Membangun arsitektur heksagonal di AWS

AWS Panduan Preskriptif

AWS Panduan Preskriptif: Membangun arsitektur heksagonal di AWS

Table of Contents

Membangun arsitektur heksagonalAWS

Furkan Oruc, Dominik Goby, Darius Kunce, dan Michal Ploski, Amazon Web Services (AWS)

Juni 2022 (riwayat dokumen)

Panduan ini menjelaskan model mental dan kumpulan pola untuk mengembangkan arsitektur perangkat lunak. Arsitektur ini mudah dipelihara, diperluas, dan diskalakan di seluruh organisasi seiring pertumbuhan adopsi produk. Hyperscaler cloud seperti Amazon Web Services (AWS) menyediakan blok bangunan bagi perusahaan kecil dan besar untuk berinovasi dan membuat produk perangkat lunak baru. Laju cepat dari layanan dan perkenalan fitur baru ini membuat para pemangku kepentingan bisnis mengharapkan tim pengembangan mereka untuk membuat prototipe produk baru yang layak minimum (MVP) lebih cepat, sehingga ide-ide baru dapat diuji dan diverifikasi sesegera mungkin. Seringkali, MVP tersebut diadopsi dan menjadi bagian dari ekosistem perangkat lunak perusahaan. Dalam proses memproduksi MVP ini, tim terkadang meninggalkan aturan pengembangan perangkat lunak dan praktik terbaik, seperti prinsip SOLID dan pengujian unit. Mereka berasumsi bahwa pendekatan ini akan mempercepat pengembangan dan mengurangi waktu ke pasar. Namun, jika mereka gagal membuat model dasar dan kerangka kerja untuk arsitektur perangkat lunak di semua tingkatan, akan sulit atau bahkan tidak mungkin untuk mengembangkan fitur baru untuk produk tersebut. Kurangnya kepastian dan perubahan persyaratan juga dapat memperlambat tim selama proses pengembangan.

Panduan ini berjalan melalui arsitektur perangkat lunak yang diusulkan, dari arsitektur heksagonal tingkat rendah hingga dekomposisi arsitektur dan organisasi tingkat tinggi, yang menggunakan desain berbasis domain (DDD) untuk mengatasi tantangan ini. DDD membantu mengelola kompleksitas bisnis dan skala tim teknik sebagai fitur baru yang dikembangkan. Ini menyelaraskan pemangku kepentingan bisnis dan teknis dengan masalah bisnis, yang disebut domain, dengan menggunakan bahasa di mana-mana. Arsitektur heksagonal adalah enabler teknis dari pendekatan ini dalam domain yang sangat spesifik, yang disebut konteks terbatas. Konteks terbatas adalah sub-area masalah bisnis yang sangat kohesif dan longgar digabungkan. Kami menyarankan Anda mengadopsi arsitektur heksagonal untuk semua proyek perangkat lunak perusahaan Anda terlepas dari kompleksitasnya.

Arsitektur heksagonal mendorong tim teknik untuk memecahkan masalah bisnis terlebih dahulu, sedangkan arsitektur berlapis klasik menggeser fokus teknik jauh dari domain untuk memecahkan masalah teknis terlebih dahulu. Selain itu, jika perangkat lunak mengikuti arsitektur heksagonal, lebih mudah untuk mengadopsi pendekatan pengembangan berbasis pengujian, yang mengurangi

loop umpan balik yang dibutuhkan pengembang untuk menguji persyaratan bisnis. Terakhir, menggunakan perintah dan penangan perintah adalah cara untuk menerapkan tanggung jawab tunggal dan prinsip-prinsip terbuka-tertutup dari SOLID. Mengikuti prinsip-prinsip ini menghasilkan basis kode yang pengembang dan arsitek bekerja pada proyek dapat dengan mudah menavigasi dan memahami, dan mengurangi risiko memperkenalkan perubahan melanggar fungsi yang ada.

Panduan ini ditujukan untuk arsitek perangkat lunak dan pengembang yang tertarik untuk memahami manfaat mengadopsi arsitektur heksagonal dan DDD untuk proyek pengembangan perangkat lunak mereka. Ini termasuk contoh merancang infrastruktur untuk aplikasi AndaAWS yang mendukung arsitektur heksagonal. Untuk contoh implementasi, lihat Menyusun proyek Python dalam arsitektur heksagonal yang digunakanAWS Lambda di situs web AWS Prescriptive Guidance.

Gambaran Umum

Desain berbasis domain (DDD)

Dalam desain berbasis domain (DDD), domain adalah inti dari sistem perangkat lunak. Model domain didefinisikan terlebih dahulu, sebelum Anda mengembangkan modul lain, dan tidak bergantung pada modul tingkat rendah lainnya. Sebagai gantinya, modul seperti database, lapisan presentasi, dan API eksternal semuanya bergantung pada domain.

Dalam DDD, arsitek menguraikan solusi ke dalam konteks terbatas dengan menggunakan dekomposisi berbasis logika bisnis bukan dekomposisi teknis. Manfaat dari pendekatan ini dibahas diHasil bisnis yang ditargetkan bagian ini.

DDD lebih mudah diterapkan ketika tim menggunakan arsitektur heksagonal. Dalam arsitektur heksagonal, inti aplikasi adalah pusat aplikasi. Ini dipisahkan dari modul lain melalui port dan adaptor, dan tidak memiliki dependensi pada modul lain. Ini sejalan sempurna dengan DDD, di mana domain adalah inti dari aplikasi yang memecahkan masalah bisnis. Panduan ini mengusulkan pendekatan di mana Anda memodelkan inti arsitektur heksagonal sebagai model domain dari konteks yang dibatasi. Bagian selanjutnya menjelaskan arsitektur heksagonal secara lebih rinci.

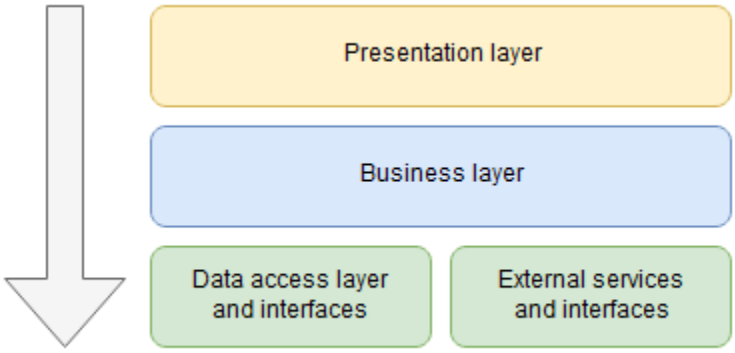
Panduan ini tidak mencakup semua aspek DDD, yang merupakan topik yang sangat luas. Untuk mendapatkan pemahaman yang lebih baik, Anda dapat meninjau sumber daya DDD yang tercantum di situs web Bahasa Domain.

arsitektur heksagonal

Arsitektur heksagonal, juga dikenal sebagai port dan adaptor atau arsitektur bawang, adalah prinsip mengelola ketergantungan inversi dalam proyek perangkat lunak. Arsitektur heksagonal mempromosikan fokus yang kuat pada logika bisnis domain inti saat mengembangkan perangkat lunak, dan memperlakukan poin integrasi eksternal sebagai sekunder. Arsitektur heksagonal membantu insinyur perangkat lunak mengadopsi praktik yang baik seperti pengembangan berbasis pengujian (TDD), yang, pada gilirannya, mempromosikan evolusi arsitektur, dan membantu Anda mengelola domain yang kompleks dalam jangka panjang.

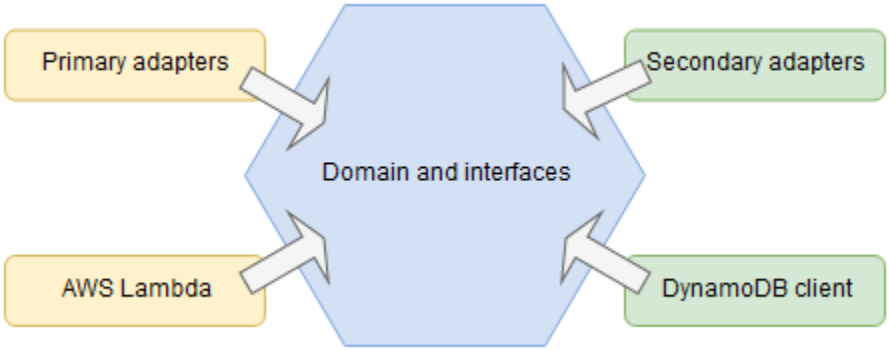
Mari kita bandingkan arsitektur heksagonal dengan arsitektur berlapis klasik, yang merupakan pilihan paling populer untuk pemodelan proyek perangkat lunak terstruktur. Ada perbedaan yang halus namun kuat antara kedua pendekatan tersebut.

Dalam arsitektur berlapis, proyek perangkat lunak disusun dalam tingkatan, yang mewakili keprihatinan luas seperti logika bisnis atau logika presentasi. Arsitektur ini menggunakan hirarki dependensi, di mana lapisan atas memiliki dependensi pada lapisan di bawahnya, tetapi tidak sebaliknya. Dalam diagram berikut, lapisan presentasi bertanggung jawab untuk interaksi pengguna, sehingga mencakup antarmuka pengguna, API, antarmuka baris perintah, dan komponen serupa. Lapisan presentasi memiliki ketergantungan pada lapisan bisnis, yang mengimplementasikan logika domain. Lapisan bisnis, pada gilirannya, memiliki ketergantungan pada lapisan akses data dan pada beberapa layanan eksternal.



Kerugian utama dari konfigurasi ini adalah struktur ketergantungan. Misalnya, jika model untuk menyimpan data dalam perubahan database, ini mempengaruhi antarmuka akses data. Setiap perubahan pada model data juga mempengaruhi lapisan bisnis, yang memiliki ketergantungan pada antarmuka akses data. Akibatnya, insinyur perangkat lunak tidak dapat membuat perubahan infrastruktur apa pun tanpa mempengaruhi logika domain. Ini, pada gilirannya, meningkatkan kemungkinan bug regresi.

arsitektur heksagonal mendefinisikan hubungan ketergantungan dengan cara yang berbeda, seperti yang digambarkan dalam diagram berikut. Ini memusatkan pengambilan keputusan di sekitar logika bisnis domain, yang mendefinisikan semua antarmuka. Komponen eksternal berinteraksi dengan logika bisnis melalui antarmuka yang disebut port. Port adalah abstraksi yang menentukan interaksi domain dengan dunia eksternal. Setiap komponen infrastruktur harus mengimplementasikan port tersebut, sehingga perubahan pada komponen tersebut tidak lagi mempengaruhi logika domain inti.



Komponen sekitarnya disebut adaptor. Adaptor adalah proxy antara dunia eksternal dan dunia internal, dan mengimplementasikan port yang didefinisikan dalam domain. Adaptor dapat dikategorikan dalam dua kelompok: primer dan sekunder. Adaptor utama adalah titik masuk ke komponen perangkat lunak. Mereka memungkinkan aktor eksternal, pengguna, dan layanan untuk berinteraksi dengan logika inti. AWS Lambdaadalah contoh yang baik dari adaptor utama. Ini terintegrasi dengan beberapaAWS layanan yang dapat memanggil fungsi Lambda sebagai titik masuk. Adaptor sekunder adalah pembungkus pustaka layanan eksternal yang menangani komunikasi dengan dunia luar. Contoh yang baik dari adaptor sekunder adalah klien Amazon DynamoDB untuk akses data.

Hasil bisnis yang ditargetkan

Arsitektur heksagonal yang dibahas dalam panduan ini membantu Anda mencapai tujuan berikut:

- Kurangi waktu ke pasar dengan meningkatkan siklus pengembangan
- Meningkatkan kualitas perangkat lunak
- Beradaptasi lebih mudah untuk berubah

Proses ini dibahas secara rinci di bagian berikut.

Meningkatkan siklus pengembangan

Mengembangkan perangkat lunak untuk cloud memperkenalkan tantangan baru bagi insinyur perangkat lunak, karena sangat sulit untuk mereplikasi lingkungan runtime secara lokal pada mesin pengembangan. Cara mudah untuk memvalidasi perangkat lunak adalah dengan menerapkannya ke cloud dan mengujinya di sana. Namun, pendekatan ini melibatkan siklus umpan balik yang panjang, terutama ketika arsitektur perangkat lunak berisi beberapa penyebaran tanpa server. Meningkatkan siklus umpan balik ini mempersingkat waktu untuk mengembangkan fitur, yang mengurangi waktu ke pasar secara signifikan.

Pengujian di cloud

Pengujian langsung di cloud adalah satu-satunya cara untuk memastikan bahwa komponen arsitektur Anda, seperti gateway di Amazon API Gateway,AWS Lambda fungsi, tabel Amazon DynamoDB, dan izinAWS Identity and Access Management (IAM), dikonfigurasi dengan benar. Ini mungkin juga satu-satunya cara yang dapat diandalkan untuk menguji integrasi komponen. Meskipun beberapaAWS layanan (seperti DynamoDB) dapat digunakan secara lokal, kebanyakan dari mereka tidak dapat direplikasi dalam pengaturan lokal. Pada saat yang sama, alat pihak ketiga seperti Moto dan LocalStackAWSlayanan tiruan untuk tujuan pengujian mungkin tidak mencerminkan kontrak API layanan nyata secara akurat, atau jumlah fitur mungkin terbatas.

Namun, bagian paling kompleks dari perangkat lunak perusahaan adalah dalam logika bisnis, bukan dalam arsitektur cloud. Arsitektur berubah lebih jarang daripada domain, yang harus mengakomodasi persyaratan bisnis baru. Dengan demikian, menguji logika bisnis di cloud menjadi proses intens untuk membuat perubahan dalam kode, memulai penyebaran, menunggu lingkungan siap, dan memvalidasi perubahan. Jika penyebaran memakan waktu hanya 5 menit, membuat dan menguji 10 perubahan dalam logika bisnis akan memakan waktu satu jam atau lebih. Jika logika bisnis lebih kompleks, pengujian mungkin memerlukan beberapa hari menunggu penyebaran selesai. Jika Anda memiliki beberapa fitur dan insinyur di tim, periode yang diperpanjang dengan cepat menjadi nyata bagi bisnis.

Pengujian secara lokal

Arsitektur heksagonal membantu pengembang fokus pada domain alih-alih teknis infrastruktur. Pendekatan ini menggunakan pengujian lokal (alat pengujian unit dalam kerangka pengembangan yang Anda pilih) untuk memenuhi persyaratan logika domain. Anda tidak perlu menghabiskan waktu

untuk memecahkan masalah integrasi teknis atau menyebarkan perangkat lunak Anda ke cloud untuk menguji logika bisnis. Anda dapat menjalankan pengujian unit secara lokal dan mengurangi loop umpan balik dari menit ke detik. Jika penerapan membutuhkan waktu 5 menit tetapi pengujian unit selesai dalam 5 detik, itu adalah pengurangan yang signifikan dalam waktu yang diperlukan untuk mendeteksi kesalahan. Menguji logika bisnis terlebih dahuluBagian kemudian dalam panduan ini secara lebih rinci.

Paralelisasi pembangunan

Pendekatan arsitektur heksagonal memungkinkan tim pengembangan untuk memparalelkan upaya pengembangan. Pengembang dapat merancang dan mengimplementasikan berbagai komponen layanan secara individual. Paralelisasi ini dimungkinkan melalui isolasi setiap komponen dan antarmuka yang ditentukan antara masing-masing komponen.

Waktu produk ke pasar

Pengujian unit lokal meningkatkan siklus umpan balik pengembangan dan mengurangi waktu untuk memasarkan produk atau fitur baru, terutama ketika ini mengandung logika bisnis yang kompleks, seperti yang dijelaskan sebelumnya. Selain itu, peningkatan cakupan kode dengan pengujian unit secara signifikan mengurangi risiko memperkenalkan bug regresi saat Anda memperbarui atau mengubah basis kode. Cakupan pengujian unit juga memungkinkan Anda untuk terus melakukan refactor basis kode agar tetap terorganisir dengan baik, yang mempercepat proses orientasi untuk teknisi baru. Ini dibahas lebih lanjut diKualitas berdasarkan desain barang bagian ini. Terakhir, jika logika bisnis terisolasi dan diuji dengan baik, ini memungkinkan pengembang untuk beradaptasi dengan cepat dengan perubahan persyaratan fungsional dan non-fungsional. Hal ini dijelaskan lebih lanjut diBeradaptasi dengan perubahan bagian ini.

Kualitas berdasarkan desain barang

Mengadopsi arsitektur heksagonal membantu mempromosikan kualitas basis kode Anda sejak awal proyek Anda. Penting untuk membangun proses yang membantu Anda memenuhi persyaratan kualitas yang diharapkan sejak awal, tanpa memperlambat proses pengembangan.

Perubahan lokal dan peningkatan keterbacaan

Menggunakan pendekatan arsitektur heksagonal memungkinkan pengembang untuk mengubah kode dalam satu kelas atau komponen tanpa mempengaruhi kelas atau komponen lain. Desain ini mempromosikan kohesi komponen yang dikembangkan. Dengan memisahkan domain dari adaptor dan menggunakan antarmuka yang terkenal, Anda dapat meningkatkan keterbacaan kode. Menjadi lebih mudah untuk mengidentifikasi masalah dan kasus sudut.

Pendekatan ini juga memfasilitasi peninjauan kode selama pengembangan dan membatasi pengenalan perubahan yang tidak terdeteksi atau utang teknis.

Menguji logika bisnis terlebih dahulu

Pengujian lokal dapat dilakukan dengan memperkenalkan end-to-end, integrasi, dan unit test untuk proyek. nd-to-end Tes E mencakup seluruh siklus hidup permintaan yang masuk. Mereka biasanya memanggil titik masuk aplikasi dan menguji untuk melihat apakah itu telah memenuhi persyaratan bisnis. Setiap proyek perangkat lunak harus memiliki setidaknya satu skenario pengujian yang menggunakan input yang diketahui dan menghasilkan output yang diharapkan. Namun, menambahkan lebih banyak skenario kasus sudut bisa menjadi rumit, karena setiap tes harus dikonfigurasi untuk mengirim permintaan melalui titik masuk (misalnya, melalui REST API atau antrian), melalui semua titik integrasi yang dibutuhkan tindakan bisnis, dan kemudian tegaskan hasilnya. Menyiapkan lingkungan untuk skenario pengujian dan menegaskan hasil dapat memakan banyak waktu pengembang.

Dalam arsitektur heksagonal, Anda menguji logika bisnis secara terpisah, dan menggunakan tes integrasi untuk menguji adaptor sekunder. Anda dapat menggunakan adaptor tiruan atau palsu dalam tes logika bisnis Anda. Anda juga dapat menggabungkan pengujian untuk kasus penggunaan bisnis dengan pengujian unit untuk model domain Anda guna mempertahankan cakupan tinggi dengan kopling rendah. Sebagai praktik yang baik, tes integrasi seharusnya tidak memvalidasi logika bisnis. Sebaliknya, mereka harus memverifikasi bahwa adaptor sekunder memanggil layanan eksternal dengan benar.

Idealnya, Anda dapat menggunakan test-driven development (TDD) dan mulai mendefinisikan entitas domain atau kasus penggunaan bisnis dengan pengujian yang tepat di awal pengembangan. Menulis tes terlebih dahulu membantu Anda membuat implementasi tiruan dari antarmuka yang diperlukan oleh domain. Ketika pengujian berhasil dan aturan logika domain terpenuhi, Anda dapat menerapkan adaptor aktual dan menyebarkan perangkat lunak ke lingkungan pengujian. Pada titik ini, implementasi logika domain Anda mungkin tidak ideal. Anda kemudian dapat bekerja pada refactoring arsitektur yang ada untuk mengembangkannya dengan memperkenalkan pola desain atau menata ulang kode secara umum. Dengan menggunakan pendekatan ini, Anda dapat menghindari memperkenalkan bug regresi dan Anda dapat meningkatkan arsitektur saat proyek tumbuh. Dengan menggabungkan pendekatan ini dengan pengujian otomatis yang Anda jalankan dalam proses integrasi berkelanjutan, Anda dapat mengurangi jumlah potensi bug sebelum mulai diproduksi.

Jika Anda menggunakan penerapan tanpa server, Anda dapat dengan cepat menyediakan instance aplikasi diAWS akun Anda untuk integrasi dan end-to-end pengujian manual. Setelah langkah-langkah implementasi ini, kami sarankan Anda mengotomatiskan pengujian dengan setiap perubahan baru yang didorong ke repositori.

Maintainability

Maintainability mengacu pada operasi dan pemantauan aplikasi untuk memastikan bahwa itu memenuhi semua persyaratan dan meminimalkan kemungkinan kegagalan sistem. Agar sistem dapat dioperasikan, Anda harus menyesuaikannya dengan lalu lintas atau persyaratan operasional di future. Anda juga harus memastikan bahwa itu tersedia dan mudah digunakan dengan dampak minimum atau tanpa dampak pada klien.

Untuk memahami keadaan sistem Anda saat ini dan historis, Anda harus membuatnya dapat diamati. Anda dapat melakukan ini dengan menyediakan metrik, log, dan pelacakan tertentu yang dapat digunakan operator untuk memastikan bahwa sistem berfungsi seperti yang diharapkan dan untuk melacak bug. Mekanisme ini juga harus memungkinkan operator untuk melakukan analisis akar penyebab tanpa harus masuk ke mesin dan membaca kode.

Arsitektur heksagonal bertujuan untuk meningkatkan pemeliharaan aplikasi web Anda sehingga kode Anda membutuhkan lebih sedikit pekerjaan secara keseluruhan. Dengan memisahkan modul, melokalisasi perubahan, dan memisahkan logika bisnis aplikasi dari implementasi adaptor, Anda dapat menghasilkan metrik dan log yang membantu operator mendapatkan pemahaman mendalam tentang sistem dan memahami ruang lingkup perubahan spesifik yang dibuat pada adaptor primer atau sekunder.

Beradaptasi dengan perubahan

Sistem perangkat lunak cenderung menjadi rumit. Salah satu alasan untuk ini bisa sering perubahan pada kebutuhan bisnis dan sedikit waktu untuk menyesuaikan arsitektur perangkat lunak yang sesuai. Alasan lain bisa menjadi investasi yang tidak memadai untuk menyiapkan arsitektur perangkat lunak pada awal proyek untuk beradaptasi dengan perubahan yang sering. Apapun alasannya, sistem perangkat lunak bisa menjadi rumit sampai pada titik di mana hampir tidak mungkin untuk melakukan perubahan. Oleh karena itu, penting untuk membangun arsitektur perangkat lunak yang dapat dipelihara sejak awal proyek. Arsitektur perangkat lunak yang baik dapat beradaptasi dengan perubahan dengan mudah.

Bagian ini menjelaskan cara merancang aplikasi yang dapat dipelihara dengan menggunakan arsitektur heksagonal yang mudah beradaptasi dengan kebutuhan non-fungsional atau bisnis.

Beradaptasi dengan persyaratan non-fungsional baru dengan menggunakan port dan adaptor

Sebagai inti dari aplikasi, model domain mendefinisikan tindakan yang diperlukan dari dunia luar untuk memenuhi persyaratan bisnis. Tindakan ini didefinisikan melalui abstraksi, yang disebut port. Port ini diimplementasikan oleh adaptor terpisah. Setiap adaptor bertanggung jawab atas interaksi dengan sistem lain. Misalnya, Anda mungkin memiliki satu adaptor untuk repositori database dan adaptor lain untuk berinteraksi dengan API pihak ketiga. Domain tidak menyadari implementasi adaptor, sehingga mudah untuk mengganti satu adaptor dengan yang lain. Misalnya, aplikasi mungkin beralih dari database SQL ke database NoSQL. Dalam hal ini, adaptor baru harus dikembangkan untuk mengimplementasikan port yang didefinisikan oleh model domain. Domain tidak memiliki dependensi pada repositori database dan menggunakan abstraksi untuk berinteraksi, sehingga tidak perlu mengubah apa pun dalam model domain. Oleh karena itu, arsitektur heksagonal menyesuaikan dengan persyaratan non-fungsional dengan mudah.

Beradaptasi dengan kebutuhan bisnis baru dengan menggunakan perintah dan penangan perintah

Dalam arsitektur berlapis klasik, domain tergantung pada lapisan persistensi. Jika Anda ingin mengubah domain, Anda juga harus mengubah layer persistensi. Sebagai perbandingan, dalam arsitektur heksagonal, domain tidak bergantung pada modul lain dalam perangkat lunak. Domain

adalah inti dari aplikasi, dan semua modul lainnya (port dan adaptor) bergantung pada model domain. Domain menggunakan prinsip ketergantungan inversi untuk berkomunikasi dengan dunia luar melalui port. Manfaat dari ketergantungan inversi adalah bahwa Anda dapat mengubah model domain bebas tanpa takut untuk melanggar bagian lain dari kode. Karena model domain mencerminkan masalah bisnis yang Anda coba selesaikan, memperbarui model domain untuk beradaptasi dengan perubahan kebutuhan bisnis bukanlah masalah.

Ketika Anda mengembangkan perangkat lunak, pemisahan kekhawatiran adalah prinsip penting untuk diikuti. Untuk mencapai pemisahan ini, Anda dapat menggunakan pola perintah yang sedikit dimodifikasi. Ini adalah pola desain perilaku di mana semua informasi yang diperlukan untuk menyelesaikan operasi dienkapsulasi dalam objek perintah. Operasi ini kemudian diproses oleh penangan perintah. Penangan perintah adalah metode yang menerima perintah, mengubah keadaan domain, dan kemudian mengembalikan respons ke pemanggil. Anda dapat menggunakan klien yang berbeda, seperti API sinkron atau antrian asinkron, untuk menjalankan perintah. Kami menyarankan Anda menggunakan perintah dan penangan perintah untuk setiap operasi pada domain. Dengan mengikuti pendekatan ini, Anda dapat menambahkan fitur baru dengan memperkenalkan perintah baru dan penangan perintah, tanpa mengubah logika bisnis yang ada. Dengan demikian, menggunakan pola perintah membuatnya lebih mudah untuk beradaptasi dengan kebutuhan bisnis baru.

Decoupling komponen dengan menggunakan façade layanan atau pola CQRS

Dalam arsitektur heksagonal, adaptor utama bertanggung jawab untuk menggabungkan permintaan baca dan tulis yang masuk secara longgar dari klien ke domain. Ada dua cara untuk mencapai kopling longgar ini: dengan menggunakan pola layanan façade atau dengan menggunakan perintah permintaan tanggung jawab segregasi (CQRS) pola.

Pola layanan façade menyediakan antarmuka menghadap ke depan untuk melayani klien seperti lapisan presentasi atau microservice. Sebuah façade layanan menyediakan klien dengan beberapa operasi baca dan tulis. Ini bertanggung jawab untuk mentransfer permintaan masuk ke domain dan memetakan respons yang diterima dari domain ke klien. Menggunakan façade layanan mudah untuk layanan mikro yang memiliki tanggung jawab tunggal dengan beberapa operasi. Namun, ketika menggunakan façade layanan, lebih sulit untuk mengikuti tanggung jawab tunggal dan prinsip-prinsip terbuka. Prinsip tanggung jawab tunggal menyatakan bahwa setiap modul harus memiliki tanggung jawab atas hanya satu fungsi perangkat lunak. Prinsip terbuka-tertutup menyatakan bahwa kode harus terbuka untuk ekstensi dan ditutup untuk modifikasi. Ketika façade layanan meluas, semua

operasi dikumpulkan dalam satu antarmuka, lebih banyak dependensi dienkapsulasi ke dalamnya, dan lebih banyak pengembang mulai memodifikasi façade yang sama. Oleh karena itu, kami sarankan menggunakan façade layanan hanya jika jelas bahwa layanan tidak akan memperpanjang banyak selama pengembangan.

Cara lain untuk menerapkan adaptor utama dalam arsitektur heksagonal adalah dengan menggunakan pola CQRS, yang memisahkan operasi baca dan tulis menggunakan kueri dan perintah. Seperti dijelaskan sebelumnya, perintah adalah objek yang berisi semua informasi yang diperlukan untuk mengubah keadaan domain. Perintah dilakukan dengan metode penangan perintah. Kueri, di sisi lain, tidak mengubah keadaan sistem. Satu-satunya tujuan mereka adalah mengembalikan data ke klien. Dalam pola CQRS, perintah dan kueri diimplementasikan dalam modul terpisah. Ini sangat menguntungkan untuk proyek yang mengikuti arsitektur berbasis peristiwa, karena perintah dapat diimplementasikan sebagai peristiwa yang diproses secara asinkron, sedangkan kueri dapat dijalankan secara sinkron dengan menggunakan API. Sebuah query juga dapat menggunakan database yang berbeda yang dioptimalkan untuk itu. Kerugian dari pola CQRS adalah bahwa dibutuhkan lebih banyak waktu untuk menerapkan daripada façade layanan. Kami merekomendasikan penggunaan pola CQRS untuk proyek-proyek yang Anda rencanakan untuk diskalakan dan dipelihara dalam jangka panjang. Perintah dan kueri menyediakan mekanisme yang efektif untuk menerapkan prinsip tanggung jawab tunggal dan mengembangkan perangkat lunak yang digabungkan secara longgar, terutama dalam proyek skala besar.

CQRS memiliki manfaat besar dalam jangka panjang, tetapi membutuhkan investasi awal. Oleh karenanya, kami menyarankan agar Anda mengevaluasi proyek Anda dengan hati-hati sebelum Anda memutuskan untuk menggunakan pola CQRs. Namun, Anda dapat menyusun aplikasi Anda dengan menggunakan perintah dan penangan perintah sejak awal tanpa memisahkan operasi baca/tulis. Ini akan membantu Anda dengan mudah refactor proyek Anda untuk CQRS jika Anda memutuskan untuk mengadopsi pendekatan itu nanti.

Penskalaan organisasi

Kombinasi arsitektur heksagonal, desain berbasis domain, dan (opsional) CQRS memungkinkan organisasi Anda untuk dengan cepat menskalakan produk Anda. Menurut Hukum Conway, arsitektur perangkat lunak cenderung berkembang untuk mencerminkan struktur komunikasi perusahaan. Pengamatan ini secara historis memiliki konotasi negatif, karena organisasi besar sering menyusun tim mereka berdasarkan keahlian teknis seperti database, bus layanan perusahaan, dan sebagainya. Masalah dengan pendekatan ini adalah bahwa pengembangan produk dan fitur selalu melibatkan masalah lintas sektor, seperti keamanan dan skalabilitas, yang memerlukan komunikasi konstan di

antara tim. Penataan tim berdasarkan fitur teknis menciptakan silo yang tidak perlu dalam organisasi, yang mengakibatkan komunikasi yang buruk, kurangnya kepemilikan, dan kehilangan pandangan dari gambaran besar. Akhirnya, masalah organisasi ini tercermin dalam arsitektur perangkat lunak.

The Inverse Conway Manuver, di sisi lain, mendefinisikan struktur organisasi berdasarkan domain yang mempromosikan arsitektur perangkat lunak. Misalnya, tim lintas fungsi diberi tanggung jawab untuk serangkaian konteks terbatas tertentu, yang diidentifikasi dengan menggunakan DDD dan penyerbuan acara. Konteks yang dibatasi itu mungkin mencerminkan fitur produk yang sangat spesifik. Misalnya, tim akun mungkin bertanggung jawab atas konteks pembayaran. Setiap fitur baru ditugaskan ke tim baru yang memiliki tanggung jawab yang sangat kohesif dan longgar, sehingga mereka hanya dapat fokus pada pengiriman fitur itu dan mengurangi waktu ke pasar. Tim dapat diskalakan sesuai dengan kompleksitas fitur, sehingga fitur kompleks dapat diberikan kepada lebih banyak insinyur.

Praktik terbaik

Model domain bisnis

Bekerja kembali dari domain bisnis ke desain perangkat lunak untuk memastikan bahwa perangkat lunak yang Anda tulis sesuai dengan kebutuhan bisnis.

Gunakan metodologi desain berbasis domain (DDD) seperti penyerbuan peristiwa untuk memodelkan domain bisnis. Acara storming memiliki format lokakarya yang fleksibel. Selama lokakarya, pakar domain dan perangkat lunak mengeksplorasi kompleksitas domain bisnis secara kolaboratif. Pakar perangkat lunak menggunakan kiriman lokakarya untuk memulai proses desain dan pengembangan komponen perangkat lunak.

Menulis dan menjalankan tes dari awal

Gunakan test-driven development (TDD) untuk memverifikasi kebenaran perangkat lunak yang Anda kembangkan. TDD bekerja paling baik pada tingkat uji unit. Pengembang merancang komponen perangkat lunak dengan menulis tes terlebih dahulu, yang memanggil komponen itu. Komponen itu tidak memiliki implementasi di awal, oleh karena itu pengujian gagal. Sebagai langkah berikutnya, pengembang mengimplementasikan fungsionalitas komponen, menggunakan perlengkapan uji dengan objek tiruan untuk mensimulasikan perilaku dependensi eksternal, atau port. Ketika tes berhasil, pengembang dapat melanjutkan dengan menerapkan adaptor nyata. Pendekatan ini meningkatkan kualitas perangkat lunak dan menghasilkan kode yang lebih mudah dibaca, karena pengembang memahami bagaimana pengguna akan menggunakan komponen. Arsitektur heksagonal mendukung metodologi TDD dengan memisahkan inti aplikasi. Pengembang menulis unit test yang berfokus pada perilaku inti domain. Mereka tidak harus menulis adapter kompleks untuk menjalankan pengujian mereka; sebaliknya, mereka dapat menggunakan objek tiruan sederhana dan perlengkapan.

Gunakan pengembangan berbasis perilaku (BDD) untuk memastikan end-to-end penerimaan pada tingkat fitur. Di BDD, pengembang menentukan skenario untuk fitur dan memverifikasinya dengan pemangku kepentingan bisnis. Tes BDD menggunakan bahasa alami sebanyak mungkin untuk mencapai hal ini. Arsitektur heksagonal mendukung metodologi BDD dengan konsep adaptor primer dan sekunder. Pengembang dapat membuat adaptor primer dan sekunder yang dapat berjalan secara lokal tanpa memanggil layanan eksternal. Mereka mengkonfigurasi BDD test suite untuk menggunakan adaptor utama lokal untuk menjalankan aplikasi.

Secara otomatis menjalankan setiap pengujian dalam pipa integrasi berkelanjutan untuk terus mengevaluasi kualitas sistem.

Tentukan perilaku domain

Terurai domain menjadi entitas, objek nilai, dan agregat (baca tentang menerapkan desain berbasis domain), dan tentukan perilakunya. Menerapkan perilaku domain sehingga tes yang ditulis pada awal proyek berhasil. Tentukan perintah yang memanggil perilaku objek domain. Mendefinisikan peristiwa yang objek domain memancarkan setelah mereka menyelesaikan perilaku.

Tentukan antarmuka yang dapat digunakan adaptor untuk berinteraksi dengan domain.

Mengotomatisasi pengujian dan deployment

Setelah bukti awal konsep, kami sarankan Anda menginvestasikan waktu menerapkan DevOps praktek. Misalnya, pipeline integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD) dan lingkungan pengujian dinamis membantu Anda menjaga kualitas kode dan menghindari kesalahan selama penerapan.

- Jalankan pengujian unit Anda di dalam proses CI Anda dan uji kode Anda sebelum digabungkan.
- Buat proses CD untuk menerapkan aplikasi Anda ke lingkungan dev/pengujian statis atau ke lingkungan yang dibuat secara dinamis yang mendukung integrasi dan end-to-end pengujian otomatis.
- Mengotomatisasi proses deployment untuk lingkungan khusus.

Skala produk Anda dengan menggunakan layanan mikro dan CQRS

Jika produk Anda berhasil, skala produk Anda dengan menguraikan proyek perangkat lunak Anda menjadi layanan mikro. Memanfaatkan portabilitas yang disediakan arsitektur heksagonal untuk meningkatkan kinerja. Membagi layanan kueri dan penangan perintah menjadi API sinkron dan asinkron terpisah. Pertimbangkan untuk mengadopsi pola pemisahan tanggung jawab permintaan perintah (CQRS) dan arsitektur berbasis peristiwa.

Jika Anda mendapatkan banyak permintaan fitur baru, pertimbangkan untuk menskalakan organisasi Anda berdasarkan pola DDD. Susun tim Anda sedemikian rupa sehingga mereka memiliki satu atau

lebih fitur sebagai konteks terbatas, seperti yang dibahas sebelumnya diPenskalaan organisasi bagian. Tim-tim ini kemudian dapat menerapkan logika bisnis dengan menggunakan arsitektur heksagonal.

Merancang struktur proyek yang memetakan konsep arsitektur heksagonal

Infrastructure as code (IAC) adalah praktik yang diadopsi secara luas dalam pengembangan cloud. Ini memungkinkan Anda menentukan dan memelihara sumber daya infrastruktur Anda (seperti jaringan, load balancer, mesin virtual, dan gateway) sebagai kode sumber. Dengan cara ini, Anda dapat melacak semua perubahan pada arsitektur Anda dengan menggunakan sistem kontrol versi. Selain itu, Anda dapat membuat dan memindahkan infrastruktur dengan mudah untuk tujuan pengujian. Kami menyarankan Anda menyimpan kode aplikasi dan kode infrastruktur di repositori yang sama saat Anda mengembangkan aplikasi cloud Anda. Pendekatan ini membuatnya mudah untuk mempertahankan infrastruktur untuk aplikasi Anda.

Kami menyarankan Anda membagi aplikasi Anda menjadi tiga folder atau proyek yang memetakan konsep arsitektur heksagonal:`entrypoints` (adaptor primer),`domain` (domain dan antarmuka), dan`adapters` (adaptor sekunder).

Struktur proyek berikut memberikan contoh pendekatan ini ketika merancang API padaAWS. Proyek ini mempertahankan kode aplikasi (`app`) dan kode infrastruktur (`infra`) dalam repositori yang sama, seperti yang direkomendasikan sebelumnya.

```
app/  # application code
|--- adapters/  # implementation of the ports defined in the domain
     |--- tests/  # adapter unit tests
|--- entrypoints/  # primary adapters, entry points
     |--- api/  # api entry point
          |--- model/  # api model
          |--- tests/  # end to end api tests
|--- domain/  # domain to implement business logic using hexagonal architecture
     |--- command_handlers/  # handlers used to run commands on the domain
     |--- commands/  # commands on the domain
     |--- events/  # events emitted by the domain
     |--- exceptions/  # exceptions defined on the domain
     |--- model/  # domain model
     |--- ports/  # abstractions used for external communication
     |--- tests/  # domain tests
```

```
infra/  # infrastructure code
```

Seperti yang dibahas sebelumnya, domain adalah inti dari aplikasi dan tidak bergantung pada modul lainnya. Kami menyarankan Anda menyusundomain folder untuk menyertakan subfolder berikut:

- command handlersberisi metode atau kelas yang menjalankan perintah pada domain.
- commandsberisi objek perintah yang menentukan informasi yang diperlukan untuk melakukan operasi pada domain.
- eventsberisi peristiwa yang dipancarkan melalui domain dan kemudian dialihkan ke layanan mikro lainnya.
- exceptionsberisi kesalahan yang diketahui didefinisikan dalam domain.
- modelberisi entitas domain, objek nilai, dan layanan domain.
- portsberisi abstraksi melalui mana domain berkomunikasi dengan database, API, atau komponen eksternal lainnya.
- testsberisi metode pengujian (seperti tes logika bisnis) yang dijalankan pada domain.

Adaptor utama adalah titik masuk ke aplikasi, seperti yang diwakili olehentrypoints folder. Contoh ini menggunakanapi folder sebagai adaptor utama. Folder ini berisi APImodel, yang mendefinisikan antarmuka yang diperlukan adaptor utama untuk berkomunikasi dengan klien. testsFolder berisi end-to-end pengujian untuk API. Ini adalah tes dangkal yang memvalidasi bahwa komponen aplikasi terintegrasi dan bekerja secara harmonis.

Adaptor sekunder, seperti yang diwakili olehadapters folder, menerapkan integrasi eksternal yang diperlukan oleh port domain. Sebuah repositori database adalah contoh yang bagus dari adaptor sekunder. Ketika sistem database berubah, Anda dapat menulis adaptor baru dengan menggunakan implementasi yang didefinisikan oleh domain. Tidak perlu mengubah domain atau logika bisnis. testsSubfolder berisi tes integrasi eksternal untuk setiap adaptor.

Contoh infrastruktur padaAWS

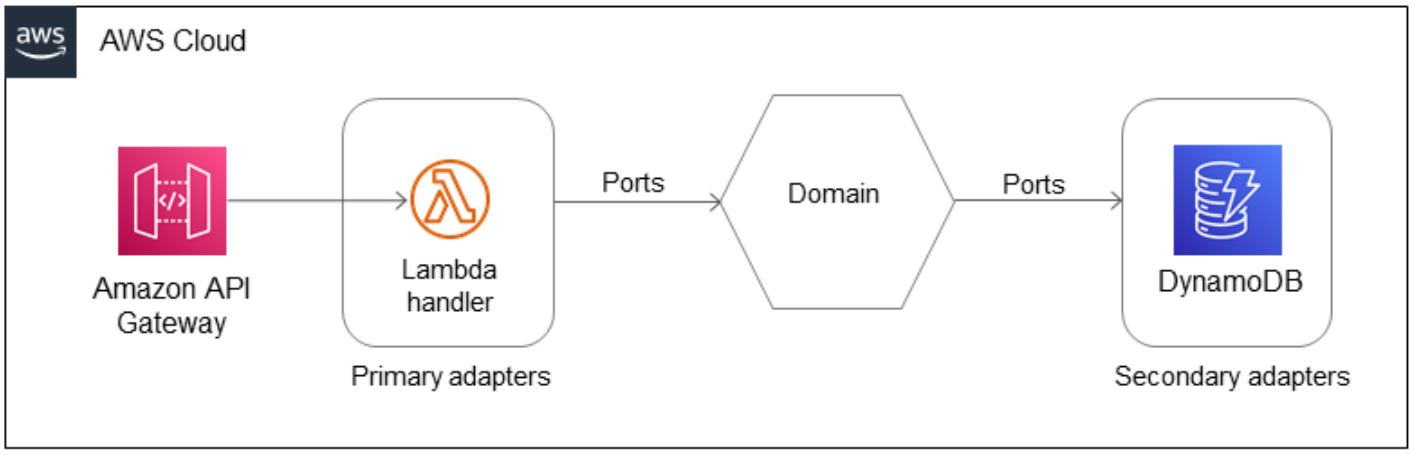
Bagian ini memberikan contoh untuk merancang infrastruktur untuk aplikasi AndaAWS yang dapat Anda gunakan untuk mengimplementasikan arsitektur heksagonal. Kami menyarankan Anda memulai dengan arsitektur sederhana untuk membangun produk layak minimum (MVP). Sebagian besar layanan mikro memerlukan satu titik masuk untuk menangani permintaan klien, lapisan komputasi untuk menjalankan kode, dan lapisan persistensi untuk menyimpan data. AWSLayanan berikut adalah kandidat yang bagus untuk digunakan sebagai klien, adaptor primer, dan adaptor sekunder dalam arsitektur heksagonal:

- Klien: Amazon API Gateway, Amazon Simple Queue Service (Amazon SQS), Elastic Load Balancing, Amazon EventBridge
- Adaptor utama:AWS Lambda Amazon Elastic Container Service (Amazon Elastic Elastic Qubernetes Service (Amazon EKS), Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Compute Cloud (Amazon EC2)
- Adaptor sekunder: Amazon DynamoDB, Amazon Relational Database Service (Amazon RDS), Amazon Aurora, API Gateway, Amazon SQS, Elastic Load Balancing, EventBridge, Amazon Simple Notification Service (Amazon SNS)

Bagian berikut membahas layanan ini dalam konteks arsitektur heksagonal secara lebih rinci.

Mulai sederhana sederhana

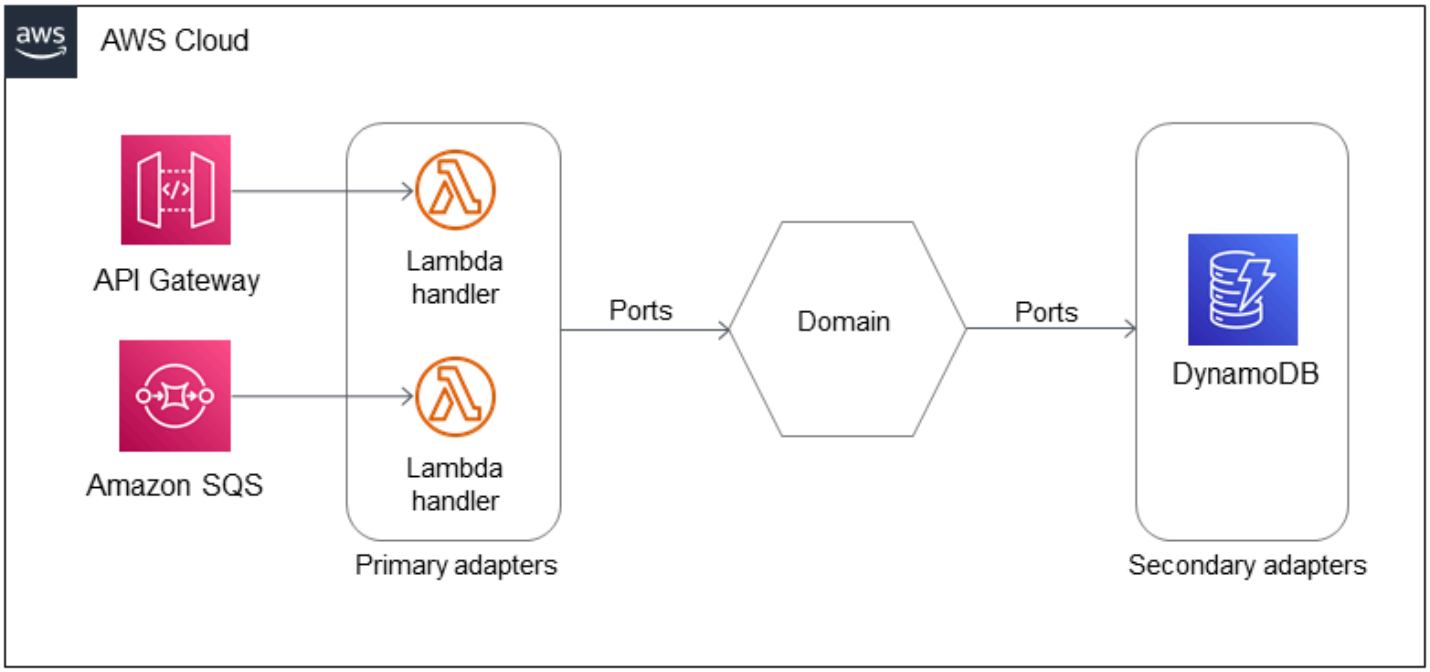
Kami menyarankan Anda memulai sederhana ketika Anda arsitek aplikasi dengan menggunakan arsitektur heksagonal. Dalam contoh ini, API Gateway digunakan sebagai klien (REST API), Lambda digunakan sebagai adaptor utama (komputasi), dan DynamoDB digunakan sebagai adaptor sekunder (persistensi). Klien gateway memanggil titik masuk, yang, dalam hal ini, adalah handler Lambda.



Arsitektur ini sepenuhnya tanpa server dan memberi arsitek titik awal yang baik. Kami menyarankan Anda menggunakan pola perintah dalam domain karena membuat kode lebih mudah untuk mempertahankan, dan menyesuaikan dengan bisnis baru dan persyaratan non-fungsional. Arsitektur ini bisa cukup untuk membangun layanan mikro sederhana dengan beberapa operasi.

Terapkan pola CQRS

Kami menyarankan Anda beralih ke pola CQRS jika jumlah operasi pada domain akan skala. Anda dapat menerapkan pola CQRS sebagai arsitektur sepenuhnya tanpa serverAWS dengan menggunakan contoh berikut.

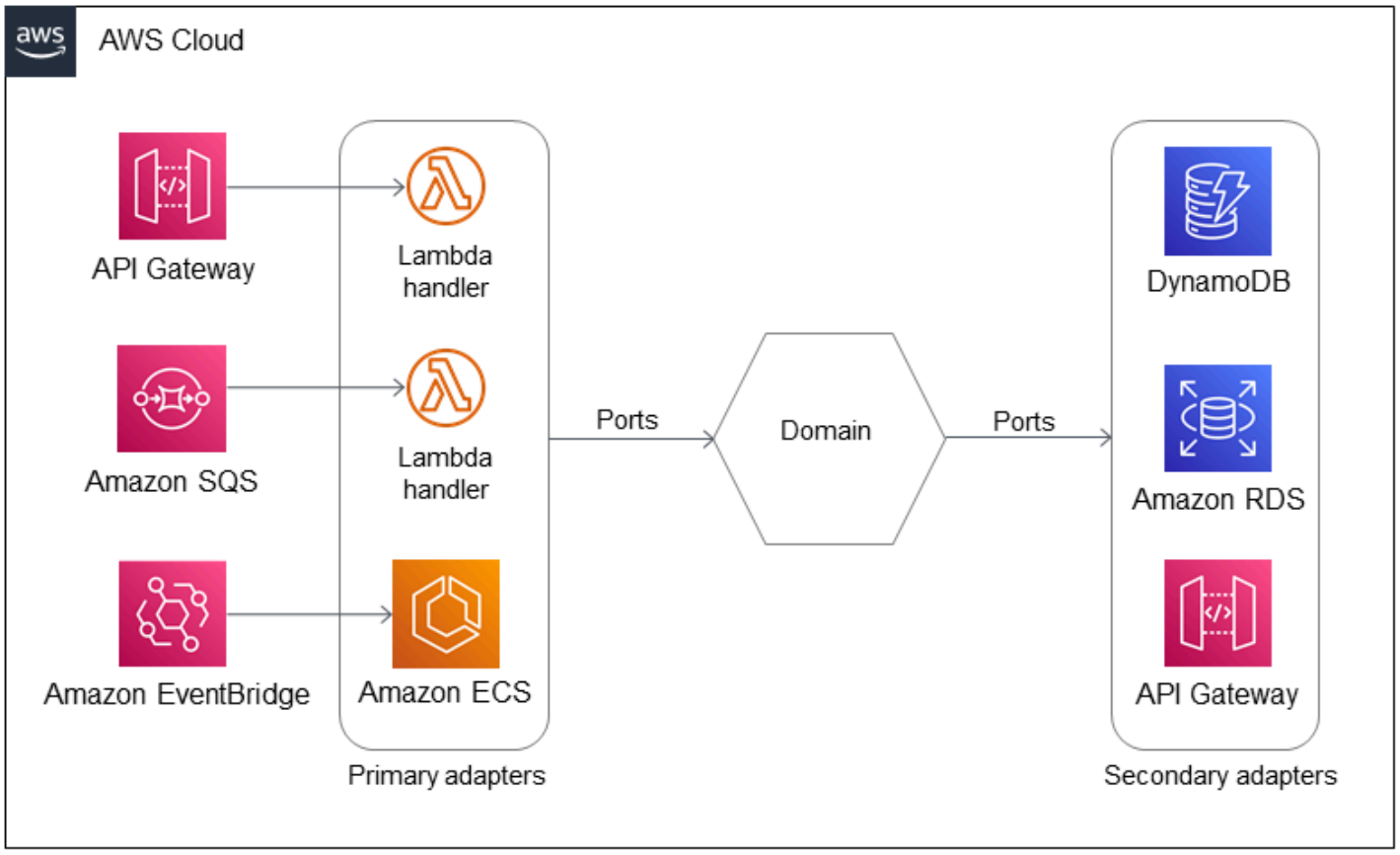


Contoh ini menggunakan dua handler Lambda, satu untuk query dan satu untuk perintah. Kueri dijalankan secara sinkron dengan menggunakan gateway API sebagai klien. Perintah dijalankan secara asinkron dengan menggunakan Amazon SQS sebagai klien.

Arsitektur ini mencakup beberapa klien (API Gateway dan Amazon SQS) dan beberapa adapter primer (Lambda), yang dipanggil oleh titik masuk yang sesuai (penangan Lambda). Semua komponen termasuk dalam konteks terbatas yang sama, sehingga mereka berada dalam domain yang sama.

Mengembangkan arsitektur dengan menambahkan kontainer, database relasional, dan API eksternal

Wadah adalah pilihan yang baik untuk tugas yang berjalan lama. Anda mungkin juga ingin menggunakan database relasional jika Anda memiliki skema data yang telah ditetapkan dan ingin mendapatkan keuntungan dari kekuatan bahasa SQL. Selain itu, domain harus berkomunikasi dengan API eksternal. Anda dapat mengembangkan contoh arsitektur untuk mendukung persyaratan ini seperti yang ditunjukkan dalam diagram berikut.

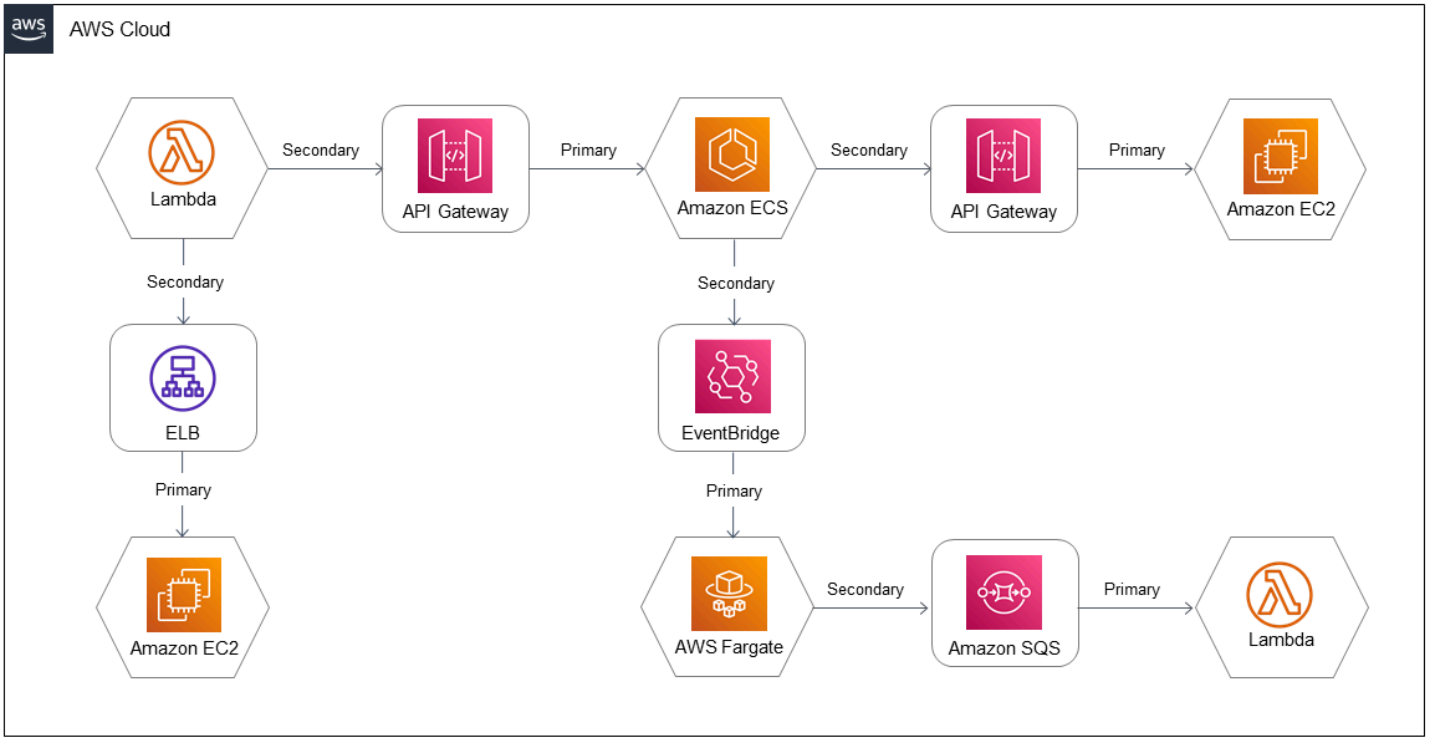


Contoh ini menggunakan Amazon ECS sebagai adaptor utama untuk meluncurkan tugas yang berjalan lama di domain. Amazon EventBridge (klien) memulai tugas Amazon ECS (titik masuk) ketika peristiwa tertentu terjadi. Arsitektur mencakup Amazon RDS sebagai adaptor sekunder lain untuk menyimpan data relasional. Ini juga menambahkan gateway API lain sebagai adaptor sekunder untuk memanggil panggilan API eksternal. Akibatnya, arsitektur menggunakan beberapa adapter primer dan sekunder yang mengandalkan lapisan komputasi yang mendasari yang berbeda dalam satu domain bisnis.

Domain selalu longgar ditambah dengan semua adapter primer dan sekunder melalui abstraksi yang disebut port. Domain mendefinisikan apa yang dibutuhkan dari dunia luar dengan menggunakan port. Karena merupakan tanggung jawab adaptor untuk mengimplementasikan port, beralih dari satu adaptor ke adaptor lainnya tidak memengaruhi domain. Misalnya, Anda dapat beralih dari Amazon DynamoDB ke Amazon RDS dengan menulis adaptor baru, tanpa memengaruhi domain.

Tambahkan lebih banyak domain (zoom out)

Arsitektur heksagonal sejajar dengan prinsip-prinsip arsitektur microservices. Contoh arsitektur yang ditunjukkan sejauh ini berisi satu domain (atau konteks terbatas). Aplikasi biasanya mencakup beberapa domain, yang perlu berkomunikasi melalui adaptor primer dan sekunder. Setiap domain mewakili layanan mikro dan digabungkan secara longgar dengan domain lain.



Dalam arsitektur ini, setiap domain menggunakan serangkaian lingkungan komputasi yang berbeda. (Setiap domain mungkin juga memiliki beberapa lingkungan komputasi, seperti pada contoh sebelumnya.) Setiap domain mendefinisikan antarmuka yang diperlukan untuk berkomunikasi dengan domain lain melalui port. Port diimplementasikan dengan menggunakan adaptor primer dan sekunder. Dengan cara ini, domain tidak terpengaruh jika ada perubahan pada adaptor. Selain itu, domain dipisahkan satu sama lain.

Dalam contoh arsitektur yang ditunjukkan dalam diagram sebelumnya, Lambda, Amazon EC2, Amazon ECS, danAWS Fargate digunakan sebagai adaptor utama. API Gateway, Elastic Load Balancing EventBridge,, dan Amazon SQS digunakan sebagai adaptor sekunder.

Pertanyaan yang Sering Diajukan

Mengapa saya harus menggunakan arsitektur heksagonal?

Arsitektur heksagonal menggeser fokus pengembang ke logika domain, menyederhanakan otomatisasi pengujian, dan meningkatkan kualitas kode dan kemampuan beradaptasi. Perbaikan ini menghasilkan waktu yang lebih cepat untuk memasarkan dan penskalaan teknis dan organisasi yang lebih mudah.

Mengapa saya harus menggunakan desain berbasis domain?

Desain berbasis domain (DDD) memungkinkan Anda membangun komponen dan konstruksi perangkat lunak dengan menggunakan bahasa yang sama antara pemangku kepentingan bisnis dan insinyur. DDD membantu Anda mengelola kompleksitas perangkat lunak dan merupakan strategi yang efektif untuk memelihara produk perangkat lunak dalam jangka panjang.

Dapatkah saya mempraktikkan pengembangan berbasis uji tanpa arsitektur heksagonal?

Ya. Test-driven development (TDD) tidak terbatas pada pola desain perangkat lunak tertentu. Namun, arsitektur heksagonal membuatnya lebih mudah untuk berlatih TDD.

Dapatkah saya menskalakan produk saya tanpa arsitektur heksagonal dan desain berbasis domain?

Ya. Penskalaan produk teknis dan organisasi dapat dicapai dengan sebagian besar pola desain. Namun, arsitektur heksagonal dan DDD membuatnya lebih mudah untuk skala dan lebih efektif untuk proyek-proyek besar dalam jangka panjang.

Teknologi apa yang harus saya gunakan untuk menerapkan arsitektur heksagonal?

Arsitektur heksagonal tidak terbatas pada tumpukan teknologi tertentu. Kami menyarankan Anda memilih teknologi yang mendukung ketergantungan inversi dan unit testing.

Saya mengembangkan produk minimum yang layak. Apakah masuk akal untuk menghabiskan waktu memikirkan arsitektur perangkat lunak?

Ya. Kami menyarankan Anda menggunakan pola desain yang akrab bagi Anda untuk MVP. Kami mendorong Anda untuk mencoba dan berlatih arsitektur heksagonal sampai teknisi Anda merasa nyaman dengannya. Membangun arsitektur heksagonal untuk proyek baru tidak memerlukan investasi waktu yang jauh lebih besar daripada memulai tanpa arsitektur apa pun.

Saya mengembangkan produk minimum yang layak dan tidak punya waktu untuk menulis tes.

Jika MVP Anda berisi logika bisnis, kami sangat menyarankan untuk menulis tes otomatis untuk itu. Ini akan mengurangi loop umpan balik dan menghemat waktu.

Pola desain tambahan apa yang dapat saya gunakan dengan arsitektur heksagonal?

Gunakan pola CQRS untuk mendukung penskalaan sistem secara keseluruhan. Gunakan pola repositori untuk menyimpan dan memulihkan model domain Anda. Gunakan unit pola kerja untuk mengelola langkah-langkah proses transaksional. Gunakan komposisi atas warisan untuk memodelkan agregat domain, entitas, dan objek nilai. Jangan membangun hirarki objek yang kompleks.

Langkah selanjutnya

- Biasakan diri Anda lebih jauh dengan konsep desain berbasis domain dengan membaca tautan yang dikumpulkan diSumber daya bagian ini.
- Jika Anda menerapkan proyek baru, gunakan template struktur proyek yang disediakan dalam panduan ini dan terapkan beberapa fitur.
- Jika Anda sedang dalam proses mengimplementasikan proyek yang ada, identifikasi kode yang dapat dibagi antara operasi hanya-baca dan hanya tulis. Abstrak read-only kode ke layanan query, dan tempat write-only kode ke penangan perintah.
- Saat struktur proyek dasar Anda ada, tulis pengujian unit, buat integrasi berkelanjutan (CI) dengan otomatisasi pengujian, dan ikuti praktik pengembangan berbasis pengujian (TDD).

Sumber daya

Referensi

- Struktur proyek Python dalam arsitektur heksagonal menggunakanAWS Lambda (pola BimbinganAWS Preskriptif)
- Tim Agile (Situs web Agile Framework Berskala)
- Pola arsitektur dengan Python, oleh Harry Percival dan Bob Gregory (O'Reilly Media, 31 Maret 2020), khususnya bab-babak berikut:
  - Perintah dan Command Handler
  - Pemisahan Tanggung Jawab Command-Query (CQRS)
  - Pola Repositori
- Menyerbu acara: Pendekatan paling cerdas untuk kolaborasi di luar batas silo, oleh Alberto Brandolini (situs web Event Storming)
- Demystifying Hukum Conway, oleh Sam Newman (Situs web Thoughtworks, 30 Juni 2014)
- Mengembangkan arsitektur evolusi denganAWS Lambda, oleh James Beswick (AWSCompute Blog, 8 Juli 2021)
- Bahasa Domain: Mengatasi Kompleksitas di Jantung Perangkat Lunak (situs web Bahasa Domain)
- Fasad, dari Dive Into Design Patterns oleh Alexander Shvets (ebook, 5 Desember 2018)
- GivenWhenThen, oleh Martin Fowler (21 Agustus 2013)
- Menerapkan Desain Berbasis Domain, oleh Vaughn Vernon (Addison-Wesley Professional, Februari 2013)
- Inverse Conway Manuver (Situs web Thoughtworks, 8 Juli 2014)
- Pola Bulan Ini: Red Green Refactor (situs web dZone, 2 Juni 2017)
- Prinsip Desain SOLID Dijelaskan: Prinsip Pembalikan Ketergantungan dengan Contoh Kode, oleh Thorben Janssen (situs Stackify, 7 Mei 2018)
- Prinsip SOLID: Penjelasan dan contoh, oleh Simon Hoiberg (Situs web ITNEXT, 1 Jan 2019)
- Seni Pengembangan Agile: Pengembangan Berbasis Tes, oleh James Shore dan Shane Warden (O'Reilly Media, 25 Maret 2010)
- Prinsip SOLID Pemrograman Berorientasi Objek Dijelaskan dalam Bahasa Inggris Biasa, oleh Yigit KemalfreeCodeCamp Erinc (posting pemrograman berorientasi objek, 20 Agustus 2020)
- Apa itu Arsitektur Berbasis Kejatan-Didorong? (AWSsitus web)

AWSjasa

- Amazon API Gateway
- Amazon Aurora
- Amazon DynamoDB
- Amazon Elastic Compute Cloud (Amazon EC2)
- Amazon Elastic Container Service (Amazon ECS)
- Amazon Elastic Kubernetes Service (Amazon EKS)
- Elastic Load Balancing
- AmazonEventBridge
- AWS Fargate
- AWS Lambda
- Amazon Relational Database Service (Amazon RDS)
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)

Alat lainnya

- Moto
- LocalStack

Riwayat dokumen

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

Perubahan	Deskripsi	Tanggal
Publikasi awal	—	15 Juni 20

AWS Glosarium Panduan Preskriptif

Berikut ini adalah istilah yang umum digunakan dalam strategi, panduan, dan pola yang disediakan oleh Panduan AWS Preskriptif. Untuk menyarankan entri, silakan gunakan tautan Berikan umpan balik di akhir glosarium.

Nomor

7 Rs

Tujuh strategi migrasi umum untuk memindahkan aplikasi ke cloud. Strategi ini dibangun di atas 5 Rs yang diidentifikasi Gartner pada tahun 2011 dan terdiri dari yang berikut:

- Refactor/Re-Architect — Memindahkan aplikasi dan memodifikasi arsitekturnya dengan memanfaatkan sepenuhnya fitur cloud-native untuk meningkatkan kelincahan, kinerja, dan skalabilitas. Ini biasanya melibatkan porting sistem operasi dan database. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Aurora PostgreSQL Compatible Edition.
- Replatform (angkat dan bentuk ulang) — Pindahkan aplikasi ke cloud, dan perkenalkan beberapa tingkat pengoptimalan untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Relational Database Service (Amazon RDS) untuk Oracle di. AWS Cloud
- Pembelian kembali (drop and shop) - Beralih ke produk yang berbeda, biasanya dengan beralih dari lisensi tradisional ke model SaaS. Contoh: Migrasikan sistem manajemen hubungan pelanggan (CRM) Anda ke Salesforce.com.
- Rehost (lift dan shift) — Pindahkan aplikasi ke cloud tanpa membuat perubahan apa pun untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Oracle pada instans EC2 di. AWS Cloud
- Relokasi (hypervisor-level lift and shift) — Pindahkan infrastruktur ke cloud tanpa membeli perangkat keras baru, menulis ulang aplikasi, atau memodifikasi operasi yang ada. Anda memigrasikan server dari platform lokal ke layanan cloud untuk platform yang sama. Contoh: Migrasikan Microsoft Hyper-V aplikasi ke AWS.
- Pertahankan (kunjungi kembali) - Simpan aplikasi di lingkungan sumber Anda. Ini mungkin termasuk aplikasi yang memerlukan refactoring besar, dan Anda ingin menunda pekerjaan itu sampai nanti, dan aplikasi lama yang ingin Anda pertahankan, karena tidak ada pembenaran bisnis untuk memigrasikannya.

- Pensiun — Menonaktifkan atau menghapus aplikasi yang tidak lagi diperlukan di lingkungan sumber Anda.

A

ABAC

Lihat kontrol akses berbasis atribut.

layanan abstrak

Lihat layanan terkelola.

ASAM

Lihat atomisitas, konsistensi, isolasi, daya tahan.

migrasi aktif-aktif

Metode migrasi database di mana database sumber dan target tetap sinkron (dengan menggunakan alat replikasi dua arah atau operasi penulisan ganda), dan kedua database menangani transaksi dari menghubungkan aplikasi selama migrasi. Metode ini mendukung migrasi dalam batch kecil yang terkontrol alih-alih memerlukan pemotongan satu kali. Ini lebih fleksibel tetapi membutuhkan lebih banyak pekerjaan daripada migrasi aktif-pasif.

migrasi aktif-pasif

Metode migrasi database di mana database sumber dan target disimpan dalam sinkron, tetapi hanya database sumber yang menangani transaksi dari menghubungkan aplikasi sementara data direplikasi ke database target. Basis data target tidak menerima transaksi apa pun selama migrasi.

fungsi agregat

Fungsi SQL yang beroperasi pada sekelompok baris dan menghitung nilai pengembalian tunggal untuk grup. Contoh fungsi agregat meliputi `SUM` dan`MAX`.

AI

Lihat kecerdasan buatan.

AIOps

Lihat operasi kecerdasan buatan.

anonimisasi

Proses menghapus informasi pribadi secara permanen dalam kumpulan data. Anonimisasi dapat membantu melindungi privasi pribadi. Data anonim tidak lagi dianggap sebagai data pribadi.

anti-pola

Solusi yang sering digunakan untuk masalah berulang di mana solusinya kontra-produktif, tidak efektif, atau kurang efektif daripada alternatif.

kontrol aplikasi

Pendekatan keamanan yang memungkinkan penggunaan hanya aplikasi yang disetujui untuk membantu melindungi sistem dari malware.

portofolio aplikasi

Kumpulan informasi rinci tentang setiap aplikasi yang digunakan oleh organisasi, termasuk biaya untuk membangun dan memelihara aplikasi, dan nilai bisnisnya. Informasi ini adalah kunci untuk penemuan portofolio dan proses analisis dan membantu mengidentifikasi dan memprioritaskan aplikasi yang akan dimigrasi, dimodernisasi, dan dioptimalkan.

kecerdasan buatan (AI)

Bidang ilmu komputer yang didedikasikan untuk menggunakan teknologi komputasi untuk melakukan fungsi kognitif yang biasanya terkait dengan manusia, seperti belajar, memecahkan masalah, dan mengenali pola. Untuk informasi lebih lanjut, lihat Apa itu Kecerdasan Buatan?

operasi kecerdasan buatan (AIOps)

Proses menggunakan teknik pembelajaran mesin untuk memecahkan masalah operasional, mengurangi insiden operasional dan intervensi manusia, dan meningkatkan kualitas layanan. Untuk informasi selengkapnya tentang cara AIOps digunakan dalam strategi AWS migrasi, lihat panduan integrasi operasi.

enkripsi asimetris

Algoritma enkripsi yang menggunakan sepasang kunci, kunci publik untuk enkripsi dan kunci pribadi untuk dekripsi. Anda dapat berbagi kunci publik karena tidak digunakan untuk dekripsi, tetapi akses ke kunci pribadi harus sangat dibatasi.

atomisitas, konsistensi, isolasi, daya tahan (ACID)

Satu set properti perangkat lunak yang menjamin validitas data dan keandalan operasional database, bahkan dalam kasus kesalahan, kegagalan daya, atau masalah lainnya.

kontrol akses berbasis atribut (ABAC)

Praktik membuat izin berbutir halus berdasarkan atribut pengguna, seperti departemen, peran pekerjaan, dan nama tim. Untuk informasi selengkapnya, lihat ABAC untuk AWS dokumentasi AWS Identity and Access Management (IAM).

sumber data otoritatif

Lokasi di mana Anda menyimpan versi utama data, yang dianggap sebagai sumber informasi yang paling dapat diandalkan. Anda dapat menyalin data dari sumber data otoritatif ke lokasi lain untuk tujuan memproses atau memodifikasi data, seperti menganonimkan, menyunting, atau membuat nama samaran.

Zona Ketersediaan

Lokasi berbeda di dalam Wilayah AWS yang terisolasi dari kegagalan di Availability Zone lainnya dan menyediakan konektivitas jaringan latensi rendah yang murah ke Availability Zone lainnya di Wilayah yang sama.

AWS Kerangka Adopsi Cloud (AWS CAF)

Kerangka pedoman dan praktik terbaik AWS untuk membantu organisasi mengembangkan rencana yang efisien dan efektif untuk bergerak dengan sukses ke cloud. AWS CAF mengatur panduan ke dalam enam area fokus yang disebut perspektif: bisnis, orang, tata kelola, platform, keamanan, dan operasi. Perspektif bisnis, orang, dan tata kelola fokus pada keterampilan dan proses bisnis; perspektif platform, keamanan, dan operasi fokus pada keterampilan dan proses teknis. Misalnya, perspektif masyarakat menargetkan pemangku kepentingan yang menangani sumber daya manusia (SDM), fungsi kepegawaian, dan manajemen orang. Untuk perspektif ini, AWS CAF memberikan panduan untuk pengembangan, pelatihan, dan komunikasi orang untuk membantu mempersiapkan organisasi untuk adopsi cloud yang sukses. Untuk informasi lebih lanjut, lihat situs web AWS CAF dan whitepaper AWS CAF.

AWS Kerangka Kualifikasi Beban Kerja (AWS WQF)

Alat yang mengevaluasi beban kerja migrasi database, merekomendasikan strategi migrasi, dan memberikan perkiraan kerja. AWS WQF disertakan dengan AWS Schema Conversion Tool ()AWS SCT. Ini menganalisis skema database dan objek kode, kode aplikasi, dependensi, dan karakteristik kinerja, dan memberikan laporan penilaian.

B

bot buruk

Bot yang dimaksudkan untuk mengganggu atau menyebabkan kerugian bagi individu atau organisasi.

BCP

Lihat perencanaan kontinuitas bisnis.

grafik perilaku

Pandangan interaktif yang terpadu tentang perilaku dan interaksi sumber daya dari waktu ke waktu. Anda dapat menggunakan grafik perilaku dengan Amazon Detective untuk memeriksa upaya logon yang gagal, panggilan API yang mencurigakan, dan tindakan serupa. Untuk informasi selengkapnya, lihat Data dalam grafik perilaku di dokumentasi Detektif.

sistem big-endian

Sistem yang menyimpan byte paling signifikan terlebih dahulu. Lihat juga endianness.

klasifikasi biner

Sebuah proses yang memprediksi hasil biner (salah satu dari dua kelas yang mungkin). Misalnya, model ML Anda mungkin perlu memprediksi masalah seperti “Apakah email ini spam atau bukan spam?” atau “Apakah produk ini buku atau mobil?”

filter mekar

Struktur data probabilistik dan efisien memori yang digunakan untuk menguji apakah suatu elemen adalah anggota dari suatu himpunan.

deployment biru/hijau

Strategi penyebaran tempat Anda membuat dua lingkungan yang terpisah namun identik. Anda menjalankan versi aplikasi saat ini di satu lingkungan (biru) dan versi aplikasi baru di lingkungan lain (hijau). Strategi ini membantu Anda dengan cepat memutar kembali dengan dampak minimal.

bot

Aplikasi perangkat lunak yang menjalankan tugas otomatis melalui internet dan mensimulasikan aktivitas atau interaksi manusia. Beberapa bot berguna atau bermanfaat, seperti perayap web yang mengindeks informasi di internet. Beberapa bot lain, yang dikenal sebagai bot buruk, dimaksudkan untuk mengganggu atau membahayakan individu atau organisasi.

botnet

Jaringan bot yang terinfeksi oleh malware dan berada di bawah kendali satu pihak, yang dikenal sebagai bot herder atau operator bot. Botnet adalah mekanisme paling terkenal untuk skala bot dan dampaknya.

cabang

Area berisi repositori kode. Cabang pertama yang dibuat dalam repositori adalah cabang utama. Anda dapat membuat cabang baru dari cabang yang ada, dan Anda kemudian dapat mengembangkan fitur atau memperbaiki bug di cabang baru. Cabang yang Anda buat untuk membangun fitur biasanya disebut sebagai cabang fitur. Saat fitur siap dirilis, Anda menggabungkan cabang fitur kembali ke cabang utama. Untuk informasi selengkapnya, lihat Tentang cabang (GitHub dokumentasi).

akses break-glass

Dalam keadaan luar biasa dan melalui proses yang disetujui, cara cepat bagi pengguna untuk mendapatkan akses ke Akun AWS yang biasanya tidak memiliki izin untuk mengaksesnya. Untuk informasi lebih lanjut, lihat indikator Implementasikan prosedur break-glass dalam panduan Well-Architected AWS .

strategi brownfield

Infrastruktur yang ada di lingkungan Anda. Saat mengadopsi strategi brownfield untuk arsitektur sistem, Anda merancang arsitektur di sekitar kendala sistem dan infrastruktur saat ini. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

cache penyangga

Area memori tempat data yang paling sering diakses disimpan.

kemampuan bisnis

Apa yang dilakukan bisnis untuk menghasilkan nilai (misalnya, penjualan, layanan pelanggan, atau pemasaran). Arsitektur layanan mikro dan keputusan pengembangan dapat didorong oleh kemampuan bisnis. Untuk informasi selengkapnya, lihat bagian Terorganisir di sekitar kemampuan bisnis dari Menjalankan layanan mikro kontainer di whitepaper. AWS

perencanaan kelangsungan bisnis (BCP)

Rencana yang membahas dampak potensial dari peristiwa yang mengganggu, seperti migrasi skala besar, pada operasi dan memungkinkan bisnis untuk melanjutkan operasi dengan cepat.

C

KAFE

Lihat Kerangka Adopsi AWS Cloud.

penyebaran kenari

Rilis versi yang lambat dan bertahap untuk pengguna akhir. Ketika Anda yakin, Anda menyebarkan versi baru dan mengganti versi saat ini secara keseluruhan.

CCoE

Lihat Cloud Center of Excellence.

CDC

Lihat mengubah pengambilan data.

ubah pengambilan data (CDC)

Proses melacak perubahan ke sumber data, seperti tabel database, dan merekam metadata tentang perubahan tersebut. Anda dapat menggunakan CDC untuk berbagai tujuan, seperti mengaudit atau mereplikasi perubahan dalam sistem target untuk mempertahankan sinkronisasi.

rekayasa kekacauan

Sengaja memperkenalkan kegagalan atau peristiwa yang mengganggu untuk menguji ketahanan sistem. Anda dapat menggunakan AWS Fault Injection Service (AWS FIS) untuk melakukan eksperimen yang menekankan AWS beban kerja Anda dan mengevaluasi responsnya.

CI/CD

Lihat integrasi berkelanjutan dan pengiriman berkelanjutan.

klasifikasi

Proses kategorisasi yang membantu menghasilkan prediksi. Model ML untuk masalah klasifikasi memprediksi nilai diskrit. Nilai diskrit selalu berbeda satu sama lain. Misalnya, model mungkin perlu mengevaluasi apakah ada mobil dalam gambar atau tidak.

Enkripsi sisi klien

Enkripsi data secara lokal, sebelum target Layanan AWS menerimanya.

Cloud Center of Excellence (CCoE)

Tim multi-disiplin yang mendorong upaya adopsi cloud di seluruh organisasi, termasuk mengembangkan praktik terbaik cloud, memobilisasi sumber daya, menetapkan jadwal migrasi, dan memimpin organisasi melalui transformasi skala besar. Untuk informasi selengkapnya, lihat posting CCoE di Blog Strategi AWS Cloud Perusahaan.

komputasi cloud

Teknologi cloud yang biasanya digunakan untuk penyimpanan data jarak jauh dan manajemen perangkat IoT. Cloud computing umumnya terhubung ke teknologi edge computing.

model operasi cloud

Dalam organisasi TI, model operasi yang digunakan untuk membangun, mematangkan, dan mengoptimalkan satu atau lebih lingkungan cloud. Untuk informasi selengkapnya, lihat Membangun Model Operasi Cloud Anda.

tahap adopsi cloud

Empat fase yang biasanya dilalui organisasi ketika mereka bermigrasi ke AWS Cloud:

- Proyek — Menjalankan beberapa proyek terkait cloud untuk bukti konsep dan tujuan pembelajaran
- Foundation — Melakukan investasi dasar untuk meningkatkan adopsi cloud Anda (misalnya, membuat landing zone, mendefinisikan CCoE, membuat model operasi)
- Migrasi — Migrasi aplikasi individual
- Re-invention — Mengoptimalkan produk dan layanan, dan berinovasi di cloud

Tahapan ini didefinisikan oleh Stephen Orban dalam posting blog The Journey Toward Cloud-First & the Stages of Adoption di blog Strategi Perusahaan. AWS Cloud Untuk informasi tentang bagaimana kaitannya dengan strategi AWS migrasi, lihat panduan kesiapan migrasi.

CMDB

Lihat database manajemen konfigurasi.

repositori kode

Lokasi di mana kode sumber dan aset lainnya, seperti dokumentasi, sampel, dan skrip, disimpan dan diperbarui melalui proses kontrol versi. Repositori cloud umum termasuk GitHub atau. AWS CodeCommit Setiap versi kode disebut cabang. Dalam struktur layanan mikro, setiap repositori

dikhususkan untuk satu bagian fungsionalitas. Pipa CI/CD tunggal dapat menggunakan beberapa repositori.

cache dingin

Cache buffer yang kosong, tidak terisi dengan baik, atau berisi data basi atau tidak relevan. Ini mempengaruhi kinerja karena instance database harus membaca dari memori utama atau disk, yang lebih lambat daripada membaca dari cache buffer.

data dingin

Data yang jarang diakses dan biasanya historis. Saat menanyakan jenis data ini, kueri lambat biasanya dapat diterima. Memindahkan data ini ke tingkat penyimpanan atau kelas yang berkinerja lebih rendah dan lebih murah dapat mengurangi biaya.

visi komputer (CV)

Bidang AI yang menggunakan pembelajaran mesin untuk menganalisis dan mengekstrak informasi dari format visual seperti gambar dan video digital. Misalnya, AWS Panorama menawarkan perangkat yang menambahkan CV ke jaringan kamera lokal, dan Amazon SageMaker menyediakan algoritme pemrosesan gambar untuk CV.

konfigurasi drift

Untuk beban kerja, konfigurasi berubah dari status yang diharapkan. Ini dapat menyebabkan beban kerja menjadi tidak patuh, dan biasanya bertahap dan tidak disengaja.

database manajemen konfigurasi (CMDB)

Repositori yang menyimpan dan mengelola informasi tentang database dan lingkungan TI, termasuk komponen perangkat keras dan perangkat lunak dan konfigurasinya. Anda biasanya menggunakan data dari CMDB dalam penemuan portofolio dan tahap analisis migrasi.

paket kesesuaian

Kumpulan AWS Config aturan dan tindakan remediasi yang dapat Anda kumpulkan untuk menyesuaikan kepatuhan dan pemeriksaan keamanan Anda. Anda dapat menerapkan paket kesesuaian sebagai entitas tunggal di Akun AWS dan Wilayah, atau di seluruh organisasi, dengan menggunakan templat YAMM. Untuk informasi selengkapnya, lihat Paket kesesuaian dalam dokumentasi. AWS Config

integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD)

Proses mengotomatiskan sumber, membangun, menguji, pementasan, dan tahap produksi dari proses rilis perangkat lunak. CI/CD umumnya digambarkan sebagai pipa. CI/CD dapat membantu

Anda mengotomatiskan proses, meningkatkan produktivitas, meningkatkan kualitas kode, dan memberikan lebih cepat. Untuk informasi lebih lanjut, lihat Manfaat pengiriman berkelanjutan. CD juga dapat berarti penerapan berkelanjutan. Untuk informasi selengkapnya, lihat Continuous Delivery vs Continuous Deployment.

CV

Lihat visi komputer.

D

data saat istirahat

Data yang stasioner di jaringan Anda, seperti data yang ada di penyimpanan.

klasifikasi data

Proses untuk mengidentifikasi dan mengkategorikan data dalam jaringan Anda berdasarkan kekritisan dan sensitivitasnya. Ini adalah komponen penting dari setiap strategi manajemen risiko keamanan siber karena membantu Anda menentukan perlindungan dan kontrol retensi yang tepat untuk data. Klasifikasi data adalah komponen pilar keamanan dalam AWS Well-Architected Framework. Untuk informasi selengkapnya, lihat Klasifikasi data.

penyimpangan data

Variasi yang berarti antara data produksi dan data yang digunakan untuk melatih model ML, atau perubahan yang berarti dalam data input dari waktu ke waktu. Penyimpangan data dapat mengurangi kualitas, akurasi, dan keadilan keseluruhan dalam prediksi model ML.

data dalam transit

Data yang aktif bergerak melalui jaringan Anda, seperti antara sumber daya jaringan.

jala data

Kerangka arsitektur yang menyediakan kepemilikan data terdistribusi dan terdesentralisasi dengan manajemen dan tata kelola terpusat.

minimalisasi data

Prinsip pengumpulan dan pemrosesan hanya data yang sangat diperlukan. Mempraktikkan minimalisasi data AWS Cloud dapat mengurangi risiko privasi, biaya, dan jejak karbon analitik Anda.

perimeter data

Satu set pagar pembatas pencegahan di AWS lingkungan Anda yang membantu memastikan bahwa hanya identitas tepercaya yang mengakses sumber daya tepercaya dari jaringan yang diharapkan. Untuk informasi selengkapnya, lihat Membangun perimeter data pada AWS.

prapemrosesan data

Untuk mengubah data mentah menjadi format yang mudah diuraikan oleh model ML Anda. Preprocessing data dapat berarti menghapus kolom atau baris tertentu dan menangani nilai yang hilang, tidak konsisten, atau duplikat.

asal data

Proses melacak asal dan riwayat data sepanjang siklus hidupnya, seperti bagaimana data dihasilkan, ditransmisikan, dan disimpan.

subjek data

Individu yang datanya dikumpulkan dan diproses.

gudang data

Sistem manajemen data yang mendukung intelijen bisnis, seperti analitik. Gudang data biasanya berisi sejumlah besar data historis, dan biasanya digunakan untuk kueri dan analisis.

bahasa definisi database (DDL)

Pernyataan atau perintah untuk membuat atau memodifikasi struktur tabel dan objek dalam database.

bahasa manipulasi basis data (DHTML)

Pernyataan atau perintah untuk memodifikasi (memasukkan, memperbarui, dan menghapus) informasi dalam database.

DDL

Lihat bahasa definisi database.

ansambel yang dalam

Untuk menggabungkan beberapa model pembelajaran mendalam untuk prediksi. Anda dapat menggunakan ansambel dalam untuk mendapatkan prediksi yang lebih akurat atau untuk memperkirakan ketidakpastian dalam prediksi.

pembelajaran mendalam

Subbidang ML yang menggunakan beberapa lapisan jaringan saraf tiruan untuk mengidentifikasi pemetaan antara data input dan variabel target yang diinginkan.

defense-in-depth

Pendekatan keamanan informasi di mana serangkaian mekanisme dan kontrol keamanan dilapisi dengan cermat di seluruh jaringan komputer untuk melindungi kerahasiaan, integritas, dan ketersediaan jaringan dan data di dalamnya. Saat Anda mengadopsi strategi ini AWS, Anda menambahkan beberapa kontrol pada lapisan AWS Organizations struktur yang berbeda untuk membantu mengamankan sumber daya. Misalnya, defense-in-depth pendekatan mungkin menggabungkan otentikasi multi-faktor, segmentasi jaringan, dan enkripsi.

administrator yang didelegasikan

Di AWS Organizations, layanan yang kompatibel dapat mendaftarkan akun AWS anggota untuk mengelola akun organisasi dan mengelola izin untuk layanan tersebut. Akun ini disebut administrator yang didelegasikan untuk layanan itu. Untuk informasi selengkapnya dan daftar layanan yang kompatibel, lihat Layanan yang berfungsi dengan AWS Organizations AWS Organizations dokumentasi.

deployment

Proses pembuatan aplikasi, fitur baru, atau perbaikan kode tersedia di lingkungan target. Deployment melibatkan penerapan perubahan dalam basis kode dan kemudian membangun dan menjalankan basis kode itu di lingkungan aplikasi.

lingkungan pengembangan

Lihat lingkungan.

kontrol detektif

Kontrol keamanan yang dirancang untuk mendeteksi, mencatat, dan memperingatkan setelah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan kedua, memperingatkan Anda tentang peristiwa keamanan yang melewati kontrol pencegahan di tempat. Untuk informasi selengkapnya, lihat Kontrol Detektif dalam Menerapkan kontrol keamanan pada. AWS

pemetaan aliran nilai pengembangan (DVSM)

Sebuah proses yang digunakan untuk mengidentifikasi dan memprioritaskan kendala yang mempengaruhi kecepatan dan kualitas dalam siklus hidup pengembangan perangkat lunak. DVSM memperluas proses pemetaan aliran nilai yang awalnya dirancang untuk praktik

manufaktur ramping. Ini berfokus pada langkah-langkah dan tim yang diperlukan untuk menciptakan dan memindahkan nilai melalui proses pengembangan perangkat lunak.

kembar digital

Representasi virtual dari sistem dunia nyata, seperti bangunan, pabrik, peralatan industri, atau jalur produksi. Kembar digital mendukung pemeliharaan prediktif, pemantauan jarak jauh, dan optimalisasi produksi.

tabel dimensi

Dalam skema bintang, tabel yang lebih kecil yang berisi atribut data tentang data kuantitatif dalam tabel fakta. Atribut tabel dimensi biasanya bidang teks atau angka diskrit yang berperilaku seperti teks. Atribut ini biasanya digunakan untuk pembatasan kueri, pemfilteran, dan pelabelan set hasil.

musibah

Peristiwa yang mencegah beban kerja atau sistem memenuhi tujuan bisnisnya di lokasi utama yang digunakan. Peristiwa ini dapat berupa bencana alam, kegagalan teknis, atau akibat dari tindakan manusia, seperti kesalahan konfigurasi yang tidak disengaja atau serangan malware.

pemulihan bencana (DR)

Strategi dan proses yang Anda gunakan untuk meminimalkan downtime dan kehilangan data yang disebabkan oleh bencana. Untuk informasi selengkapnya, lihat Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML~

Lihat bahasa manipulasi database.

desain berbasis domain

Pendekatan untuk mengembangkan sistem perangkat lunak yang kompleks dengan menghubungkan komponennya ke domain yang berkembang, atau tujuan bisnis inti, yang dilayani oleh setiap komponen. Konsep ini diperkenalkan oleh Eric Evans dalam bukunya, Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003). Untuk informasi tentang cara menggunakan desain berbasis domain dengan pola gambar pencekik, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

DR

Lihat pemulihan bencana.

deteksi drift

Melacak penyimpangan dari konfigurasi dasar. Misalnya, Anda dapat menggunakan AWS CloudFormation untuk mendeteksi penyimpangan dalam sumber daya sistem, atau Anda dapat menggunakannya AWS Control Tower untuk mendeteksi perubahan di landing zone yang mungkin memengaruhi kepatuhan terhadap persyaratan tata kelola.

DVSM

Lihat pemetaan aliran nilai pengembangan.

E

EDA

Lihat analisis data eksplorasi.

komputasi tepi

Teknologi yang meningkatkan daya komputasi untuk perangkat pintar di tepi jaringan IoT. Jika dibandingkan dengan komputasi awan, komputasi tepi dapat mengurangi latensi komunikasi dan meningkatkan waktu respons.

enkripsi

Proses komputasi yang mengubah data plaintext, yang dapat dibaca manusia, menjadi ciphertext.

kunci enkripsi

String kriptografi dari bit acak yang dihasilkan oleh algoritma enkripsi. Panjang kunci dapat bervariasi, dan setiap kunci dirancang agar tidak dapat diprediksi dan unik.

endianness

Urutan byte disimpan dalam memori komputer. Sistem big-endian menyimpan byte paling signifikan terlebih dahulu. Sistem little-endian menyimpan byte paling tidak signifikan terlebih dahulu.

titik akhir

Lihat titik akhir layanan.

layanan endpoint

Layanan yang dapat Anda host di cloud pribadi virtual (VPC) untuk dibagikan dengan pengguna lain. Anda dapat membuat layanan endpoint dengan AWS PrivateLink dan memberikan izin

kepada prinsipal lain Akun AWS atau ke AWS Identity and Access Management (IAM). Akun atau prinsipal ini dapat terhubung ke layanan endpoint Anda secara pribadi dengan membuat titik akhir VPC antarmuka. Untuk informasi selengkapnya, lihat Membuat layanan titik akhir di dokumentasi Amazon Virtual Private Cloud (Amazon VPC).

perencanaan sumber daya perusahaan (ERP)

Sistem yang mengotomatiskan dan mengelola proses bisnis utama (seperti akuntansi, MES, dan manajemen proyek) untuk suatu perusahaan.

enkripsi amplop

Proses mengenkripsi kunci enkripsi dengan kunci enkripsi lain. Untuk informasi selengkapnya, lihat Enkripsi amplop dalam dokumentasi AWS Key Management Service (AWS KMS).

lingkungan

Sebuah contoh dari aplikasi yang sedang berjalan. Berikut ini adalah jenis lingkungan yang umum dalam komputasi awan:

- Development Environment — Sebuah contoh dari aplikasi yang berjalan yang hanya tersedia untuk tim inti yang bertanggung jawab untuk memelihara aplikasi. Lingkungan pengembangan digunakan untuk menguji perubahan sebelum mempromosikannya ke lingkungan atas. Jenis lingkungan ini kadang-kadang disebut sebagai lingkungan pengujian.
- lingkungan yang lebih rendah — Semua lingkungan pengembangan untuk aplikasi, seperti yang digunakan untuk build awal dan pengujian.
- lingkungan produksi — Sebuah contoh dari aplikasi yang berjalan yang pengguna akhir dapat mengakses. Dalam pipa CI/CD, lingkungan produksi adalah lingkungan penyebaran terakhir.
- lingkungan atas — Semua lingkungan yang dapat diakses oleh pengguna selain tim pengembangan inti. Ini dapat mencakup lingkungan produksi, lingkungan praproduksi, dan lingkungan untuk pengujian penerimaan pengguna.

epik

Dalam metodologi tangkas, kategori fungsional yang membantu mengatur dan memprioritaskan pekerjaan Anda. Epik memberikan deskripsi tingkat tinggi tentang persyaratan dan tugas implementasi. Misalnya, epos keamanan AWS CAF mencakup manajemen identitas dan akses, kontrol detektif, keamanan infrastruktur, perlindungan data, dan respons insiden. Untuk informasi selengkapnya tentang epos dalam strategi AWS migrasi, lihat panduan implementasi program.

ERP

Lihat perencanaan sumber daya perusahaan.

analisis data eksplorasi (EDA)

Proses menganalisis dataset untuk memahami karakteristik utamanya. Anda mengumpulkan atau mengumpulkan data dan kemudian melakukan penyelidikan awal untuk menemukan pola, mendeteksi anomali, dan memeriksa asumsi. EDA dilakukan dengan menghitung statistik ringkasan dan membuat visualisasi data.

F

tabel fakta

Tabel tengah dalam skema bintang. Ini menyimpan data kuantitatif tentang operasi bisnis. Biasanya, tabel fakta berisi dua jenis kolom: kolom yang berisi ukuran dan yang berisi kunci asing ke tabel dimensi.

gagal cepat

Filosofi yang menggunakan pengujian yang sering dan bertahap untuk mengurangi siklus hidup pengembangan. Ini adalah bagian penting dari pendekatan tangkas.

batas isolasi kesalahan

Dalam AWS Cloud, batas seperti Availability Zone, Wilayah AWS, control plane, atau data plane yang membatasi efek kegagalan dan membantu meningkatkan ketahanan beban kerja. Untuk informasi selengkapnya, lihat Batas Isolasi AWS Kesalahan.

cabang fitur

Lihat cabang.

fitur

Data input yang Anda gunakan untuk membuat prediksi. Misalnya, dalam konteks manufaktur, fitur bisa berupa gambar yang diambil secara berkala dari lini manufaktur.

pentingnya fitur

Seberapa signifikan fitur untuk prediksi model. Ini biasanya dinyatakan sebagai skor numerik yang dapat dihitung melalui berbagai teknik, seperti Shapley Additive Explanations (SHAP) dan gradien terintegrasi. Untuk informasi lebih lanjut, lihat Interpretabilitas model pembelajaran mesin dengan:.AWS

transformasi fitur

Untuk mengoptimalkan data untuk proses ML, termasuk memperkaya data dengan sumber tambahan, menskalakan nilai, atau mengekstrak beberapa set informasi dari satu bidang data. Hal ini memungkinkan model ML untuk mendapatkan keuntungan dari data. Misalnya, jika Anda memecah tanggal “2021-05-27 00:15:37" menjadi “2021", “Mei”, “Kamis”, dan “15", Anda dapat membantu algoritme pembelajaran mempelajari pola bernuansa yang terkait dengan komponen data yang berbeda.

FGAC

Lihat kontrol akses berbutir halus.

kontrol akses berbutir halus (FGAC)

Penggunaan beberapa kondisi untuk mengizinkan atau menolak permintaan akses.

migrasi flash-cut

Metode migrasi database yang menggunakan replikasi data berkelanjutan melalui pengambilan data perubahan untuk memigrasikan data dalam waktu sesingkat mungkin, alih-alih menggunakan pendekatan bertahap. Tujuannya adalah untuk menjaga downtime seminimal mungkin.

G

pemblokiran geografis

Lihat pembatasan geografis.

pembatasan geografis (pemblokiran geografis)

Di Amazon CloudFront, opsi untuk mencegah pengguna di negara tertentu mengakses distribusi konten. Anda dapat menggunakan daftar izinkan atau daftar blokir untuk menentukan negara yang disetujui dan dilarang. Untuk informasi selengkapnya, lihat Membatasi distribusi geografis konten Anda dalam dokumentasi. CloudFront

Alur kerja Gitflow

Pendekatan di mana lingkungan bawah dan atas menggunakan cabang yang berbeda dalam repositori kode sumber. Alur kerja Gitflow dianggap warisan, dan alur kerja berbasis batang adalah pendekatan modern yang lebih disukai.

strategi greenfield

Tidak adanya infrastruktur yang ada di lingkungan baru. Saat mengadopsi strategi greenfield untuk arsitektur sistem, Anda dapat memilih semua teknologi baru tanpa batasan kompatibilitas dengan infrastruktur yang ada, juga dikenal sebagai brownfield. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

pagar pembatas

Aturan tingkat tinggi yang membantu mengatur sumber daya, kebijakan, dan kepatuhan di seluruh unit organisasi (OU). Pagar pembatas preventif menegakkan kebijakan untuk memastikan keselarasan dengan standar kepatuhan. Mereka diimplementasikan dengan menggunakan kebijakan kontrol layanan dan batas izin IAM. Detective guardrails mendeteksi pelanggaran kebijakan dan masalah kepatuhan, dan menghasilkan peringatan untuk remediasi. Mereka diimplementasikan dengan menggunakan AWS Config, AWS Security Hub, Amazon GuardDuty AWS Trusted Advisor, Amazon Inspector, dan pemeriksaan khusus AWS Lambda .

H

HA

Lihat ketersediaan tinggi.

migrasi database heterogen

Memigrasi database sumber Anda ke database target yang menggunakan mesin database yang berbeda (misalnya, Oracle ke Amazon Aurora). Migrasi heterogen biasanya merupakan bagian dari upaya arsitektur ulang, dan mengubah skema dapat menjadi tugas yang kompleks. AWS menyediakan AWS SCT yang membantu dengan konversi skema.

ketersediaan tinggi (HA)

Kemampuan beban kerja untuk beroperasi terus menerus, tanpa intervensi, jika terjadi tantangan atau bencana. Sistem HA dirancang untuk gagal secara otomatis, secara konsisten memberikan kinerja berkualitas tinggi, dan menangani beban dan kegagalan yang berbeda dengan dampak kinerja minimal.

modernisasi sejarawan

Pendekatan yang digunakan untuk memodernisasi dan meningkatkan sistem teknologi operasional (OT) untuk melayani kebutuhan industri manufaktur dengan lebih baik. Sejarawan

adalah jenis database yang digunakan untuk mengumpulkan dan menyimpan data dari berbagai sumber di pabrik.

migrasi database homogen

Memigrasi database sumber Anda ke database target yang berbagi mesin database yang sama (misalnya, Microsoft SQL Server ke Amazon RDS for SQL Server). Migrasi homogen biasanya merupakan bagian dari upaya rehosting atau replatforming. Anda dapat menggunakan utilitas database asli untuk memigrasi skema.

data panas

Data yang sering diakses, seperti data real-time atau data translasi terbaru. Data ini biasanya memerlukan tingkat atau kelas penyimpanan berkinerja tinggi untuk memberikan respons kueri yang cepat.

perbaikan terbaru

Perbaikan mendesak untuk masalah kritis dalam lingkungan produksi. Karena urgensinya, perbaikan terbaru biasanya dibuat di luar alur kerja DevOps rilis biasa.

periode hypercare

Segera setelah cutover, periode waktu ketika tim migrasi mengelola dan memantau aplikasi yang dimigrasi di cloud untuk mengatasi masalah apa pun. Biasanya, periode ini panjangnya 1-4 hari. Pada akhir periode hypercare, tim migrasi biasanya mentransfer tanggung jawab untuk aplikasi ke tim operasi cloud.

I

IAc

Lihat infrastruktur sebagai kode.

kebijakan berbasis identitas

Kebijakan yang dilampirkan pada satu atau beberapa prinsip IAM yang mendefinisikan izin mereka dalam lingkungan. AWS Cloud

aplikasi idle

Aplikasi yang memiliki penggunaan CPU dan memori rata-rata antara 5 dan 20 persen selama periode 90 hari. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini atau mempertahankannya di tempat.

IIoT

Lihat Internet of Things industri.

infrastruktur yang tidak dapat diubah

Model yang menyebarkan infrastruktur baru untuk beban kerja produksi alih-alih memperbarui, menambal, atau memodifikasi infrastruktur yang ada. Infrastruktur yang tidak dapat diubah secara inheren lebih konsisten, andal, dan dapat diprediksi daripada infrastruktur yang dapat berubah. Untuk informasi selengkapnya, lihat praktik terbaik Deploy using immutable infrastructure di AWS Well-Architected Framework.

masuk (masuknya) VPC

Dalam arsitektur AWS multi-akun, VPC yang menerima, memeriksa, dan merutekan koneksi jaringan dari luar aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

migrasi inkremental

Strategi cutover di mana Anda memigrasikan aplikasi Anda dalam bagian-bagian kecil alih-alih melakukan satu cutover penuh. Misalnya, Anda mungkin hanya memindahkan beberapa layanan mikro atau pengguna ke sistem baru pada awalnya. Setelah Anda memverifikasi bahwa semuanya berfungsi dengan baik, Anda dapat secara bertahap memindahkan layanan mikro atau pengguna tambahan hingga Anda dapat menonaktifkan sistem lama Anda. Strategi ini mengurangi risiko yang terkait dengan migrasi besar.

Industri 4.0

Sebuah istilah yang diperkenalkan oleh Klaus Schwab pada tahun 2016 untuk merujuk pada modernisasi proses manufaktur melalui kemajuan dalam konektivitas, data real-time, otomatisasi, analitik, dan AI/ML.

infrastruktur

Semua sumber daya dan aset yang terkandung dalam lingkungan aplikasi.

infrastruktur sebagai kode (IAc)

Proses penyediaan dan pengelolaan infrastruktur aplikasi melalui satu set file konfigurasi. IAc dirancang untuk membantu Anda memusatkan manajemen infrastruktur, menstandarisasi sumber daya, dan menskalakan dengan cepat sehingga lingkungan baru dapat diulang, andal, dan konsisten.

Internet of Things industri (IIoT)

Penggunaan sensor dan perangkat yang terhubung ke internet di sektor industri, seperti manufaktur, energi, otomotif, perawatan kesehatan, ilmu kehidupan, dan pertanian. Untuk informasi selengkapnya, lihat Membangun strategi transformasi digital Internet of Things (IIoT) industri.

inspeksi VPC

Dalam arsitektur AWS multi-akun, VPC terpusat yang mengelola inspeksi lalu lintas jaringan antara VPC (dalam hal yang sama atau berbeda Wilayah AWS), internet, dan jaringan lokal. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

Internet of Things (IoT)

Jaringan objek fisik yang terhubung dengan sensor atau prosesor tertanam yang berkomunikasi dengan perangkat dan sistem lain melalui internet atau melalui jaringan komunikasi lokal. Untuk informasi selengkapnya, lihat Apa itu IoT?

interpretabilitas

Karakteristik model pembelajaran mesin yang menggambarkan sejauh mana manusia dapat memahami bagaimana prediksi model bergantung pada inputnya. Untuk informasi selengkapnya, lihat Interpretabilitas model pembelajaran mesin dengan AWS.

IoT

Lihat Internet of Things.

Perpustakaan informasi TI (ITIL)

Serangkaian praktik terbaik untuk memberikan layanan TI dan menyelaraskan layanan ini dengan persyaratan bisnis. ITIL menyediakan dasar untuk ITSM.

Manajemen layanan TI (ITSM)

Kegiatan yang terkait dengan merancang, menerapkan, mengelola, dan mendukung layanan TI untuk suatu organisasi. Untuk informasi tentang mengintegrasikan operasi cloud dengan alat ITSM, lihat panduan integrasi operasi.

ITIL

Lihat perpustakaan informasi TI.

ITSM

Lihat manajemen layanan TI.

L

kontrol akses berbasis label (LBAC)

Implementasi kontrol akses wajib (MAC) di mana pengguna dan data itu sendiri masing-masing secara eksplisit diberi nilai label keamanan. Persimpangan antara label keamanan pengguna dan label keamanan data menentukan baris dan kolom mana yang dapat dilihat oleh pengguna.

landing zone

Landing zone adalah AWS lingkungan multi-akun yang dirancang dengan baik yang dapat diskalakan dan aman. Ini adalah titik awal dari mana organisasi Anda dapat dengan cepat meluncurkan dan menyebarkan beban kerja dan aplikasi dengan percaya diri dalam lingkungan keamanan dan infrastruktur mereka. Untuk informasi selengkapnya tentang zona pendaratan, lihat Menyiapkan lingkungan multi-akun AWS yang aman dan dapat diskalakan.

migrasi besar

Migrasi 300 atau lebih server.

LBAC

Lihat kontrol akses berbasis label.

hak istimewa paling sedikit

Praktik keamanan terbaik untuk memberikan izin minimum yang diperlukan untuk melakukan tugas. Untuk informasi selengkapnya, lihat Menerapkan izin hak istimewa terkecil dalam dokumentasi IAM.

angkat dan geser

Lihat 7 Rs.

sistem endian kecil

Sebuah sistem yang menyimpan byte paling tidak signifikan terlebih dahulu. Lihat juga endianness.

lingkungan yang lebih rendah

Lihat lingkungan.

M

pembelajaran mesin (ML)

Jenis kecerdasan buatan yang menggunakan algoritma dan teknik untuk pengenalan pola dan pembelajaran. ML menganalisis dan belajar dari data yang direkam, seperti data Internet of Things (IoT), untuk menghasilkan model statistik berdasarkan pola. Untuk informasi selengkapnya, lihat Machine Learning.

cabang utama

Lihat cabang.

malware

Perangkat lunak yang dirancang untuk membahayakan keamanan atau privasi komputer. Malware dapat mengganggu sistem komputer, membocorkan informasi sensitif, atau mendapatkan akses yang tidak sah. Contoh malware termasuk virus, worm, ransomware, Trojan horse, spyware, dan keyloggers.

layanan terkelola

Layanan AWS yang AWS mengoperasikan lapisan infrastruktur, sistem operasi, dan platform, dan Anda mengakses titik akhir untuk menyimpan dan mengambil data. Amazon Simple Storage Service (Amazon S3) dan Amazon DynamoDB adalah contoh layanan terkelola. Ini juga dikenal sebagai layanan abstrak.

sistem eksekusi manufaktur (MES)

Sistem perangkat lunak untuk melacak, memantau, mendokumentasikan, dan mengendalikan proses produksi yang mengubah bahan baku menjadi produk jadi di lantai toko.

PETA

Lihat Program Percepatan Migrasi.

mekanisme

Proses lengkap di mana Anda membuat alat, mendorong adopsi alat, dan kemudian memeriksa hasilnya untuk melakukan penyesuaian. Mekanisme adalah siklus yang memperkuat dan meningkatkan dirinya sendiri saat beroperasi. Untuk informasi lebih lanjut, lihat Membangun mekanisme di AWS Well-Architected Framework.

akun anggota

Semua Akun AWS selain akun manajemen yang merupakan bagian dari organisasi di AWS Organizations. Akun dapat menjadi anggota dari hanya satu organisasi pada suatu waktu.

MES

Lihat sistem eksekusi manufaktur.

Transportasi Telemetri Antrian Pesan (MQTT)

Protokol komunikasi ringan machine-to-machine (M2M), berdasarkan pola terbitkan/berlangganan, untuk perangkat IoT yang dibatasi sumber daya.

layanan mikro

Layanan kecil dan independen yang berkomunikasi melalui API yang terdefinisi dengan baik dan biasanya dimiliki oleh tim kecil yang mandiri. Misalnya, sistem asuransi mungkin mencakup layanan mikro yang memetakan kemampuan bisnis, seperti penjualan atau pemasaran, atau subdomain, seperti pembelian, klaim, atau analitik. Manfaat layanan mikro termasuk kelincahan, penskalaan yang fleksibel, penyebaran yang mudah, kode yang dapat digunakan kembali, dan ketahanan. Untuk informasi selengkapnya, lihat Mengintegrasikan layanan mikro dengan menggunakan layanan tanpa AWS server.

arsitektur microservices

Pendekatan untuk membangun aplikasi dengan komponen independen yang menjalankan setiap proses aplikasi sebagai layanan mikro. Layanan mikro ini berkomunikasi melalui antarmuka yang terdefinisi dengan baik dengan menggunakan API ringan. Setiap layanan mikro dalam arsitektur ini dapat diperbarui, digunakan, dan diskalakan untuk memenuhi permintaan fungsi tertentu dari suatu aplikasi. Untuk informasi selengkapnya, lihat Menerapkan layanan mikro di AWS.

Program Percepatan Migrasi (MAP)

AWS Program yang menyediakan dukungan konsultasi, pelatihan, dan layanan untuk membantu organisasi membangun fondasi operasional yang kuat untuk pindah ke cloud, dan untuk membantu mengimbangi biaya awal migrasi. MAP mencakup metodologi migrasi untuk mengeksekusi migrasi lama dengan cara metodis dan seperangkat alat untuk mengotomatisasi dan mempercepat skenario migrasi umum.

migrasi dalam skala

Proses memindahkan sebagian besar portofolio aplikasi ke cloud dalam gelombang, dengan lebih banyak aplikasi bergerak pada tingkat yang lebih cepat di setiap gelombang. Fase

ini menggunakan praktik terbaik dan pelajaran yang dipetik dari fase sebelumnya untuk mengimplementasikan pabrik migrasi tim, alat, dan proses untuk merampingkan migrasi beban kerja melalui otomatisasi dan pengiriman tangkas. Ini adalah fase ketiga dari strategi AWS migrasi.

pabrik migrasi

Tim lintas fungsi yang merampingkan migrasi beban kerja melalui pendekatan otomatis dan gesit. Tim pabrik migrasi biasanya mencakup operasi, analis dan pemilik bisnis, insinyur migrasi, pengembang, dan DevOps profesional yang bekerja di sprint. Antara 20 dan 50 persen portofolio aplikasi perusahaan terdiri dari pola berulang yang dapat dioptimalkan dengan pendekatan pabrik. Untuk informasi selengkapnya, lihat diskusi tentang pabrik migrasi dan panduan Pabrik Migrasi Cloud di kumpulan konten ini.

metadata migrasi

Informasi tentang aplikasi dan server yang diperlukan untuk menyelesaikan migrasi. Setiap pola migrasi memerlukan satu set metadata migrasi yang berbeda. Contoh metadata migrasi termasuk subnet target, grup keamanan, dan akun. AWS

pola migrasi

Tugas migrasi berulang yang merinci strategi migrasi, tujuan migrasi, dan aplikasi atau layanan migrasi yang digunakan. Contoh: Rehost migrasi ke Amazon EC2 AWS dengan Layanan Migrasi Aplikasi.

Penilaian Portofolio Migrasi (MPA)

Alat online yang menyediakan informasi untuk memvalidasi kasus bisnis untuk bermigrasi ke. AWS Cloud MPA menyediakan penilaian portofolio terperinci (ukuran kanan server, harga, perbandingan TCO, analisis biaya migrasi) serta perencanaan migrasi (analisis data aplikasi dan pengumpulan data, pengelompokan aplikasi, prioritas migrasi, dan perencanaan gelombang). Alat MPA (memerlukan login) tersedia gratis untuk semua AWS konsultan dan konsultan APN Partner.

Penilaian Kesiapan Migrasi (MRA)

Proses mendapatkan wawasan tentang status kesiapan cloud organisasi, mengidentifikasi kekuatan dan kelemahan, dan membangun rencana aksi untuk menutup kesenjangan yang diidentifikasi, menggunakan CAF. AWS Untuk informasi selengkapnya, lihat panduan kesiapan migrasi. MRA adalah tahap pertama dari strategi AWS migrasi.

strategi migrasi

Pendekatan yang digunakan untuk memigrasikan beban kerja ke file. AWS Cloud Untuk informasi lebih lanjut, lihat entri 7 Rs di glosarium ini dan lihat Memobilisasi organisasi Anda untuk mempercepat migrasi skala besar.

ML

Lihat pembelajaran mesin.

modernisasi

Mengubah aplikasi usang (warisan atau monolitik) dan infrastrukturnya menjadi sistem yang gesit, elastis, dan sangat tersedia di cloud untuk mengurangi biaya, mendapatkan efisiensi, dan memanfaatkan inovasi. Untuk informasi selengkapnya, lihat Strategi untuk memodernisasi aplikasi di. AWS Cloud

penilaian kesiapan modernisasi

Evaluasi yang membantu menentukan kesiapan modernisasi aplikasi organisasi; mengidentifikasi manfaat, risiko, dan dependensi; dan menentukan seberapa baik organisasi dapat mendukung keadaan masa depan aplikasi tersebut. Hasil penilaian adalah cetak biru arsitektur target, peta jalan yang merinci fase pengembangan dan tonggak untuk proses modernisasi, dan rencana aksi untuk mengatasi kesenjangan yang diidentifikasi. Untuk informasi lebih lanjut, lihat Mengevaluasi kesiapan modernisasi untuk aplikasi di. AWS Cloud

aplikasi monolitik (monolit)

Aplikasi yang berjalan sebagai layanan tunggal dengan proses yang digabungkan secara ketat. Aplikasi monolitik memiliki beberapa kelemahan. Jika satu fitur aplikasi mengalami lonjakan permintaan, seluruh arsitektur harus diskalakan. Menambahkan atau meningkatkan fitur aplikasi monolitik juga menjadi lebih kompleks ketika basis kode tumbuh. Untuk mengatasi masalah ini, Anda dapat menggunakan arsitektur microservices. Untuk informasi lebih lanjut, lihat Mengurai monolit menjadi layanan mikro.

MPA

Lihat Penilaian Portofolio Migrasi.

MQTT

Lihat Transportasi Telemetri Antrian Pesan.

klasifikasi multiclass

Sebuah proses yang membantu menghasilkan prediksi untuk beberapa kelas (memprediksi satu dari lebih dari dua hasil). Misalnya, model ML mungkin bertanya “Apakah produk ini buku, mobil, atau telepon?” atau “Kategori produk mana yang paling menarik bagi pelanggan ini?”

infrastruktur yang bisa berubah

Model yang memperbarui dan memodifikasi infrastruktur yang ada untuk beban kerja produksi. Untuk meningkatkan konsistensi, keandalan, dan prediktabilitas, AWS Well-Architected Framework merekomendasikan penggunaan infrastruktur yang tidak dapat diubah sebagai praktik terbaik.

O

OAC

Lihat kontrol akses asal.

OAI

Lihat identitas akses asal.

OCM

Lihat manajemen perubahan organisasi.

migrasi offline

Metode migrasi di mana beban kerja sumber diturunkan selama proses migrasi. Metode ini melibatkan waktu henti yang diperpanjang dan biasanya digunakan untuk beban kerja kecil dan tidak kritis.

OI

Lihat integrasi operasi.

OLA

Lihat perjanjian tingkat operasional.

migrasi online

Metode migrasi di mana beban kerja sumber disalin ke sistem target tanpa diambil offline. Aplikasi yang terhubung ke beban kerja dapat terus berfungsi selama migrasi. Metode ini melibatkan waktu henti nol hingga minimal dan biasanya digunakan untuk beban kerja produksi yang kritis.

OPC-UA

Lihat Komunikasi Proses Terbuka - Arsitektur Terpadu.

Komunikasi Proses Terbuka - Arsitektur Terpadu (OPC-UA)

Protokol komunikasi machine-to-machine (M2M) untuk otomasi industri. OPC-UA menyediakan standar interoperabilitas dengan enkripsi data, otentikasi, dan skema otorisasi.

perjanjian tingkat operasional (OLA)

Perjanjian yang menjelaskan apa yang dijanjikan kelompok TI fungsional untuk diberikan satu sama lain, untuk mendukung perjanjian tingkat layanan (SLA).

Tinjauan Kesiapan Operasional (ORR)

Daftar pertanyaan dan praktik terbaik terkait yang membantu Anda memahami, mengevaluasi, mencegah, atau mengurangi ruang lingkup insiden dan kemungkinan kegagalan. Untuk informasi lebih lanjut, lihat Ulasan Kesiapan Operasional (ORR) dalam Kerangka Kerja Well-Architected AWS .

teknologi operasional (OT)

Sistem perangkat keras dan perangkat lunak yang bekerja dengan lingkungan fisik untuk mengendalikan operasi industri, peralatan, dan infrastruktur. Di bidang manufaktur, integrasi sistem OT dan teknologi informasi (TI) adalah fokus utama untuk transformasi Industri 4.0.

integrasi operasi (OI)

Proses modernisasi operasi di cloud, yang melibatkan perencanaan kesiapan, otomatisasi, dan integrasi. Untuk informasi selengkapnya, lihat panduan integrasi operasi.

jejak organisasi

Jejak yang dibuat oleh AWS CloudTrail itu mencatat semua peristiwa untuk semua Akun AWS dalam organisasi di AWS Organizations. Jejak ini dibuat di setiap Akun AWS bagian organisasi dan melacak aktivitas di setiap akun. Untuk informasi selengkapnya, lihat Membuat jejak untuk organisasi dalam CloudTrail dokumentasi.

manajemen perubahan organisasi (OCM)

Kerangka kerja untuk mengelola transformasi bisnis utama yang mengganggu dari perspektif orang, budaya, dan kepemimpinan. OCM membantu organisasi mempersiapkan, dan transisi ke, sistem dan strategi baru dengan mempercepat adopsi perubahan, mengatasi masalah transisi,

dan mendorong perubahan budaya dan organisasi. Dalam strategi AWS migrasi, kerangka kerja ini disebut percepatan orang, karena kecepatan perubahan yang diperlukan dalam proyek adopsi cloud. Untuk informasi lebih lanjut, lihat panduan OCM.

kontrol akses asal (OAC)

Di CloudFront, opsi yang disempurnakan untuk membatasi akses untuk mengamankan konten Amazon Simple Storage Service (Amazon S3) Anda. OAC mendukung semua bucket S3 di semua Wilayah AWS, enkripsi sisi server dengan AWS KMS (SSE-KMS), dan dinamis dan permintaan ke bucket S3. `PUT` `DELETE`

identitas akses asal (OAI)

Di CloudFront, opsi untuk membatasi akses untuk mengamankan konten Amazon S3 Anda. Saat Anda menggunakan OAI, CloudFront buat prinsipal yang dapat diautentikasi oleh Amazon S3. Prinsipal yang diautentikasi dapat mengakses konten dalam bucket S3 hanya melalui distribusi tertentu. CloudFront Lihat juga OAC, yang menyediakan kontrol akses yang lebih terperinci dan ditingkatkan.

ORR

Lihat tinjauan kesiapan operasional.

OT

Lihat teknologi operasional.

keluar (jalan keluar) VPC

Dalam arsitektur AWS multi-akun, VPC yang menangani koneksi jaringan yang dimulai dari dalam aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

P

batas izin

Kebijakan manajemen IAM yang dilampirkan pada prinsipal IAM untuk menetapkan izin maksimum yang dapat dimiliki pengguna atau peran. Untuk informasi selengkapnya, lihat Batas izin dalam dokumentasi IAM.

Informasi Identifikasi Pribadi (PII)

Informasi yang, jika dilihat secara langsung atau dipasangkan dengan data terkait lainnya, dapat digunakan untuk menyimpulkan identitas individu secara wajar. Contoh PII termasuk nama, alamat, dan informasi kontak.

PII

Lihat informasi yang dapat diidentifikasi secara pribadi.

buku pedoman

Serangkaian langkah yang telah ditentukan sebelumnya yang menangkap pekerjaan yang terkait dengan migrasi, seperti mengirimkan fungsi operasi inti di cloud. Buku pedoman dapat berupa skrip, runbook otomatis, atau ringkasan proses atau langkah-langkah yang diperlukan untuk mengoperasikan lingkungan modern Anda.

PLC

Lihat pengontrol logika yang dapat diprogram.

PLM

Lihat manajemen siklus hidup produk.

kebijakan

Objek yang dapat menentukan izin (lihat kebijakan berbasis identitas), menentukan kondisi akses (lihat kebijakanberbasis sumber daya), atau menentukan izin maksimum untuk semua akun dalam organisasi di (lihat kebijakan kontrol layanan). AWS Organizations

persistensi poliglot

Secara independen memilih teknologi penyimpanan data microservice berdasarkan pola akses data dan persyaratan lainnya. Jika layanan mikro Anda memiliki teknologi penyimpanan data yang sama, mereka dapat menghadapi tantangan implementasi atau mengalami kinerja yang buruk. Layanan mikro lebih mudah diimplementasikan dan mencapai kinerja dan skalabilitas yang lebih baik jika mereka menggunakan penyimpanan data yang paling sesuai dengan kebutuhan mereka. Untuk informasi selengkapnya, lihat Mengaktifkan persistensi data di layanan mikro.

penilaian portofolio

Proses menemukan, menganalisis, dan memprioritaskan portofolio aplikasi untuk merencanakan migrasi. Untuk informasi selengkapnya, lihat Mengevaluasi kesiapan migrasi.

predikat

Kondisi kueri yang mengembalikan `true` atau`false`, biasanya terletak di `WHERE` klausa.

predikat pushdown

Teknik pengoptimalan kueri database yang menyaring data dalam kueri sebelum transfer. Ini mengurangi jumlah data yang harus diambil dan diproses dari database relasional, dan meningkatkan kinerja kueri.

kontrol preventif

Kontrol keamanan yang dirancang untuk mencegah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan pertama untuk membantu mencegah akses tidak sah atau perubahan yang tidak diinginkan ke jaringan Anda. Untuk informasi selengkapnya, lihat Kontrol pencegahan dalam Menerapkan kontrol keamanan pada. AWS

principal

Entitas AWS yang dapat melakukan tindakan dan mengakses sumber daya. Entitas ini biasanya merupakan pengguna root untuk Akun AWS, peran IAM, atau pengguna. Untuk informasi selengkapnya, lihat Prinsip dalam istilah dan konsep Peran dalam dokumentasi IAM.

Privasi oleh Desain

Pendekatan dalam rekayasa sistem yang memperhitungkan privasi di seluruh proses rekayasa.

zona host pribadi

Container yang menyimpan informasi tentang bagaimana Anda ingin Amazon Route 53 merespons kueri DNS untuk domain dan subdomainnya dalam satu atau beberapa VPC. Untuk informasi selengkapnya, lihat Bekerja dengan zona yang dihosting pribadi di dokumentasi Route 53.

kontrol proaktif

Kontrol keamanan yang dirancang untuk mencegah penyebaran sumber daya yang tidak sesuai. Kontrol ini memindai sumber daya sebelum disediakan. Jika sumber daya tidak sesuai dengan kontrol, maka itu tidak disediakan. Untuk informasi selengkapnya, lihat panduan referensi Kontrol dalam AWS Control Tower dokumentasi dan lihat Kontrol proaktif dalam Menerapkan kontrol keamanan pada AWS.

manajemen siklus hidup produk (PLM)

Manajemen data dan proses untuk suatu produk di seluruh siklus hidupnya, mulai dari desain, pengembangan, dan peluncuran, melalui pertumbuhan dan kematangan, hingga penurunan dan penghapusan.

lingkungan produksi

Lihat lingkungan.

pengontrol logika yang dapat diprogram (PLC)

Di bidang manufaktur, komputer yang sangat andal dan mudah beradaptasi yang memantau mesin dan mengotomatiskan proses manufaktur.

pseudonimisasi

Proses penggantian pengidentifikasi pribadi dalam kumpulan data dengan nilai placeholder. Pseudonimisasi dapat membantu melindungi privasi pribadi. Data pseudonim masih dianggap sebagai data pribadi.

terbitkan/berlangganan (pub/sub)

Pola yang memungkinkan komunikasi asinkron antara layanan mikro untuk meningkatkan skalabilitas dan daya tanggap. Misalnya, dalam MES berbasis layanan mikro, layanan mikro dapat mempublikasikan pesan peristiwa ke saluran yang dapat berlangganan layanan mikro lainnya. Sistem dapat menambahkan layanan mikro baru tanpa mengubah layanan penerbitan.

Q

rencana kueri

Serangkaian langkah, seperti instruksi, yang digunakan untuk mengakses data dalam sistem database relasional SQL.

regresi rencana kueri

Ketika pengoptimal layanan database memilih rencana yang kurang optimal daripada sebelum perubahan yang diberikan ke lingkungan database. Hal ini dapat disebabkan oleh perubahan statistik, kendala, pengaturan lingkungan, pengikatan parameter kueri, dan pembaruan ke mesin database.

R

Matriks RACI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

ransomware

Perangkat lunak berbahaya yang dirancang untuk memblokir akses ke sistem komputer atau data sampai pembayaran dilakukan.

Matriks RASCI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

RCAC

Lihat kontrol akses baris dan kolom.

replika baca

Salinan database yang digunakan untuk tujuan read-only. Anda dapat merutekan kueri ke replika baca untuk mengurangi beban pada database utama Anda.

arsitek ulang

Lihat 7 Rs.

tujuan titik pemulihan (RPO)

Jumlah waktu maksimum yang dapat diterima sejak titik pemulihan data terakhir. Ini menentukan apa yang dianggap sebagai hilangnya data yang dapat diterima antara titik pemulihan terakhir dan gangguan layanan.

tujuan waktu pemulihan (RTO)

Penundaan maksimum yang dapat diterima antara gangguan layanan dan pemulihan layanan.

refactor

Lihat 7 Rs.

Wilayah

Kumpulan AWS sumber daya di wilayah geografis. Masing-masing Wilayah AWS terisolasi dan independen dari yang lain untuk memberikan toleransi kesalahan, stabilitas, dan ketahanan. Untuk informasi selengkapnya, lihat Menentukan Wilayah AWS akun yang dapat digunakan.

regresi

Teknik ML yang memprediksi nilai numerik. Misalnya, untuk memecahkan masalah “Berapa harga rumah ini akan dijual?” Model ML dapat menggunakan model regresi linier untuk memprediksi harga jual rumah berdasarkan fakta yang diketahui tentang rumah (misalnya, luas persegi).

rehost

Lihat 7 Rs.

melepaskan

Dalam proses penyebaran, tindakan mempromosikan perubahan pada lingkungan produksi.

memindahkan

Lihat 7 Rs.

memplatform ulang

Lihat 7 Rs.

pembelian kembali

Lihat 7 Rs.

ketahanan

Kemampuan aplikasi untuk melawan atau pulih dari gangguan. Ketersediaan tinggi dan pemulihan bencana adalah pertimbangan umum ketika merencanakan ketahanan di. AWS Cloud Untuk informasi lebih lanjut, lihat AWS Cloud Ketahanan.

kebijakan berbasis sumber daya

Kebijakan yang dilampirkan ke sumber daya, seperti bucket Amazon S3, titik akhir, atau kunci enkripsi. Jenis kebijakan ini menentukan prinsip mana yang diizinkan mengakses, tindakan yang didukung, dan kondisi lain yang harus dipenuhi.

matriks yang bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI)

Matriks yang mendefinisikan peran dan tanggung jawab untuk semua pihak yang terlibat dalam kegiatan migrasi dan operasi cloud. Nama matriks berasal dari jenis tanggung jawab yang didefinisikan dalam matriks: bertanggung jawab (R), akuntabel (A), dikonsultasikan (C), dan diinformasikan (I). Jenis dukungan (S) adalah opsional. Jika Anda menyertakan dukungan, matriks disebut matriks RASCI, dan jika Anda mengecualikannya, itu disebut matriks RACI.

kontrol responsif

Kontrol keamanan yang dirancang untuk mendorong remediasi efek samping atau penyimpangan dari garis dasar keamanan Anda. Untuk informasi selengkapnya, lihat Kontrol responsif dalam Menerapkan kontrol keamanan pada AWS.

melestarikan

Lihat 7 Rs.

pensiun

Lihat 7 Rs.

rotasi

Proses memperbarui rahasia secara berkala untuk membuatnya lebih sulit bagi penyerang untuk mengakses kredensil.

kontrol akses baris dan kolom (RCAC)

Penggunaan ekspresi SQL dasar dan fleksibel yang telah menetapkan aturan akses. RCAC terdiri dari izin baris dan topeng kolom.

RPO

Lihat tujuan titik pemulihan.

RTO

Lihat tujuan waktu pemulihan.

buku runbook

Satu set prosedur manual atau otomatis yang diperlukan untuk melakukan tugas tertentu. Ini biasanya dibangun untuk merampingkan operasi berulang atau prosedur dengan tingkat kesalahan yang tinggi.

D

SAML 2.0

Standar terbuka yang digunakan oleh banyak penyedia identitas (IdPs). Fitur ini memungkinkan sistem masuk tunggal gabungan (SSO), sehingga pengguna dapat masuk ke AWS Management Console atau memanggil operasi AWS API tanpa Anda harus membuat pengguna di IAM untuk

semua orang di organisasi Anda. Untuk informasi lebih lanjut tentang federasi berbasis SAMP 2.0, lihat Tentang federasi berbasis SAMP 2.0 dalam dokumentasi IAM.

PENIPUAN

Lihat kontrol pengawasan dan akuisisi data.

SCP

Lihat kebijakan kontrol layanan.

Rahasia

Dalam AWS Secrets Manager, informasi rahasia atau terbatas, seperti kata sandi atau kredensil pengguna, yang Anda simpan dalam bentuk terenkripsi. Ini terdiri dari nilai rahasia dan metadata-nya. Nilai rahasia dapat berupa biner, string tunggal, atau beberapa string. Untuk informasi selengkapnya, lihat Apa yang ada di rahasia Secrets Manager? dalam dokumentasi Secrets Manager.

kontrol keamanan

Pagar pembatas teknis atau administratif yang mencegah, mendeteksi, atau mengurangi kemampuan pelaku ancaman untuk mengeksploitasi kerentanan keamanan. Ada empat jenis kontrol keamanan utama: preventif, detektif, responsif, dan proaktif.

pengerasan keamanan

Proses mengurangi permukaan serangan untuk membuatnya lebih tahan terhadap serangan. Ini dapat mencakup tindakan seperti menghapus sumber daya yang tidak lagi diperlukan, menerapkan praktik keamanan terbaik untuk memberikan hak istimewa paling sedikit, atau menonaktifkan fitur yang tidak perlu dalam file konfigurasi.

sistem informasi keamanan dan manajemen acara (SIEM)

Alat dan layanan yang menggabungkan sistem manajemen informasi keamanan (SIM) dan manajemen acara keamanan (SEM). Sistem SIEM mengumpulkan, memantau, dan menganalisis data dari server, jaringan, perangkat, dan sumber lain untuk mendeteksi ancaman dan pelanggaran keamanan, dan untuk menghasilkan peringatan.

otomatisasi respons keamanan

Tindakan yang telah ditentukan dan diprogram yang dirancang untuk secara otomatis merespons atau memulihkan peristiwa keamanan. Otomatisasi ini berfungsi sebagai kontrol keamanan detektif atau responsif yang membantu Anda menerapkan praktik terbaik AWS keamanan. Contoh

tindakan respons otomatis termasuk memodifikasi grup keamanan VPC, menambal instans Amazon EC2, atau memutar kredensil.

enkripsi sisi server

Enkripsi data di tujuannya, oleh Layanan AWS yang menerimanya.

kebijakan kontrol layanan (SCP)

Kebijakan yang menyediakan kontrol terpusat atas izin untuk semua akun di organisasi. AWS Organizations SCP menentukan pagar pembatas atau menetapkan batasan pada tindakan yang dapat didelegasikan oleh administrator kepada pengguna atau peran. Anda dapat menggunakan SCP sebagai daftar izin atau daftar penolakan, untuk menentukan layanan atau tindakan mana yang diizinkan atau dilarang. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam AWS Organizations dokumentasi.

titik akhir layanan

URL titik masuk untuk file Layanan AWS. Anda dapat menggunakan endpoint untuk terhubung secara terprogram ke layanan target. Untuk informasi selengkapnya, lihat Layanan AWS titik akhir di Referensi Umum AWS.

perjanjian tingkat layanan (SLA)

Perjanjian yang menjelaskan apa yang dijanjikan tim TI untuk diberikan kepada pelanggan mereka, seperti waktu kerja dan kinerja layanan.

indikator tingkat layanan (SLI)

Pengukuran aspek kinerja layanan, seperti tingkat kesalahan, ketersediaan, atau throughputnya.

tujuan tingkat layanan (SLO)

Metrik target yang mewakili kesehatan layanan, yang diukur dengan indikator tingkat layanan.

model tanggung jawab bersama

Model yang menjelaskan tanggung jawab yang Anda bagikan AWS untuk keamanan dan kepatuhan cloud. AWS bertanggung jawab atas keamanan cloud, sedangkan Anda bertanggung jawab atas keamanan di cloud. Untuk informasi selengkapnya, lihat Model tanggung jawab bersama.

SIEM

Lihat informasi keamanan dan sistem manajemen acara.

titik kegagalan tunggal (SPOF)

Kegagalan dalam satu komponen penting dari aplikasi yang dapat mengganggu sistem.

SLA

Lihat perjanjian tingkat layanan.

SLI

Lihat indikator tingkat layanan.

SLO

Lihat tujuan tingkat layanan.

split-and-seed model

Pola untuk menskalakan dan mempercepat proyek modernisasi. Ketika fitur baru dan rilis produk didefinisikan, tim inti berpisah untuk membuat tim produk baru. Ini membantu meningkatkan kemampuan dan layanan organisasi Anda, meningkatkan produktivitas pengembang, dan mendukung inovasi yang cepat. Untuk informasi lebih lanjut, lihat Pendekatan bertahap untuk memodernisasi aplikasi di. AWS Cloud

SPOF

Lihat satu titik kegagalan.

skema bintang

Struktur organisasi database yang menggunakan satu tabel fakta besar untuk menyimpan data transaksional atau terukur dan menggunakan satu atau lebih tabel dimensi yang lebih kecil untuk menyimpan atribut data. Struktur ini dirancang untuk digunakan dalam gudang data atau untuk tujuan intelijen bisnis.

pola ara pencekik

Pendekatan untuk memodernisasi sistem monolitik dengan menulis ulang secara bertahap dan mengganti fungsionalitas sistem sampai sistem warisan dapat dinonaktifkan. Pola ini menggunakan analogi pohon ara yang tumbuh menjadi pohon yang sudah mapan dan akhirnya mengatasi dan menggantikan inangnya. Pola ini diperkenalkan oleh Martin Fowler sebagai cara untuk mengelola risiko saat menulis ulang sistem monolitik. Untuk contoh cara menerapkan pola ini, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

subnet

Rentang alamat IP dalam VPC Anda. Subnet harus berada di Availability Zone tunggal.

kontrol pengawasan dan akuisisi data (SCADA)

Di bidang manufaktur, sistem yang menggunakan perangkat keras dan perangkat lunak untuk memantau aset fisik dan operasi produksi.

enkripsi simetris

Algoritma enkripsi yang menggunakan kunci yang sama untuk mengenkripsi dan mendekripsi data.

pengujian sintetis

Menguji sistem dengan cara yang mensimulasikan interaksi pengguna untuk mendeteksi potensi masalah atau untuk memantau kinerja. Anda dapat menggunakan Amazon CloudWatch Synthetics untuk membuat tes ini.

T

tag

Pasangan nilai kunci yang bertindak sebagai metadata untuk mengatur sumber daya Anda. AWS Tanda dapat membantu Anda mengelola, mengidentifikasi, mengatur, dan memfilter sumber daya. Untuk informasi selengkapnya, lihat Menandai AWS sumber daya Anda.

variabel target

Nilai yang Anda coba prediksi dalam ML yang diawasi. Ini juga disebut sebagai variabel hasil. Misalnya, dalam pengaturan manufaktur, variabel target bisa menjadi cacat produk.

daftar tugas

Alat yang digunakan untuk melacak kemajuan melalui runbook. Daftar tugas berisi ikhtisar runbook dan daftar tugas umum yang harus diselesaikan. Untuk setiap tugas umum, itu termasuk perkiraan jumlah waktu yang dibutuhkan, pemilik, dan kemajuan.

lingkungan uji

Lihat lingkungan.

pelatihan

Untuk menyediakan data bagi model ML Anda untuk dipelajari. Data pelatihan harus berisi jawaban yang benar. Algoritma pembelajaran menemukan pola dalam data pelatihan yang memetakan atribut data input ke target (jawaban yang ingin Anda prediksi). Ini menghasilkan model ML yang menangkap pola-pola ini. Anda kemudian dapat menggunakan model ML untuk membuat prediksi pada data baru yang Anda tidak tahu targetnya.

gerbang transit

Hub transit jaringan yang dapat Anda gunakan untuk menghubungkan VPC dan jaringan lokal Anda. Untuk informasi selengkapnya, lihat Apa itu gateway transit dalam AWS Transit Gateway dokumentasi.

alur kerja berbasis batang

Pendekatan di mana pengembang membangun dan menguji fitur secara lokal di cabang fitur dan kemudian menggabungkan perubahan tersebut ke cabang utama. Cabang utama kemudian dibangun untuk pengembangan, praproduksi, dan lingkungan produksi, secara berurutan.

akses tepercaya

Memberikan izin ke layanan yang Anda tentukan untuk melakukan tugas di organisasi Anda di dalam AWS Organizations dan di akunnya atas nama Anda. Layanan tepercaya menciptakan peran terkait layanan di setiap akun, ketika peran itu diperlukan, untuk melakukan tugas manajemen untuk Anda. Untuk informasi selengkapnya, lihat Menggunakan AWS Organizations dengan AWS layanan lain dalam AWS Organizations dokumentasi.

penyetelan

Untuk mengubah aspek proses pelatihan Anda untuk meningkatkan akurasi model ML. Misalnya, Anda dapat melatih model ML dengan membuat set pelabelan, menambahkan label, dan kemudian mengulangi langkah-langkah ini beberapa kali di bawah pengaturan yang berbeda untuk mengoptimalkan model.

tim dua pizza

Sebuah DevOps tim kecil yang bisa Anda beri makan dengan dua pizza. Ukuran tim dua pizza memastikan peluang terbaik untuk berkolaborasi dalam pengembangan perangkat lunak.

U

waswas

Sebuah konsep yang mengacu pada informasi yang tidak tepat, tidak lengkap, atau tidak diketahui yang dapat merusak keandalan model ML prediktif. Ada dua jenis ketidakpastian: ketidakpastian epistemik disebabkan oleh data yang terbatas dan tidak lengkap, sedangkan ketidakpastian aleatorik disebabkan oleh kebisingan dan keacakan yang melekat dalam data. Untuk informasi lebih lanjut, lihat panduan Mengukur ketidakpastian dalam sistem pembelajaran mendalam.

tugas yang tidak terdiferensiasi

Juga dikenal sebagai angkat berat, pekerjaan yang diperlukan untuk membuat dan mengoperasikan aplikasi tetapi itu tidak memberikan nilai langsung kepada pengguna akhir atau memberikan keunggulan kompetitif. Contoh tugas yang tidak terdiferensiasi termasuk pengadaan, pemeliharaan, dan perencanaan kapasitas.

lingkungan atas

Lihat lingkungan.

V

menyedot debu

Operasi pemeliharaan database yang melibatkan pembersihan setelah pembaruan tambahan untuk merebut kembali penyimpanan dan meningkatkan kinerja.

kendali versi

Proses dan alat yang melacak perubahan, seperti perubahan kode sumber dalam repositori.

Peering VPC

Koneksi antara dua VPC yang memungkinkan Anda merutekan lalu lintas dengan menggunakan alamat IP pribadi. Untuk informasi selengkapnya, lihat Apa itu peering VPC di dokumentasi VPC Amazon.

kerentanan

Kelemahan perangkat lunak atau perangkat keras yang membahayakan keamanan sistem.

W

cache hangat

Cache buffer yang berisi data terkini dan relevan yang sering diakses. Instance database dapat membaca dari cache buffer, yang lebih cepat daripada membaca dari memori utama atau disk.

data hangat

Data yang jarang diakses. Saat menanyakan jenis data ini, kueri yang cukup lambat biasanya dapat diterima.

fungsi jendela

Fungsi SQL yang melakukan perhitungan pada sekelompok baris yang berhubungan dengan catatan saat ini. Fungsi jendela berguna untuk memproses tugas, seperti menghitung rata-rata bergerak atau mengakses nilai baris berdasarkan posisi relatif dari baris saat ini.

beban kerja

Kumpulan sumber daya dan kode yang memberikan nilai bisnis, seperti aplikasi yang dihadapi pelanggan atau proses backend.

aliran kerja

Grup fungsional dalam proyek migrasi yang bertanggung jawab atas serangkaian tugas tertentu. Setiap alur kerja independen tetapi mendukung alur kerja lain dalam proyek. Misalnya, alur kerja portofolio bertanggung jawab untuk memprioritaskan aplikasi, perencanaan gelombang, dan mengumpulkan metadata migrasi. Alur kerja portofolio mengirimkan aset ini ke alur kerja migrasi, yang kemudian memigrasikan server dan aplikasi.

CACING

Lihat menulis sekali, baca banyak.

WQF

Lihat Kerangka Kualifikasi Beban Kerja AWS.

tulis sekali, baca banyak (WORM)

Model penyimpanan yang menulis data satu kali dan mencegah data dihapus atau dimodifikasi. Pengguna yang berwenang dapat membaca data sebanyak yang diperlukan, tetapi mereka tidak dapat mengubahnya. Infrastruktur penyimpanan data ini dianggap tidak dapat diubah.

Z

eksploitasi zero-day

Serangan, biasanya malware, yang memanfaatkan kerentanan zero-day.

kerentanan zero-day

Cacat atau kerentanan yang tak tanggung-tanggung dalam sistem produksi. Aktor ancaman dapat menggunakan jenis kerentanan ini untuk menyerang sistem. Pengembang sering menyadari kerentanan sebagai akibat dari serangan tersebut.

aplikasi zombie

Aplikasi yang memiliki CPU rata-rata dan penggunaan memori di bawah 5 persen. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini.

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