AWS Whitepaper

Menerapkan Layanan Mikro pada AWS

Menerapkan Layanan Mikro pada AWS: AWS Whitepaper

Table of Contents

Menerapkan Layanan Mikro pada AWS

Tanggal publikasi: 31 Juli 2023 () Riwayat dokumen

Microservices menawarkan pendekatan yang efisien untuk pengembangan perangkat lunak yang mempercepat penyebaran, mendorong inovasi, meningkatkan pemeliharaan, dan meningkatkan skalabilitas. Metode ini bergantung pada layanan kecil yang digabungkan secara longgar yang berkomunikasi melalui definisi yang baik APIs, yang dikelola oleh tim otonom. Mengadopsi layanan mikro menawarkan manfaat, seperti peningkatan skalabilitas, ketahanan, fleksibilitas, dan siklus pengembangan yang lebih cepat.

Whitepaper ini mengeksplorasi tiga pola layanan mikro populer: API driven, event driven, dan data streaming. Kami memberikan gambaran umum tentang setiap pendekatan, menguraikan fitur utama layanan mikro, mengatasi tantangan dalam pengembangannya, dan mengilustrasikan bagaimana Amazon Web Services (AWS) dapat membantu tim aplikasi mengatasi hambatan ini.

Mempertimbangkan sifat kompleks topik seperti penyimpanan data, komunikasi asinkron, dan penemuan layanan, Anda dianjurkan untuk mempertimbangkan kebutuhan spesifik aplikasi Anda dan kasus penggunaan di samping panduan yang diberikan saat membuat keputusan arsitektur.

Pengantar

Arsitektur microservices menggabungkan konsep yang sukses dan terbukti dari berbagai bidang, seperti:

- Pengembangan perangkat lunak tangkas
- Arsitektur berorientasi layanan
- Desain API-first
- Terus menerusIntegration/Continuous Delivery (CI/CD)

Seringkali, layanan mikro menggabungkan pola desain dari Aplikasi Dua Belas Faktor.

Meskipun layanan mikro menawarkan banyak manfaat, penting untuk menilai persyaratan unik kasus penggunaan Anda dan biaya terkait. Arsitektur monolitik atau pendekatan alternatif mungkin lebih tepat dalam beberapa kasus. Memutuskan antara layanan mikro atau monolit harus dibuat atas case-by-case dasar, dengan mempertimbangkan faktor-faktor seperti skala, kompleksitas, dan kasus penggunaan tertentu.

Kami pertama-tama mengeksplorasi arsitektur layanan mikro yang sangat skalabel dan toleran terhadap kesalahan (antarmuka pengguna, implementasi layanan mikro, penyimpanan data) dan mendemonstrasikan cara membangunnya menggunakan teknologi kontainer. AWS Kami kemudian menyarankan AWS layanan untuk mengimplementasikan arsitektur layanan mikro tanpa server yang khas, mengurangi kompleksitas operasional.

Tanpa server dicirikan oleh prinsip-prinsip berikut:

- Tidak ada infrastruktur untuk menyediakan atau mengelola
- Penskalaan secara otomatis berdasarkan unit konsumsi
- Model penagihan “Bayar untuk nilai”
- Ketersediaan bawaan dan toleransi kesalahan
- Arsitektur Berbasis Acara (EDA)

Terakhir, kami memeriksa keseluruhan sistem dan membahas aspek lintas layanan dari arsitektur layanan mikro, seperti pemantauan terdistribusi, pencatatan, penelusuran, audit, konsistensi data, dan komunikasi asinkron.

Dokumen ini berfokus pada beban kerja yang berjalan di AWS Cloud, tidak termasuk skenario hibrida dan strategi migrasi. Untuk informasi tentang strategi migrasi, lihat whitepaper Metodologi Migrasi Kontainer.

Apakah Anda sudah Well-Architected?

Kerangka Kerja AWS Well-Architected membantu Anda memahami pro dan kontra dari keputusan yang Anda buat saat membangun sistem di cloud. Enam pilar dari Kerangka Kerja ini memungkinkan Anda mempelajari praktik terbaik arsitektural untuk merancang dan mengoperasikan sistem yang andal, aman, efisien, hemat biaya, dan berkelanjutan. Dengan menggunakan AWS Well-Architected Tool, tersedia tanpa biaya di Konsol Manajemen AWS, Anda dapat meninjau beban kerja Anda terhadap praktik terbaik ini dengan menjawab serangkaian pertanyaan untuk setiap pilar.

Di Lensa Aplikasi Tanpa Server, kami fokus pada praktik terbaik untuk merancang aplikasi tanpa server Anda. AWS

Untuk panduan lebih lanjut dari para ahli dan praktik terbaik untuk arsitektur cloud Anda—referensi penerapan arsitektur, diagram, dan laporan resmi—lihat Pusat Arsitektur AWS.

Modernisasi ke layanan mikro

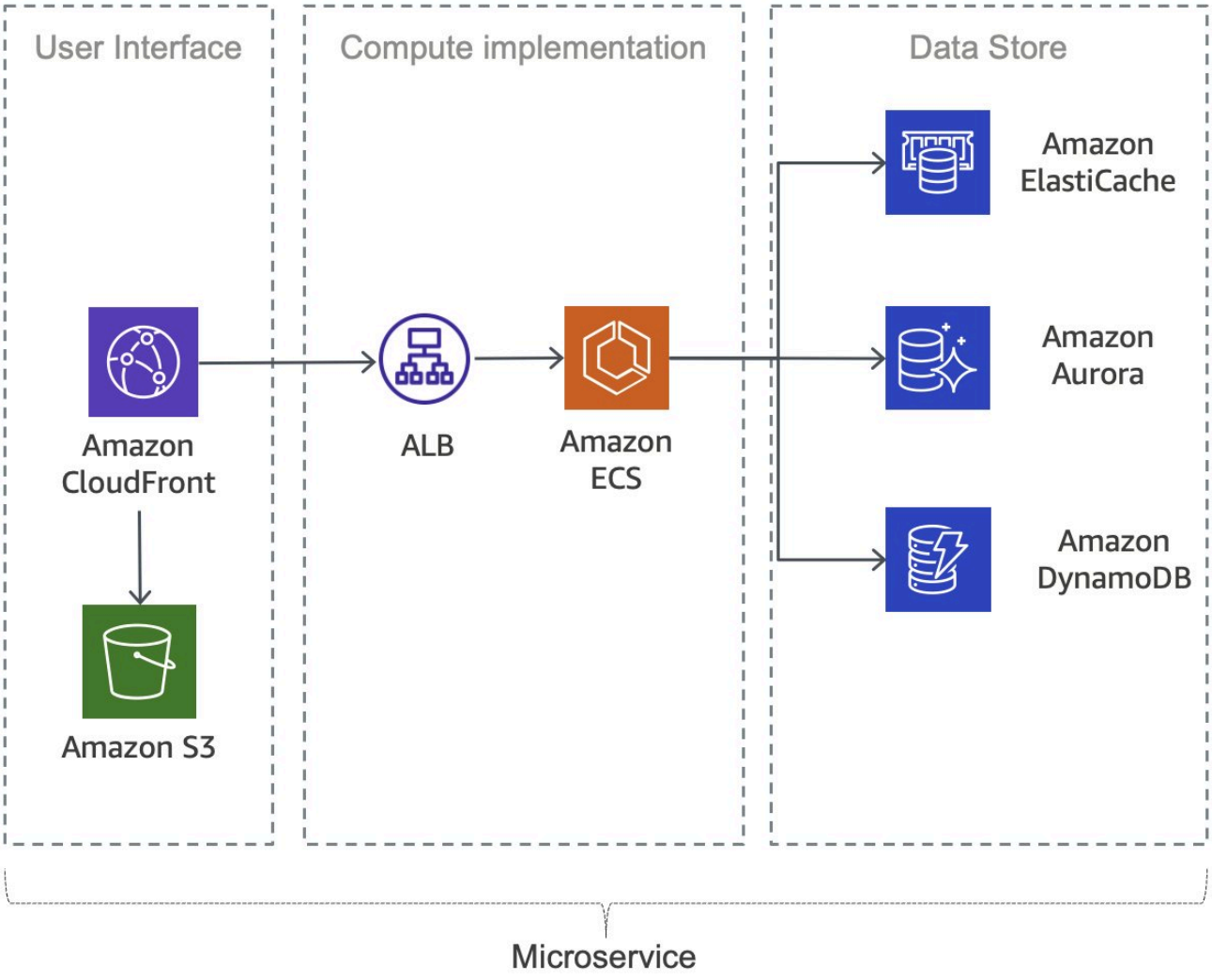
Layanan mikro pada dasarnya adalah unit kecil dan independen yang membentuk aplikasi. Transisi dari struktur monolitik tradisional ke layanan mikro dapat mengikuti berbagai strategi.

Transisi ini juga memengaruhi cara organisasi Anda beroperasi:

- Ini mendorong pengembangan tangkas, di mana tim bekerja dalam siklus cepat.
- Tim biasanya kecil, kadang-kadang digambarkan sebagai dua tim pizza — cukup kecil sehingga dua pizza bisa memberi makan seluruh tim.
- Tim bertanggung jawab penuh atas layanan mereka, mulai dari pembuatan hingga penyebaran dan pemeliharaan.

Arsitektur microservices sederhana di AWS

Aplikasi monolitik khas terdiri dari lapisan yang berbeda: lapisan presentasi, lapisan aplikasi, dan lapisan data. Arsitektur layanan mikro, di sisi lain, memisahkan fungsionalitas menjadi vertikal kohesif menurut domain tertentu, bukan lapisan teknologi. Gambar 1 mengilustrasikan arsitektur referensi untuk aplikasi layanan mikro yang khas pada. AWS



Gambar 1: Aplikasi layanan mikro yang khas pada AWS

Antarmuka pengguna

Aplikasi web modern sering menggunakan JavaScript kerangka kerja untuk mengembangkan aplikasi satu halaman yang berkomunikasi dengan backend. APIs Ini APIs biasanya dibangun menggunakan

Representational State Transfer (REST) atau RESTful APIs, atau APIs GraphQL. Konten web statis dapat disajikan menggunakan Amazon Simple Storage Service (Amazon S3) dan Amazon. CloudFront

Layanan mikro

APIs dianggap sebagai pintu depan layanan mikro, karena merupakan titik masuk untuk logika aplikasi. Biasanya, API layanan RESTful web atau APIs GraphQL digunakan. Ini APIs mengelola dan memproses panggilan klien, menangani fungsi seperti manajemen lalu lintas, pemfilteran permintaan, perutean, caching, otentikasi, dan otorisasi.

Implementasi Microservices

AWS menawarkan blok bangunan untuk mengembangkan layanan mikro, termasuk Amazon ECS dan Amazon EKS sebagai pilihan untuk mesin orkestrasi kontainer dan AWS Fargate dan EC2 sebagai opsi hosting. AWS Lambda adalah cara lain tanpa server untuk membangun layanan mikro. AWS Pilihan antara opsi hosting ini tergantung pada kebutuhan pelanggan untuk mengelola infrastruktur yang mendasarinya.

AWS Lambda memungkinkan Anda untuk mengunggah kode Anda, secara otomatis menskalakan dan mengelola pelaksanaannya dengan ketersediaan tinggi. Ini menghilangkan kebutuhan akan manajemen infrastruktur, sehingga Anda dapat bergerak cepat dan fokus pada logika bisnis Anda. Lambda mendukung beberapa bahasa pemrograman dan dapat dipicu oleh AWS layanan lain atau dipanggil langsung dari web atau aplikasi seluler.

Aplikasi berbasis kontainer telah mendapatkan popularitas karena portabilitas, produktivitas, dan efisiensi.AWS menawarkan beberapa layanan untuk membangun, menyebarkan, dan mengelola kontainer.

- App2Container, alat baris perintah untuk memigrasi dan memodernisasi aplikasi web Java dan .NET ke dalam format kontainer. AWS A2C menganalisis dan membuat inventaris aplikasi yang berjalan di bare metal, mesin virtual, instans Amazon Elastic Compute Cloud (EC2), atau di cloud.

- Amazon Elastic Container Service (Amazon ECS) dan Amazon Elastic Kubernetes Service (Amazon EKS) mengelola infrastruktur container Anda, sehingga memudahkan peluncuran dan pemeliharaan aplikasi kontainer.

- Amazon EKS adalah layanan Kubernetes terkelola untuk menjalankan Kubernetes di AWS cloud dan pusat data lokal (Amazon EKS Anywhere). Ini memperluas layanan cloud ke lingkungan lokal untuk latensi rendah, pemrosesan data lokal, biaya transfer data tinggi, atau persyaratan residensi data (lihat whitepaper di "Menjalankan Beban Kerja Kontainer Hibrid Dengan Amazon EKS Anywhere “). Anda dapat menggunakan semua plug-in dan tooling yang ada dari komunitas Kubernetes dengan EKS.
- Amazon Elastic Container Service (Amazon ECS) adalah layanan orkestrasi kontainer yang dikelola sepenuhnya yang menyederhanakan penerapan, pengelolaan, dan penskalaan aplikasi kontainer Anda. Pelanggan memilih ECS untuk kesederhanaan dan integrasi mendalam dengan AWS layanan.

Untuk bacaan lebih lanjut, lihat blog Amazon ECS vs Amazon EKS: memahami layanan AWS kontainer.

- AWS App Runneradalah layanan aplikasi kontainer yang dikelola sepenuhnya yang memungkinkan Anda membangun, menyebarkan, dan menjalankan aplikasi web dan layanan API kontainer tanpa infrastruktur atau pengalaman kontainer sebelumnya.
- AWS Fargate, mesin komputasi tanpa server, bekerja dengan Amazon ECS dan Amazon EKS untuk secara otomatis mengelola sumber daya komputasi untuk aplikasi kontainer.
- Amazon ECR adalah registri kontainer yang dikelola sepenuhnya yang menawarkan hosting berkinerja tinggi, sehingga Anda dapat menyebarkan gambar dan artefak aplikasi dengan andal di mana saja.

Integrasi berkelanjutan dan penyebaran berkelanjutan (CI/CD)

Integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD) adalah bagian penting dari DevOps inisiatif untuk perubahan perangkat lunak yang cepat. AWS menawarkan layanan untuk diterapkan CI/CD untuk layanan mikro, tetapi diskusi terperinci berada di luar cakupan dokumen ini. Untuk informasi selengkapnya, lihat Mempraktikkan Integrasi Berkelanjutan dan Pengiriman Berkelanjutan di AWS whitepaper.

Jaringan pribadi

AWS PrivateLink adalah teknologi yang meningkatkan keamanan layanan mikro dengan memungkinkan koneksi pribadi antara Virtual Private Cloud (VPC) dan layanan yang didukung. AWS

Ini membantu mengisolasi dan mengamankan lalu lintas layanan mikro, memastikannya tidak pernah melintasi internet publik. Ini sangat berguna untuk mematuhi peraturan seperti PCI atau HIPAA.

Penyimpanan data

Penyimpanan data digunakan untuk mempertahankan data yang dibutuhkan oleh layanan mikro. Toko populer untuk data sesi adalah cache dalam memori seperti Memcached atau Redis. AWS menawarkan kedua teknologi sebagai bagian dari ElastiCache layanan Amazon yang dikelola.

Menempatkan cache antara server aplikasi dan database adalah mekanisme umum untuk mengurangi beban baca pada database, yang, pada gilirannya, memungkinkan sumber daya digunakan untuk mendukung lebih banyak penulisan. Cache juga dapat meningkatkan latensi.

Database relasional masih sangat populer untuk menyimpan data terstruktur dan objek bisnis. AWS menawarkan enam mesin database (Microsoft SQL Server, Oracle, MySQL, MariaDB, PostgreSQL, dan Amazon Aurora) sebagai layanan terkelola melalui Amazon Relational Database Service (Amazon RDS).

Database relasional, bagaimanapun, tidak dirancang untuk skala tanpa akhir, yang dapat mempersulit dan intensif waktu untuk menerapkan teknik untuk mendukung sejumlah besar pertanyaan.

Database NoSQL telah dirancang untuk mendukung skalabilitas, kinerja, dan ketersediaan daripada konsistensi database relasional. Salah satu elemen penting dari database NoSQL adalah bahwa mereka biasanya tidak menerapkan skema yang ketat. Data didistribusikan melalui partisi yang dapat diskalakan secara horizontal dan diambil menggunakan kunci partisi.

Karena layanan mikro individu dirancang untuk melakukan satu hal dengan baik, mereka biasanya memiliki model data yang disederhanakan yang mungkin cocok untuk ketekunan NoSQL. Penting untuk dipahami bahwa database NoSQL memiliki pola akses yang berbeda dari database relasional. Misalnya, tidak mungkin untuk bergabung dengan tabel. Jika ini perlu, logika harus diimplementasikan dalam aplikasi. Anda dapat menggunakan Amazon DynamoDB untuk membuat tabel database yang dapat menyimpan dan mengambil sejumlah data dan melayani setiap tingkat lalu lintas permintaan. DynamoDB memberikan kinerja milidetik satu digit, namun, ada kasus penggunaan tertentu yang memerlukan waktu respons dalam mikrodetik. DynamoDB Accelerator (DAX) menyediakan kemampuan caching untuk mengakses data.

DynamoDB juga menawarkan fitur penskalaan otomatis untuk menyesuaikan kapasitas throughput secara dinamis sebagai respons terhadap lalu lintas aktual. Namun, ada kasus di mana perencanaan

kapasitas sulit atau tidak mungkin karena lonjakan aktivitas besar dengan durasi pendek dalam aplikasi Anda. Untuk situasi seperti itu, DynamoDB menyediakan opsi sesuai permintaan, yang menawarkan harga sederhana. pay-per-request DynamoDB on-demand mampu melayani ribuan permintaan per detik secara instan tanpa perencanaan kapasitas.

Untuk informasi selengkapnya, lihat Manajemen data terdistribusi dan Cara Memilih Database.

Menyederhanakan operasi

Untuk lebih menyederhanakan upaya operasional yang diperlukan untuk menjalankan, memelihara, dan memantau layanan mikro, kita dapat menggunakan arsitektur tanpa server sepenuhnya.

Menyebarkan aplikasi berbasis Lambda

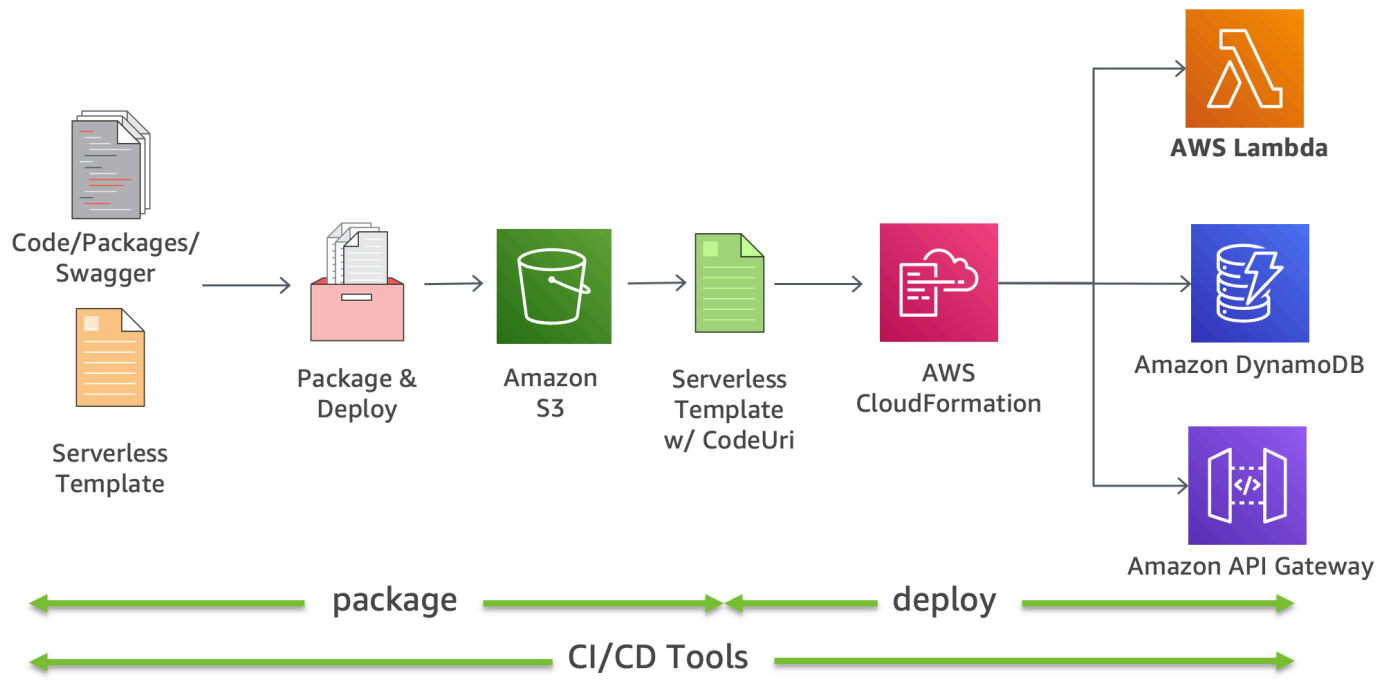
Anda dapat menerapkan kode Lambda dengan mengunggah `zip` arsip file atau dengan membuat dan mengunggah gambar kontainer melalui UI konsol menggunakan URI gambar Amazon ECR yang valid. Namun, ketika fungsi Lambda menjadi kompleks, artinya memiliki lapisan, dependensi, dan izin, mengunggah melalui UI dapat menjadi berat untuk perubahan kode.

Menggunakan AWS CloudFormation dan AWS Serverless Application Model (AWS SAM), AWS Cloud Development Kit (AWS CDK), atau Terraform merampingkan proses mendefinisikan aplikasi tanpa server. AWS SAM, didukung secara native oleh CloudFormation, menawarkan sintaks yang disederhanakan untuk menentukan sumber daya tanpa server.AWS Lambda Lapisan membantu mengelola pustaka bersama di beberapa fungsi Lambda, meminimalkan jejak fungsi, memusatkan pustaka sadar penyewa, dan meningkatkan pengalaman pengembang. Lambda SnapStart untuk Java meningkatkan kinerja startup untuk aplikasi yang sensitif terhadap latensi.

Untuk menerapkan, tentukan kebijakan sumber daya dan izin dalam CloudFormation templat, artefak penerapan paket, dan terapkan templat. SAM Local, sebuah AWS CLI alat, memungkinkan pengembangan lokal, pengujian, dan analisis aplikasi tanpa server sebelum mengunggah ke Lambda.

Integrasi dengan alat seperti AWS Cloud9 IDE, AWS CodeBuild, AWS CodeDeploy, dan AWS CodePipeline merampingkan penulisan, pengujian, debugging, dan penerapan aplikasi berbasis SAM.

Diagram berikut menunjukkan penyebaran AWS Serverless Application Model sumber daya menggunakan CloudFormation dan alat AWS CI/CD.



Gambar 2: AWS Serverless Application Model (AWS SAM)

Mengabstraksikan kompleksitas multi-tenancy

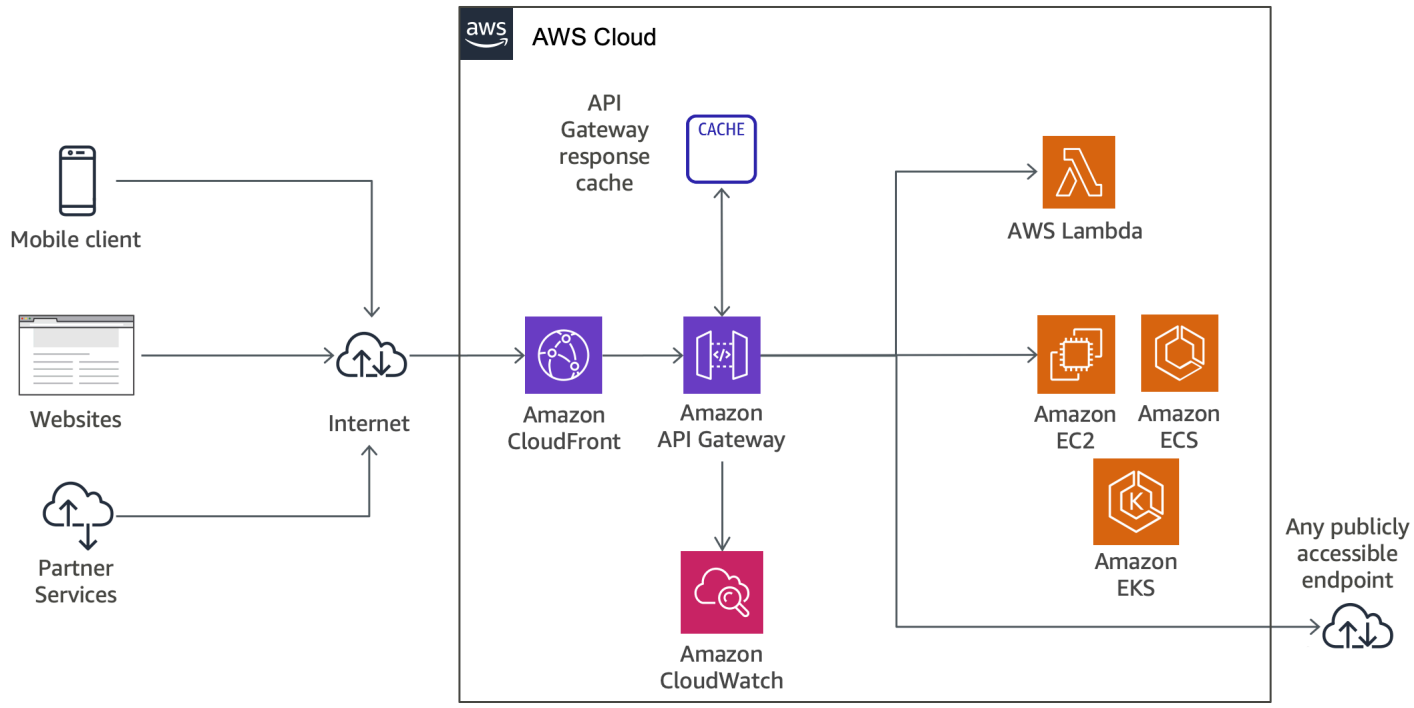
Dalam lingkungan multi-penyewa seperti platform SaaS, sangat penting untuk merampingkan seluk-beluk yang terkait dengan multi-tenancy, membebaskan pengembang untuk berkonsentrasi pada pengembangan fitur dan fungsionalitas. Ini dapat dicapai dengan menggunakan alat seperti AWS Lambda Layers, yang menawarkan pustaka bersama untuk mengatasi masalah lintas sektor. Alasan di balik pendekatan ini adalah bahwa perpustakaan dan alat bersama, bila digunakan dengan benar, mengelola konteks penyewa secara efisien.

Namun, mereka tidak boleh meluas ke enkapsulasi logika bisnis karena kompleksitas dan risiko yang mungkin mereka perkenalkan. Masalah mendasar dengan pustaka bersama adalah meningkatnya kompleksitas seputar pembaruan, membuatnya lebih menantang untuk dikelola dibandingkan dengan duplikasi kode standar. Oleh karena itu, penting untuk mencapai keseimbangan antara penggunaan perpustakaan bersama dan duplikasi dalam pencarian abstraksi yang paling efektif.

Manajemen API

Mengelola APIs dapat memakan waktu, terutama ketika mempertimbangkan beberapa versi, tahapan siklus pengembangan, otorisasi, dan fitur lain seperti throttling dan caching. Selain API Gateway, beberapa pelanggan juga menggunakan ALB (Application Load Balancer) atau NLB (Network Load Balancer) untuk manajemen API. Amazon API Gateway membantu mengurangi kompleksitas operasional pembuatan dan pemeliharaan RESTful APIs. Ini memungkinkan Anda untuk membuat APIs secara terprogram, berfungsi sebagai “pintu depan” untuk mengakses data, logika bisnis, atau fungsionalitas dari layanan backend Anda, Otorisasi dan kontrol akses, pembatasan tarif, caching, pemantauan, dan manajemen lalu lintas dan berjalan tanpa mengelola server. APIs

Gambar 3 menggambarkan bagaimana API Gateway menangani panggilan API dan berinteraksi dengan komponen lain. Permintaan dari perangkat seluler, situs web, atau layanan backend lainnya dialihkan ke CloudFront Point of Presence (PoP) terdekat untuk mengurangi latensi dan memberikan pengalaman pengguna yang optimal.

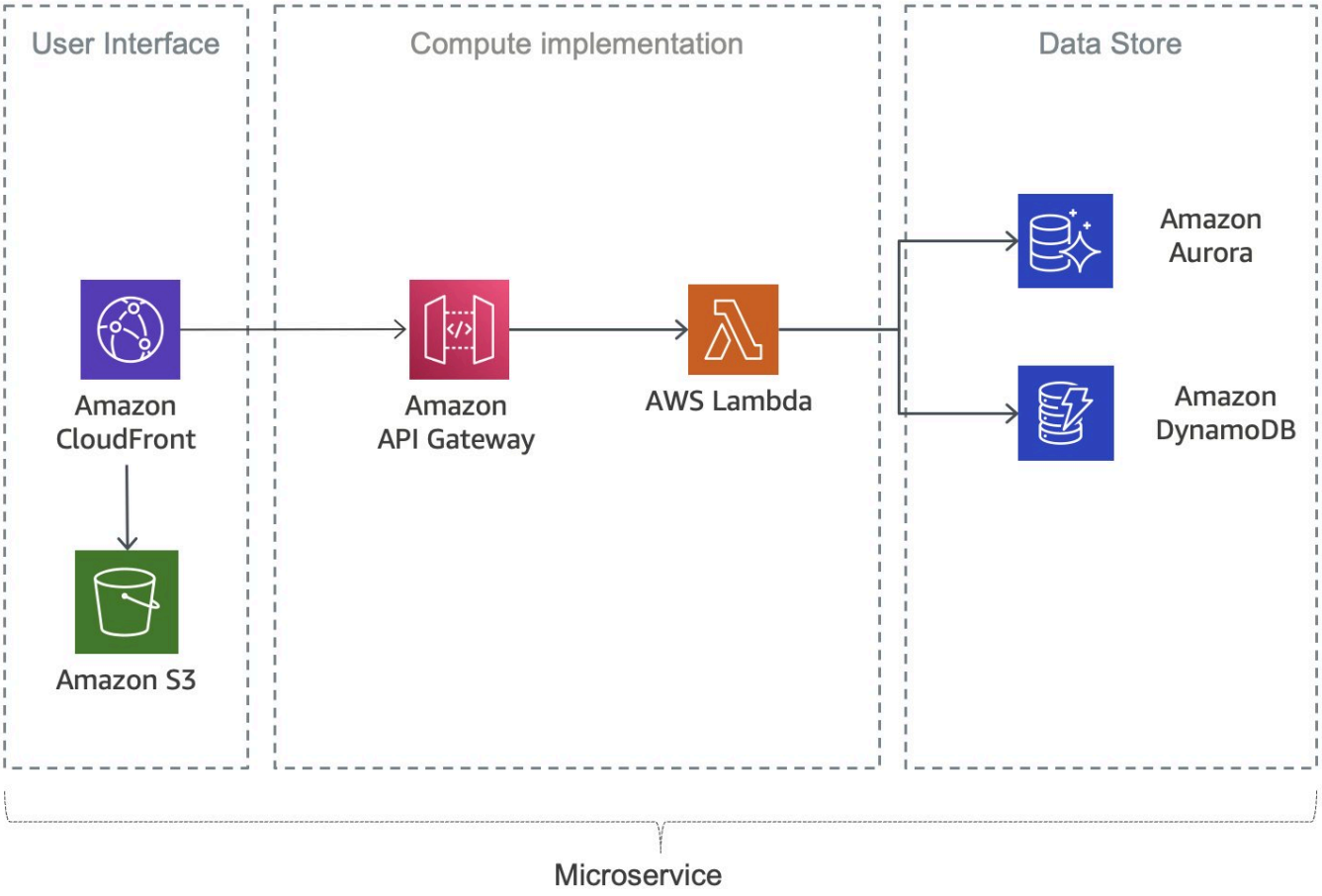


Gambar 3: Alur panggilan API Gateway

Layanan mikro pada teknologi tanpa server

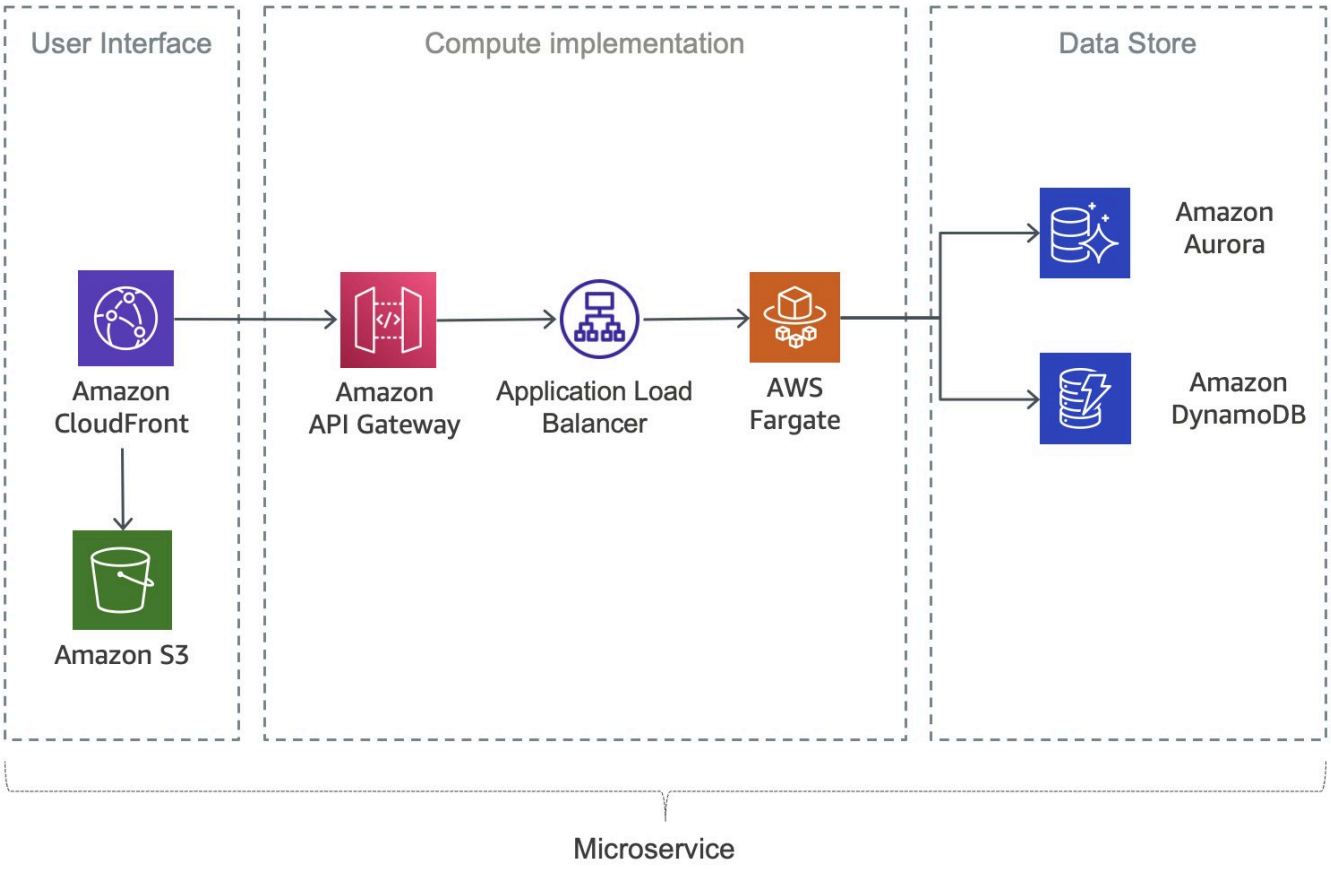
Menggunakan layanan mikro dengan teknologi tanpa server dapat sangat mengurangi kompleksitas operasional. AWS Lambda dan AWS Fargate, terintegrasi dengan API Gateway, memungkinkan pembuatan aplikasi tanpa server sepenuhnya. Mulai 7 April 2023, fungsi Lambda dapat secara progresif mengalirkan muatan respons kembali ke klien, meningkatkan kinerja untuk aplikasi web dan seluler. Sebelum ini, aplikasi berbasis Lambda yang menggunakan model pemanggilan permintaan-respons tradisional harus menghasilkan dan menyangga respons sebelum mengembalikannya ke klien, yang dapat menunda waktu ke byte pertama. Dengan streaming respons, fungsi dapat mengirim respons sebagian kembali ke klien saat mereka siap, secara signifikan meningkatkan waktu ke byte pertama, yang sangat sensitif terhadap aplikasi web dan seluler.

Gambar 4 menunjukkan arsitektur microservice tanpa server menggunakan AWS Lambda dan mengelola layanan. Arsitektur tanpa server ini mengurangi kebutuhan untuk merancang skala dan ketersediaan tinggi, dan mengurangi upaya yang diperlukan untuk menjalankan dan memantau infrastruktur yang mendasarinya.



Gambar 4: Layanan mikro tanpa server menggunakan AWS Lambda

Gambar 5 menampilkan implementasi tanpa server serupa menggunakan kontainer dengan AWS Fargate, menghilangkan kekhawatiran tentang infrastruktur yang mendasarinya. Ini juga dilengkapi Amazon Aurora Serverless, basis data auto-scaling sesuai permintaan yang secara otomatis menyesuaikan kapasitas berdasarkan kebutuhan aplikasi Anda.



Gambar 5: Layanan mikro tanpa server menggunakan AWS Fargate

Sistem yang tangguh dan efisien

Pemulihan bencana (DR)

Aplikasi microservices sering mengikuti pola Aplikasi Dua Belas Faktor, di mana proses bersifat stateless, dan data persisten disimpan dalam layanan pendukung stateful seperti database. Ini menyederhanakan pemulihan bencana (DR) karena jika layanan gagal, mudah untuk meluncurkan instance baru untuk memulihkan fungsionalitas.

Strategi pemulihan bencana untuk layanan mikro harus fokus pada layanan hilir yang mempertahankan status aplikasi, seperti sistem file, database, atau antrian. Organizations harus merencanakan Recovery Time Objective (RTO) dan Recovery Point Objective (RPO). RTO adalah penundaan maksimum yang dapat diterima antara gangguan layanan dan pemulihan, sedangkan RPO adalah waktu maksimum sejak titik pemulihan data terakhir.

Untuk informasi lebih lanjut tentang strategi pemulihan bencana, lihat Disaster Recovery of Workloads on AWS: Recovery in the Cloud whitepaper.

Ketersediaan tinggi (HA)

Kami akan memeriksa ketersediaan tinggi (HA) untuk berbagai komponen arsitektur layanan mikro.

Amazon EKS menyediakan ketersediaan tinggi dengan menjalankan kontrol Kubernetes dan instance pesawat data di beberapa Availability Zone. Ini secara otomatis mendeteksi dan menggantikan instance pesawat kontrol yang tidak sehat dan menyediakan peningkatan dan penambalan versi otomatis.

Amazon ECR menggunakan Amazon Simple Storage Service (Amazon S3) untuk penyimpanan agar gambar kontainer Anda sangat tersedia dan dapat diakses. Ia bekerja dengan Amazon EKS, Amazon ECS, dan AWS Lambda, menyederhanakan pengembangan ke alur kerja produksi.

Amazon ECS adalah layanan regional yang menyederhanakan penampung berjalan dengan cara yang sangat tersedia di beberapa Availability Zone dalam suatu Wilayah, menawarkan beberapa strategi penjadwalan yang menempatkan container untuk kebutuhan sumber daya dan persyaratan ketersediaan.

AWS Lambda beroperasi di beberapa Availability Zone, memastikan ketersediaan selama gangguan layanan dalam satu zona. Jika menghubungkan fungsi Anda ke VPC, tentukan subnet di beberapa Availability Zone untuk ketersediaan tinggi.

Komponen sistem terdistribusi

Dalam arsitektur layanan mikro, penemuan layanan mengacu pada proses secara dinamis menemukan dan mengidentifikasi lokasi jaringan (alamat IP dan port) dari layanan mikro individu dalam sistem terdistribusi.

Saat memilih pendekatan AWS, pertimbangkan faktor-faktor seperti:

- Modifikasi kode: Bisakah Anda mendapatkan manfaat tanpa memodifikasi kode?
- Lalu lintas lintas VPC atau lintas akun: Jika diperlukan, apakah sistem Anda memerlukan manajemen komunikasi yang efisien di berbagai atau? VPCs Akun AWS
- Strategi penyebaran: Apakah sistem Anda menggunakan atau berencana untuk menggunakan strategi penyebaran lanjutan seperti penerapan biru-hijau atau canary?
- Pertimbangan kinerja: Jika arsitektur Anda sering berkomunikasi dengan layanan eksternal, apa yang akan berdampak pada kinerja secara keseluruhan?

AWS menawarkan beberapa metode untuk menerapkan penemuan layanan dalam arsitektur layanan mikro Anda:

- Amazon ECS Service Discovery: Amazon ECS mendukung penemuan layanan menggunakan metode berbasis DNS-nya atau dengan mengintegrasikan dengan AWS Cloud Map (lihat Penemuan Layanan ECS). ECS Service Connect semakin meningkatkan manajemen koneksi, yang dapat sangat bermanfaat untuk aplikasi yang lebih besar dengan beberapa layanan yang berinteraksi.
- Amazon Route 53: Route 53 terintegrasi dengan ECS dan AWS layanan lainnya, seperti EKS, untuk memfasilitasi penemuan layanan. Dalam konteks ECS, Route 53 dapat menggunakan fitur ECS Service Discovery, yang memanfaatkan Auto Naming API untuk secara otomatis mendaftar dan membatalkan pendaftaran layanan.
- AWS Cloud Map: Opsi ini menawarkan penemuan layanan berbasis API dinamis, yang menyebarkan perubahan di seluruh layanan Anda.

Untuk kebutuhan komunikasi yang lebih maju, Amazon VPC Lattice adalah layanan jaringan aplikasi yang secara konsisten menghubungkan, memantau, dan mengamankan komunikasi antar layanan Anda, membantu meningkatkan produktivitas sehingga pengembang Anda dapat fokus membangun fitur yang penting bagi bisnis Anda. Anda dapat menentukan kebijakan untuk manajemen lalu lintas

jaringan, akses, dan pemantauan untuk menghubungkan layanan komputasi dengan cara yang disederhanakan dan konsisten di seluruh instance, container, dan aplikasi tanpa server.

Jika Anda sudah menggunakan perangkat lunak pihak ketiga, seperti HashiCorp Konsul, atau Netflix Eureka untuk penemuan layanan, Anda mungkin lebih suka terus menggunakan ini saat Anda bermigrasi AWS, memungkinkan transisi yang lebih lancar.

Pilihan antara opsi-opsi ini harus selaras dengan kebutuhan spesifik Anda. Untuk persyaratan yang lebih sederhana, solusi berbasis DNS seperti Amazon ECS atau AWS Cloud Map mungkin cukup. Untuk sistem yang lebih kompleks atau lebih besar, jerat layanan seperti Amazon VPC Lattice mungkin lebih cocok.

Kesimpulannya, merancang arsitektur layanan mikro AWS adalah tentang memilih alat yang tepat untuk memenuhi kebutuhan spesifik Anda. Dengan mengingat pertimbangan yang dibahas, Anda dapat memastikan bahwa Anda membuat keputusan berdasarkan informasi untuk mengoptimalkan penemuan layanan sistem dan komunikasi antar-layanan Anda.

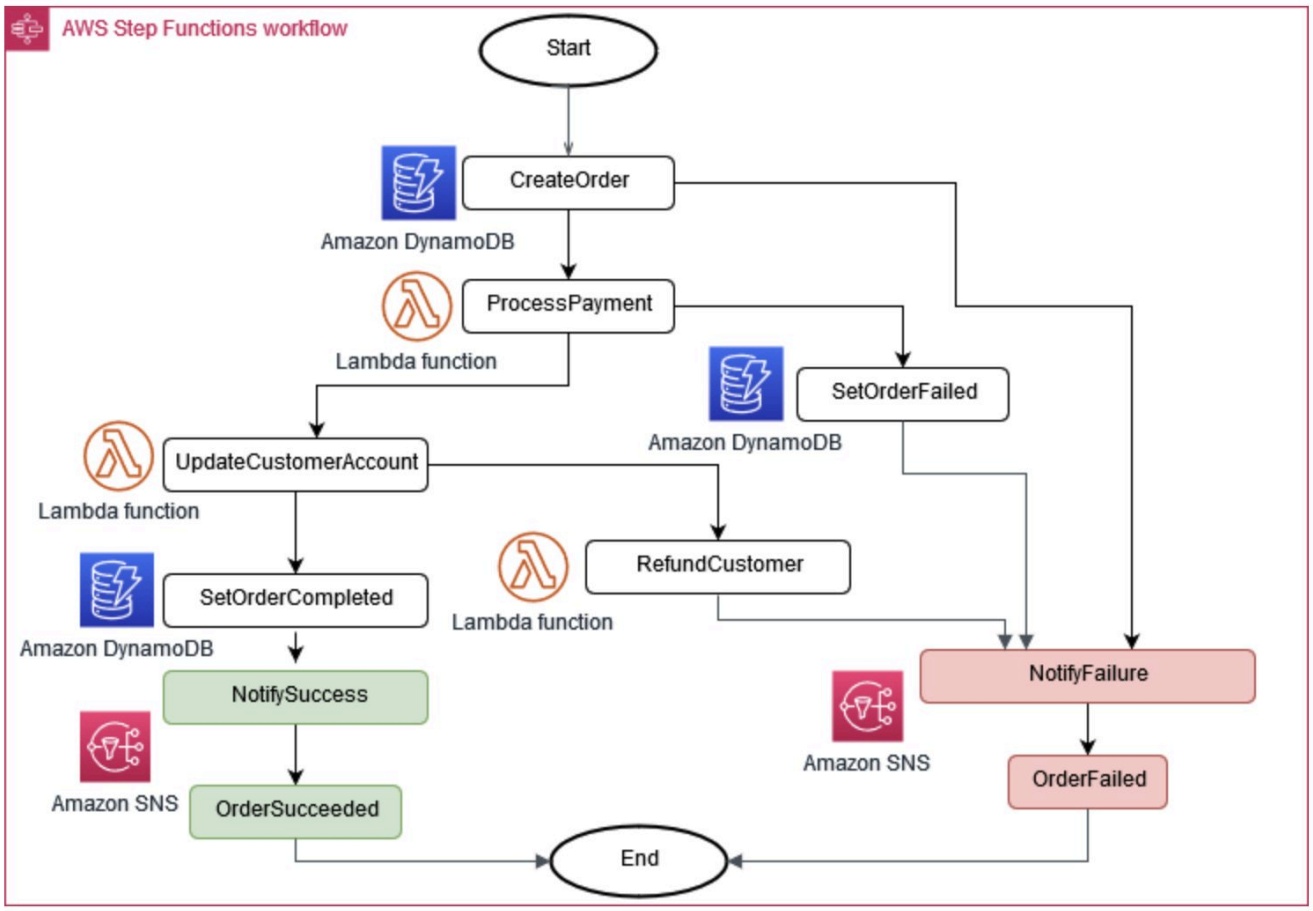
Manajemen data terdistribusi

Dalam aplikasi tradisional, semua komponen sering berbagi satu database. Sebaliknya, setiap komponen aplikasi berbasis layanan mikro mempertahankan datanya sendiri, mempromosikan independensi dan desentralisasi. Pendekatan ini, yang dikenal sebagai manajemen data terdistribusi, membawa tantangan baru.

Salah satu tantangan tersebut muncul dari trade-off antara konsistensi dan kinerja dalam sistem terdistribusi. Seringkali lebih praktis untuk menerima sedikit keterlambatan dalam pembaruan data (konsistensi akhirnya) daripada bersikeras pada pembaruan instan (konsistensi langsung).

Terkadang, operasi bisnis membutuhkan beberapa layanan mikro untuk bekerja sama. Jika satu bagian gagal, Anda mungkin harus membatalkan beberapa tugas yang diselesaikan. Pola Saga membantu mengelola ini dengan mengoordinasikan serangkaian tindakan kompensasi.

Untuk membantu layanan mikro tetap sinkron, penyimpanan data terpusat dapat digunakan. Toko ini, dikelola dengan alat seperti AWS Lambda, AWS Step Functions, dan Amazon EventBridge, dapat membantu membersihkan dan menghapus duplikasi data.

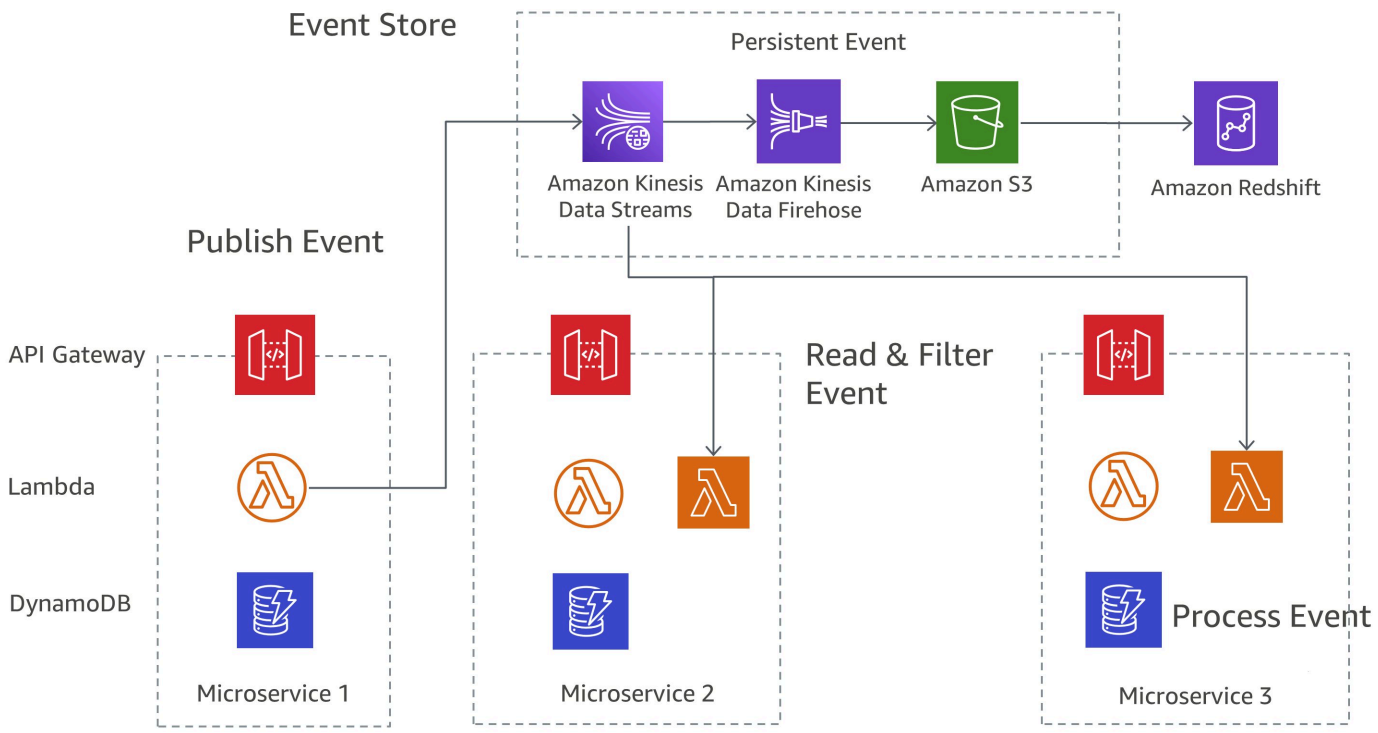


Gambar 6: Koordinator Eksekusi Saga

Pendekatan umum dalam mengelola perubahan di seluruh layanan mikro adalah sumber acara. Setiap perubahan dalam aplikasi dicatat sebagai peristiwa, menciptakan garis waktu status sistem. Pendekatan ini tidak hanya membantu debug dan audit tetapi juga memungkinkan berbagai bagian aplikasi bereaksi terhadap peristiwa yang sama.

Event sourcing sering bekerja hand-in-hand dengan pola Command Query Responsibility Segregation (CQRS), yang memisahkan modifikasi data dan kueri data ke dalam modul yang berbeda untuk kinerja dan keamanan yang lebih baik.

Pada AWS, Anda dapat menerapkan pola-pola ini menggunakan kombinasi layanan. Seperti yang dapat Anda lihat pada Gambar 7, Amazon Kinesis Data Streams dapat berfungsi sebagai toko acara pusat Anda, sementara Amazon S3 menyediakan penyimpanan yang tahan lama untuk semua catatan acara. AWS Lambda, Amazon DynamoDB, dan Amazon API Gateway bekerja sama untuk menangani dan memproses peristiwa ini.



Gambar 7: Pola sumber acara pada AWS

Ingat, dalam sistem terdistribusi, acara mungkin dikirimkan beberapa kali karena percobaan ulang, jadi penting untuk merancang aplikasi Anda untuk menangani ini.

Manajemen konfigurasi

Dalam arsitektur microservices, setiap layanan berinteraksi dengan berbagai sumber daya seperti database, antrian, dan layanan lainnya. Cara yang konsisten untuk mengonfigurasi koneksi dan lingkungan operasi setiap layanan sangat penting. Idealnya, aplikasi harus beradaptasi dengan konfigurasi baru tanpa perlu restart. Pendekatan ini merupakan bagian dari prinsip Aplikasi Dua Belas Faktor, yang merekomendasikan penyimpanan konfigurasi dalam variabel lingkungan.

Pendekatan yang berbeda adalah dengan menggunakan AWS App Config. Ini adalah fitur AWS Systems Manager yang memudahkan pelanggan untuk mengkonfigurasi, memvalidasi, dan menyebarkan flag fitur dan konfigurasi aplikasi dengan cepat dan aman. Data flag dan konfigurasi fitur Anda dapat divalidasi secara sintaksis atau semantik pada fase pra-penerapan, dan dapat dipantau dan diputar kembali secara otomatis jika alarm yang telah Anda konfigurasi dipicu. AppConfig dapat diintegrasikan dengan Amazon ECS dan Amazon EKS dengan menggunakan AWS AppConfig agen. Agen berfungsi sebagai wadah sespan yang berjalan bersama aplikasi penampung Amazon ECS dan Amazon EKS Anda. Jika Anda menggunakan flag AWS AppConfig fitur atau data konfigurasi dinamis lainnya dalam fungsi Lambda, maka kami sarankan Anda menambahkan ekstensi AWS AppConfig Lambda sebagai lapisan ke fungsi Lambda Anda.

GitOpsadalah pendekatan inovatif untuk manajemen konfigurasi yang menggunakan Git sebagai sumber kebenaran untuk semua perubahan konfigurasi. Ini berarti bahwa setiap perubahan yang dilakukan pada file konfigurasi Anda secara otomatis dilacak, diversi, dan diaudit melalui Git.

Manajemen rahasia

Keamanan adalah yang terpenting, jadi kredensional tidak boleh diteruskan dalam teks biasa. AWS menawarkan layanan aman untuk ini, seperti AWS Systems Manager Parameter Store dan AWS Secrets Manager. Alat-alat ini dapat mengirim rahasia ke wadah di Amazon EKS sebagai volume, atau ke Amazon ECS sebagai variabel lingkungan. Dalam AWS Lambda, variabel lingkungan dibuat tersedia untuk kode Anda secara otomatis. Untuk alur kerja Kubernetes, Operator Rahasia Eksternal mengambil rahasia langsung dari layanan seperti AWS Secrets Manager, membuat Rahasia Kubernetes yang sesuai. Ini memungkinkan integrasi yang mulus dengan konfigurasi Kubernetes-native.

Optimalisasi biaya dan keberlanjutan

Arsitektur layanan mikro dapat meningkatkan optimalisasi biaya dan keberlanjutan. Dengan memecah aplikasi menjadi bagian-bagian yang lebih kecil, Anda hanya dapat meningkatkan layanan yang membutuhkan lebih banyak sumber daya, mengurangi biaya dan pemborosan. Ini sangat berguna ketika berhadapan dengan lalu lintas variabel. Layanan mikro dikembangkan secara independen. Jadi pengembang dapat melakukan pembaruan yang lebih kecil, dan mengurangi sumber daya yang dihabiskan untuk pengujian ujung ke ujung. Saat memperbarui, mereka harus menguji hanya sebagian dari fitur yang bertentangan dengan monolit.

Komponen stateless (layanan yang menyimpan status di penyimpanan data eksternal alih-alih penyimpanan data lokal) dalam arsitektur Anda dapat menggunakan Instans EC2 Spot Amazon, yang menawarkan EC2 kapasitas yang tidak terpakai di cloud. AWS Instans ini lebih hemat biaya daripada instans sesuai permintaan dan sangat cocok untuk beban kerja yang dapat menangani gangguan. Ini selanjutnya dapat memangkas biaya sambil mempertahankan ketersediaan tinggi.

Dengan layanan terisolasi, Anda dapat menggunakan opsi komputasi yang dioptimalkan biaya untuk setiap grup auto-scaling. Misalnya, AWS Graviton menawarkan opsi komputasi berkinerja tinggi yang hemat biaya untuk beban kerja yang sesuai dengan instans berbasis ARM.

Mengoptimalkan biaya dan penggunaan sumber daya juga membantu meminimalkan dampak lingkungan, selaras dengan pilar Keberlanjutan dari Kerangka Well-Architected. Anda dapat memantau kemajuan Anda dalam mengurangi emisi karbon menggunakan AWS Customer Carbon Footprint Tool. Alat ini memberikan wawasan tentang dampak lingkungan dari AWS penggunaan Anda.

Mekanisme komunikasi

Dalam paradigma microservices, berbagai komponen aplikasi harus berkomunikasi melalui jaringan. Pendekatan umum untuk ini termasuk pesan berbasis REST, berbasis GraphQL, berbasis GRPC, dan asinkron.

Komunikasi berbasis istirahat

HTTP/S Protokol, yang digunakan secara luas untuk komunikasi sinkron antara layanan mikro, sering beroperasi melalui. RESTful APIs API Gateway menawarkan cara yang efisien untuk membangun API yang berfungsi sebagai titik akses terpusat ke layanan backend, menangani tugas-tugas seperti manajemen lalu lintas, otorisasi, pemantauan, dan kontrol versi.

Komunikasi berbasis GraphQL

Demikian pula, GraphQL adalah metode luas untuk komunikasi sinkron, menggunakan protokol yang sama seperti REST tetapi membatasi eksposur ke titik akhir tunggal. Dengan AWS AppSync, Anda dapat membuat dan menerbitkan aplikasi GraphQL yang berinteraksi AWS dengan layanan dan datastores secara langsung, atau menggabungkan fungsi Lambda untuk logika bisnis.

Komunikasi berbasis GRPC

gRPC adalah protokol komunikasi RPC open-source yang sinkron, ringan, berkinerja tinggi. gRPC meningkatkan protokol yang mendasarinya dengan menggunakan HTTP/2 dan memungkinkan lebih banyak fitur seperti kompresi dan prioritas aliran. Ini menggunakan Protobuf Interface Definition Language (IDL) yang dikodekan biner dan dengan demikian memanfaatkan pembingkaian biner HTTP/2.

Pesan asinkron dan event passing

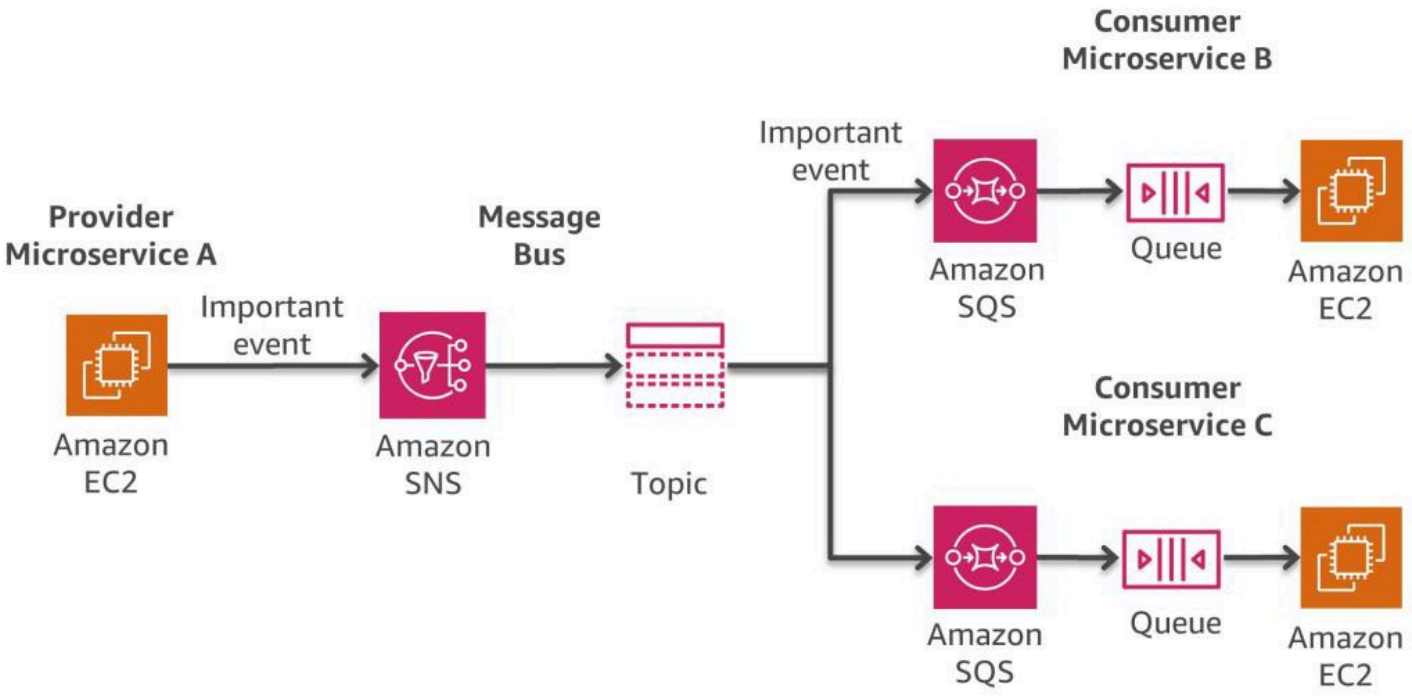
Pesan asinkron memungkinkan layanan untuk berkomunikasi dengan mengirim dan menerima pesan melalui antrian. Hal ini memungkinkan layanan untuk tetap longgar digabungkan dan mempromosikan penemuan layanan.

Pesan dapat didefinisikan dari tiga jenis berikut:

- Antrian Pesan: Antrian pesan bertindak sebagai penyangga yang memisahkan pengirim (produsen) dan penerima (konsumen) pesan. Produsen mengantrekan pesan ke dalam antrian, dan konsumen menghapus antrean dan memprosesnya. Pola ini berguna untuk komunikasi asinkron, leveling beban, dan penanganan semburan lalu lintas.
- Publish-Subscribe: Dalam pola berlangganan terbitkan, pesan dipublikasikan ke suatu topik, dan beberapa pelanggan yang tertarik menerima pesan tersebut. Pola ini memungkinkan penyiaran acara atau pesan ke beberapa konsumen secara asinkron.
- Event-Driven Messaging: Event-driven messaging melibatkan menangkap dan bereaksi terhadap peristiwa yang terjadi dalam sistem. Acara dipublikasikan ke broker pesan, dan layanan yang tertarik berlangganan jenis acara tertentu. Pola ini memungkinkan kopling longgar dan memungkinkan layanan bereaksi terhadap peristiwa tanpa dependensi langsung.

Untuk mengimplementasikan masing-masing jenis pesan ini, AWS menawarkan berbagai layanan terkelola seperti Amazon SQS, Amazon SNS, Amazon, EventBridge Amazon MQ, dan Amazon MSK. Layanan ini memiliki fitur unik yang disesuaikan dengan kebutuhan spesifik:

- Amazon Simple Queue Service (Amazon SQS) dan Amazon Simple Notification Service (Amazon SNS): Seperti yang Anda lihat pada Gambar 8, kedua layanan ini saling melengkapi, dengan Amazon SQS menyediakan ruang untuk menyimpan pesan dan Amazon SNS memungkinkan pengiriman pesan ke beberapa pelanggan. Mereka efektif ketika pesan yang sama perlu dikirim ke beberapa tujuan.



Gambar 8: Pola bus pesan aktif AWS

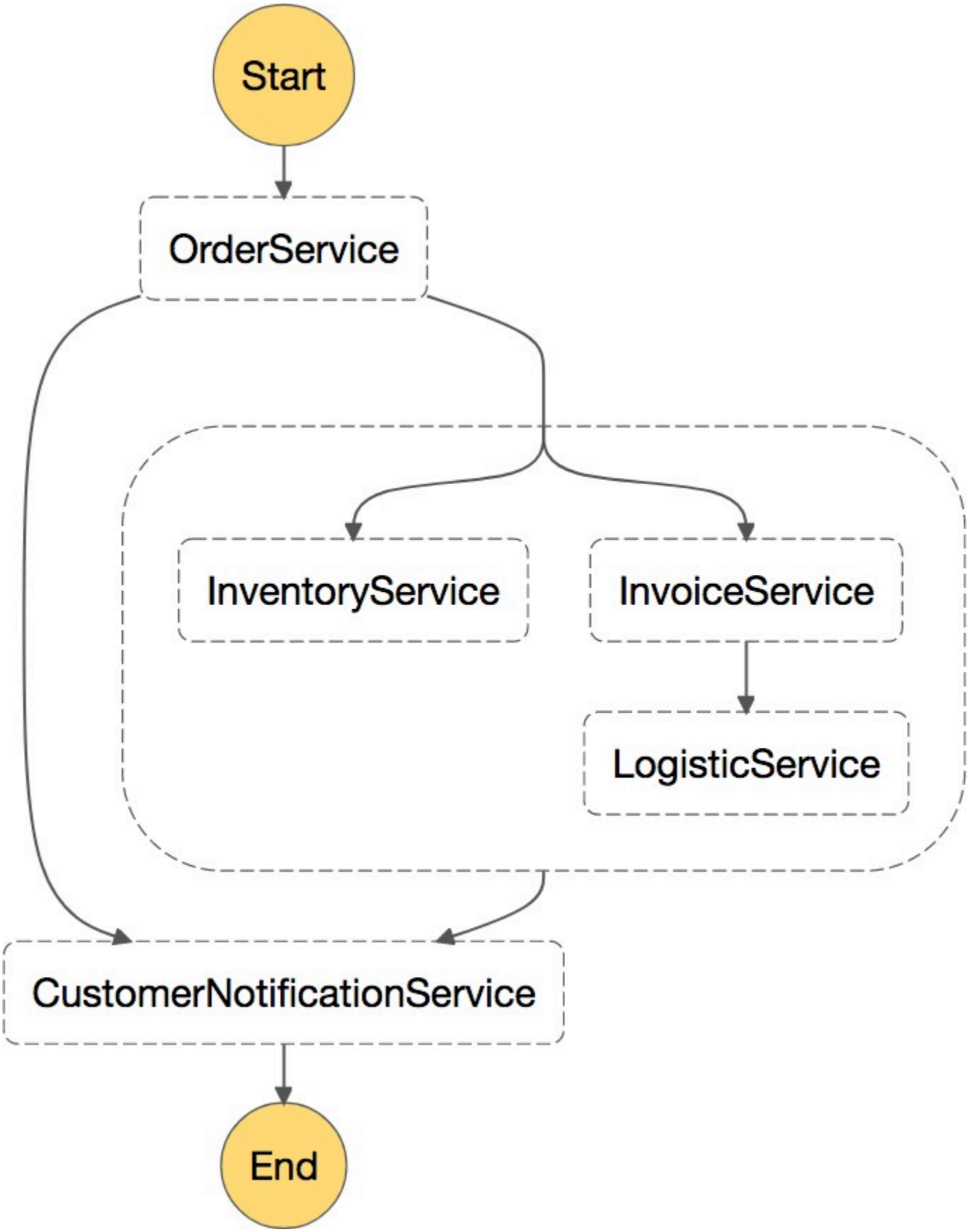
- Amazon EventBridge: layanan tanpa server yang menggunakan peristiwa untuk menghubungkan komponen aplikasi bersama-sama, sehingga memudahkan Anda untuk membangun aplikasi berbasis peristiwa yang dapat diskalakan. Gunakan untuk merutekan acara dari sumber seperti aplikasi rumahan, AWS layanan, dan perangkat lunak pihak ketiga ke aplikasi konsumen di seluruh organisasi Anda. EventBridge menyediakan cara sederhana dan konsisten untuk menelan, memfilter, mengubah, dan menyampaikan acara sehingga Anda dapat membangun aplikasi baru dengan cepat. EventBridge bus acara sangat cocok untuk many-to-many perutean acara antara layanan yang digerakkan oleh acara.
- Amazon MQ: pilihan yang baik jika Anda memiliki sistem pesan yang sudah ada sebelumnya yang menggunakan protokol standar seperti JMS, AMQP, atau sejenisnya. Layanan terkelola ini menyediakan pengganti untuk sistem Anda tanpa mengganggu operasi.
- Amazon MSK (Managed Kafka): sistem pesan untuk menyimpan dan membaca pesan, berguna untuk kasus di mana pesan harus diproses beberapa kali. Ini juga mendukung streaming pesan real-time.
- Amazon Kinesis: pemrosesan dan analisis data streaming secara real-time. Hal ini memungkinkan pengembangan aplikasi real-time dan menyediakan integrasi yang mulus dengan AWS ekosistem.

Ingat, layanan terbaik untuk Anda tergantung pada kebutuhan spesifik Anda, jadi penting untuk memahami apa yang ditawarkan masing-masing dan bagaimana mereka selaras dengan kebutuhan Anda.

Orkestrasi dan manajemen negara

Orkestrasi layanan mikro mengacu pada pendekatan terpusat, di mana komponen sentral, yang dikenal sebagai orkestrator, bertanggung jawab untuk mengelola dan mengoordinasikan interaksi antara layanan mikro. Mengatur alur kerja di beberapa layanan mikro dapat menjadi tantangan. Menyematkan kode orkestrasi langsung ke layanan tidak disarankan, karena memperkenalkan kopling yang lebih ketat dan menghalangi penggantian layanan individual.

Step Functions menyediakan mesin alur kerja untuk mengelola kompleksitas orkestrasi layanan, seperti penanganan kesalahan dan serialisasi. Ini memungkinkan Anda untuk menskalakan dan mengubah aplikasi dengan cepat tanpa menambahkan kode koordinasi. Step Functions adalah bagian dari platform AWS tanpa server dan mendukung fungsi Lambda, Amazon EC2, Amazon EKS, Amazon ECS, AI, dan banyak lagi. SageMaker AWS Glue


Start
OrderService
InventoryService
InvoiceService
LogisticService
CustomerNotificationService
End

Gambar 9: Contoh alur kerja layanan mikro dengan langkah paralel dan sekuensial yang dipanggil oleh AWS Step Functions

Amazon Managed Workflows for Apache Airflow (Amazon MWAA) adalah alternatif untuk Step Functions. Anda harus menggunakan Amazon MWAA jika Anda memprioritaskan open source dan portabilitas. Airflow memiliki komunitas open-source yang besar dan aktif yang menyumbangkan fungsionalitas dan integrasi baru secara teratur.

Observabilitas

Karena arsitektur microservices secara inheren terdiri dari banyak komponen terdistribusi, observabilitas di semua komponen tersebut menjadi penting. Amazon CloudWatch memungkinkan ini, mengumpulkan dan melacak metrik, memantau file log, dan bereaksi terhadap perubahan di lingkungan Anda AWS . Ini dapat memantau AWS sumber daya dan metrik khusus yang dihasilkan oleh aplikasi dan layanan Anda.

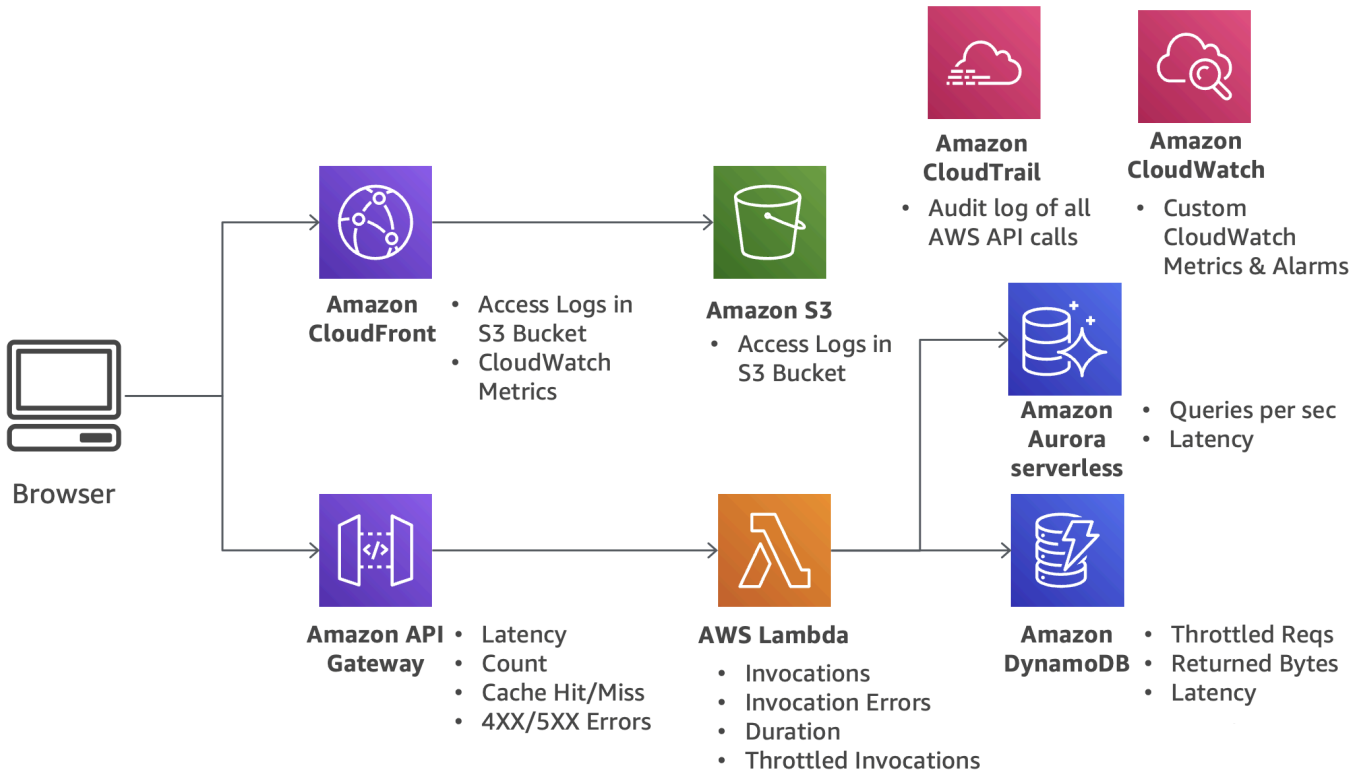
Topik

Memantau

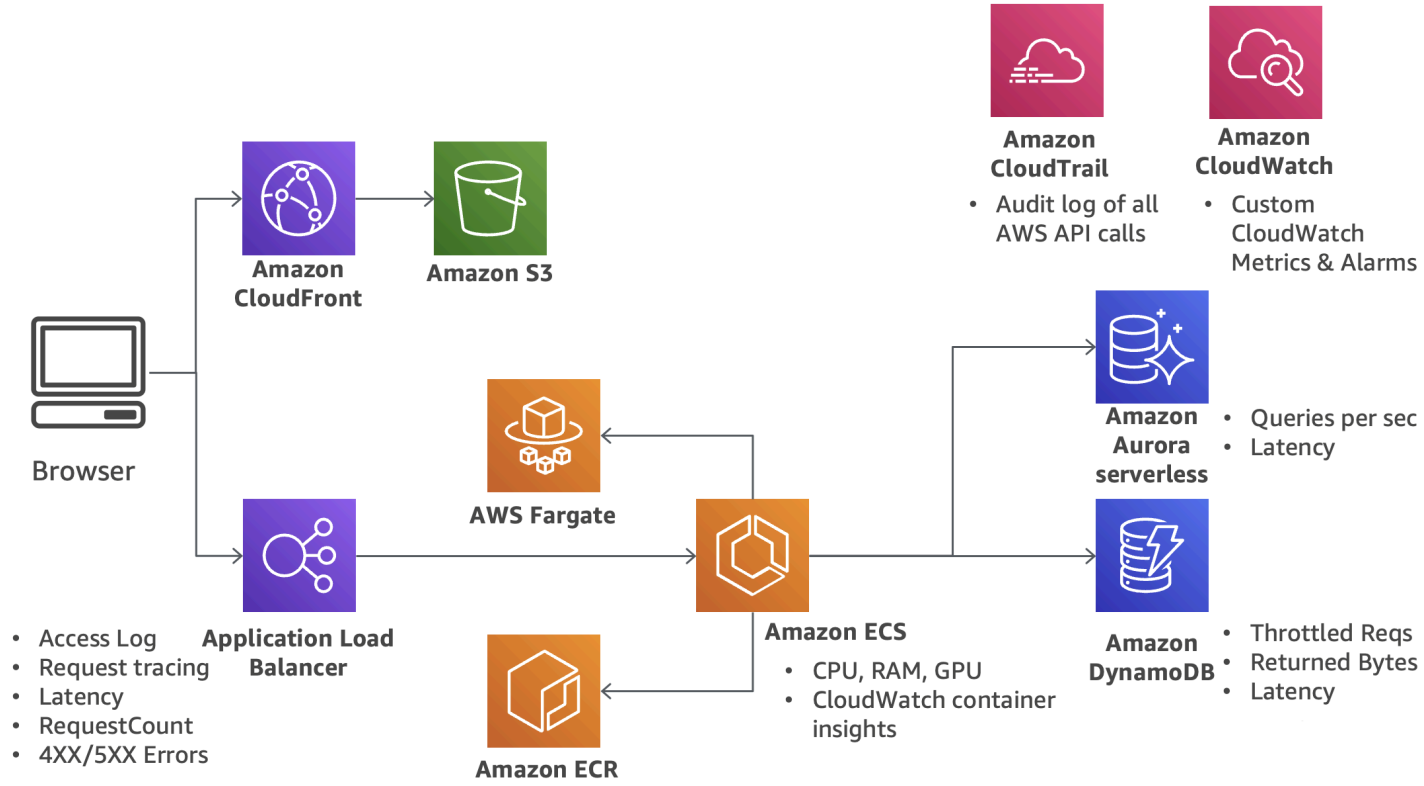
CloudWatch menawarkan visibilitas seluruh sistem ke dalam pemanfaatan sumber daya, kinerja aplikasi, dan kesehatan operasional. Dalam arsitektur layanan mikro, pemantauan metrik khusus bermanfaat, karena pengembang dapat memilih metrik mana yang akan dikumpulkan. CloudWatch Penskalaan dinamis juga dapat didasarkan pada metrik khusus ini.

CloudWatch Container Insights memperluas fungsionalitas ini, secara otomatis mengumpulkan metrik untuk banyak sumber daya seperti CPU, memori, disk, dan jaringan. Ini membantu dalam mendiagnosis masalah terkait wadah, merampingkan resolusi.

Untuk Amazon EKS, pilihan yang sering disukai adalah Prometheus, platform sumber terbuka yang menyediakan kemampuan pemantauan dan peringatan yang komprehensif. Ini biasanya digabungkan dengan Grafana untuk visualisasi metrik intuitif. Amazon Managed Service for Prometheus (AMP) menawarkan layanan pemantauan yang sepenuhnya kompatibel dengan Prometheus, memungkinkan Anda mengawasi aplikasi kontainer dengan mudah. Selain itu, Amazon Managed Grafana (AMG) menyederhanakan analisis dan visualisasi metrik Anda, sehingga tidak perlu mengelola infrastruktur yang mendasarinya.



Gambar 10: Arsitektur tanpa server dengan komponen pemantauan

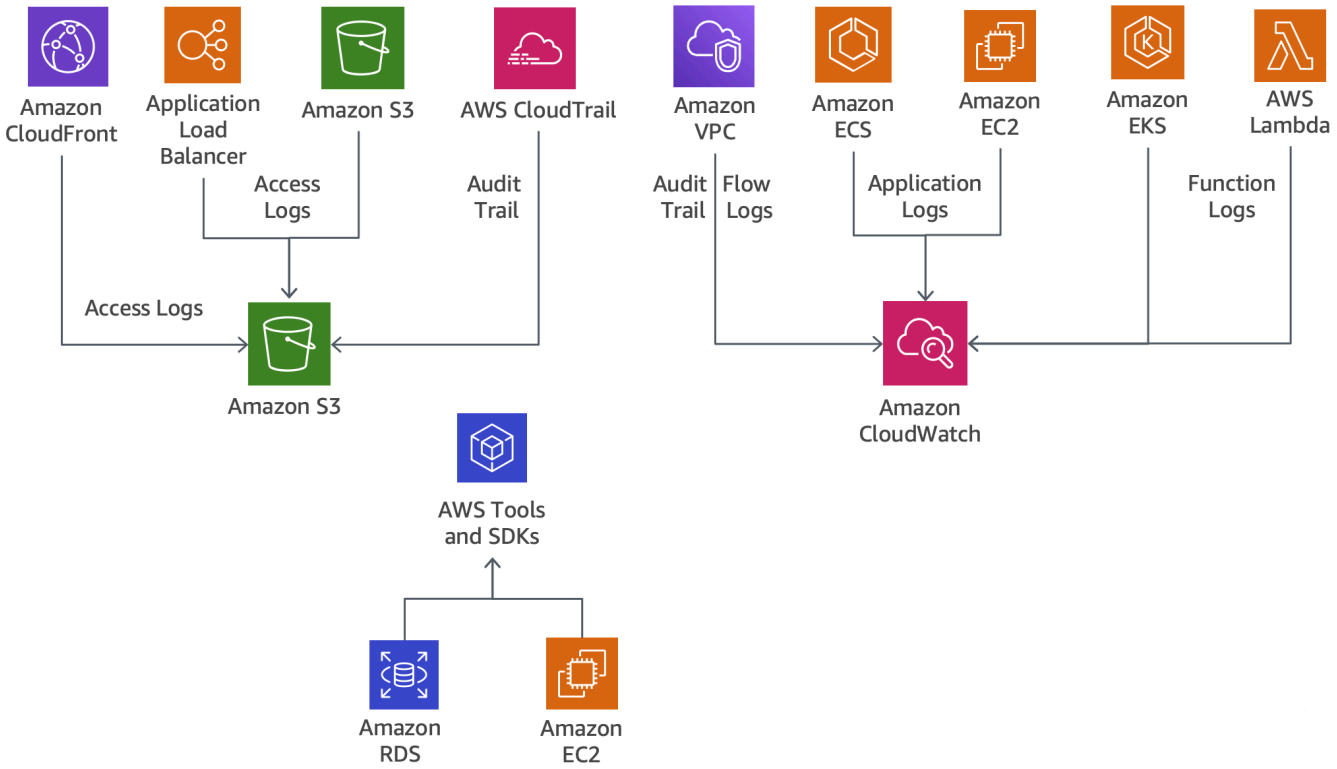


Gambar 11: Arsitektur berbasis kontainer dengan komponen pemantauan

Memusatkan log

Logging adalah kunci untuk menentukan dan menyelesaikan masalah. Dengan layanan mikro, Anda dapat merilis lebih sering dan bereksperimen dengan fitur-fitur baru. AWS menyediakan layanan seperti Amazon S3, CloudWatch Log, dan Amazon OpenSearch Service untuk memusatkan file log. Amazon EC2 menggunakan daemon untuk mengirim log ke, CloudWatch sementara Lambda dan Amazon ECS secara native mengirim output log mereka ke sana. Untuk Amazon EKS, Fluent Bit atau Fluentd dapat digunakan untuk meneruskan log ke CloudWatch pelaporan menggunakan OpenSearch dan Kibana. Namun, karena footprint yang lebih kecil dan keunggulan kinerja, Fluent Bit direkomendasikan daripada Fluentd.

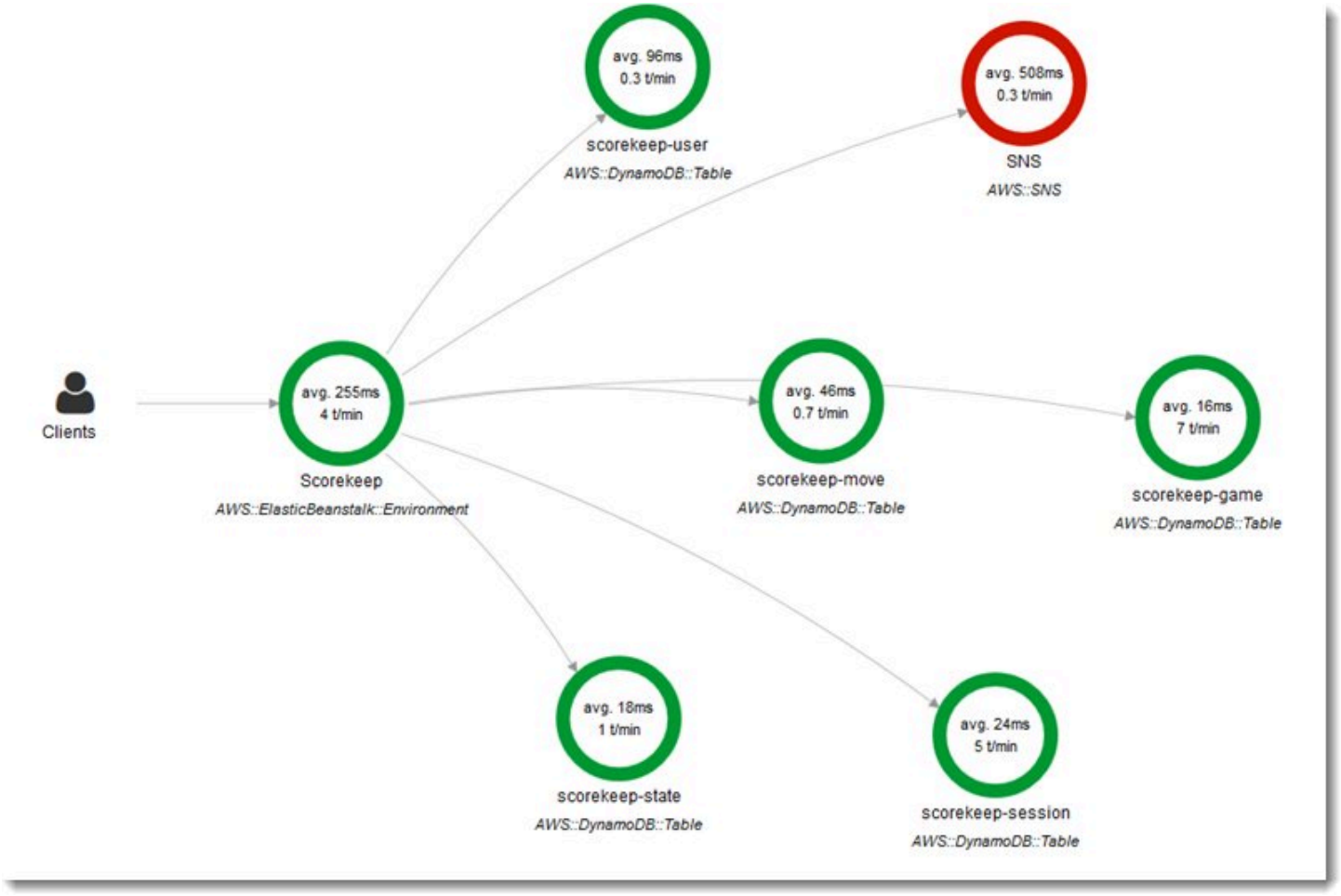
Gambar 12 mengilustrasikan bagaimana log dari berbagai AWS layanan diarahkan ke Amazon CloudWatch S3 dan. Log terpusat ini dapat dianalisis lebih lanjut menggunakan OpenSearch Layanan Amazon, termasuk Kibana untuk visualisasi data. Selain itu, Amazon Athena dapat digunakan untuk kueri ad hoc terhadap log yang disimpan di Amazon S3.



Gambar 12: Kemampuan pencatatan AWS layanan

Penelusuran terdistribusi

Layanan mikro sering bekerja sama untuk menangani permintaan. AWS X-Ray menggunakan ID korelasi untuk melacak permintaan di seluruh layanan ini. X-Ray bekerja dengan Amazon EC2, Amazon ECS, Lambda, dan Elastic Beanstalk.

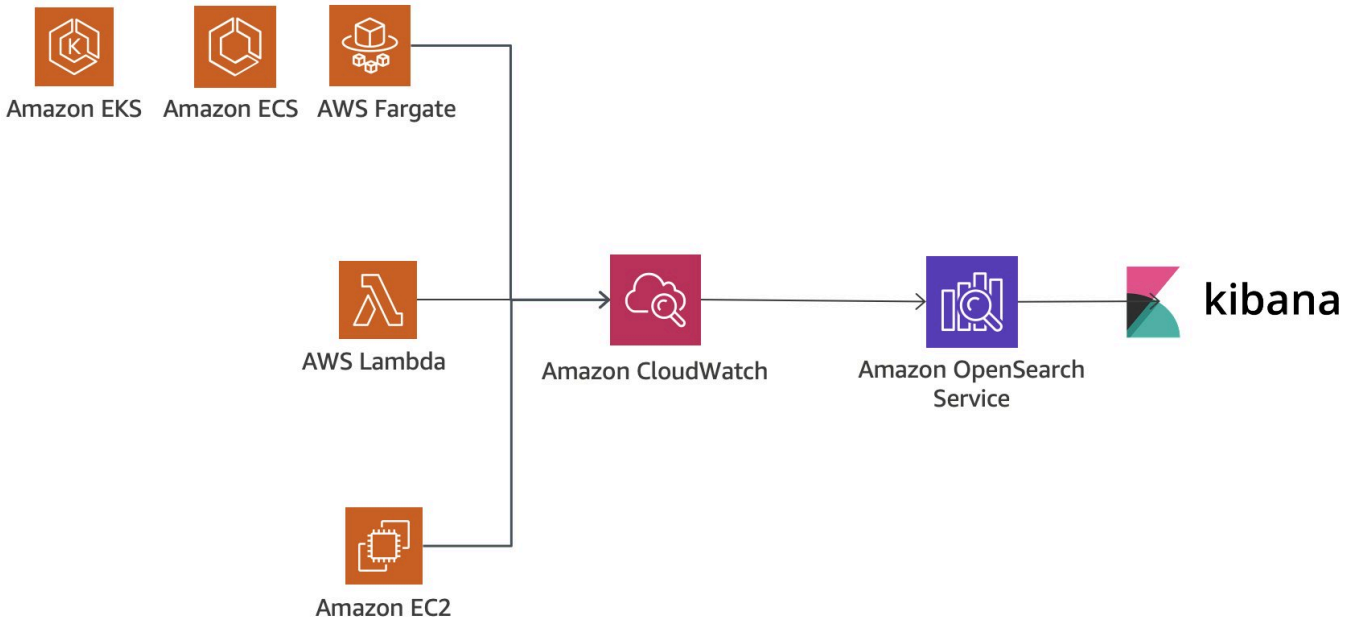


Gambar 13: peta AWS X-Ray layanan

AWS Distro for OpenTelemetry adalah bagian dari OpenTelemetry proyek dan menyediakan sumber terbuka dan agen untuk mengumpulkan jejak APIs dan metrik terdistribusi, meningkatkan pemantauan aplikasi Anda. Ini mengirimkan metrik dan jejak ke beberapa solusi pemantauan AWS dan mitra. Dengan mengumpulkan metadata dari AWS sumber daya Anda, ini menyelaraskan kinerja aplikasi dengan data infrastruktur yang mendasarinya, mempercepat pemecahan masalah. Plus, ini kompatibel dengan berbagai AWS layanan dan dapat digunakan di tempat.

Analisis log pada AWS

Amazon CloudWatch Logs Insights memungkinkan eksplorasi, analisis, dan visualisasi log secara real-time. Untuk analisis file log lebih lanjut, Amazon OpenSearch Service, yang mencakup Kibana, adalah alat yang ampuh. CloudWatch Log dapat mengalirkan entri log ke OpenSearch Layanan secara real time. Kibana, terintegrasi dengan mulus OpenSearch, memvisualisasikan data ini dan menawarkan antarmuka pencarian yang intuitif.

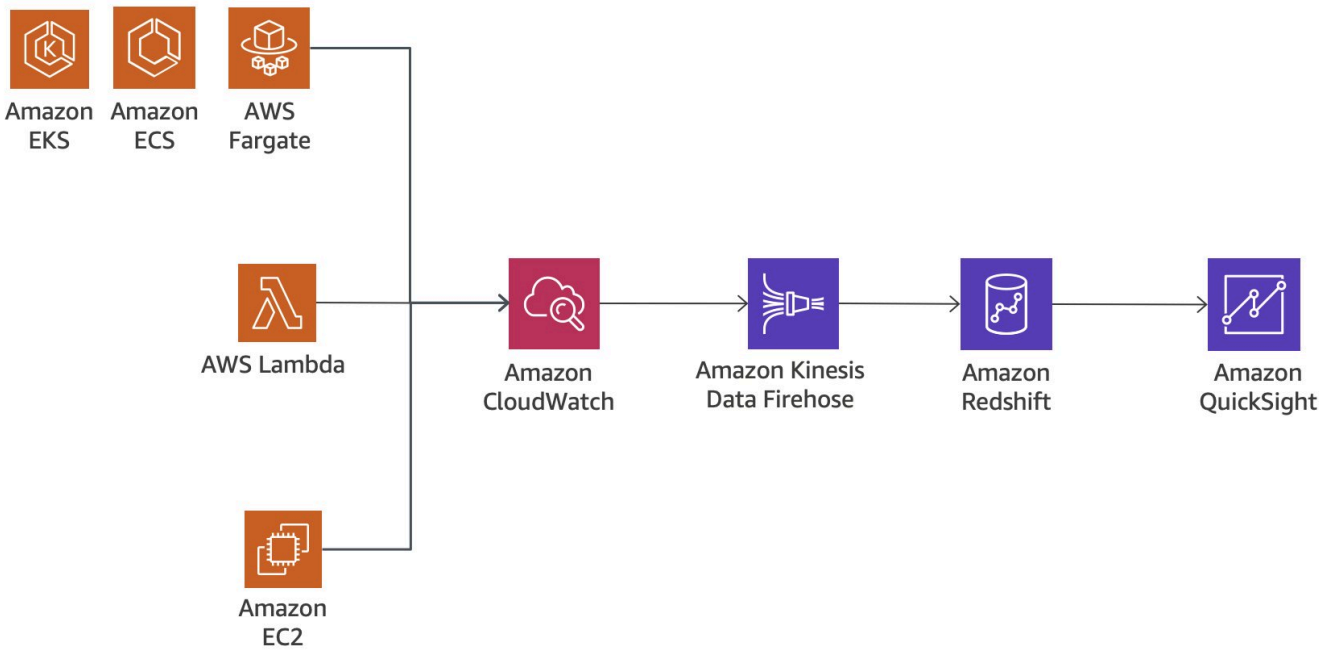


Gambar 14: Analisis log dengan Amazon OpenSearch Service

Opsi lain untuk analisis

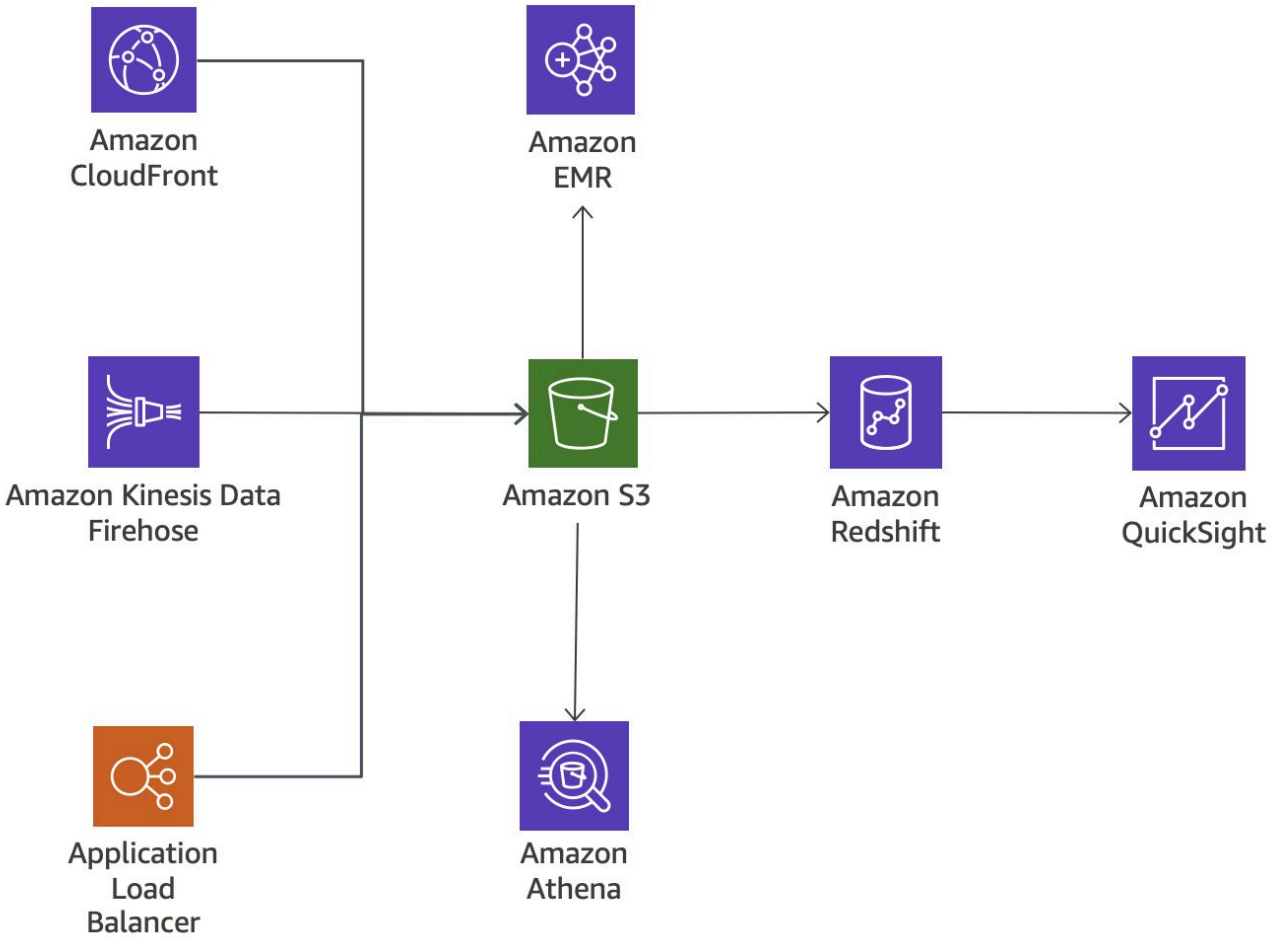
Untuk analisis log lebih lanjut, Amazon Redshift, layanan gudang data yang dikelola sepenuhnya, dan Quick, layanan intelijen bisnis yang dapat diskalakan, menawarkan solusi yang efektif. QuickSight menyediakan konektivitas yang mudah ke berbagai layanan AWS data seperti Redshift, RDS, Aurora, EMR, DynamoDB, Amazon S3, dan Kinesis, menyederhanakan akses data.

CloudWatch Log dapat mengalirkan entri log ke Amazon Data Firehose, layanan untuk mengirimkan data streaming waktu nyata. QuickSight kemudian menggunakan data yang disimpan di Redshift untuk analisis, pelaporan, dan visualisasi yang komprehensif.



Gambar 15: Analisis log dengan Amazon Redshift dan Quick

Selain itu, ketika log disimpan dalam bucket S3, layanan penyimpanan objek, data dapat dimuat ke dalam layanan seperti Redshift atau EMR, platform data besar berbasis cloud, memungkinkan analisis menyeluruh dari data log yang disimpan.



Gambar 16: Merampingkan Analisis Log: Dari AWS layanan ke QuickSight

Mengelola obrolan dalam komunikasi layanan mikro

Chattiness mengacu pada komunikasi yang berlebihan antara layanan mikro, yang dapat menyebabkan inefisiensi karena peningkatan latensi jaringan. Sangat penting untuk mengelola obrolan secara efektif untuk sistem yang berfungsi dengan baik.

Beberapa alat utama untuk mengelola obrolan adalah REST APIs, HTTP dan APIs gRPC. APIs REST APIs menawarkan berbagai fitur canggih seperti kunci API, pembatasan per klien, validasi permintaan, AWS WAF integrasi, atau titik akhir API pribadi. HTTP APIs dirancang dengan fitur minimal dan karenanya datang dengan harga yang lebih rendah. Untuk detail selengkapnya tentang topik ini dan daftar fitur inti yang tersedia di REST APIs dan HTTP APIs, lihat Memilih antara REST APIs dan HTTP APIs.

Seringkali, microservices menggunakan REST melalui HTTP untuk komunikasi karena penggunaannya yang luas. Tetapi dalam situasi volume tinggi, overhead REST dapat menyebabkan masalah kinerja. Itu karena komunikasi menggunakan jabat tangan TCP yang diperlukan untuk setiap permintaan baru. Dalam kasus seperti itu, gRPC API adalah pilihan yang lebih baik. gRPC mengurangi latensi karena memungkinkan beberapa permintaan melalui koneksi TCP tunggal. gRPC juga mendukung streaming dua arah, memungkinkan klien dan server untuk mengirim dan menerima pesan pada saat yang bersamaan. Ini mengarah pada komunikasi yang lebih efisien, terutama untuk transfer data besar atau real-time.

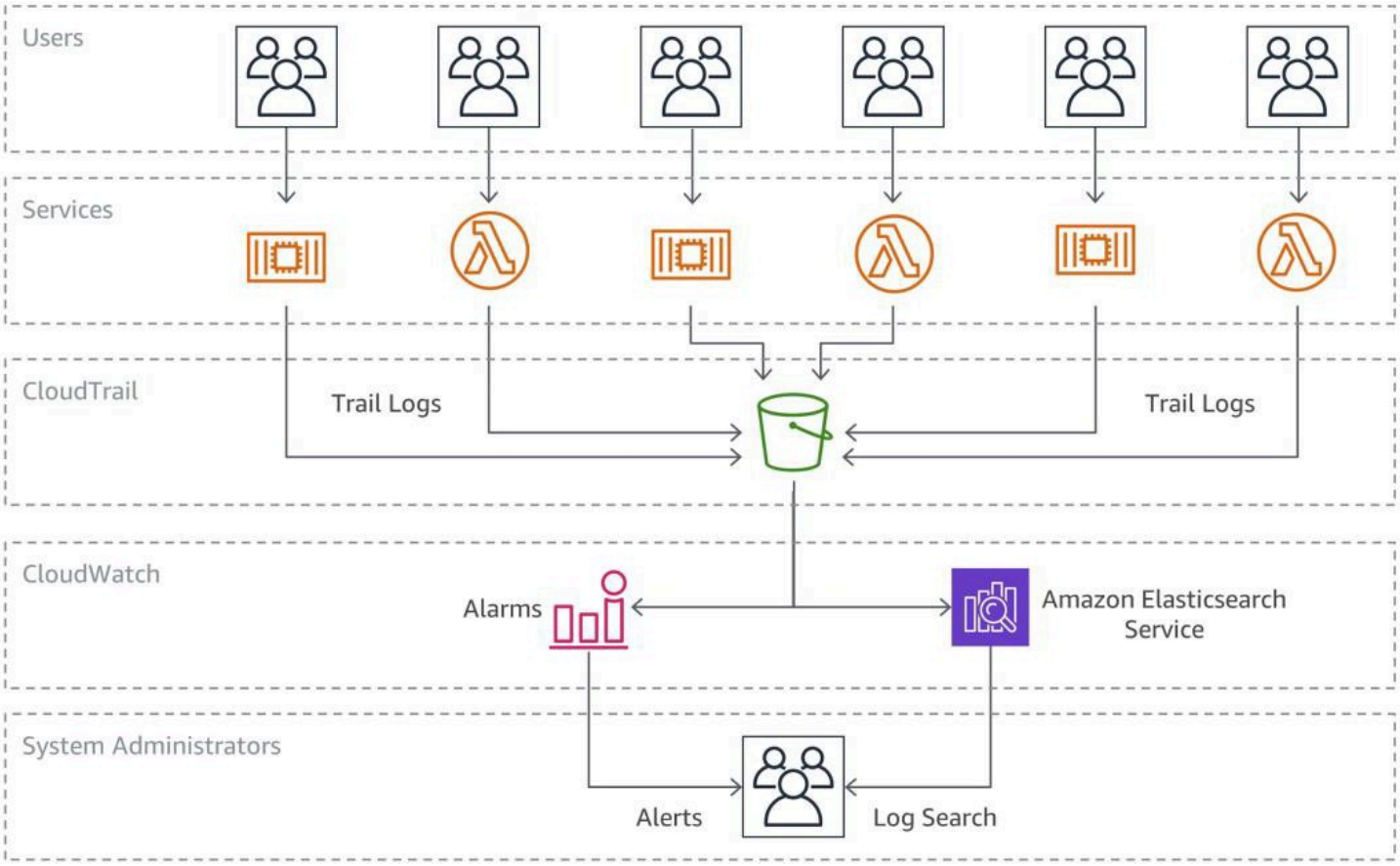
Jika obrolan tetap ada meskipun memilih jenis API yang tepat, mungkin perlu mengevaluasi kembali arsitektur layanan mikro Anda. Mengkonsolidasikan layanan atau merevisi model domain Anda dapat mengurangi obrolan dan meningkatkan efisiensi.

Menggunakan protokol dan caching

Microservices sering menggunakan protokol seperti gRPC dan REST untuk komunikasi (lihat bagian sebelumnya di.Mekanisme komunikasi) gRPC menggunakan HTTP/2 untuk transportasi, sedangkan REST biasanya menggunakan HTTP/1.1. gRPC menggunakan buffer protokol untuk serialisasi, sedangkan REST biasanya menggunakan JSON atau XHTML. Untuk mengurangi latensi dan overhead komunikasi, caching dapat diterapkan. Layanan seperti Amazon ElastiCache atau lapisan caching di API Gateway dapat membantu mengurangi jumlah panggilan antar layanan mikro.

Audit

Dalam arsitektur layanan mikro, sangat penting untuk memiliki visibilitas ke dalam tindakan pengguna di semua layanan. AWS menyediakan alat seperti AWS CloudTrail, yang mencatat semua panggilan API yang dibuat AWS, dan AWS CloudWatch, yang digunakan untuk menangkap log aplikasi. Ini memungkinkan Anda melacak perubahan dan menganalisis perilaku di seluruh layanan mikro Anda. Amazon EventBridge dapat bereaksi terhadap perubahan sistem dengan cepat, memberi tahu orang yang tepat atau bahkan memulai alur kerja secara otomatis untuk menyelesaikan masalah.

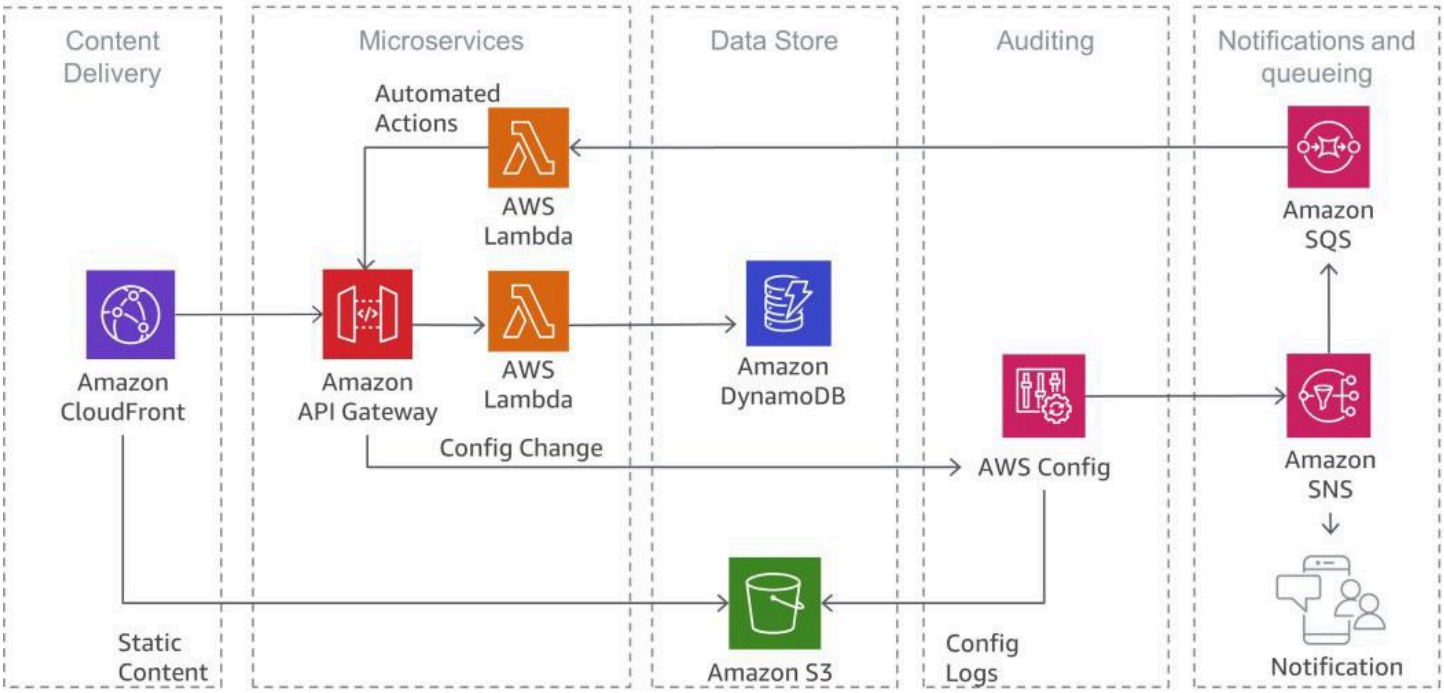


Gambar 17: Audit dan remediasi di seluruh layanan mikro Anda

Inventaris sumber daya dan manajemen perubahan

Dalam lingkungan pengembangan yang gesit dengan konfigurasi infrastruktur yang berkembang pesat, audit dan kontrol otomatis sangat penting. Aturan AWS Config memberikan pendekatan terkelola untuk memantau perubahan ini di seluruh layanan mikro. Mereka memungkinkan definisi kebijakan keamanan tertentu yang secara otomatis mendeteksi, melacak, dan mengirim peringatan tentang pelanggaran kebijakan.

Misalnya, jika konfigurasi API Gateway dalam layanan mikro diubah untuk menerima lalu lintas HTTP masuk, bukan hanya permintaan HTTPS, AWS Config aturan yang telah ditentukan dapat mendeteksi pelanggaran keamanan ini. Ini mencatat perubahan untuk audit dan memicu pemberitahuan SNS, memulihkan status yang sesuai.



Gambar 18: Mendeteksi pelanggaran keamanan dengan AWS Config

Kesimpulan

Arsitektur layanan mikro, pendekatan desain serbaguna yang memberikan alternatif untuk sistem monolitik tradisional, membantu dalam penskalaan aplikasi, meningkatkan kecepatan pengembangan, dan mendorong pertumbuhan organisasi. Dengan kemampuan beradaptasi, dapat diimplementasikan menggunakan wadah, pendekatan tanpa server, atau perpaduan keduanya, disesuaikan dengan kebutuhan spesifik.

Namun, itu bukan one-size-fits-all solusi. Setiap kasus penggunaan memerlukan evaluasi yang cermat mengingat potensi peningkatan kompleksitas arsitektur dan tuntutan operasional. Tetapi ketika didekati secara strategis, manfaat layanan mikro dapat secara signifikan lebih besar daripada tantangan ini. Kuncinya adalah dalam perencanaan proaktif, terutama di bidang observabilitas, keamanan, dan manajemen perubahan.

Penting juga untuk dicatat bahwa di luar layanan mikro, ada kerangka kerja arsitektur yang sama sekali berbeda seperti arsitektur Generative AI seperti Retrieval Augmented Generation (RAG), menyediakan berbagai opsi yang paling sesuai dengan kebutuhan Anda.

AWS, dengan rangkaian layanan terkelola yang kuat, memberdayakan tim untuk membangun arsitektur layanan mikro yang efisien dan meminimalkan kompleksitas secara efektif. Whitepaper ini bertujuan untuk memandu Anda melalui AWS layanan yang relevan dan penerapan pola-pola utama. Tujuannya adalah untuk membekali Anda dengan pengetahuan untuk memanfaatkan kekuatan layanan mikro AWS, memungkinkan Anda memanfaatkan manfaatnya dan mengubah perjalanan pengembangan aplikasi Anda.

Kontributor

Individu dan organisasi berikut berkontribusi terhadap dokumen ini:

- Sascha Möllering, Solusi Arsitektur, Amazon Web Services
- Christian Müller, Arsitektur Solusi, Amazon Web Services
- Matthias Jung, Arsitektur Solusi, Amazon Web Services
- Peter Dalbhanjan, Arsitektur Solusi, Amazon Web Services
- Peter Chapman, Arsitektur Solusi, Amazon Web Services
- Christoph Kassen, Arsitektur Solusi, Amazon Web Services
- Umair Ishaq, Arsitektur Solusi, Amazon Web Services
- Rajiv Kumar, Arsitektur Solusi, Amazon Web Services
- Ramesh Dwarakanath, Arsitektur Solusi, Amazon Web Services
- Andrew Watkins, Arsitektur Solusi, Amazon Web Services
- Yann Stoneman, Arsitektur Solusi, Amazon Web Services
- Mainak Chaudhuri, Solusi Arsitektur, Amazon Web Services
- Gaurav Acharya, Arsitektur Solusi, Amazon Web Services

Riwayat dokumen

Untuk mengetahui jika ada perubahan pada laporan resmi ini, Anda dapat berlangganan umpan RSS.

Perubahan	Deskripsi	Tanggal
Pembaruan besar	Menambahkan informasi tentang Alat Jejak Karbon AWS Pelanggan, Amazon EventBridge, AWS AppSync (GraphQL), AWS Lambda Lapisan, SnapStart Lambda, Model Bahasa Besar (), Amazon Managed Streaming for Apache Kafka (MSKLLMs) , Alur Kerja Terkelola Amazon untuk Apache Airflow (MWAA), Amazon VPC Lattice,. AWS AppConfig Ditambahkan bagian terpisah tentang optimasi biaya dan keberlanj utan.	31 Juli 2023
Pembaruan kecil	Ditambahkan Well-Architected ke abstrak.	13 April 2022
Laporan resmi diperbarui	Integrasi Amazon EventBrid ge, AWS OpenTelemetry, AMP, AMG, Wawasan Kontainer, perubahan teks kecil.	November 9, 2021
Pembaruan kecil	Tata letak halaman disesuaik an	30 April 2021
Pembaruan kecil	Perubahan teks kecil.	1 Agustus 2019

Laporan resmi diperbarui	Integrasi Amazon EKS, AWS Fargate, Amazon MQ, PrivateLink AWS, AWS App Mesh, AWS Cloud Map	1 Juni 2019
Laporan resmi diperbarui	Integrasi AWS Step Functions , AWS X-Ray, dan aliran acara ECS.	1 September 2017
Publikasi awal	Menerapkan Layanan Mikro di AWS diterbitkan.	1 Desember 2016

> **Note**
>
> Untuk berlangganan pembaruan RSS, Anda harus mengaktifkan plug-in RSS untuk browser yang Anda gunakan.

Pemberitahuan

Pelanggan bertanggung jawab untuk membuat penilaian independen mereka sendiri atas informasi dalam dokumen ini. Dokumen ini: (a) hanya untuk tujuan informasi, (b) mewakili penawaran dan praktik AWS produk saat ini, yang dapat berubah tanpa pemberitahuan, dan (c) tidak membuat komitmen atau jaminan apa pun dari AWS dan afiliasinya, pemasok, atau pemberi lisensinya. AWS produk atau layanan disediakan “sebagaimana adanya” tanpa jaminan, representasi, atau kondisi apa pun, baik tersurat maupun tersirat. Tanggung jawab dan kewajiban AWS kepada pelanggannya dikendalikan oleh AWS perjanjian, dan dokumen ini bukan bagian dari, juga tidak mengubah, perjanjian apa pun antara AWS dan pelanggannya.

AWS Glosarium

Untuk AWS terminologi terbaru, lihat AWS glosarium di Referensi.Glosarium AWS

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