluruhan.

Ilustrasi berikut menunjukkan bagaimana monolit asuransi dapat didekomposisi menjadi subdomain setelah didekomposisi oleh kemampuan bisnis.



Ilustrasi menunjukkan bahwa layanan Penjualan dan Pemasaran dipecah menjadi layanan mikro yang lebih kecil. Model Pembelian dan Klaim adalah pembeda bisnis penting untuk Penjualan, dan dibagi menjadi dua layanan mikro terpisah. Pemasaran didekomposisi dengan menggunakan fungsionalitas bisnis pendukung seperti Kampanye, Analisis, dan Laporan.

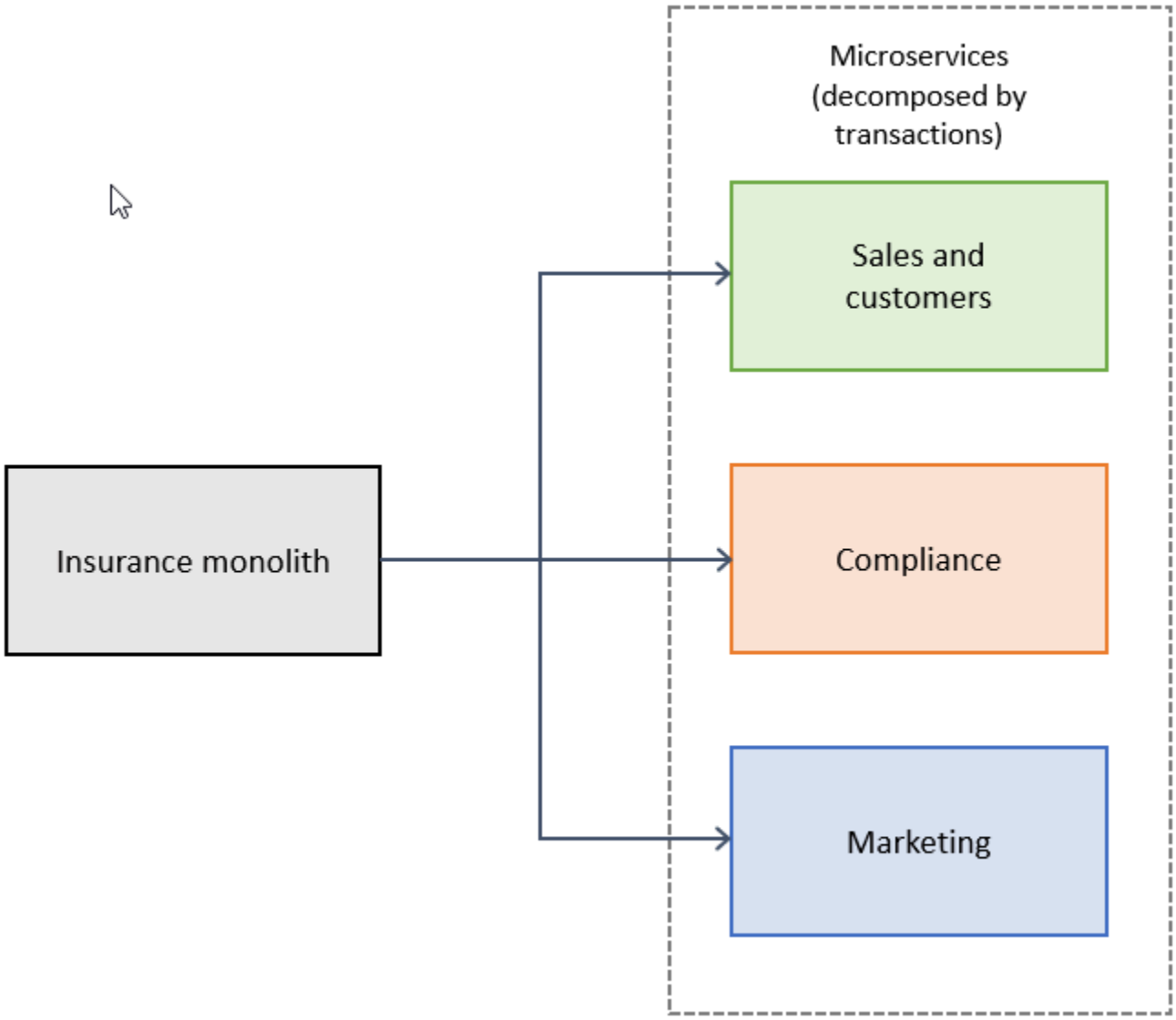
Terurai oleh transaksi

Dalam sistem terdistribusi, aplikasi biasanya harus memanggil beberapa layanan mikro untuk menyelesaikan satu transaksi bisnis. Untuk menghindari masalah latensi atau masalah komit dua fase, Anda dapat mengelompokkan layanan mikro berdasarkan transaksi. Pola ini sesuai jika Anda

menganggap waktu respons penting dan modul Anda yang berbeda tidak membuat monolit setelah Anda mengemasnya. Tabel berikut menjelaskan kelebihan dan kekurangan menggunakan pola ini.

Keuntungan	Kekurangan
• Waktu respons yang lebih cepat. • Anda tidak perlu khawatir tentang konsistensi data. • Peningkatan ketersediaan.	• Beberapa modul dapat dikemas bersama, dan ini dapat membuat monolit. • Beberapa fungsi diimplementasikan dalam layanan mikro tunggal alih-alih layanan mikro terpisah, yang meningkatkan biaya dan kompleksitas. • Transaction-oriented Layanan mikro dapat tumbuh jika jumlah domain bisnis dan dependensi di antara mereka tinggi. • Versi yang tidak konsisten dapat digunakan pada saat yang sama untuk domain bisnis yang sama.

Dalam ilustrasi berikut, monolit asuransi dipecah menjadi beberapa layanan mikro berdasarkan transaksi.



Dalam sistem asuransi, permintaan klaim biasanya ditandai ke pelanggan setelah diajukan. Ini berarti bahwa layanan klaim tidak dapat ada tanpa layanan mikro Pelanggan. Penjualan dan Pelanggan dikemas bersama dalam satu paket microservice, dan transaksi bisnis memerlukan koordinasi dengan keduanya.

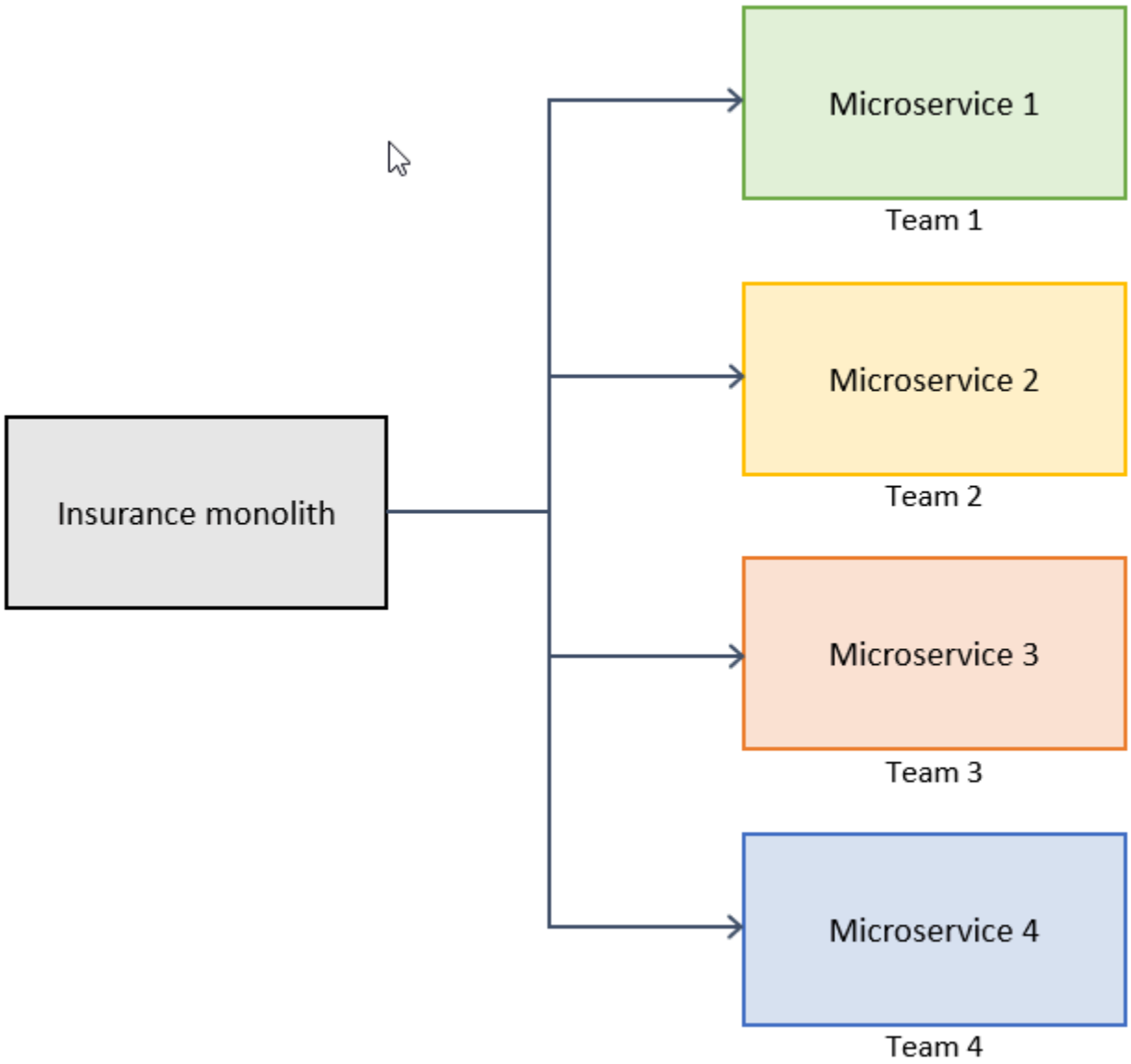
Layanan per pola tim

Alih-alih menguraikan monolit berdasarkan kemampuan atau layanan bisnis, pola layanan per tim memecahnya menjadi layanan mikro yang dikelola oleh tim individu. Setiap tim bertanggung jawab atas kemampuan bisnis dan memiliki basis kode kapabilitas. Tim secara independen mengembangkan, menguji, menyebarkan, atau menskalakan layanannya, dan terutama berinteraksi

dengan tim lain untuk menegosiasikan API. Kami menyarankan Anda menetapkan setiap layanan mikro ke satu tim. Namun, jika tim cukup besar, beberapa subtim dapat memiliki layanan mikro terpisah dalam struktur tim yang sama. Tabel berikut menjelaskan kelebihan dan kekurangan menggunakan pola ini.

Keuntungan	Kekurangan
• Tim bertindak secara independen dengan koordinasi minimal. • Basis kode dan layanan mikro tidak dibagikan oleh beberapa tim. • Tim dapat dengan cepat berinovasi dan mengulangi fitur produk. • Tim yang berbeda dapat menggunakan teknologi, kerangka kerja, atau bahasa pemrograman yang berbeda. Penting: Ini harus disembunyikan di balik API publik yang terdefinisi dengan baik dan stabil.	• Mungkin sulit untuk menyelaraskan tim dengan fungsionalitas pengguna akhir atau kemampuan bisnis. • Upaya tambahan diperlukan untuk memberikan peningkatan aplikasi yang lebih besar dan terkoordinasi, terutama jika ada dependensi melingkar antar tim.

Ilustrasi berikut menunjukkan bagaimana monolit dapat dibagi menjadi layanan mikro yang dikelola, dipelihara, dan disampaikan oleh tim individu.



Pola ara pencekik

Pola desain yang dibahas sejauh ini dalam panduan ini berlaku untuk aplikasi penguraian untuk proyek greenfield. Bagaimana dengan proyek brownfield yang melibatkan aplikasi monolitik besar? Menerapkan pola desain sebelumnya pada mereka akan sulit, karena memecahnya menjadi potongan-potongan kecil saat digunakan secara aktif adalah tugas besar.

Pola ara pencekik adalah pola desain populer yang diperkenalkan oleh Martin Fowler, yang terinspirasi oleh jenis ara tertentu yang berbiji di cabang-cabang atas pohon. Pohon yang ada awalnya menjadi struktur pendukung untuk ara baru. Ara kemudian mengirimkan akarnya ke tanah, secara bertahap menyelimuti pohon asli dan hanya menyisakan ara baru yang mandiri di tempatnya.

Pola ini biasanya digunakan untuk secara bertahap mengubah aplikasi monolitik menjadi layanan mikro dengan mengganti fungsi tertentu dengan layanan baru. Tujuannya adalah agar versi warisan dan modern baru hidup berdampingan. Sistem baru ini awalnya didukung oleh, dan membungkus, sistem yang ada. Dukungan ini memberi sistem baru waktu untuk tumbuh dan berpotensi menggantikan sistem lama sepenuhnya.

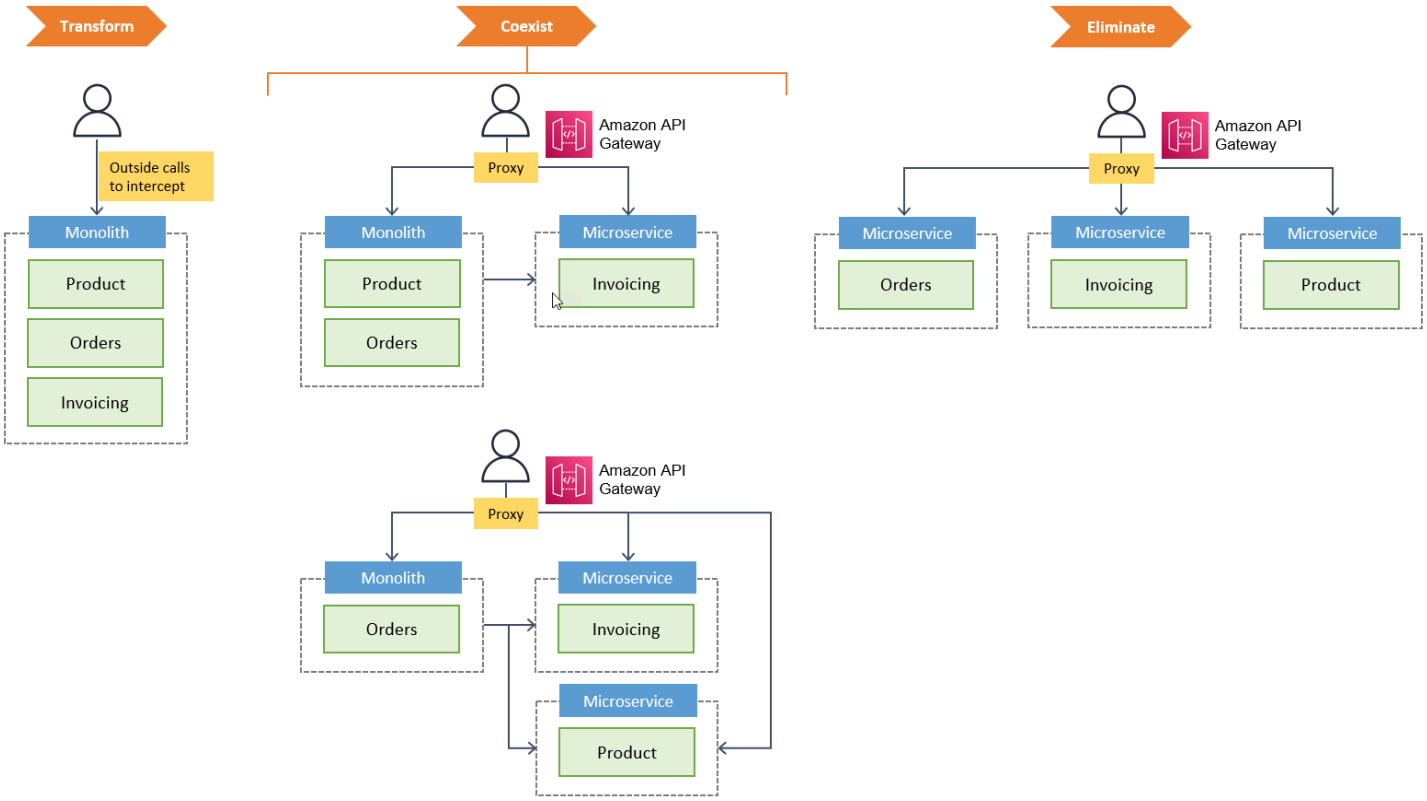
Proses transisi dari aplikasi monolitik ke layanan mikro dengan menerapkan pola ara pencekik terdiri dari tiga langkah: mengubah, hidup berdampingan, dan menghilangkan:

- Transform — Mengidentifikasi dan membuat komponen modern baik dengan porting atau menulis ulang mereka secara paralel dengan aplikasi lama.
- Coexist - Simpan aplikasi monolit untuk rollback. Mencegat panggilan sistem luar dengan memasukkan proxy HTTP (misalnya, Amazon API Gateway) di perimeter monolit Anda dan arahkan lalu lintas ke versi modern. Ini membantu Anda menerapkan fungsionalitas secara bertahap.
- Hilangkan — Pensiun fungsi lama dari monolit karena lalu lintas dialihkan dari monolit warisan ke layanan modern.

Tabel berikut menjelaskan keuntungan dan kerugian menggunakan pola ara pencekik.

Keuntungan	Kekurangan
• Memungkinkan migrasi yang anggun dari layanan ke satu atau lebih layanan pengganti . • Menyimpan layanan lama dalam permainan saat refactoring ke versi yang diperbarui. • Memberikan kemampuan untuk menambahk an layanan dan fungsionalitas baru sambil memfaktorkan ulang layanan lama. • Pola ini dapat digunakan untuk pembuatan versi API. • Pola ini dapat digunakan untuk interaksi lama untuk solusi yang tidak atau tidak akan ditingkatkan.	• Tidak cocok untuk sistem kecil di mana kompleksitasnya rendah dan ukurannya kecil. • Tidak dapat digunakan dalam sistem di mana permintaan ke sistem backend tidak dapat dicegat dan dirutekan. • Lapisan proxy atau fasad dapat menjadi satu titik kegagalan atau hambatan kinerja jika tidak dirancang dengan benar. • Membutuhkan rencana rollback untuk setiap layanan refactored untuk kembali ke cara lama melakukan sesuatu dengan cepat dan aman jika ada yang salah.

Ilustrasi berikut menunjukkan bagaimana monolit dapat dibagi menjadi layanan mikro dengan menerapkan pola ara pencekik ke arsitektur aplikasi. Kedua sistem berfungsi secara paralel, tetapi Anda akan mulai memindahkan fungsionalitas di luar basis kode monolit dan meningkatkannya dengan kemampuan baru. Kemampuan baru ini memberi Anda kesempatan untuk merancang layanan mikro dengan cara yang paling sesuai dengan kebutuhan Anda. Anda akan terus menghapus kemampuan dari monolit sampai semuanya digantikan oleh layanan mikro. Pada saat itu, Anda dapat menghilangkan aplikasi monolit. Poin kunci yang perlu diperhatikan di sini adalah bahwa monolit dan layanan mikro akan hidup bersama untuk jangka waktu tertentu.



Cabang dengan pola abstraksi

Pola ara pencekik bekerja dengan baik ketika Anda dapat mencegat panggilan di perimeter monolit. Namun, jika Anda ingin memodernisasi komponen yang ada lebih dalam di tumpukan aplikasi lama dan memiliki dependensi hulu, kami merekomendasikan cabang dengan pola abstraksi. Pola ini memungkinkan Anda membuat perubahan pada basis kode yang ada untuk memungkinkan versi modern hidup berdampingan dengan aman bersama versi lama tanpa menyebabkan gangguan.

Untuk menggunakan cabang dengan pola abstraksi dengan sukses, ikuti proses ini:

1. Identifikasi komponen monolit yang memiliki dependensi hulu.

2. Buat layer abstraksi yang mewakili interaksi antara kode yang akan dimodernisasi dan kliennya.
3. Ketika abstraksi sudah ada, ubah klien yang ada untuk menggunakan abstraksi baru.
4. Buat implementasi abstraksi baru dengan fungsionalitas yang dikerjakan ulang di luar monolit.
5. Alihkan abstraksi ke implementasi baru saat siap.
6. Ketika implementasi baru menyediakan semua fungsionalitas yang diperlukan untuk pengguna dan monolit tidak lagi digunakan, bersihkan implementasi yang lebih lama.

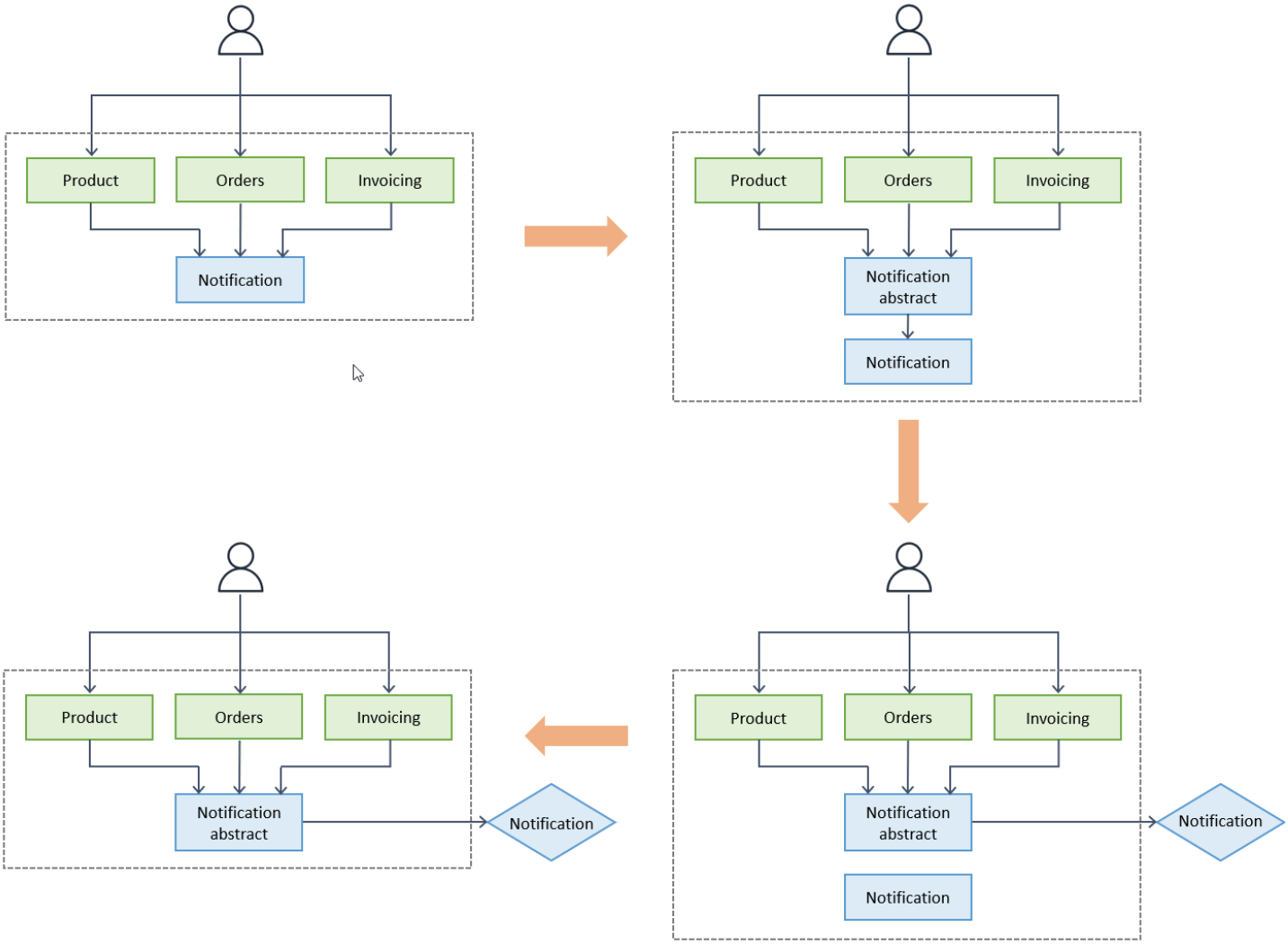
Cabang berdasarkan pola abstraksi sering dikacaukan dengan sakelar fitur, yang juga memungkinkan Anda untuk membuat perubahan pada sistem Anda secara bertahap. Perbedaannya adalah bahwa fitur toggle dimaksudkan untuk memungkinkan pengembangan fitur baru dan menjaga fitur-fitur tersebut tidak terlihat oleh pengguna saat sistem berjalan. Sakelar fitur dengan demikian digunakan pada waktu penerapan atau runtime untuk memilih apakah fitur atau serangkaian fitur tertentu terlihat dalam aplikasi. Branch by abstraction adalah teknik pengembangan dan dapat dikombinasikan dengan fitur toggle untuk beralih antara implementasi lama dan baru.

Tabel berikut menjelaskan keuntungan dan kerugian menggunakan cabang dengan pola abstraksi.

Keuntungan	Kekurangan
• Memungkinkan perubahan tambahan yang dapat dibalik jika terjadi kesalahan (kompatibel ke belakang). • Memungkinkan Anda mengekstrak fungsionalitas yang jauh di dalam monolit ketika Anda tidak dapat mencegat panggilan ke sana di tepi monolit. • Memungkinkan beberapa implementasi untuk hidup berdampingan dalam sistem perangkat lunak. • Menyediakan cara mudah untuk menerapkan mekanisme fallback dengan menggunakan langkah verifikasi perantara untuk memanggil fungsionalitas baru dan lama.	• Tidak cocok jika konsistensi data terlibat. • Membutuhkan perubahan pada sistem yang ada. • Mungkin menambahkan lebih banyak overhead ke proses pengembangan, terutama jika basis kode tidak terstrukt ur dengan baik. (Dalam banyak kasus, sisi positifnya sepadan dengan usaha ekstra, dan semakin besar restrukturisasi, semakin penting untuk mempertimbangkan penggunaan cabang dengan pola abstraksi.)

Keuntungan	Kekurangan
• Mendukung pengiriman berkelanjutan, karena kode Anda bekerja setiap saat sepanjang fase restrukturisasi.	

Ilustrasi berikut menunjukkan cabang dengan pola abstraksi untuk komponen Notifikasi dalam monolit asuransi. Dimulai dengan membuat abstrak atau antarmuka untuk fungsionalitas notifikasi. Dalam peningkatan kecil, klien yang ada diubah untuk menggunakan abstraksi baru. Ini mungkin memerlukan pencarian basis kode untuk panggilan ke API yang terkait dengan komponen Notification. Anda membuat implementasi baru fungsionalitas notifikasi sebagai layanan mikro di luar monolit Anda dan menghostingnya dalam arsitektur modern. Di dalam monolit Anda, antarmuka abstraksi Anda yang baru dibuat bertindak sebagai broker dan memanggil implementasi baru. Secara bertahap, Anda mem-port fungsionalitas notifikasi ke implementasi baru, yang tetap tidak aktif hingga sepenuhnya diuji dan siap. Ketika implementasi baru siap, Anda mengalihkan abstraksi Anda untuk menggunakannya. Anda ingin menggunakan mekanisme switching yang dapat dibalik dengan mudah (seperti fitur toggle) sehingga Anda dapat beralih kembali ke fungsi lama dengan mudah jika Anda mengalami masalah. Ketika implementasi baru mulai menyediakan semua fungsionalitas notifikasi kepada pengguna Anda dan monolit tidak lagi digunakan, Anda dapat membersihkan implementasi yang lebih lama dan menghapus flag fitur switching yang mungkin telah Anda terapkan


Product
Orders
Invoicing
Notification
Product
Orders
Invoicing
Notification abstract
Notification
Product
Orders
Invoicing
Notification abstract
Notification
Product
Orders
Invoicing
Notification abstract
Notification
Notification

FAQ monolit yang membusuk

Bagian ini memberikan jawaban atas pertanyaan yang sering diajukan tentang monolit yang membusuk.

Bisakah Anda menggunakan beberapa pola untuk memecah satu monolit?

Ya, Anda dapat menggunakan beberapa pola untuk menguraikan monolit. Cara yang paling umum adalah menguraikan monolit dengan dekomposisi oleh pola kemampuan bisnis dan kemudian menggunakan pola dekomposisi oleh subdomain untuk memecahnya lebih banyak.

Bagaimana penguraian monolit menjadi layanan mikro mempengaruhi proses? DevOps

Karena Anda tidak perlu menerapkan kembali semuanya setelah perubahan dilakukan pada aplikasi, Anda harus memiliki dukungan dan kepemilikan layanan mikro yang baru dibuat yang ditambahkan ke proses penyebaran. Ini bisa membuat DevOps prosesnya lebih kompleks.

Sumber daya

Panduan terkait

- Strategi untuk memodernisasi aplikasi di Cloud AWS
- Pendekatan bertahap untuk memodernisasi aplikasi di Cloud AWS
- Mengevaluasi kesiapan modernisasi untuk aplikasi di Cloud AWS
- Mengintegrasikan layanan mikro dengan menggunakan layanan tanpa server AWS

Sumber daya lainnya

- Memecah Aplikasi Monolitik menjadi Microservices dengan AWS Copilot, Amazon ECS, Docker, dan AWS Fargate

Riwayat dokumen

Tabel berikut menjelaskan perubahan signifikan pada panduan ini. Jika Anda ingin diberi tahu tentang pembaruan masa depan, Anda dapat berlangganan umpan RSS.

Perubahan	Deskripsi	Tanggal
Ditambahkan pola baru	Menambahkan informasi tentang ara dan cabang pencekik dengan pola abstraksi.	6 April 2023
Publikasi awal	—	11 Januari 2021

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.