Laporan Resmi AWS

Menerapkan Layanan Mikro di AWS

Menerapkan Layanan Mikro di AWS: Laporan Resmi AWS

Table of Contents

Menerapkan Layanan Mikro di AWS

Tanggal publikasi: 9 November 2021 (Riwayat dan kontributor dokumen)

Abstrak

Layanan mikro adalah pendekatan arsitektur dan organisasi terhadap pengembangan perangkat lunak yang dibuat untuk mempercepat siklus deployment, mendorong inovasi dan kepemilikan, meningkatkan pemeliharaan dan skalabilitas aplikasi perangkat lunak, serta menskalakan organisasi yang mengirimkan perangkat lunak dan layanan menggunakan pendekatan tangkas yang membantu tim bekerja secara independen. Dengan pendekatan layanan mikro, perangkat lunak terdiri dari layanan kecil yang berkomunikasi melalui Antarmuka Program Aplikasi (API) yang terdefinisi dengan baik yang dapat di-deploy secara independen. Layanan ini dimiliki oleh tim kecil yang otonom. Pendekatan tangkas ini adalah kunci untuk keberhasilan dalam menskalakan organisasi Anda.

Tiga pola umum telah diamati ketika pelanggan AWS membangun layanan mikro: didorong API, didorong kejadian, dan aliran data. Laporan resmi ini memperkenalkan ketiga pendekatan tersebut dan merangkum karakteristik umum layanan mikro, membahas tantangan utama membangun layanan mikro, serta menjelaskan bagaimana tim produk dapat menggunakan Amazon Web Services (AWS) untuk mengatasi tantangan ini.

Karena laporan resmi ini membahas berbagai topik yang saling berkaitan dengan erat, termasuk penyimpanan data, komunikasi asinkron, dan penemuan layanan, sebaiknya Anda mempertimbangkan persyaratan tertentu dan kasus penggunaan aplikasinya, selain panduan yang diberikan, sebelum membuat pilihan arsitektur.

Pengantar

Arsitektur layanan mikro bukanlah pendekatan yang sepenuhnya baru untuk rekayasa perangkat lunak, melainkan merupakan kombinasi dari berbagai konsep yang sukses dan terbukti, seperti:

- Pengembangan perangkat lunak yang tangkas
- Arsitektur berorientasi layanan
- Desain yang mengutamakan API
- Integrasi berkelanjutan/pengiriman berkelanjutan (CI/CD)

Dalam banyak kasus, pola desain Aplikasi Dua Belas Faktor digunakan untuk layanan mikro.

Laporan resmi ini pertama-tama menjelaskan berbagai aspek arsitektur layanan mikro yang sangat dapat diskalakan dan toleran terhadap kesalahan (antarmuka pengguna, penerapan layanan mikro, dan penyimpanan data) serta cara membangunnya di AWS menggunakan teknologi kontainer. Kemudian, laporan resmi ini merekomendasikan layanan AWS untuk menerapkan arsitektur layanan mikro nirserver yang biasa untuk mengurangi kompleksitas operasional.

Nirserver didefinisikan sebagai model operasional dengan prinsip berikut:

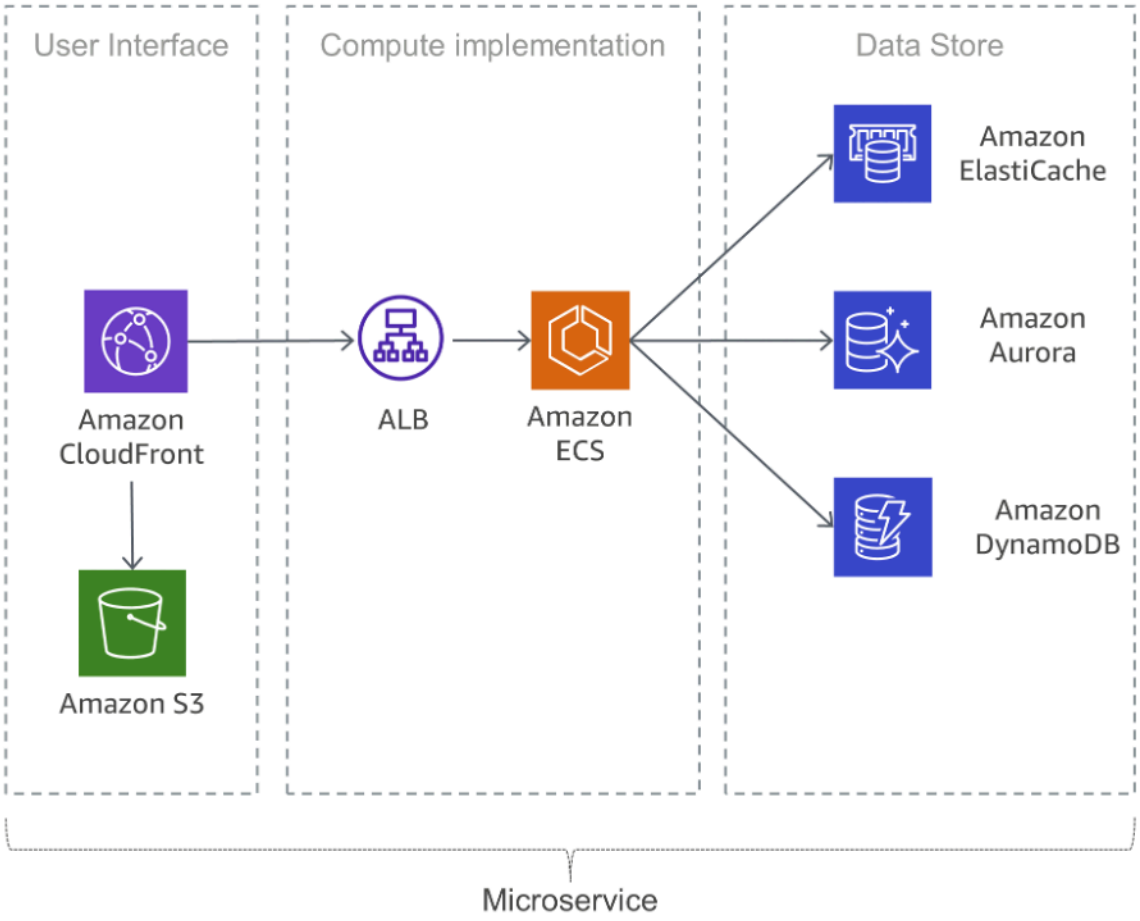
- Tidak ada infrastruktur yang perlu disediakan atau dikelola.
- Penskalaan secara otomatis berdasarkan unit konsumsi
- Model penagihan Bayar sesuai nilai
- Ketersediaan dan toleransi kesalahan bawaan

Terakhir, laporan resmi ini menjelaskan sistem secara keseluruhan dan membahas aspek lintas layanan arsitektur layanan mikro, seperti pemantauan dan pengauditan terdistribusi, konsistensi data, serta komunikasi asinkron.

Laporan resmi ini hanya berfokus pada beban kerja yang berjalan di AWS Cloud. Laporan resmi ini tidak mencakup skenario hybrid atau strategi migrasi. Untuk informasi selengkapnya tentang migrasi, lihat laporan resmi Metodologi Migrasi Kontainer).

Arsitektur layanan mikro di AWS

Aplikasi monolitik yang biasa dibangun menggunakan lapisan yang berbeda-beda—lapisan antarmuka pengguna (UI), lapisan bisnis, dan lapisan persistensi. Gagasan inti dari arsitektur layanan mikro adalah membagi fungsionalitas menjadi beberapa vertikal yang kohesif — bukan berdasarkan lapisan teknologi, tetapi dengan menerapkan domain tertentu. Gambar berikut menampilkan arsitektur referensi untuk aplikasi layanan mikro yang biasa di AWS.



Aplikasi layanan mikro yang biasa di AWS

Topik

- Antarmuka pengguna
- Layanan mikro
- Penyimpanan data

Antarmuka pengguna

Aplikasi web modern sering menggunakan kerangka kerja JavaScript untuk menerapkan aplikasi satu halaman yang berkomunikasi dengan API transfer status representasional (REST) atau RESTful. Konten web statis dapat disajikan menggunakan Amazon Simple Storage Service (S3) dan Amazon CloudFront.

Karena klien layanan mikro dilayani dari lokasi edge terdekat dan mendapatkan respons dari cache atau server proksi dengan koneksi yang dioptimalkan ke sumber asal, latensinya dapat dikurangi secara signifikan. Namun, beberapa layanan mikro yang berjalan secara berdekatan dengan satu sama lain tidak akan mendapatkan manfaat dari jaringan pengiriman konten. Dalam beberapa kasus, pendekatan ini mungkin justru akan lebih meningkatkan latensi. Salah satu praktik terbaik adalah menerapkan mekanisme caching lainnya untuk mengurangi chattiness dan meminimalkan latensi. Untuk informasi selengkapnya, lihat topik the section called “Chattiness”).

Layanan mikro

API adalah pintu depan layanan mikro, yang berarti bahwa API berfungsi sebagai titik masuk untuk logika aplikasi di balik satu set antarmuka program, biasanya API layanan web RESTful. API ini menerima dan memproses panggilan dari klien, dan mungkin menerapkan fungsionalitas, seperti manajemen lalu lintas, pemfilteran permintaan, perutean, caching, autentikasi, dan otorisasi.

Penerapan layanan mikro

AWS memiliki komponen terintegrasi yang mendukung pengembangan layanan mikro. Dua pendekatan yang populer diterapkan menggunakan kontainer AWS Lambda dan Docker dengan AWS Fargate.

Dengan AWS Lambda, Anda mengunggah kode dan membiarkan Lambda mengurus semua yang diperlukan untuk menjalankan serta menskalakan penerapan guna memenuhi kurva permintaan Anda yang sebenarnya dengan ketersediaan tinggi. Administrasi infrastruktur tidak diperlukan. Lambda mendukung beberapa bahasa pemrograman dan dapat dipanggil dari layanan AWS lain atau dipanggil langsung dari web atau aplikasi seluler apa pun. Salah satu keuntungan terbesar AWS Lambda adalah Anda dapat bergerak cepat: Anda dapat berfokus pada logika bisnis karena keamanan dan penskalaan dikelola oleh AWS. Pendekatan dogmatis Lambda akan mendorong platform yang dapat diskalakan.

Salah satu pendekatan yang umum untuk mengurangi upaya operasional untuk deployment adalah deployment berbasis kontainer. Teknologi kontainer, seperti Docker telah meningkat popularitasnya

dalam beberapa tahun terakhir karena manfaat, seperti portabilitas, produktivitas, dan efisiensi. Kontainer dapat cukup sulit untuk dipelajari dan Anda harus memikirkan perbaikan keamanan untuk image dan pemantauan Docker Anda. Amazon Elastic Container Service (Amazon ECS) dan Amazon Elastic Kubernetes Service (Amazon EKS) menghilangkan kebutuhan untuk menginstal, mengoperasikan, dan menskalakan infrastruktur manajemen klaster Anda sendiri. Dengan panggilan API, Anda dapat meluncurkan dan menghentikan aplikasi yang didukung Docker, mengkueri status lengkap klaster Anda, serta mengakses banyak fitur yang sudah dikenal, seperti grup keamanan, Penyeimbangan Beban, volume Amazon Elastic Block Store (Amazon EBS), dan peran AWS Identity and Access Management (IAM).

AWS Fargate adalah mesin komputasi nirserver untuk kontainer yang beroperasi dengan Amazon ECS dan Amazon EKS. Dengan Fargate, Anda tidak perlu lagi memikirkan penyediaan sumber daya komputasi yang cukup untuk aplikasi kontainer Anda. Fargate dapat meluncurkan puluhan ribu kontainer dan diskalakan dengan mudah untuk menjalankan aplikasi Anda yang sangat penting.

Amazon ECS mendukung strategi dan batas penempatan kontainer untuk menyesuaikan cara Amazon ECS menempatkan dan mengakhiri tugas. Batas penempatan tugas adalah aturan yang dipertimbangkan selama penempatan tugas. Anda dapat mengaitkan atribut, yang pada dasarnya adalah pasangan kunci-nilai, ke instans kontainer Anda lalu menggunakan batas untuk menempatkan tugas berdasarkan atribut ini. Misalnya, Anda dapat menggunakan batas untuk menempatkan layanan mikro tertentu berdasarkan tipe instans atau kemampuan instans, seperti instans yang didukung GPU.

Amazon EKS menjalankan perangkat lunak Kubernetes sumber terbuka versi terbaru, sehingga Anda dapat menggunakan semua plugin dan alat yang ada dari komunitas Kubernetes. Aplikasi yang berjalan di Amazon EKS sepenuhnya kompatibel dengan aplikasi yang berjalan di lingkungan Kubernetes standar apa pun, baik yang berjalan di pusat data on-premise maupun cloud publik. Amazon EKS mengintegrasikan IAM dengan Kubernetes, sehingga memungkinkan Anda mendaftarkan entitas IAM dengan sistem autentikasi native di Kubernetes. Tidak perlu menyiapkan kredensial secara manual untuk melakukan autentikasi dengan bidang kendali Kubernetes. Integrasi IAM memungkinkan Anda menggunakan IAM untuk langsung melakukan autentikasi dengan bidang kendali itu sendiri serta memberikan akses yang sangat terperinci ke titik akhir publik pada bidang kendali Kubernetes Anda.

Image Docker yang digunakan di Amazon ECS dan Amazon EKS dapat disimpan di (Amazon Elastic Container Registry (Amazon ECR). Amazon ECR membuat Anda tidak perlu lagi mengoperasikan dan menskalakan infrastruktur yang diperlukan untuk mendukung registri kontainer Anda.

Integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD) adalah praktik terbaik dan bagian penting dari inisiatif DevOps yang memungkinkan perubahan perangkat lunak yang cepat sambil mempertahankan stabilitas dan keamanan sistem. Namun, hal ini di luar lingkup untuk laporan resmi ini. Untuk informasi selengkapnya, lihat laporan resmi Praktik Integrasi Berkelanjutan dan Pengiriman Berkelanjutan di AWS.

Tautan privat

AWS PrivateLink adalah teknologi yang berketersediaan tinggi dan dapat diskalakan yang memungkinkan Anda menghubungkan Virtual Private Cloud (VPC) secara privat ke layanan AWS yang didukung, layanan yang di-host oleh akun AWS lainnya (layanan titik akhir VPC), serta layanan partner AWS Marketplace yang didukung. Anda tidak memerlukan gateway internet, perangkat terjemahan alamat jaringan, alamat IP publik, koneksi AWS Direct Connect, atau koneksi VPN untuk berkomunikasi dengan layanan ini. Lalu lintas antara VPC Anda dan layanan tersebut tidak meninggalkan jaringan Amazon.

Tautan privat adalah cara yang bagus untuk meningkatkan isolasi dan keamanan arsitektur layanan mikro. Layanan mikro, misalnya, dapat digunakan dalam VPC yang sepenuhnya terpisah, berada di balik penyeimbang beban, dan diekspos ke layanan mikro lainnya melalui titik akhir AWS PrivateLink. Dengan penyiapan ini, menggunakan AWS PrivateLink, lalu lintas jaringan ke dan dari layanan mikro tidak pernah melintasi internet publik. Satu kasus penggunaan untuk isolasi tersebut mencakup kepatuhan terhadap peraturan untuk layanan yang menangani data sensitif, seperti PCI, HIPPA, dan Privacy Shield Uni Eropa/AS. Selain itu, AWS PrivateLink memungkinkan layanan mikro terhubung dari berbagai akun dan Amazon VPC, tanpa memerlukan aturan firewall, definisi jalur, atau tabel rute, sehingga menyederhanakan manajemen jaringan. Dengan memanfaatkan PrivateLink, penyedia Perangkat Lunak sebagai Layanan (SaaS) dan ISV juga dapat menawarkan solusi berbasis layanan mikro mereka dengan isolasi operasional yang menyeluruh dan akses aman.

Penyimpanan data

Penyimpanan data digunakan untuk melakukan persistensi data yang dibutuhkan oleh layanan mikro. Penyimpanan populer untuk data sesi adalah cache dalam memori, seperti Memcached atau Redis. AWS menawarkan kedua teknologi tersebut sebagai bagian dari layanan Amazon ElastiCache terkelola.

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

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

Namun, basis data relasional tidak dirancang untuk penskalaan tanpa batas, yang dapat menyulitkan dan menghabiskan banyak waktu guna menerapkan teknik untuk mendukung sejumlah besar kueri.

Basis data NoSQL telah dirancang untuk mendukung skalabilitas, performa, dan ketersediaan dibandingkan dengan konsistensi yang dimiliki basis data relasional. Salah satu elemen penting dari basis data NoSQL adalah bahwa basis data ini biasanya tidak memberlakukan skema yang ketat. Data didistribusikan ke beberapa partisi yang dapat diskalakan secara horizontal dan diambil menggunakan kunci partisi.

Karena setiap layanan mikro dirancang untuk melakukan satu hal dengan baik, layanan mikro ini biasanya memiliki model data yang disederhanakan yang mungkin cocok untuk persistensi NoSQL. Penting untuk dipahami bahwa basis data NoSQL memiliki pola akses yang berbeda dari basis data relasional. Misalnya, penggabungan tabel tidak dapat dilakukan. Jika hal ini diperlukan, logikanya harus diterapkan dalam aplikasi. Anda dapat menggunakan Amazon DynamoDB untuk membuat tabel basis data yang dapat menyimpan dan mengambil data dalam jumlah apa pun dan melayani tingkat lalu lintas permintaan apa pun. DynamoDB memberikan performa milidetik digit tunggal, tetapi ada kasus penggunaan tertentu yang memerlukan waktu respons dalam mikrodetik. Amazon 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 sebenarnya. Namun, ada kasus saat perencanaan kapasitas sulit dilakukan atau tidak memungkinkan karena lonjakan aktivitas besar yang berdurasi singkat dalam aplikasi Anda. Untuk situasi seperti itu, DynamoDB menyediakan opsi sesuai permintaan, yang menawarkan harga bayar sesuai permintaan yang sederhana. DynamoDB sesuai permintaan mampu melayani ribuan permintaan per detik secara instan tanpa perencanaan kapasitas.

Mengurangi kompleksitas operasional

Arsitektur yang sebelumnya dijelaskan dalam laporan resmi ini sudah menggunakan layanan terkelola, tetapi instans Amazon Elastic Compute Cloud (Amazon EC2) masih perlu dikelola. Upaya operasional yang diperlukan untuk menjalankan, memelihara, dan memantau layanan mikro dapat dikurangi secara lebih lanjut menggunakan arsitektur yang sepenuhnya nirserver.

Topik

- Penerapan API
- Layanan mikro nirserver
- Pemulihan bencana
- Ketersediaan tinggi
- Melakukan deployment aplikasi berbasis Lambda

Penerapan API

Perancangan, deployment, pemantauan, peningkatan berkelanjutan, dan pemeliharaan API dapat menjadi tugas yang memakan waktu. Terkadang versi API yang berbeda-beda perlu dijalankan untuk memastikan kompatibilitas mundur untuk semua klien. Tahapan yang berbeda-beda dalam siklus pengembangan (misalnya, pengembangan, pengujian, dan produksi) makin menambah upaya operasional.

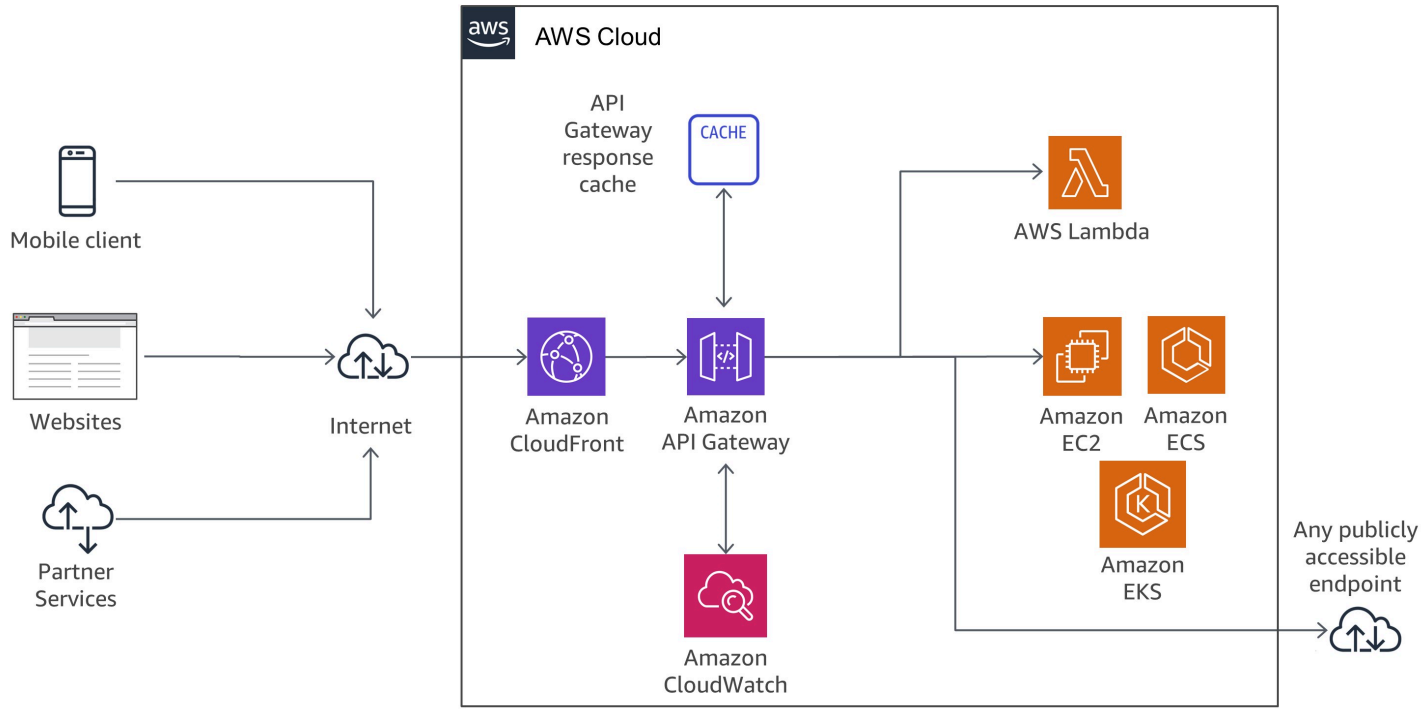
Otorisasi adalah fitur penting untuk semua API, tetapi biasanya kompleks untuk dibuat dan memerlukan pekerjaan yang berulang. Ketika API dipublikasikan dan berhasil, tantangan berikutnya adalah mengelola, memantau, dan memonetisasi ekosistem developer pihak ketiga yang memanfaatkan API tersebut.

Fitur dan tantangan penting lainnya mencakup throttling permintaan untuk melindungi layanan backend, caching respons API, penanganan transformasi permintaan dan respons, serta membuat definisi dan dokumentasi API dengan alat seperti Swagger.

Amazon API Gateway mengatasi tantangan tersebut dan mengurangi kompleksitas operasional dalam membuat dan memelihara API RESTful. API Gateway memungkinkan Anda membuat API Anda secara programatis dengan mengimpor definisi Swagger menggunakan API AWS atau Konsol Manajemen AWS. API Gateway berfungsi sebagai pintu depan untuk aplikasi web apa pun yang

berjalan di Amazon EC2, Amazon ECS, AWS Lambda, atau di lingkungan on-premise apa pun. Pada dasarnya, API Gateway memungkinkan Anda menjalankan API tanpa harus mengelola server.

Gambar berikut menampilkan 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 meminimalkan latensi dan memberikan pengalaman pengguna yang optimal.



Alur panggilan API Gateway

Layanan mikro nirserver

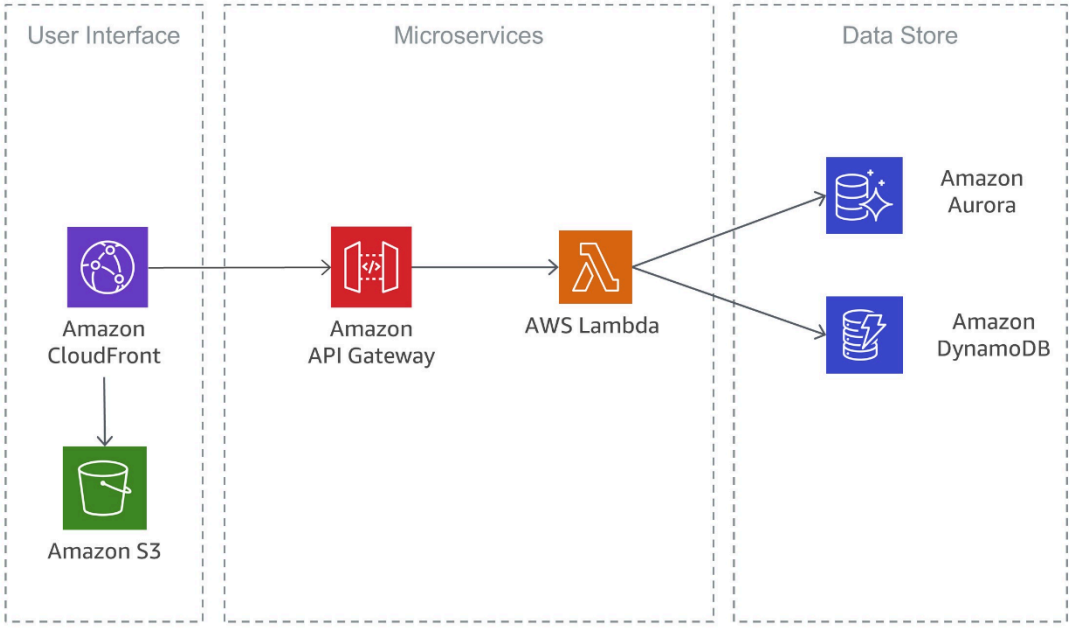
"Nirserver lebih mudah dikelola daripada tidak ada server sama sekali".

Menyingkirkan server adalah cara yang bagus untuk menghilangkan kompleksitas operasional.

Lambda terintegrasi secara erat dengan API Gateway. Kemampuan untuk melakukan panggilan sinkron dari API Gateway ke Lambda memungkinkan pembuatan aplikasi yang sepenuhnya nirserver dan dijelaskan secara mendetail dalam Panduan Developer Amazon API Gateway.

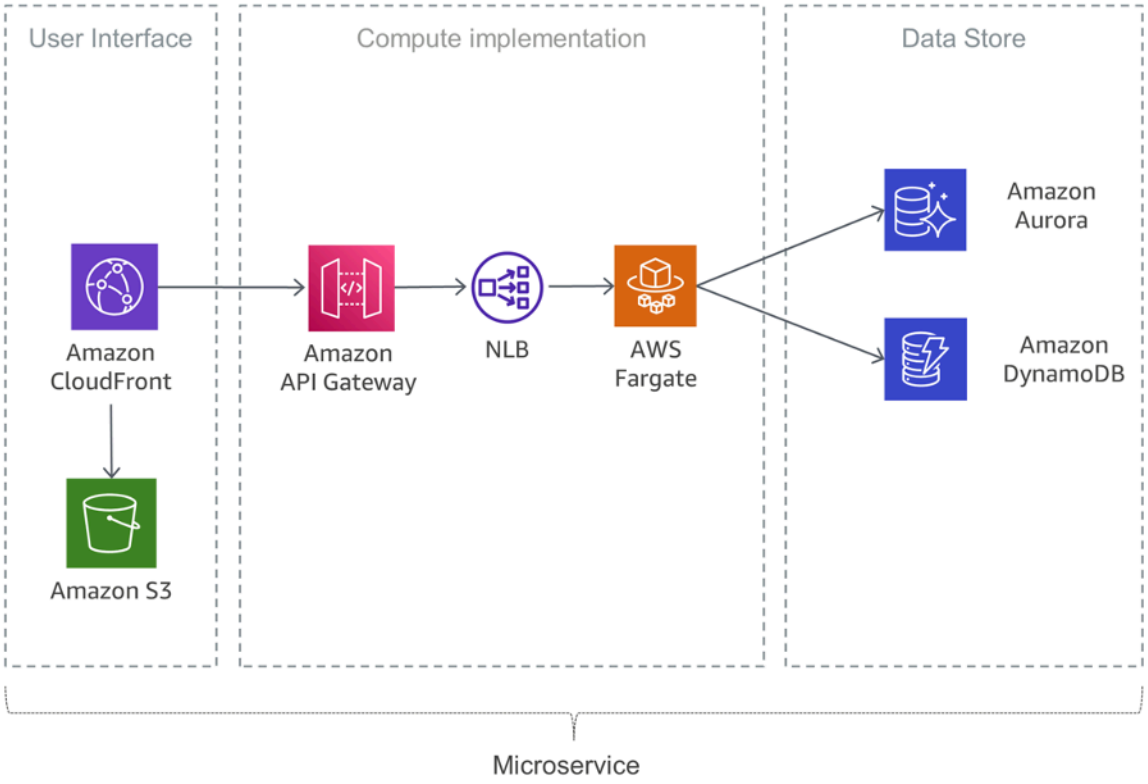
Gambar berikut menampilkan arsitektur layanan mikro nirserver dengan AWS Lambda tempat layanan lengkap dibangun dari layanan terkelola, sehingga menghilangkan beban arsitektur yang

dirancang untuk ketersediaan tinggi dan skalabilitas, serta menghilangkan upaya operasional dalam menjalankan dan memantau infrastruktur yang mendasari layanan mikro.



Layanan mikro nirserver menggunakan AWS Lambda

Salah satu penerapan serupa yang juga didasarkan pada layanan nirserver ditunjukkan pada gambar berikut. Dalam arsitektur ini, kontainer Docker digunakan dengan Fargate, sehingga tidak perlu memikirkan infrastruktur yang mendasarinya. Selain DynamoDB, Amazon Aurora Serverless juga digunakan, yaitu konfigurasi penskalaan otomatis sesuai permintaan untuk Amazon Aurora (edisi yang kompatibel dengan MySQL), tempat basis data akan otomatis aktif, nonaktif, dan menaikkan atau menurunkan skala kapasitas sesuai kebutuhan aplikasi Anda.



Layanan mikro nirserver menggunakan Fargate

Pemulihan bencana

Seperti disebutkan sebelumnya dalam pengantar di laporan resmi ini, aplikasi layanan mikro yang biasa akan diterapkan menggunakan pola Aplikasi Dua Belas Faktor. Bagian Proses menyatakan bahwa “Proses dua belas faktor bersifat stateless dan share-nothing. Setiap data yang perlu dipersistensi harus disimpan dalam layanan stateful backing, biasanya basis data.”

Untuk arsitektur layanan mikro yang biasa, artinya fokus utama untuk pemulihan bencana harus diberikan pada layanan downstream yang mempertahankan status aplikasi. Misalnya, hal ini dapat berupa sistem file, basis data, atau antrean. Saat membuat strategi pemulihan bencana, organisasi paling sering merencanakan tujuan waktu pemulihan dan tujuan titik pemulihan.

Tujuan waktu pemulihan adalah penundaan maksimum yang dapat diterima antara gangguan layanan dan pemulihan layanan. Tujuan ini ditetapkan oleh organisasi dan menentukan hal yang dianggap sebagai periode waktu yang dapat diterima ketika layanan tidak tersedia.

Tujuan titik pemulihan adalah jumlah waktu maksimum yang dapat diterima sejak titik pemulihan data terakhir. Tujuan ini ditetapkan oleh organisasi dan menentukan hal yang dianggap sebagai kehilangan data yang dapat diterima antara titik pemulihan terakhir dan gangguan layanan.

Untuk informasi selengkapnya, lihat laporan resmi Pemulihan Bencana Beban Kerja di AWS: Pemulihan di Cloud.

Ketersediaan tinggi

Bagian ini mengamati lebih dekat ketersediaan tinggi untuk opsi komputasi yang berbeda-beda.

Amazon EKS menjalankan instans bidang kendali dan bidang data Kubernetes di beberapa Zona Ketersediaan untuk memastikan ketersediaan tinggi. Amazon EKS secara otomatis mendeteksi dan mengganti instans bidang kendali yang tidak berkondisi baik, serta menyediakan upgrade dan patching otomatis untuk instans bidang kendali tersebut. Bidang kendali ini terdiri dari minimal dua node server API dan tiga node etcd yang berjalan di tiga Zona Ketersediaan dalam sebuah wilayah. Amazon EKS menggunakan arsitektur Wilayah AWS untuk menjaga ketersediaan tinggi.

Amazon ECR meng-host image dalam arsitektur yang berketersediaan tinggi dan berperforma tinggi, sehingga memungkinkan Anda melakukan deployment image secara andal untuk aplikasi kontainer di seluruh Zona Ketersediaan. Amazon ECR beroperasi dengan Amazon EKS, Amazon ECS, dan AWS Lambda, sehingga menyederhanakan pengembangan ke alur kerja produksi.

Amazon ECS adalah layanan regional yang menyederhanakan proses dalam menjalankan kontainer dengan ketersediaan tinggi di beberapa Zona Ketersediaan dalam sebuah Wilayah AWS. Amazon ECS menyertakan beberapa strategi penjadwalan yang menempatkan kontainer di seluruh klaster Anda berdasarkan kebutuhan sumber daya Anda (misalnya, CPU atau RAM) dan persyaratan ketersediaan.

AWS Lambda menjalankan fungsi Anda di beberapa Zona Ketersediaan untuk memastikan fungsi tersebut tersedia untuk memproses kejadian jika terjadi gangguan layanan di satu zona. Jika Anda mengonfigurasikan fungsi untuk terhubung ke Virtual Private Cloud (VPC) di akun Anda, tentukan subnet di beberapa Zona Ketersediaan untuk memastikan ketersediaan tinggi.

Melakukan deployment aplikasi berbasis Lambda

Anda dapat menggunakan AWS CloudFormation untuk menentukan, melakukan deployment, dan mengonfigurasikan aplikasi nirserver.

AWS Serverless Application Model (AWS SAM) adalah cara yang mudah untuk menentukan aplikasi nirserver. AWS SAM didukung secara native oleh CloudFormation dan menentukan sintaks yang disederhanakan untuk menyatakan sumber daya nirserver. Untuk melakukan deployment aplikasi

Anda, tentukan sumber daya yang dibutuhkan sebagai bagian dari aplikasi Anda, bersama dengan kebijakan izin terkait dalam templat CloudFormation, paketkan artefak deployment Anda, lalu lakukan deployment templat tersebut. Berdasarkan AWS SAM, SAM Local adalah alat AWS Command Line Interface (AWS CLI) yang menyediakan lingkungan bagi Anda untuk mengembangkan, menguji, dan menganalisis aplikasi nirserver secara lokal sebelum mengunggahnya ke runtime Lambda. Anda dapat menggunakan AWS SAM Local untuk membuat lingkungan pengujian lokal yang mensimulasikan lingkungan runtime AWS.

Komponen sistem terdistribusi

Setelah melihat bagaimana AWS dapat memecahkan tantangan yang terkait dengan setiap layanan mikro, fokusnya akan tertuju pada tantangan lintas layanan, seperti penemuan layanan, konsistensi data, komunikasi asinkron, serta pemantauan dan audit terdistribusi.

Topik

Penemuan layanan

Salah satu tantangan utama dengan arsitektur layanan mikro adalah memungkinkan layanan menemukan dan berinteraksi satu sama lain. Karakteristik arsitektur layanan mikro yang didistribusikan tidak hanya mempersulit layanan untuk berkomunikasi, tetapi juga menghadirkan tantangan lain, seperti memeriksa kondisi sistem tersebut dan mengumumkan kapan aplikasi baru tersedia. Anda juga harus memutuskan bagaimana dan di mana menyimpan informasi meta-store, seperti data konfigurasi, yang dapat digunakan oleh aplikasi. Pada bagian ini, beberapa teknik untuk melakukan penemuan layanan di AWS untuk arsitektur berbasis layanan mikro akan dibahas.

Penemuan layanan berbasis DNS

Amazon ECS sekarang dilengkapi penemuan layanan terintegrasi yang memudahkan layanan terkontainerisasi Anda untuk menemukan dan terhubung satu sama lain.

Sebelumnya, untuk memastikan layanan dapat menemukan dan terhubung satu sama lain, Anda harus mengonfigurasikan dan menjalankan sistem penemuan layanan Anda sendiri berdasarkan Amazon Route 53, AWS Lambda, dan ECS Event Stream, atau menghubungkan setiap layanan ke penyeimbang beban.

Amazon ECS membuat dan mengelola registri nama layanan menggunakan API Route 53 Auto Naming. Nama secara otomatis dipetakan ke satu set catatan DNS sehingga Anda dapat merujuk

ke layanan berdasarkan nama dalam kode Anda dan menulis kueri DNS untuk melakukan resolving nama menjadi titik akhir layanan pada saat runtime. Anda dapat menetapkan ketentuan pemeriksaan kondisi dalam definisi tugas layanan lalu Amazon ECS akan memastikan bahwa hanya titik akhir layanan berkondisi baik yang dikembalikan oleh pencarian layanan.

Selain itu, Anda juga dapat menggunakan penemuan layanan terpadu untuk layanan yang dikelola oleh Kubernetes. Untuk memungkinkan integrasi ini, AWS telah berkontribusi pada proyek External DNS, yaitu sebuah proyek inkubator Kubernetes.

Opsi lainnya adalah menggunakan kemampuan AWS Cloud Map. AWS Cloud Map memperluas kemampuan API Auto Naming dengan menyediakan registri layanan untuk sumber daya, seperti Protokol Internet (IP), Lokator Sumber Seragam (URL), dan Nama Sumber Daya Amazon (ARN), serta menawarkan mekanisme penemuan layanan berbasis API dengan propagasi perubahan yang lebih cepat dan kemampuan untuk menggunakan atribut guna mempersempit set sumber daya yang ditemukan. Sumber daya Auto Naming Route 53 Penamaan yang ada akan otomatis di-upgrade ke AWS Cloud Map.

Perangkat lunak pihak ketiga

Salah satu pendekatan yang berbeda untuk menerapkan penemuan layanan adalah menggunakan perangkat lunak pihak ketiga, seperti HashiCorp Consul, etcd, atau Netflix Eureka. Ketiga contoh ini adalah penyimpanan nilai-kunci terdistribusi yang andal. Untuk HashiCorp Consul, ada AWS Quick Start yang menyiapkan lingkungan AWS Cloud yang fleksibel dan dapat diskalakan serta meluncurkan HashiCorp Consul secara otomatis ke dalam konfigurasi pilihan Anda.

Mesh layanan

Dalam arsitektur layanan mikro lanjutan, aplikasi sebenarnya dapat terdiri dari ratusan, atau bahkan ribuan, layanan. Sering kali bagian yang paling kompleks dari aplikasi bukanlah layanan itu sendiri, tetapi komunikasi di antara layanan tersebut. Mesh layanan adalah lapisan tambahan untuk menangani komunikasi antarlayanan, yang bertanggung jawab untuk memantau dan mengontrol lalu lintas dalam arsitektur layanan mikro. Hal ini memungkinkan tugas, seperti penemuan layanan, sepenuhnya ditangani oleh lapisan ini.

Biasanya, mesh layanan dibagi menjadi bidang data dan bidang kendali. Bidang data terdiri dari satu set proksi cerdas yang di-deploy dengan kode aplikasi sebagai proksi sidecar khusus yang mengintersepsi semua komunikasi jaringan di antara layanan mikro. Bidang kendali bertanggung jawab untuk berkomunikasi dengan proksi.

Mesh layanan bersifat transparan, yang berarti bahwa developer aplikasi tidak harus mengetahui lapisan tambahan ini dan tidak perlu membuat perubahan pada kode aplikasi yang ada. AWS App Mesh adalah mesh layanan yang memberikan jaringan tingkat aplikasi untuk memungkinkan layanan Anda berkomunikasi dengan satu sama lain di beberapa jenis infrastruktur komputasi. App Mesh menstandardisasikan cara layanan Anda berkomunikasi, sehingga memberikan visibilitas yang menyeluruh dan memastikan ketersediaan tinggi untuk aplikasi Anda.

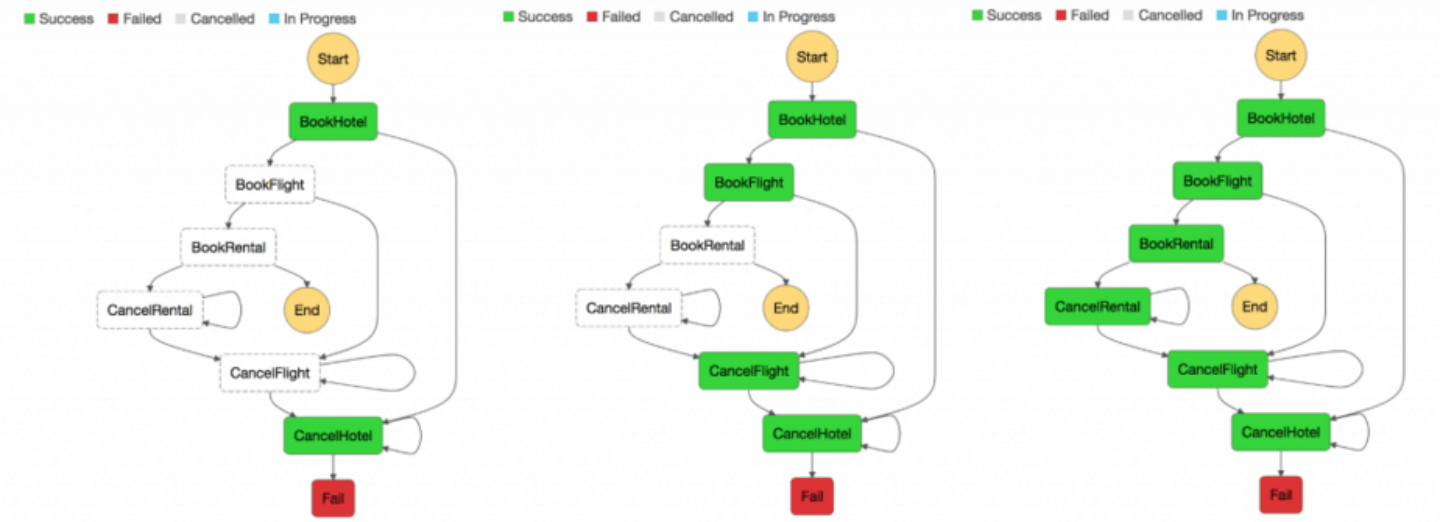
Anda dapat menggunakan layanan mikro AWS App Mesh yang sudah ada atau baru yang berjalan di AWS Fargate, Amazon ECS, Amazon EKS, dan Kubernetes yang dikelola sendiri di AWS. App Mesh dapat memantau dan mengontrol komunikasi untuk layanan mikro yang berjalan di klaster, sistem orkestrasi, atau VPC sebagai aplikasi tunggal tanpa perubahan kode.

Manajemen data terdistribusi

Aplikasi monolitik biasanya didukung oleh basis data relasional besar, yang mendefinisikan model data tunggal yang umum untuk semua komponen aplikasi. Dalam pendekatan layanan mikro, basis data sentral semacam itu akan mencegah sasaran membangun komponen yang terdesentralisasi dan independen. Setiap komponen layanan mikro harus memiliki lapisan persistensi data sendiri.

Namun, manajemen data terdistribusi akan menimbulkan tantangan baru. Sebagai akibat dari Teorema CAP, arsitektur layanan mikro yang terdistribusi secara inheren akan menimbulkan tarik ulur antara konsistensi dan performa serta memerlukan penerapan eventual consistency.

Dalam sistem terdistribusi, transaksi bisnis dapat mencakup beberapa layanan mikro. Karena layanan mikro tidak dapat menggunakan transaksi ACID tunggal, Anda dapat mengalami eksekusi parsial. Dalam hal ini, kita akan membutuhkan beberapa logika kontrol untuk mengulang transaksi yang sudah diproses. Untuk tujuan ini, pola Saga terdistribusi umumnya digunakan. Jika transaksi bisnis gagal, Saga akan mengatur serangkaian transaksi pengompensasi yang mengurungkan perubahan yang dilakukan oleh transaksi sebelumnya. AWS Step Functions memudahkan penerapan koordinator eksekusi Saga, seperti yang ditunjukkan pada gambar berikut.



Koordinator eksekusi Saga

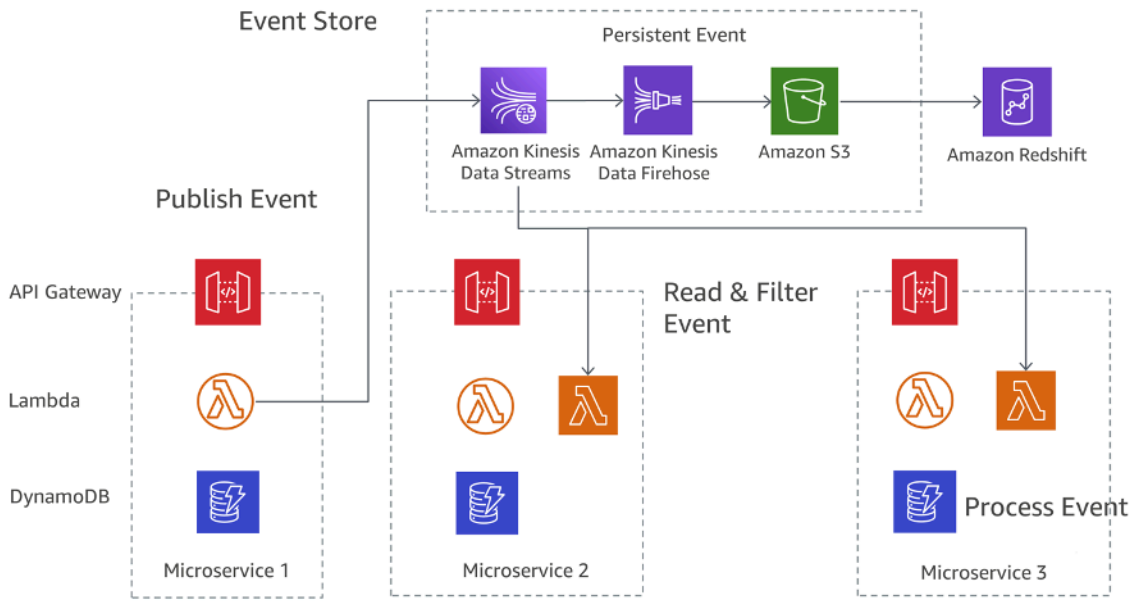
Dengan membangun penyimpanan terpusat data referensi penting yang dikurasi oleh alat dan prosedur manajemen data inti, akan tersedia sarana untuk layanan mikro guna menyinkronkan data penting dan mungkin juga status roll back-nya. Menggunakan Lambda dengan Amazon CloudWatch Events terjadwal, Anda dapat membuat mekanisme pembersihan dan deduplikasi sederhana.

Sangat umum bahwa perubahan status akan memengaruhi lebih dari satu layanan mikro. Dalam kasus tersebut, event sourcing telah terbukti menjadi pola yang berguna. Ide inti di balik event sourcing adalah untuk mempresentasikan dan melakukan persistensi setiap perubahan aplikasi sebagai catatan kejadian. Bukannya melakukan persistensi status aplikasi, data akan disimpan sebagai aliran kejadian. Sistem pencatatan log dan kontrol versi transaksi basis data adalah dua contoh umum untuk event sourcing. Event sourcing memiliki beberapa manfaat: status dapat ditentukan dan direkonstruksi untuk setiap titik waktu. Event sourcing secara otomatis menghasilkan jejak audit persisten dan juga memfasilitasi debugging.

Dalam konteks arsitektur layanan mikro, event sourcing memungkinkan pemisahan (decoupling) berbagai bagian aplikasi menggunakan pola publish/subscribe, dan event sourcing akan mengumpankan data kejadian yang sama ke dalam model data yang berbeda-beda untuk layanan mikro terpisah. Event sourcing sering digunakan bersamaan dengan pola Command Query Responsibility Secregation (CQRS) untuk melakukan decoupling beban kerja baca dari beban kerja tulis serta mengoptimalkan performa, skalabilitas, dan keamanan. Dalam sistem manajemen data tradisional, perintah dan kueri dijalankan berdasarkan repositori data yang sama.

Gambar berikut menampilkan bagaimana pola event sourcing dapat diterapkan di AWS. Amazon Kinesis Data Streams berfungsi sebagai komponen utama penyimpanan kejadian pusat, yang

menangkap perubahan aplikasi sebagai kejadian dan melakukan persistensi kejadian tersebut di Amazon S3. Gambar tersebut menampilkan tiga layanan mikro berbeda, yang terdiri dari Amazon API Gateway AWS Lambda, dan Amazon DynamoDB. Tanda panah menunjukkan alur kejadian: ketika Layanan mikro 1 mengalami perubahan status kejadian, layanan mikro ini memublikasikan sebuah kejadian dengan menulis pesan ke Kinesis Data Streams. Semua layanan mikro menjalankan aplikasi Kinesis Data Streams-nya sendiri di AWS Lambda yang membaca salinan pesan, memfilternya berdasarkan relevansi untuk layanan mikro, dan mungkin meneruskannya untuk diproses lebih lanjut. Jika fungsi Anda mengembalikan kesalahan, Lambda akan mencoba ulang batch tersebut sampai pemrosesannya berhasil atau datanya kedaluwarsa. Agar menghindari kemacetan shard, Anda dapat mengonfigurasikan pemetaan event sourcing untuk mencoba lagi dengan ukuran batch yang lebih kecil, membatasi jumlah percobaan ulang, atau membuang catatan yang sudah lama. Untuk mempertahankan kejadian yang dibuang, Anda dapat mengonfigurasikan pemetaan event sourcing agar mengirimkan detail tentang batch yang gagal ke antrean Amazon Simple Queue Service (Amazon SQS) atau topik Amazon Simple Notification Service (Amazon SNS).



Pola event sourcing di AWS

Amazon S3 menyimpan semua kejadian secara tahan lama di semua layanan mikro dan merupakan sumber kebenaran tunggal dalam hal debugging, memulihkan status aplikasi, atau mengaudit perubahan aplikasi. Ada dua alasan utama mengapa catatan dapat dikirimkan lebih dari satu kali ke aplikasi Kinesis Data Streams Anda: percobaan ulang produsen dan percobaan ulang konsumen.

Aplikasi Anda harus mengantisipasi dan menangani pemrosesan setiap catatan dengan tepat beberapa kali.

Manajemen konfigurasi

Dalam arsitektur layanan mikro yang biasa dengan puluhan layanan yang berbeda, setiap layanan membutuhkan akses ke beberapa layanan downstream dan komponen infrastruktur yang mengekspos data ke layanan. Contohnya dapat berupa antrean pesan, basis data, dan layanan mikro lainnya. Salah satu tantangan utamanya adalah mengonfigurasikan setiap layanan secara konsisten untuk memberikan informasi tentang koneksi ke layanan downstream dan infrastruktur. Selain itu, konfigurasinya juga harus berisi informasi tentang lingkungan tempat layanan beroperasi, dan pengaktifan ulang aplikasi untuk menggunakan data konfigurasi baru seharusnya tidak diperlukan.

Prinsip ketiga dari pola Aplikasi Dua Belas Faktor mencakup topik ini: “Aplikasi dua belas faktor menyimpan konfigurasi dalam variabel lingkungan (environment) (sering disingkat menjadi env vars atau env).” Untuk Amazon ECS, variabel lingkungan dapat diteruskan ke kontainer menggunakan parameter ketentuan kontainer lingkungan yang dipetakan ke opsi `--env` untuk docker run. Variabel lingkungan dapat diteruskan ke kontainer Anda secara massal menggunakan parameter ketentuan kontainer `environmentFiles` untuk mencantumkan satu atau beberapa file yang berisi variabel lingkungan. File ini harus di-host di Amazon S3. Di AWS Lambda, runtime menyediakan variabel lingkungan untuk kode Anda dan menetapkan variabel lingkungan tambahan yang berisi informasi tentang permintaan fungsi dan pemanggilan. Untuk Amazon EKS, Anda dapat menentukan variabel lingkungan di bidang env dalam manifes konfigurasi untuk pod yang sesuai. Salah satu cara yang berbeda untuk menggunakan variabel env adalah menggunakan ConfigMap.

Komunikasi asinkron dan olahpesan ringan

Komunikasi dalam aplikasi tradisional dan monolitik bersifat sederhana — satu bagian dari aplikasi menggunakan panggilan metode atau mekanisme distribusi kejadian internal untuk berkomunikasi dengan bagian lain. Jika aplikasi yang sama diterapkan menggunakan layanan mikro yang di-decoupling, komunikasi antara berbagai bagian aplikasi harus diterapkan menggunakan komunikasi jaringan.

Komunikasi berbasis REST

Protokol HTTP/S adalah cara paling populer untuk menerapkan komunikasi sinkron di antara layanan mikro. Dalam kebanyakan kasus, API RESTful menggunakan HTTP sebagai lapisan transport. Gaya arsitektur REST bergantung pada komunikasi stateless, antarmuka seragam, dan metode standar.

Dengan API Gateway, Anda dapat membuat API yang berfungsi sebagai pintu depan aplikasi untuk mengakses data, logika bisnis, atau fungsionalitas dari layanan backend Anda. Developer API dapat membuat API yang mengakses AWS atau layanan web lainnya, serta data yang disimpan di AWS Cloud. Objek API yang didefinisikan dengan layanan API Gateway adalah grup sumber daya dan metode.

Sumber daya adalah typed object dalam domain API dan mungkin telah mengaitkan sebuah model data atau hubungan dengan sumber daya lainnya. Setiap sumber daya dapat dikonfigurasikan untuk merespons satu atau beberapa metode, yaitu kata kerja HTTP standar, seperti GET, POST, atau PUT. API REST dapat di-deploy ke tahap yang berbeda-beda, serta diberi versi dan diklon ke versi baru.

API Gateway menangani semua tugas yang diperlukan dalam penerimaan dan pemrosesan hingga ratusan ribu panggilan API secara bersamaan, termasuk pengelolaan lalu lintas, otorisasi dan kontrol akses, pemantauan, dan pengelolaan versi API.

Olahpesan asinkron dan penerusan kejadian

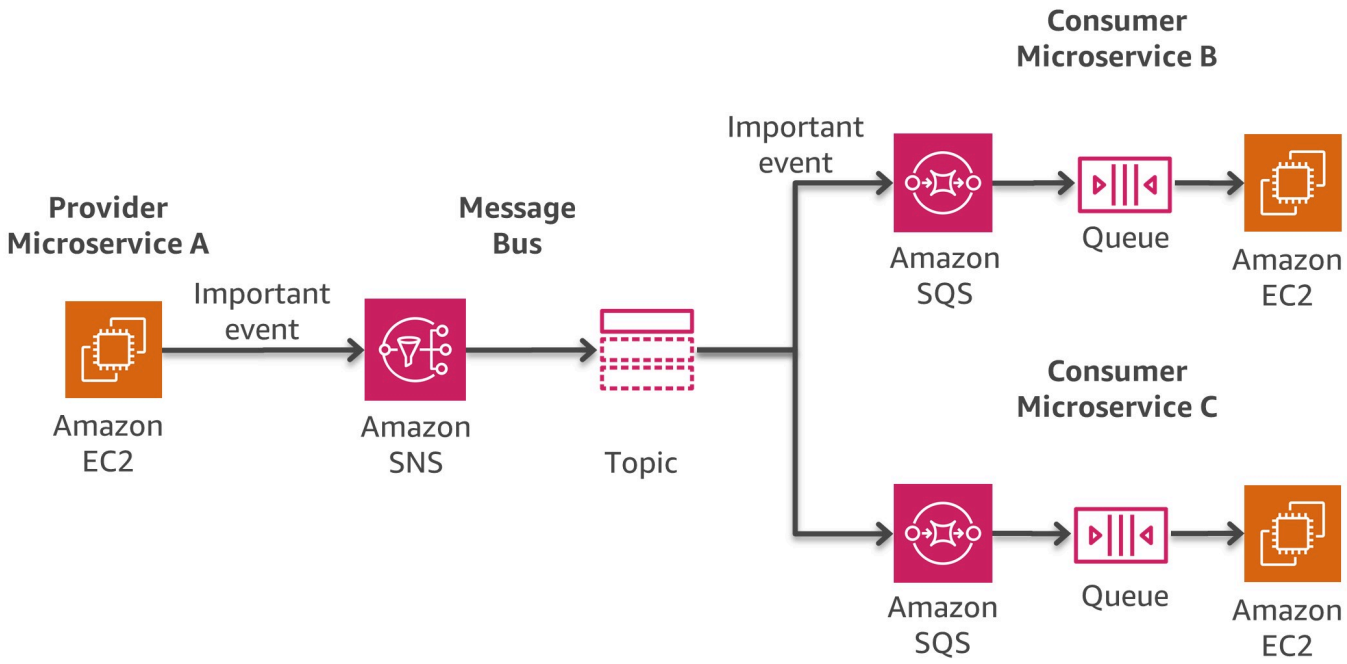
Penerusan pesan adalah pola tambahan yang digunakan untuk menerapkan komunikasi di antara layanan mikro. Layanan berkomunikasi dengan bertukar pesan melalui antrean. Salah satu manfaat utama dari gaya komunikasi ini adalah tidak memerlukan penemuan layanan dan layanannya tidak di-coupling secara erat.

Sistem sinkron memiliki coupling yang erat, yang berarti masalah dalam dependensi downstream sinkron akan memiliki dampak langsung pada pemanggil upstream. Percobaan ulang dari pemanggil upstream dapat menyebarkan dan memperbesar masalah dengan cepat.

Bergantung pada persyaratan tertentu, seperti protokol, AWS menawarkan berbagai layanan yang membantu menerapkan pola ini. Satu kemungkinan penerapan menggunakan kombinasi antrean Amazon Simple Queue Service (Amazon SQS) atau Amazon Simple Notification Service (Amazon SNS).

Kedua layanan ini beroperasi bersama secara erat: Amazon SNS memungkinkan aplikasi mengirimkan pesan ke beberapa pelanggan melalui mekanisme push. Dengan Amazon SNS dan

Amazon SQS bersama-sama, satu pesan dapat dikirimkan ke beberapa konsumen. Gambar berikut menampilkan integrasi Amazon SNS dan Amazon SQS.



Pola bus pesan di AWS

Jika Anda mengatur antrean Amazon SQS agar berlangganan topik SNS, Anda dapat memublikasikan pesan ke topik tersebut, dan Amazon SNS akan mengirimkan sebuah pesan ke antrean Amazon SQS yang berlangganan. Pesan ini berisi subjek dan pesan yang dipublikasikan ke topik bersama dengan informasi metadata dalam format JSON.

Opsi lain untuk membangun arsitektur berbasis kejadian dengan event sourcing yang mencakup aplikasi internal, aplikasi SaaS pihak ketiga, dan layanan AWS dalam skala besar adalah Amazon EventBridge. Sebagai layanan bus kejadian terkelola penuh, EventBridge akan menerima kejadian dari sumber yang berbeda-beda, mengidentifikasi target berdasarkan aturan perutean, dan mengirimkan data secara hampir waktu nyata ke target tersebut, termasuk AWS Lambda, Amazon SNS, Amazon Kinesis Streams, dll. Kejadian yang masuk juga dapat disesuaikan oleh input transformer sebelum pengiriman.

Untuk pengembangan aplikasi berbasis kejadian yang secara signifikan lebih cepat, registri skema EventBridge akan mengumpulkan dan mengatur skema, termasuk skema untuk semua kejadian yang dihasilkan oleh layanan AWS. Pelanggan juga dapat menentukan skema kustom atau

menggunakan opsi Infer Schema (Inferensikan Skema) untuk menemukan skema secara otomatis. Namun setelah mempertimbangkan segala aspeknya, semua fitur tersebut disertai dengan sebuah potensi kerugian, yaitu nilai latensi yang relatif lebih tinggi untuk pengiriman EventBridge. Selain itu, throughput dan kuota default untuk EventBridge mungkin memerlukan penambahan berdasarkan kasus penggunaannya, yang dapat dilakukan dengan mengajukan permintaan dukungan.

Sebuah strategi penerapan yang berbeda berdasarkan Amazon MQ dapat digunakan jika perangkat lunak yang ada menggunakan API dan protokol standar terbuka untuk olahpesan, termasuk JMS, NMS, AMQP, STOMP, MQTT, dan WebSocket. Amazon SQS akan mengekspos API kustom. Dengan demikian, jika Anda memiliki aplikasi yang sudah ada yang ingin dimigrasikan dari—misalnya, lingkungan on-premise ke AWS—maka perubahan kode diperlukan. Dengan Amazon MQ, dalam banyak kasus, hal ini tidak diperlukan.

Amazon MQ mengelola administrasi dan pemeliharaan ActiveMQ, broker pesan sumber terbuka yang populer. Infrastruktur yang mendasarinya secara otomatis disediakan untuk ketersediaan tinggi dan ketahanan pesan untuk mendukung keandalan aplikasi Anda.

Orkestrasi dan manajemen state

Karakter terdistribusi dari layanan mikro menyulitkan untuk mengatur alur kerja ketika beberapa layanan mikro diperlukan. Developer mungkin cenderung ingin menambahkan kode orkestrasi ke layanan mereka secara langsung. Namun, hal ini harus dihindari karena akan menimbulkan penggabungan (coupling) yang lebih erat dan mempersulit penggantian setiap layanan dengan cepat.

Anda dapat menggunakan AWS Step Functions untuk membangun aplikasi dari setiap komponen yang masing-masing melakukan fungsi diskret. Step Functions menyediakan mesin status yang menyembunyikan kompleksitas orkestrasi layanan, seperti penanganan kesalahan, serialisasi, dan paralelisasi. Hal ini memungkinkan Anda menskalakan dan mengubah aplikasi dengan cepat sambil menghindari kode koordinasi tambahan di dalam layanan.

Step Functions adalah cara yang dapat diandalkan untuk mengoordinasikan komponen dan melakukan step through fungsi aplikasi Anda. Step Functions menyediakan konsol grafis untuk mengatur dan memvisualisasikan komponen aplikasi Anda sebagai serangkaian langkah. Hal ini mempermudah untuk membangun dan menjalankan layanan terdistribusi.

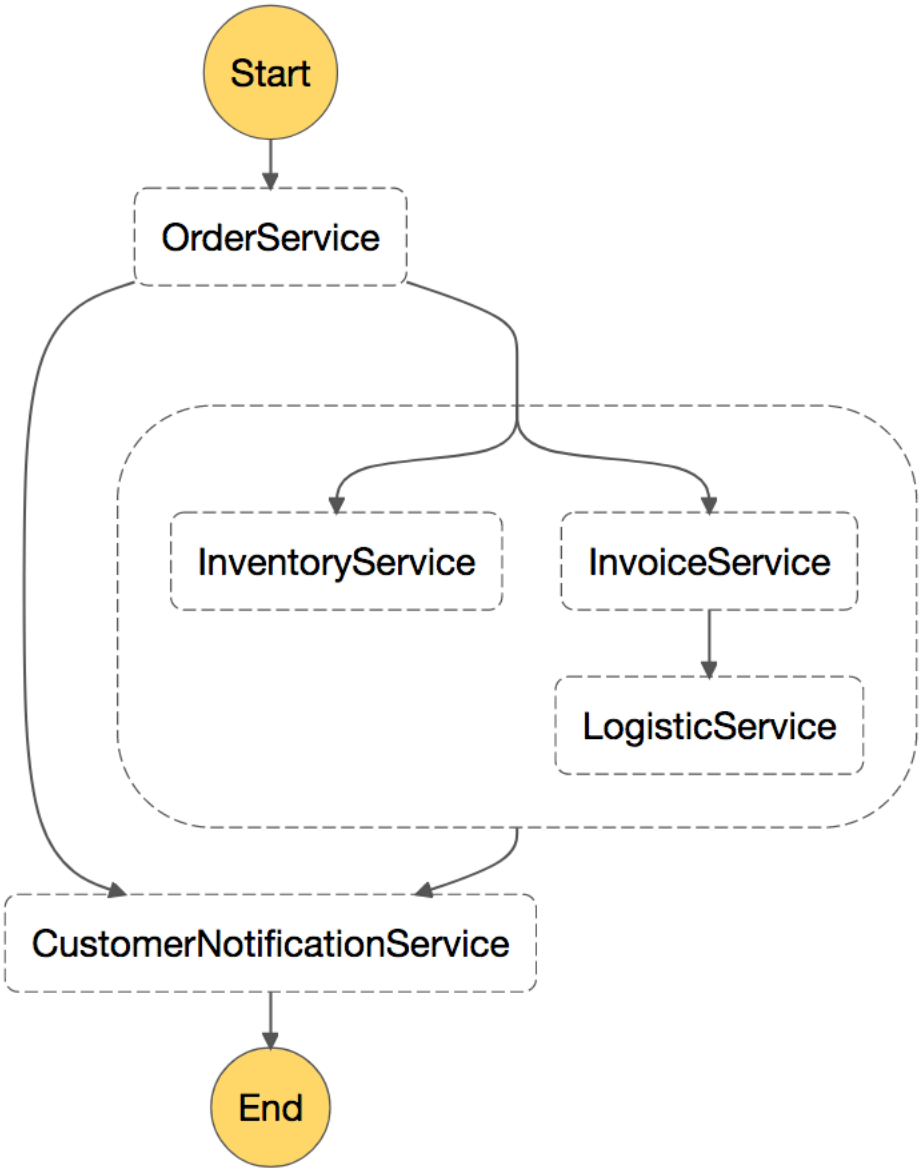
Step Functions secara otomatis memulai dan melacak setiap langkah, serta mencoba lagi ketika ada kesalahan, sehingga aplikasi dijalankan sesuai urutan dan seperti yang diharapkan. Step Functions mencatat status setiap langkah, jadi ketika ada yang salah, Anda dapat mendiagnosis dan menyelesaikan masalah dengan cepat. Anda dapat mengubah dan menambahkan langkah bahkan

tanpa menulis kode, sehingga Anda dapat mengembangkan aplikasi dengan mudah dan berinovasi dengan lebih cepat.

Step Functions adalah bagian dari AWS Serverless Platform serta mendukung orkestrasi fungsi dan aplikasi Lambda berdasarkan sumber daya komputasi, seperti Amazon EC2, Amazon EKS, Amazon ECS, dan layanan tambahan, seperti Amazon SageMaker dan AWS Glue. Step Functions mengelola operasi dan infrastruktur yang mendasar bagi Anda untuk membantu memastikan aplikasi Anda tersedia dalam skala apa pun.

Untuk membangun alur kerja, Step Functions menggunakan Amazon States Language. Alur kerja dapat berisi langkah-langkah berurutan atau paralel serta langkah branching.

Gambar berikut menampilkan contoh alur kerja untuk arsitektur layanan mikro yang menggabungkan langkah berurutan dan paralel. Pemanggilan alur kerja seperti itu dapat dilakukan baik melalui API Step Functions maupun dengan API Gateway.



Contoh alur kerja layanan mikro yang dipanggil oleh Step Functions

Pemantauan terdistribusi

Arsitektur layanan mikro terdiri dari berbagai bagian terdistribusi yang harus dipantau. Anda dapat menggunakan Amazon CloudWatch untuk mengumpulkan atau melacak metrik, memusatkan dan memantau file log, mengatur alarm, dan secara otomatis bereaksi terhadap perubahan di lingkungan AWS Anda. CloudWatch dapat memantau sumber daya AWS, seperti instans Amazon EC2, tabel DynamoDB, dan instans DB Amazon RDS, serta metrik kustom yang dibuat oleh aplikasi dan layanan, dan file log apa pun yang dibuat oleh aplikasi Anda.

Pemantauan

Anda dapat menggunakan CloudWatch untuk mendapatkan visibilitas tingkat sistem terkait penggunaan sumber daya, performa aplikasi, dan kondisi operasional. CloudWatch menyediakan solusi pemantauan yang andal, dapat diskalakan, dan fleksibel yang dapat mulai Anda gunakan dalam hitungan menit. Anda tidak perlu lagi menyiapkan, mengelola, serta menskalakan sistem dan infrastruktur pemantauan Anda sendiri. Dalam arsitektur layanan mikro, kemampuan memantau metrik kustom menggunakan CloudWatch merupakan manfaat tambahan karena developer dapat memutuskan metrik mana yang harus dikumpulkan untuk setiap layanan. Selain itu, penskalaan dinamis dapat diterapkan berdasarkan metrik kustom.

Selain Amazon CloudWatch, Anda juga dapat menggunakan CloudWatch Container Insights untuk mengumpulkan, menggabungkan, serta meringkas metrik dan log dari aplikasi dan layanan mikro terkontainerisasi Anda. CloudWatch Container Insights secara otomatis mengumpulkan metrik untuk banyak sumber daya, seperti CPU, memori, disk, serta jaringan dan agregat sebagai metrik CloudWatch di tingkat klaster, node, pod, tugas, dan layanan. Dengan CloudWatch Container Insights, Anda dapat memperoleh akses ke metrik dasbor CloudWatch Container Insights. Layanan ini juga menyediakan informasi diagnostik, seperti kegagalan pengaktifan ulang kontainer, untuk membantu Anda mengisolasi masalah dan menyelesaikannya dengan cepat. Anda juga dapat mengatur alarm CloudWatch pada metrik yang dikumpulkan oleh Container Insights.

Container Insights tersedia untuk platform Amazon ECS, Amazon EKS, dan Kubernetes di Amazon EC2. Dukungan Amazon ECS mencakup dukungan untuk Fargate.

Opsi lain yang populer, terutama untuk Amazon EKS, adalah menggunakan Prometheus. Prometheus adalah kit alat pemantauan dan pemberitahuan sumber terbuka yang sering digunakan dalam kombinasi dengan Grafana untuk memvisualisasikan metrik yang dikumpulkan. Banyak komponen Kubernetes menyimpan metrik di `/metrics` dan Prometheus dapat melakukan scraping terhadap metrik ini secara berkala.

Amazon Managed Service for Prometheus (AMP) adalah layanan pemantauan yang kompatibel dengan Prometheus yang memudahkan pemantauan aplikasi terkontainerisasi dalam skala besar. Dengan AMP, Anda dapat menggunakan bahasa kueri Prometheus (PromQL) sumber terbuka untuk memantau performa beban kerja terkontainerisasi tanpa harus mengelola infrastruktur dasar yang diperlukan untuk mengelola penyerapan, penyimpanan, dan pengkuerian metrik operasional. Anda dapat mengumpulkan metrik Prometheus dari lingkungan Amazon EKS dan Amazon ECS, menggunakan server AWS Distro for OpenTelemetry atau Prometheus sebagai agen pengumpulan.

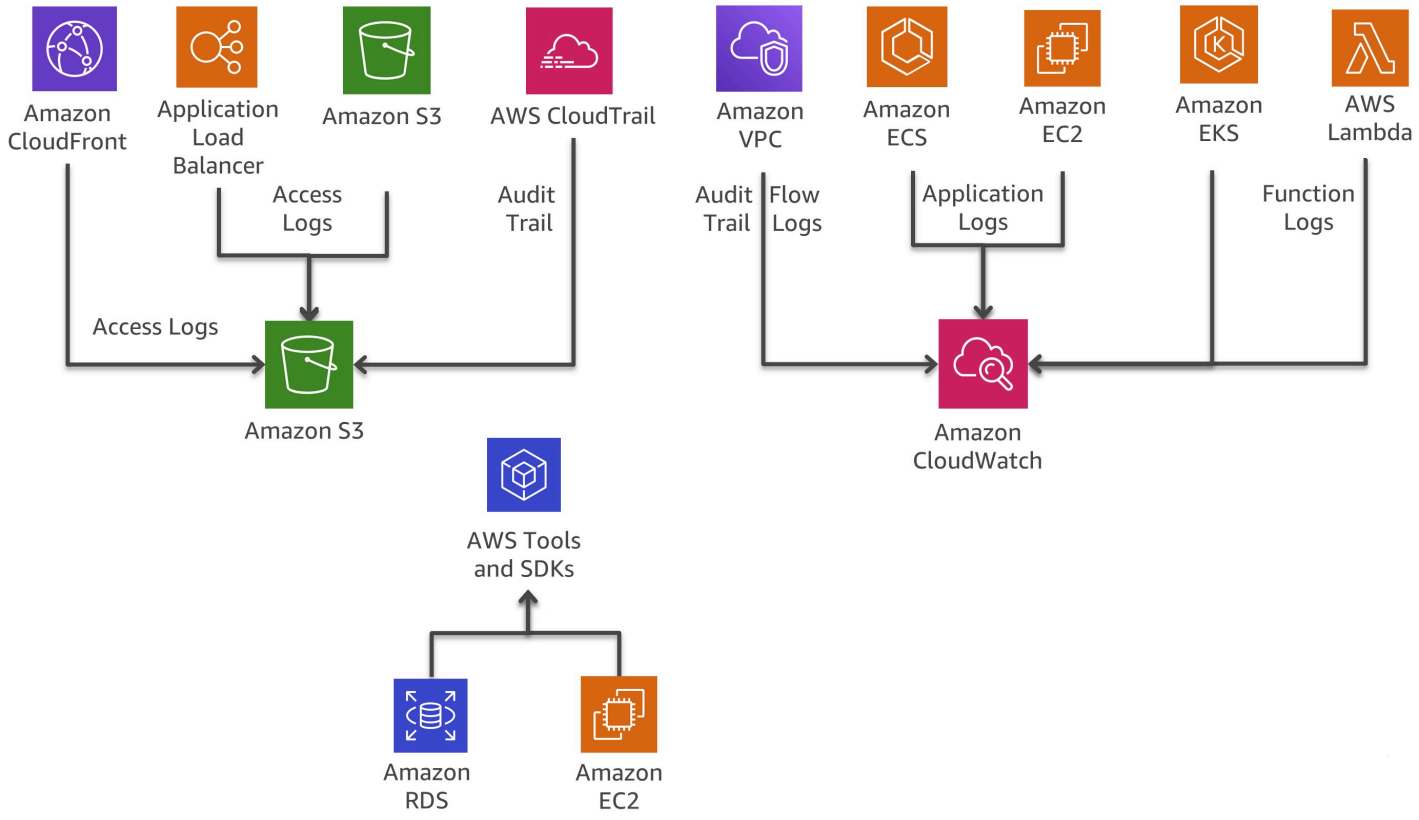
AMP sering digunakan dalam kombinasi dengan Amazon Managed Service for Grafana (AMG). AMG memudahkan pengkuerian, visualisasi, pemberitahuan, dan pemahaman terhadap metrik Anda di mana pun metrik tersebut disimpan. Dengan AMG, Anda dapat menganalisis metrik, log, dan jejak tanpa harus menyediakan server, mengonfigurasikan dan memperbarui perangkat lunak, atau melakukan pekerjaan berat yang diperlukan untuk mengamankan dan menskalakan Grafana dalam produksi.

Memusatkan log

Pencatatan log yang konsisten sangat penting untuk mengatasi dan mengidentifikasi masalah. Layanan mikro memungkinkan tim mengirimkan lebih banyak rilis daripada sebelumnya dan mendorong tim rekayasa untuk menjalankan eksperimen pada fitur baru dalam produksi. Pemahaman dampak bagi pelanggan sangat penting untuk meningkatkan sebuah aplikasi secara bertahap.

Secara default, sebagian besar layanan AWS memusatkan file log-nya. Tujuan utama untuk file log di AWS adalah Amazon S3 dan Amazon CloudWatch Logs. Untuk aplikasi yang berjalan di instans Amazon EC2, daemon tersedia untuk mengirimkan file log ke CloudWatch Logs. Fungsi Lambda secara native mengirimkan output log-nya ke CloudWatch Logs dan Amazon ECS menyertakan dukungan untuk driver log awslogs yang memungkinkan pemusatan log kontainer ke CloudWatch Logs. Untuk Amazon EKS, Fluent Bit atau Fluentd dapat meneruskan log dari setiap instans dalam klaster ke pencatatan log terpusat CloudWatch Logs tempat log tersebut digabungkan untuk pelaporan tingkat lebih tinggi menggunakan Amazon OpenSearch Service dan Kibana. Karena ukuran yang lebih kecil dan keunggulan performa-nya, Fluent Bit lebih direkomendasikan dibandingkan dengan FluentD.

Gambar berikut menampilkan kemampuan pencatatan log beberapa layanan. Tim kemudian dapat mencari dan menganalisis log ini menggunakan alat, seperti Amazon OpenSearch Service dan Kibana. Amazon Athena dapat digunakan untuk menjalankan kueri satu kali terhadap file log terpusat di Amazon S3.



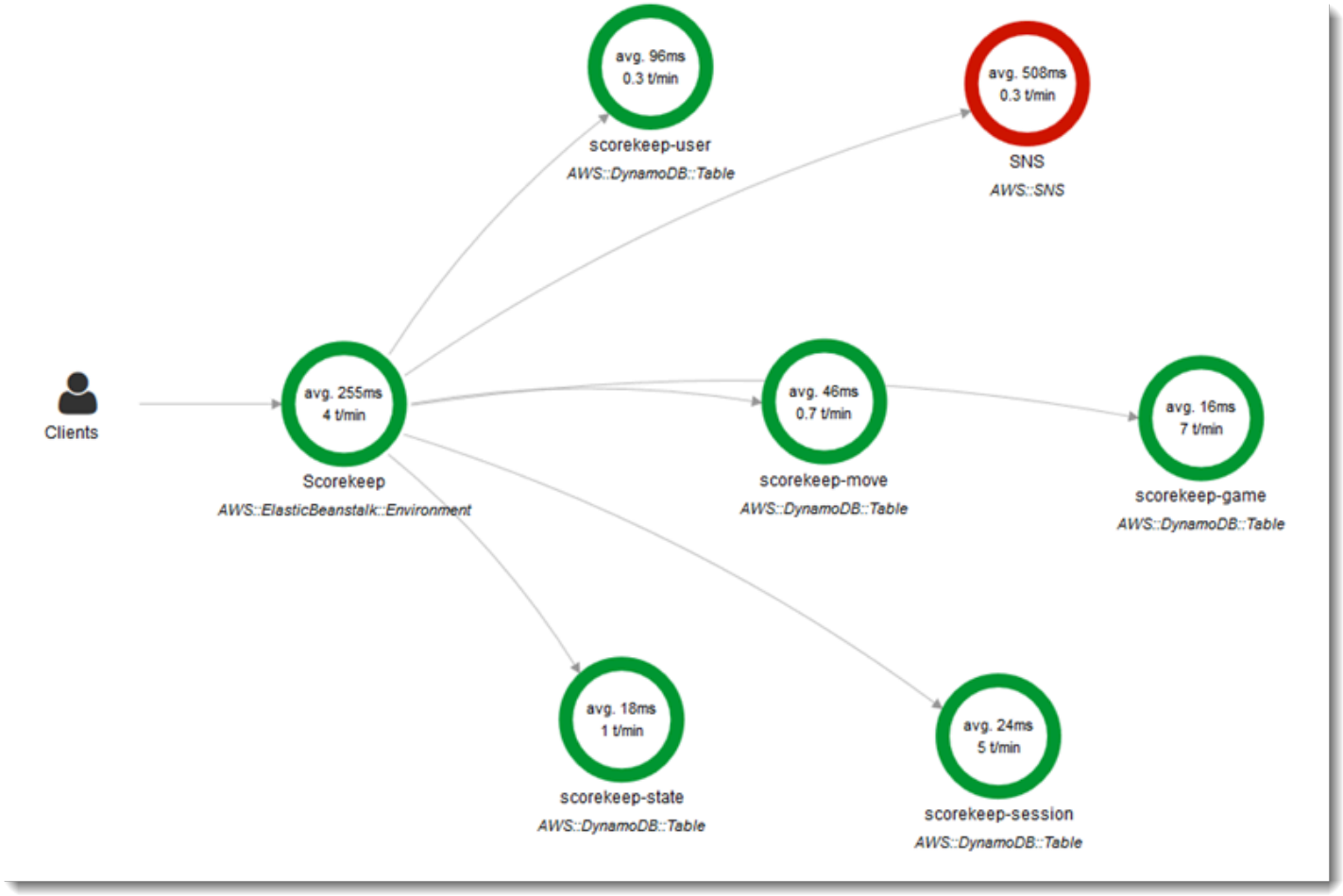
Kemampuan pencatatan log layanan AWS

Penelusuran terdistribusi

Dalam banyak kasus, satu set layanan mikro beroperasi bersama untuk menangani sebuah permintaan. Bayangkan sebuah sistem kompleks yang terdiri dari puluhan layanan mikro tempat sebuah kesalahan terjadi di salah satu layanan dalam rantai panggilan. Meskipun setiap layanan mikro mencatat log dengan benar dan log dikonsolidasikan dalam sistem pusat, mungkin sulit untuk menemukan semua pesan log yang relevan.

Gagasan utama AWS X-Ray adalah penggunaan ID korelasi, yang merupakan pengidentifikasi unik yang dilampirkan ke semua permintaan dan pesan yang terkait dengan rantai kejadian tertentu. ID jejak akan ditambahkan ke permintaan HTTP di header penelusuran tertentu bernama `X-Amzn-Trace-Id` saat permintaan diterima di layanan terintegrasi X-Ray pertama (misalnya, Application Load Balancer atau API Gateway) dan disertakan dalam respons. Melalui SDK X-Ray, setiap layanan mikro dapat membaca, dan juga dapat menambahkan atau memperbarui header ini.

X-Ray beroperasi dengan Amazon EC2, Amazon ECS, Lambda, dan AWS Elastic Beanstalk. Anda dapat menggunakan X-Ray dengan aplikasi yang tertulis dalam Java, Node.js, dan .NET yang di-deploy pada layanan ini.



Peta layanan AWS X-Ray

Epsagon adalah SaaS terkelola penuh yang mencakup penelusuran untuk semua layanan AWS, API pihak ketiga (melalui panggilan HTTP), serta layanan umum lainnya, seperti Redis, Kafka, dan Elastic. Layanan Epsagon mencakup kemampuan pemantauan, pemberitahuan ke beberapa layanan yang paling umum, dan visibilitas muatan di setiap panggilan yang dibuat oleh kode Anda.

AWS Distro for OpenTelemetry adalah distribusi proyek OpenTelemetry yang aman, siap produksi, dan didukung AWS. Sebagai bagian dari Cloud Native Computing Foundation, AWS Distro for OpenTelemetry menyediakan API, pustaka, dan agen sumber terbuka guna mengumpulkan jejak dan metrik terdistribusi untuk pemantauan aplikasi. Dengan AWS Distro for OpenTelemetry, Anda dapat menginstrumentasikan aplikasi Anda sekali saja untuk mengirimkan metrik dan jejak yang berkorelasi ke beberapa solusi pemantauan AWS dan partner. Gunakan agen instrumentasi otomatis

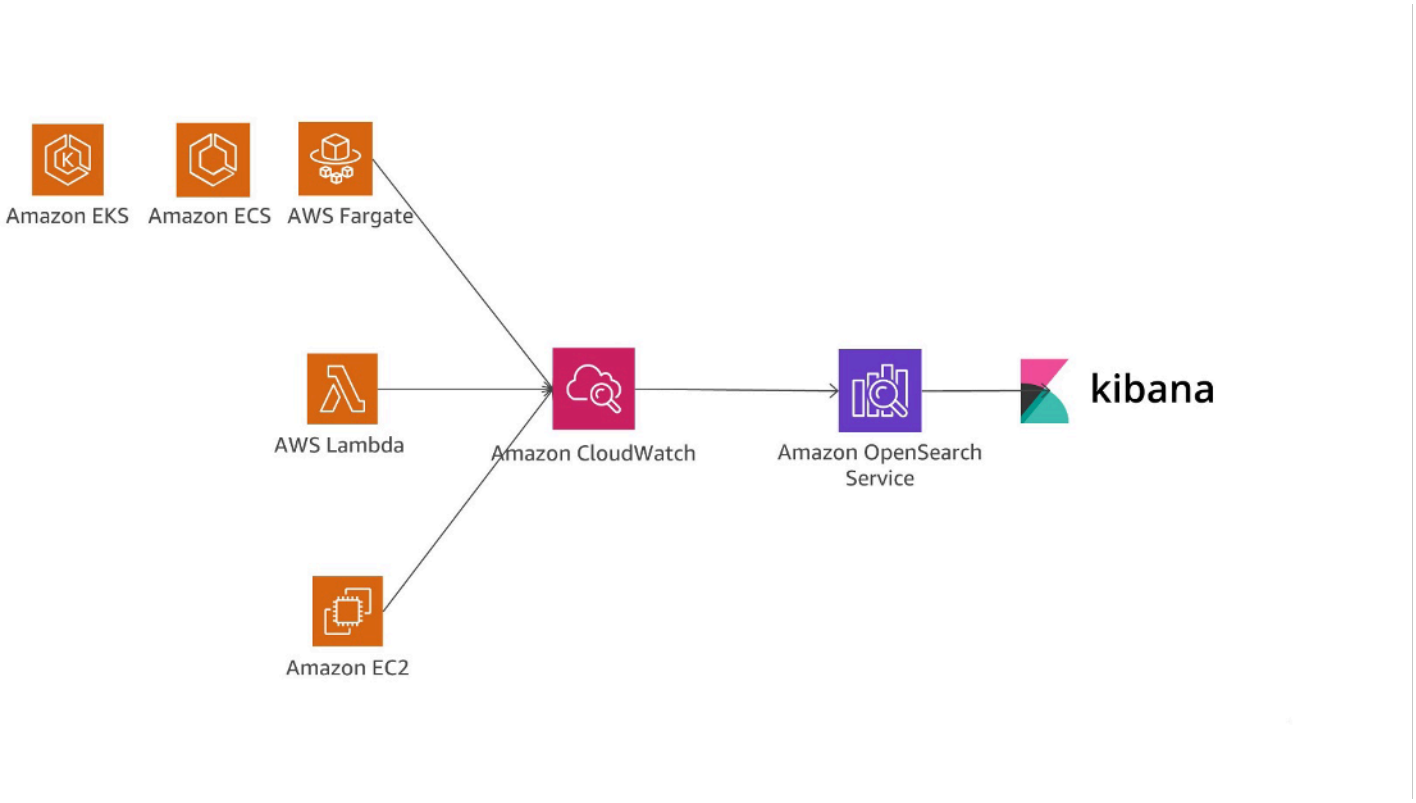
untuk mengumpulkan jejak tanpa mengubah kode Anda. AWS Distro for OpenTelemetry juga mengumpulkan metadata dari sumber daya dan layanan terkelola AWS untuk mengorelasikan data performa aplikasi dengan data infrastruktur yang mendasarinya, sehingga mengurangi waktu rata-rata untuk penyelesaian masalah. Gunakan AWS Distro for OpenTelemetry untuk menginstrumentasikan aplikasi Anda yang berjalan di Amazon EC2, Amazon ECS, Amazon EKS di Amazon EC2, Fargate, dan AWS Lambda, serta on-premise.

Opsi untuk analisis log di AWS

Mencari, menganalisis, dan memvisualisasikan data log merupakan aspek penting dalam memahami sistem terdistribusi. Amazon CloudWatch Logs Insights memungkinkan Anda menjelajahi, menganalisis, dan memvisualisasikan log secara instan. Hal ini memungkinkan Anda memecahkan masalah operasional. Sebuah opsi lain untuk menganalisis file log adalah menggunakan Amazon OpenSearch Service bersama dengan Kibana.

Amazon OpenSearch Service dapat digunakan untuk pencarian teks lengkap, pencarian terstruktur, analitik, dan ketiganya dalam kombinasi. Kibana adalah plugin visualisasi data sumber terbuka yang terintegrasi sempurna dengan Amazon OpenSearch Service.

Gambar berikut menampilkan analisis log dengan Amazon OpenSearch Service dan Kibana. CloudWatch Logs dapat dikonfigurasikan untuk melakukan streaming entri log ke Amazon OpenSearch Service secara waktu nyata melalui langganan CloudWatch Logs. Kibana memvisualisasikan data dan mengekspos antarmuka pencarian yang mudah digunakan untuk menyimpan data di Amazon OpenSearch Service. Solusi ini dapat digunakan dalam kombinasi dengan perangkat lunak seperti ElastAlert untuk menerapkan sistem pemberitahuan untuk mengirimkan notifikasi dan email SNS, membuat tiket JIRA, dan sebagainya, jika anomali, lonjakan, atau pola penting lainnya terdeteksi dalam data.



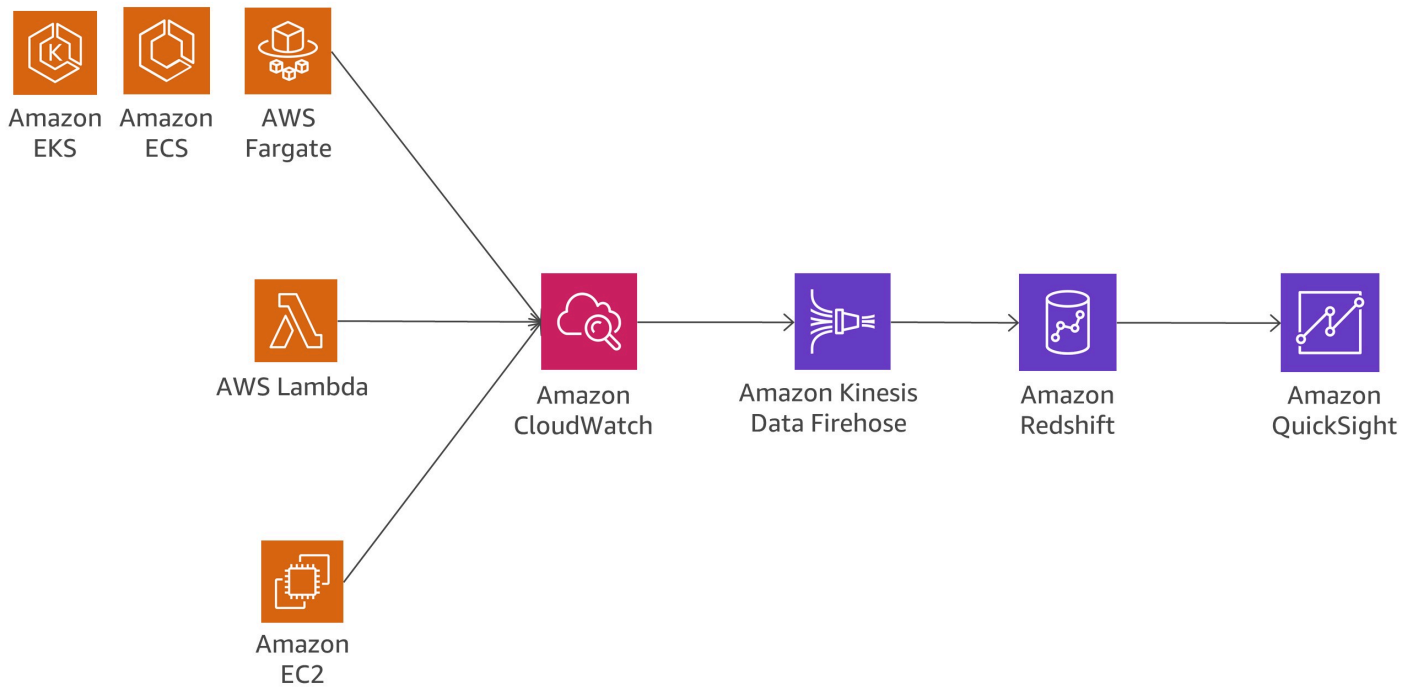
Analisis log dengan Amazon OpenSearch Service dan Kibana

Opsi lain untuk menganalisis file log adalah menggunakan Amazon Redshift dengan Amazon QuickSight.

QuickSight dapat dengan mudah dihubungkan ke layanan data AWS, termasuk Amazon Redshift, Amazon RDS, Amazon Aurora, Amazon EMR, DynamoDB, Amazon S3, dan Amazon Kinesis

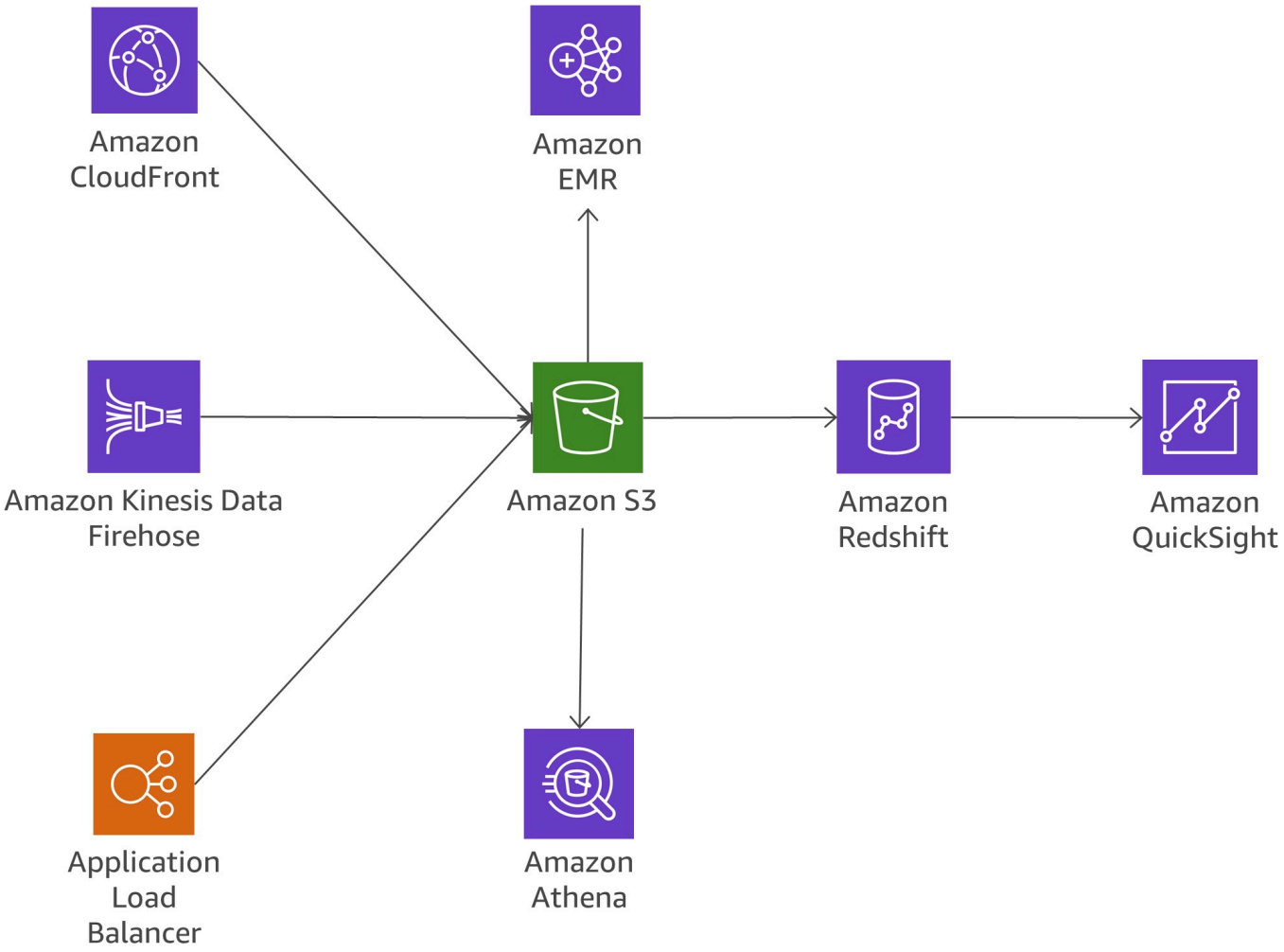
CloudWatch Logs dapat bertindak sebagai penyimpanan terpusat untuk data log, dan, selain hanya menyimpan data, entri log juga dapat dialirkan ke Amazon Kinesis Data Firehose.

Gambar berikut menampilkan skenario saat entri log dialirkan dari berbagai sumber ke Amazon Redshift menggunakan CloudWatch Logs dan Kinesis Data Firehose. Amazon QuickSight menggunakan data yang disimpan di Amazon Redshift untuk analisis, pelaporan, dan visualisasi.



Analisis log dengan Amazon Redshift dan Amazon QuickSight

Gambar berikut menampilkan skenario analisis log di Amazon S3. Ketika log disimpan dalam bucket Amazon S3, data log dapat dimuat dalam layanan data AWS yang berbeda-beda, seperti Amazon Redshift atau Amazon EMR, untuk menganalisis data yang disimpan dalam aliran log dan menemukan anomali.



Analisis log di Amazon S3

Chattiness

Dengan memecah aplikasi monolitik menjadi layanan mikro kecil, overhead komunikasi meningkat karena layanan mikro harus saling berkomunikasi. Dalam banyak penerapan, REST melalui HTTP digunakan karena merupakan protokol komunikasi ringan, tetapi volume pesan yang tinggi dapat menyebabkan masalah. Dalam beberapa kasus, Anda mungkin mempertimbangkan untuk mengonsolidasikan layanan yang mengirimkan banyak pesan secara bolak-balik. Jika Anda mendapati diri Anda dalam situasi saat Anda mengonsolidasikan jumlah layanan yang meningkat hanya untuk mengurangi chattiness, Anda harus meninjau domain dan model domain Anda yang bermasalah.

Protokol

Sebelumnya di laporan resmi ini, pada bagian the section called “Komunikasi asinkron dan olahpesan ringan”, berbagai protokol yang dapat digunakan telah dibahas. Untuk layanan mikro, penggunaan protokol sederhana seperti HTTP adalah hal yang umum. Pesan yang dipertukarkan di antara layanan dapat dienkodekan dengan cara yang berbeda-beda, misalnya format yang dapat dibaca manusia, seperti JSON atau YAML, atau format biner yang efisien, seperti Avro atau Protocol Buffers.

Caching

Cache adalah cara yang bagus untuk mengurangi latensi dan chattiness arsitektur layanan mikro. Beberapa lapisan caching dimungkinkan, bergantung pada kasus penggunaan sebenarnya dan bottleneck. Banyak aplikasi layanan mikro yang berjalan di AWS menggunakan ElastiCache untuk mengurangi volume panggilan ke layanan mikro lain dengan melakukan caching hasil secara lokal. API Gateway menyediakan lapisan caching bawaan untuk mengurangi beban pada server backend. Selain itu, caching juga berguna untuk mengurangi beban dari lapisan persistensi data. Tantangan untuk setiap mekanisme caching adalah menemukan keseimbangan yang tepat antara cache hit rate yang baik, serta ketepatan waktu dan konsistensi data.

Pengauditan

Tantangan lain yang perlu diatasi dalam arsitektur layanan mikro, yang berpotensi memiliki ratusan layanan terdistribusi, adalah memastikan visibilitas tindakan pengguna di setiap layanan dan mampu mendapatkan gambaran secara keseluruhan yang baik tentang semua layanan di tingkat organisasi. Untuk membantu menegakkan kebijakan keamanan, penting untuk mengaudit akses sumber daya dan aktivitas yang mengarah ke perubahan sistem.

Perubahan harus dilacak pada setiap tingkat layanan serta di seluruh layanan yang berjalan di sistem yang lebih luas. Biasanya, perubahan sering terjadi pada arsitektur layanan mikro, yang membuat perubahan audit menjadi makin lebih penting. Bagian ini membahas layanan dan fitur utama dalam AWS yang dapat membantu Anda mengaudit arsitektur layanan mikro.

Jejak audit

AWS CloudTrail adalah alat yang berguna untuk melacak perubahan dalam layanan mikro karena memungkinkan semua panggilan API yang dibuat AWS Cloud untuk dicatat dan dikirimkan ke CloudWatch Logs secara waktu nyata, atau ke Amazon S3 dalam beberapa menit.

Semua tindakan pengguna dan sistem otomatis menjadi dapat dicari dan dapat dianalisis untuk perilaku yang tidak terduga, pelanggaran kebijakan perusahaan, atau debugging. Informasi yang

dicatat mencakup stempel waktu, informasi pengguna dan akun, layanan yang dipanggil, tindakan layanan yang diminta, alamat IP pemanggil, serta parameter permintaan dan elemen respons.

CloudTrail memungkinkan definisi beberapa jejak untuk akun yang sama, yang memungkinkan berbagai pemangku kepentingan, seperti administrator keamanan, developer perangkat lunak, atau auditor IT, membuat dan mengelola jejak mereka sendiri. Jika tim layanan mikro memiliki akun AWS yang berbeda-beda, jejak dapat digabungkan ke dalam satu bucket S3.

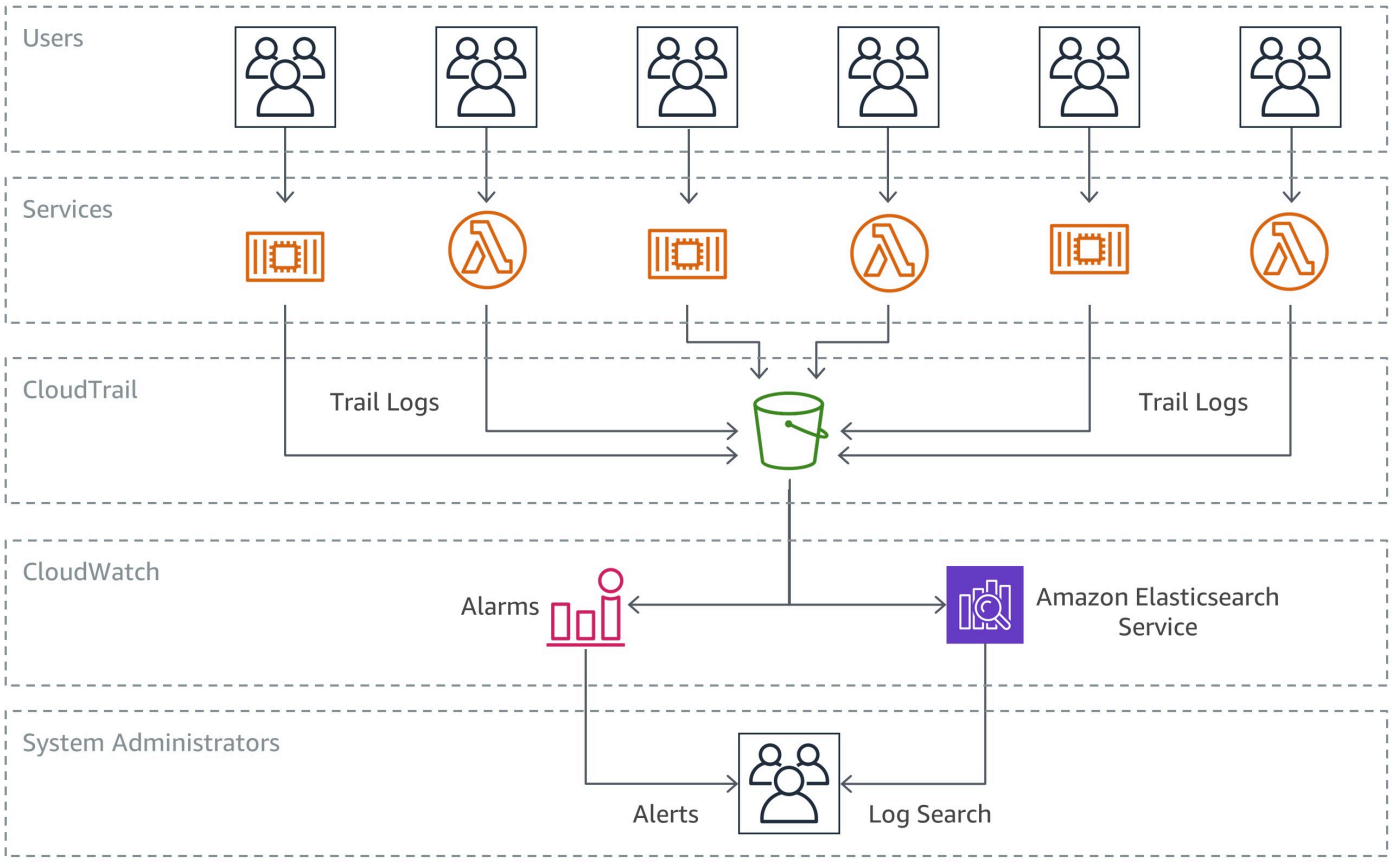
Keuntungan menyimpan jejak audit di CloudWatch adalah data jejak audit ditangkap secara waktu nyata, dan mudah dalam mengalihkan informasi ke Amazon OpenSearch Service untuk pencarian dan visualisasi. Anda dapat mengonfigurasikan CloudTrail untuk masuk ke Amazon S3 dan CloudWatch Logs.

Kejadian dan tindakan waktu nyata

Perubahan tertentu dalam arsitektur sistem harus direspons dengan cepat dan tindakan yang diambil untuk memperbaiki situasi atau prosedur tata kelola tertentu untuk mengotorisasi perubahan harus diinisiasikan. Integrasi Amazon CloudWatch Events dengan CloudTrail memungkinkan layanan ini membuat kejadian untuk semua panggilan API yang bermutasi di semua layanan AWS. Penentuan kejadian kustom atau pembuatan kejadian berdasarkan jadwal yang tetap dapat dilakukan.

Ketika suatu kejadian dipicu dan cocok dengan aturan yang ditetapkan, grup orang yang telah ditentukan sebelumnya di organisasi Anda dapat segera diberi tahu, sehingga mereka dapat mengambil tindakan yang tepat. Jika tindakan yang diperlukan dapat diotomatisasikan, aturan dapat secara otomatis memicu alur kerja bawaan atau memanggil fungsi Lambda untuk mengatasi masalah tersebut.

Gambar berikut menampilkan lingkungan tempat CloudTrail dan CloudWatch Events beroperasi bersama untuk memenuhi persyaratan audit dan remediasi dalam arsitektur layanan mikro. Semua layanan mikro dilacak oleh CloudTrail dan jejak auditnya disimpan dalam bucket Amazon S3. CloudWatch Events akan mengetahui perubahan operasional yang terjadi. CloudWatch Events merespons perubahan operasional ini dan mengambil tindakan korektif seperlunya, dengan mengirimkan pesan untuk merespons lingkungan, mengaktifkan fungsi, membuat perubahan, dan menangkap informasi status. CloudWatch Events berada di CloudTrail dan memicu pemberitahuan saat perubahan tertentu dilakukan pada arsitektur Anda.



Pengauditan dan remediasi

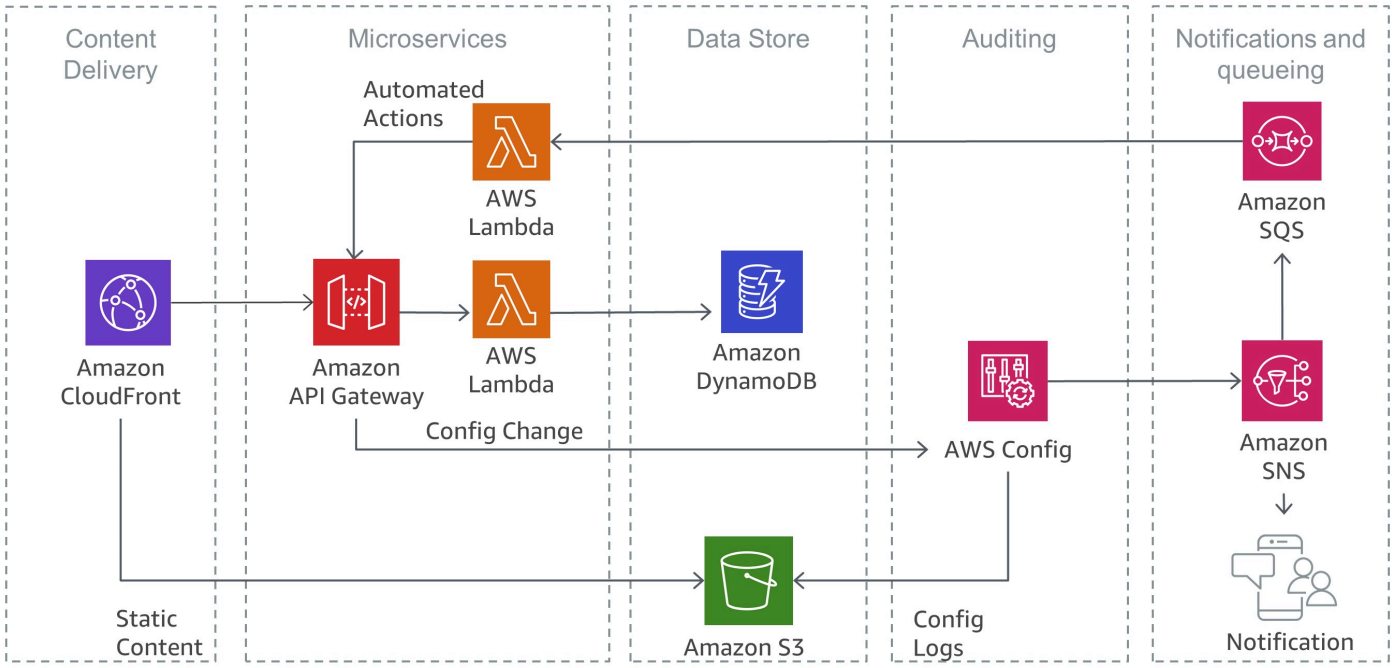
Inventaris sumber daya dan manajemen perubahan

Agar dapat mempertahankan kontrol atas konfigurasi infrastruktur yang berubah dengan cepat dalam lingkungan pengembangan yang tangkas, diperlukan pendekatan yang lebih otomatis dan terkelola untuk mengaudit dan mengendalikan arsitektur Anda.

Meskipun CloudTrail dan CloudWatch Events merupakan komponen penting untuk melacak dan merespons perubahan infrastruktur di seluruh layanan mikro, aturan AWS Config memungkinkan perusahaan menentukan kebijakan keamanan dengan aturan tertentu guna mendeteksi, melacak, dan memberi tahu Anda secara otomatis tentang kebijakan pelanggaran.

Contoh berikutnya menunjukkan cara untuk memungkinkan deteksi, pengiriman informasi, dan reaksi otomatis terhadap perubahan konfigurasi yang tidak mematuhi aturan dalam arsitektur layanan mikro Anda. Seorang anggota tim pengembangan telah membuat perubahan pada API Gateway untuk layanan mikro guna memungkinkan titik akhir menerima lalu lintas HTTP yang masuk, bukan hanya mengizinkan permintaan HTTPS.

Karena situasi ini sebelumnya telah diidentifikasi sebagai masalah kepatuhan keamanan oleh organisasinya, aturan AWS Config telah memantau kondisi ini. Aturan tersebut mengidentifikasi perubahan sebagai pelanggaran keamanan, lalu melakukan dua tindakan: aturan tersebut membuat log perubahan yang terdeteksi dalam bucket Amazon S3 untuk audit dan membuat notifikasi SNS. Amazon SNS digunakan untuk dua tujuan dalam skenario kami: untuk mengirimkan email ke grup yang ditentukan guna menginformasikan tentang pelanggaran keamanan dan menambahkan pesan ke antrean SQS. Selanjutnya, pesan diambil dan status yang mematuhi aturan dipulihkan dengan mengubah konfigurasi API Gateway.



Mendeteksi pelanggaran keamanan dengan AWS Config

Kesimpulan

Arsitektur layanan mikro adalah pendekatan desain terdistribusi yang dimaksudkan untuk mengatasi keterbatasan arsitektur monolitik tradisional. Layanan mikro membantu menskalakan aplikasi dan organisasi sambil meningkatkan waktu siklus. Namun, layanan ini juga disertai dengan beberapa tantangan yang mungkin menambah kompleksitas arsitektur tambahan dan beban operasional.

AWS menawarkan portofolio lengkap layanan terkelola yang dapat membantu tim produk membangun arsitektur layanan mikro dan meminimalkan kompleksitas arsitektur dan operasional. Laporan resmi ini menjelaskan layanan AWS yang relevan dan cara menerapkan pola yang biasa, seperti penemuan layanan atau event sourcing, secara native dengan layanan AWS.

Sumber daya

- Pusat Arsitektur AWS
- Laporan Resmi AWS
- AWS Architecture Monthly
- Blog Arsitektur AWS
- Video This Is My Architecture
- AWS Answers
- Dokumentasi AWS

Riwayat dan kontributor dokumen

Riwayat Dokumen

Untuk mendapatkan notifikasi tentang pembaruan laporan resmi ini, sebaiknya berlangganan umpan RSS.

perubahan-riwayat-pembaruan	deskripsi-riwayat-pembaruan	tanggal-riwayat-pembaruan
Laporan resmi diperbarui	Integrasi Amazon EventBridge, AWS OpenTelemetry, AMP, AMG, Container Insights, perubahan teks kecil.	9 November 2021
Pembaruan kecil	Tata letak halaman disesuaikan	30 April 2021
Pembaruan kecil	Perubahan teks kecil.	1 Agustus 2019
Laporan resmi diperbarui	Integrasi Amazon EKS, AWS Fargate, Amazon MQ, AWS PrivateLink, AWS App Mesh, AWS Cloud Map	1 Juni 2019
Laporan resmi diperbarui	Integrasi AWS Step Functions, AWS X-Ray, dan alur kejadian ECS.	1 September 2017
Publikasi awal	Menerapkan Layanan Mikro di AWS dipublikasikan.	1 Desember 2016

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

Kontributor

Individu dan organisasi berikut berkontribusi pada dokumen ini:

- Sascha Möllering, Arsitektur Solusi (Solutions Architecture), AWS
- Christian Müller, Arsitektur Solusi (Solutions Architecture), AWS
- Matthias Jung, Arsitektur Solusi (Solutions Architecture), AWS
- Peter Dalbhanjan, Arsitektur Solusi (Solutions Architecture), AWS
- Peter Chapman, Arsitektur Solusi (Solutions Architecture), AWS
- Christoph Kassen, Arsitektur Solusi (Solutions Architecture), AWS
- Umair Ishaq, Arsitektur Solusi (Solutions Architecture), AWS
- Rajiv Kumar, Arsitektur Solusi (Solutions Architecture), AWS

Pemberitahuan

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