Otorisasi SaaS multi-penyewa dan kontrol akses API: Opsi implementasi dan praktik terbaik

AWS Bimbingan Preskriptif

AWS Bimbingan Preskriptif: Otorisasi SaaS multi-penyewa dan kontrol akses API: Opsi implementasi dan praktik terbaik

Table of Contents

Otorisasi SaaS multi-penyewa dan kontrol akses API: Opsi implementasi dan praktik terbaik

Tabby Ward, Thomas Davis, Gideon Landeman, dan Tomas Riha, Amazon Web Services ()AWS

Mei 2024 (riwayat dokumen)

Otorisasi dan kontrol akses API merupakan tantangan bagi banyak aplikasi perangkat lunak—khususnya, untuk aplikasi multi-tenant software as a service (SaaS). Kompleksitas ini terbukti ketika Anda mempertimbangkan proliferasi API layanan mikro yang harus diamankan dan sejumlah besar kondisi akses yang muncul dari penyewa yang berbeda, karakteristik pengguna, dan status aplikasi. Untuk mengatasi masalah ini secara efektif, solusi harus menerapkan kontrol akses di banyak API yang disajikan oleh layanan mikro, lapisan Backend for Frontend (BFF), dan komponen lain dari aplikasi SaaS multi-penyewa. Pendekatan ini harus disertai dengan mekanisme yang mampu membuat keputusan akses yang kompleks berdasarkan banyak faktor dan atribut.

Secara tradisional, kontrol akses API dan otorisasi ditangani oleh logika kustom dalam kode aplikasi. Pendekatan ini rawan kesalahan dan tidak aman, karena pengembang yang memiliki akses ke kode ini dapat secara tidak sengaja atau sengaja mengubah logika otorisasi, yang dapat mengakibatkan akses tidak sah. Mengaudit keputusan yang dibuat oleh logika kustom dalam kode aplikasi itu sulit, karena auditor harus membenamkan diri dalam logika khusus untuk menentukan efektivitasnya dalam menegakkan standar tertentu. Selain itu, kontrol akses API umumnya tidak diperlukan, karena tidak ada banyak API yang harus diamankan. Pergeseran paradigma dalam desain aplikasi untuk mendukung layanan mikro dan arsitektur berorientasi layanan telah meningkatkan jumlah API yang harus menggunakan bentuk otorisasi dan kontrol akses. Selain itu, kebutuhan untuk mempertahankan akses berbasis penyewa dalam aplikasi SaaS multi-penyewa menghadirkan tantangan otorisasi tambahan untuk mempertahankan penyewaan. Praktik terbaik yang diuraikan dalam panduan ini memberikan beberapa manfaat:

- Logika otorisasi dapat dipusatkan dan ditulis dalam bahasa deklaratif tingkat tinggi yang tidak spesifik untuk bahasa pemrograman apa pun.
- Logika otorisasi diabstraksikan dari kode aplikasi dan dapat diterapkan sebagai pola berulang untuk semua API dalam aplikasi.
- Abstraksi mencegah perubahan yang tidak disengaja oleh pengembang ke logika otorisasi.
- Integrasi ke dalam aplikasi SaaS konsisten dan sederhana.

- Abstraksi mencegah kebutuhan untuk menulis logika otorisasi khusus untuk setiap titik akhir API.
- Audit disederhanakan, karena auditor tidak perlu lagi meninjau kode untuk menentukan izin.
- Pendekatan yang diuraikan dalam panduan ini mendukung penggunaan paradigma kontrol akses ganda tergantung pada persyaratan organisasi.
- Pendekatan otorisasi dan kontrol akses ini menyediakan cara sederhana dan mudah untuk mempertahankan isolasi data penyewa di lapisan API dalam aplikasi SaaS.
- Praktik terbaik memberikan pendekatan yang konsisten untuk penyewa orientasi dan offboarding sehubungan dengan otorisasi.
- Pendekatan ini menawarkan model penerapan otorisasi yang berbeda (dikumpulkan atau silo), yang memiliki kelebihan dan kekurangan, seperti yang dibahas dalam panduan ini.

Hasil bisnis yang ditargetkan

Panduan preskriptif ini menjelaskan pola desain berulang untuk otorisasi dan kontrol akses API yang dapat diimplementasikan untuk aplikasi SaaS multi-penyewa. Panduan ini ditujukan untuk tim mana pun yang mengembangkan aplikasi dengan persyaratan otorisasi yang kompleks atau kebutuhan kontrol akses API yang ketat. Arsitektur merinci pembuatan titik keputusan kebijakan (PDP) atau mesin kebijakan dan integrasi poin penegakan kebijakan (PEP) dalam API. Dua opsi khusus untuk membuat PDP dibahas: menggunakan Izin Terverifikasi Amazon dengan SDK Cedar dan menggunakan Open Policy Agent (OPA) dengan bahasa kebijakan Rego. Panduan ini juga membahas pengambilan keputusan akses berdasarkan model kontrol akses berbasis atribut (ABAC) atau model kontrol akses berbasis peran (RBAC), atau kombinasi dari kedua model. Kami menyarankan Anda menggunakan pola dan konsep desain yang disediakan dalam panduan ini untuk menginformasikan dan menstandarisasi implementasi otorisasi dan kontrol akses API Anda dalam aplikasi SaaS multi-penyewa. Panduan ini membantu mencapai hasil bisnis berikut:

- Arsitektur otorisasi API standar untuk aplikasi SaaS multi-penyewa — Arsitektur ini membedakan antara tiga komponen: titik administrasi kebijakan (PAP) tempat kebijakan disimpan dan dikelola, titik keputusan kebijakan (PDP) di mana kebijakan tersebut dievaluasi untuk mencapai keputusan otorisasi, dan titik penegakan kebijakan (PEP) yang memberlakukan keputusan tersebut. Layanan otorisasi yang dihosting, Izin Terverifikasi, berfungsi sebagai PAP dan PDP. Atau, Anda dapat membangun PDP Anda sendiri dengan menggunakan mesin open source seperti Cedar atau OPA.
- Pemisahan logika otorisasi dari aplikasi — Logika otorisasi, ketika disematkan dalam kode aplikasi atau diimplementasikan melalui mekanisme penegakan ad hoc, dapat mengalami perubahan yang tidak disengaja atau berbahaya yang menyebabkan akses data lintas penyewa yang tidak

disengaja atau pelanggaran keamanan lainnya. Untuk membantu mengurangi kemungkinan ini, Anda dapat menggunakan PAP, seperti Izin Terverifikasi, untuk menyimpan kebijakan otorisasi secara independen dari kode aplikasi, dan untuk menerapkan tata kelola yang kuat untuk pengelolaan kebijakan tersebut. Kebijakan dapat dipertahankan secara terpusat dalam bahasa deklaratif tingkat tinggi, yang membuat pemeliharaan logika otorisasi jauh lebih sederhana daripada ketika Anda menyematkan kebijakan di beberapa bagian kode aplikasi. Pendekatan ini juga memastikan bahwa pembaruan diterapkan secara konsisten.

- Pendekatan fleksibel untuk model kontrol akses — Kontrol akses berbasis peran (RBAC), kontrol akses berbasis atribut (ABAC), atau kombinasi kedua model adalah pendekatan yang valid untuk kontrol akses. Model-model ini berusaha memenuhi persyaratan otorisasi untuk bisnis dengan menggunakan pendekatan yang berbeda. Panduan ini membandingkan dan membedakan model-model ini untuk membantu Anda memilih model yang sesuai untuk organisasi Anda. Panduan ini juga membahas bagaimana model ini berlaku untuk bahasa kebijakan otorisasi yang berbeda, seperti OPA/Rego dan Cedar. Arsitektur yang dibahas dalam panduan ini memungkinkan salah satu atau kedua model untuk diadopsi dengan sukses.
- Kontrol akses API yang ketat — Panduan ini menyediakan metode untuk mengamankan API secara konsisten dan meluas dalam aplikasi dengan sedikit usaha. Ini sangat berharga untuk arsitektur aplikasi berorientasi layanan atau layanan mikro yang umumnya menggunakan sejumlah besar API untuk memfasilitasi komunikasi intra-aplikasi. Kontrol akses API yang ketat membantu meningkatkan keamanan aplikasi dan membuatnya kurang rentan terhadap serangan atau eksploitasi.

Isolasi penyewa dan otorisasi multi-penyewa

Panduan ini mengacu pada konsep isolasi penyewa dan otorisasi multi-penyewa. Isolasi penyewa mengacu pada mekanisme eksplisit yang Anda gunakan dalam sistem SaaS untuk memastikan bahwa sumber daya setiap penyewa, bahkan ketika mereka beroperasi pada infrastruktur bersama, terisolasi. Otorisasi multi-penyewa mengacu pada otorisasi tindakan masuk dan mencegahnya diterapkan pada penyewa yang salah. Pengguna hipotetis dapat diautentikasi dan diotorisasi, dan masih mengakses sumber daya penyewa lain. Otentikasi dan otorisasi tidak akan memblokir akses ini—Anda perlu menerapkan isolasi penyewa untuk mencapai tujuan ini. Untuk diskusi yang lebih luas tentang perbedaan antara kedua konsep ini, lihat bagian isolasi Penyewa dari whitepaper Dasar-Dasar Arsitektur SaaS.

Jenis kontrol akses

Anda dapat menggunakan dua model yang didefinisikan secara luas untuk menerapkan kontrol akses: kontrol akses berbasis peran (RBAC) dan kontrol akses berbasis atribut (ABAC). Setiap model memiliki kelebihan dan kekurangan, yang dibahas secara singkat di bagian ini. Model yang harus Anda gunakan tergantung pada kasus penggunaan spesifik Anda. Arsitektur yang dibahas dalam panduan ini mendukung kedua model.

RBAC

Role-based access control (RBAC) menentukan akses ke sumber daya berdasarkan peran yang biasanya selaras dengan logika bisnis. Izin dikaitkan dengan peran yang sesuai. Misalnya, peran pemasaran akan memberi wewenang kepada pengguna untuk melakukan aktivitas pemasaran dalam sistem terbatas. Ini adalah model kontrol akses yang relatif sederhana untuk diterapkan karena selaras dengan logika bisnis yang mudah dikenali.

Model RBAC kurang efektif ketika:

- Anda memiliki pengguna unik yang tanggung jawabnya mencakup beberapa peran.
- Anda memiliki logika bisnis yang kompleks yang membuat peran sulit untuk didefinisikan.
- Penskalaan hingga ukuran besar memerlukan administrasi dan pemetaan izin yang konstan ke peran baru dan yang sudah ada.
- Otorisasi didasarkan pada parameter dinamis.

ABAC

Attribute-based access control (ABAC) menentukan akses ke sumber daya berdasarkan atribut. Atribut dapat dikaitkan dengan pengguna, sumber daya, lingkungan, atau bahkan status aplikasi. Kebijakan atau aturan Anda mereferensikan atribut dan dapat menggunakan logika Boolean dasar untuk menentukan apakah pengguna diizinkan melakukan tindakan. Berikut adalah contoh dasar izin:

Dalam sistem pembayaran, semua pengguna di departemen Keuangan diizinkan untuk memproses pembayaran di titik akhir API `/payments` selama jam kerja.

Keanggotaan di departemen Keuangan adalah atribut pengguna yang menentukan akses ke`/payments`. Ada juga atribut sumber daya yang terkait dengan titik akhir `/payments` API yang mengizinkan akses hanya selama jam kerja. Di ABAC, apakah pengguna dapat memproses pembayaran atau tidak ditentukan oleh kebijakan yang mencakup keanggotaan departemen Keuangan sebagai atribut pengguna, dan waktu sebagai atribut sumber daya. `/payments`

Model ABAC sangat fleksibel dalam memungkinkan keputusan otorisasi yang dinamis, kontekstual, dan granular. Namun, model ABAC sulit diterapkan pada awalnya. Mendefinisikan aturan dan kebijakan serta menghitung atribut untuk semua vektor akses yang relevan memerlukan investasi awal yang signifikan untuk diterapkan.

Pendekatan hibrida RBAC-ABAC

Menggabungkan RBAC dan ABAC dapat memberikan beberapa keunggulan kedua model. RBAC, yang selaras dengan logika bisnis, lebih mudah diterapkan daripada ABAC. Untuk memberikan lapisan granularitas tambahan saat membuat keputusan otorisasi, Anda dapat menggabungkan ABAC dengan RBAC. Pendekatan hibrida ini menentukan akses dengan menggabungkan peran pengguna (dan izin yang ditetapkan) dengan atribut tambahan untuk membuat keputusan akses. Menggunakan kedua model memungkinkan administrasi sederhana dan penetapan izin sementara juga memungkinkan peningkatan fleksibilitas dan granularitas yang berkaitan dengan keputusan otorisasi.

Perbandingan model kontrol akses

Tabel berikut membandingkan tiga model kontrol akses yang dibahas sebelumnya. Perbandingan ini dimaksudkan untuk menjadi informatif dan tingkat tinggi. Menggunakan model akses dalam situasi tertentu mungkin tidak selalu berkorelasi dengan perbandingan yang dibuat dalam tabel ini.

Faktor	RBAC	ABAC	Hibrida
Fleksibilitas	Sedang	Tinggi	Tinggi
Kesederhanaan	Tinggi	Rendah	Sedang
Granularitas	Rendah	Tinggi	Sedang
Keputusan dan aturan dinamis	Tidak	Ya	Ya

Sadar konteks	Tidak	Ya	Agak
Upaya implementasi	Rendah	Tinggi	Sedang

Menerapkan PDP

Policy decision point (PDP) dapat dicirikan sebagai mesin kebijakan atau aturan. Komponen ini bertanggung jawab untuk menerapkan kebijakan atau aturan dan mengembalikan keputusan apakah akses tertentu diizinkan. PDP dapat berfungsi dengan model kontrol akses berbasis peran (RBAC) dan kontrol akses berbasis atribut (ABAC); Namun, PDP adalah persyaratan untuk ABAC. PDP memungkinkan logika otorisasi dalam kode aplikasi untuk diturunkan ke sistem terpisah. Ini dapat menyederhanakan kode aplikasi. Ini juga menyediakan antarmuka easy-to-use berulang untuk membuat keputusan otorisasi untuk API, microservices, Backend for Frontend (BFF) layer, atau komponen aplikasi lainnya.

Bagian berikut membahas tiga metode untuk menerapkan PDP. Namun, ini bukan daftar lengkap.

Metode implementasi PDP:

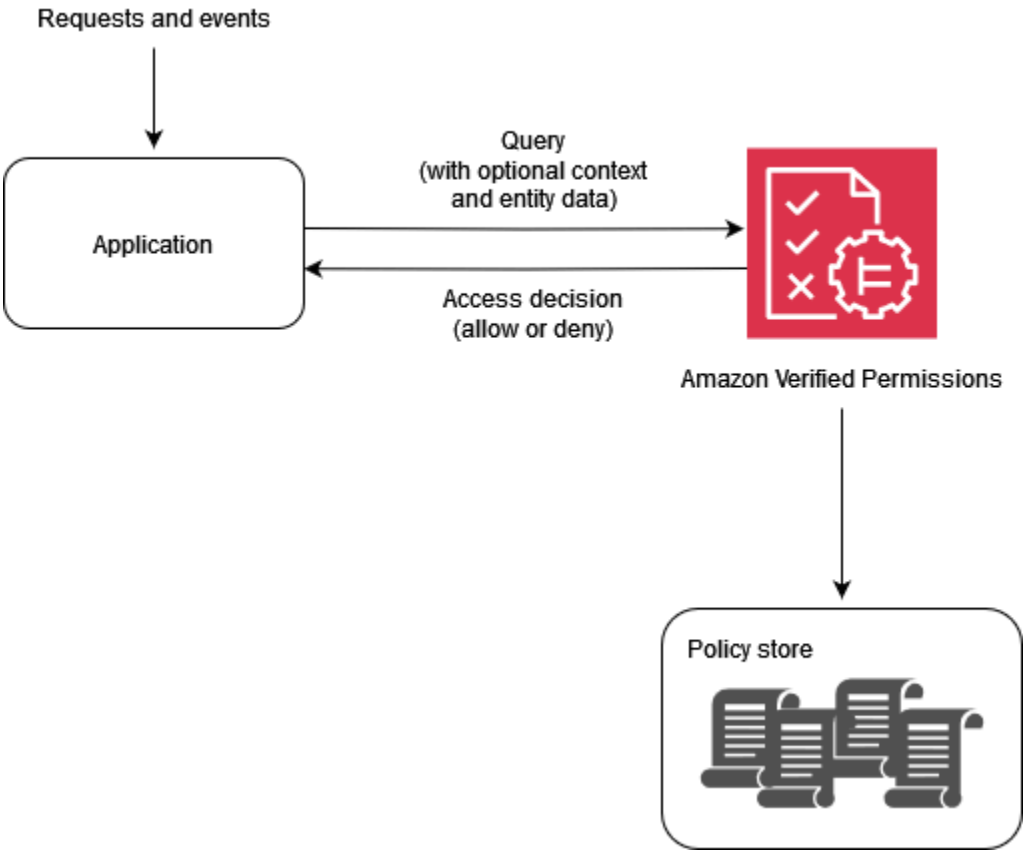
- Menerapkan PDP dengan menggunakan Izin Terverifikasi Amazon
- Menerapkan PDP dengan menggunakan OPA
- Menggunakan mesin kebijakan khusus

Menerapkan PDP dengan menggunakan Izin Terverifikasi Amazon

Izin Terverifikasi Amazon adalah layanan pengelolaan izin dan otorisasi yang dapat diskalakan dan berbutir halus yang dapat Anda gunakan untuk menerapkan titik keputusan kebijakan (PDP). Sebagai mesin kebijakan, ini dapat membantu aplikasi Anda memverifikasi tindakan pengguna secara real time dan menyoroti izin yang terlalu istimewa atau tidak valid. Ini membantu pengembang Anda membangun aplikasi yang lebih aman lebih cepat dengan mengeksternalisasi otorisasi dan memusatkan manajemen dan administrasi kebijakan. Dengan memisahkan logika otorisasi dari logika aplikasi, Izin Terverifikasi mendukung decoupling kebijakan.

Dengan menggunakan Izin Terverifikasi untuk menerapkan PDP dan menerapkan hak istimewa paling sedikit dan verifikasi berkelanjutan dalam aplikasi, pengembang dapat menyelaraskan akses aplikasi mereka dengan prinsip Zero Trust. Selain itu, tim keamanan dan audit dapat menganalisis dan mengaudit dengan lebih baik siapa yang memiliki akses ke sumber daya mana dalam suatu aplikasi. Izin Terverifikasi menggunakan Cedar, yang merupakan bahasa kebijakan sumber terbuka yang dibuat khusus dan mengutamakan keamanan, untuk menentukan kontrol akses berbasis kebijakan berdasarkan kontrol akses berbasis peran (RBAC) dan kontrol akses berbasis atribut (ABAC) untuk kontrol akses yang lebih terperinci dan sadar konteks.

Izin Terverifikasi menyediakan beberapa fitur berguna untuk aplikasi SaaS, seperti kemampuan untuk mengaktifkan otorisasi multi-penyewa dengan menggunakan beberapa penyedia identitas seperti Amazon Cognito, Google, dan Facebook. Fitur Izin Terverifikasi lainnya yang sangat membantu untuk aplikasi SaaS adalah dukungan untuk peran khusus berdasarkan per-penyewa. Jika Anda merancang sistem manajemen hubungan pelanggan (CRM), satu penyewa mungkin menentukan perincian akses dengan peluang penjualan berdasarkan satu set kriteria tertentu. Penyewa lain mungkin memiliki definisi lain. Sistem izin yang mendasari dalam Izin Terverifikasi dapat mendukung variasi ini, yang menjadikannya kandidat yang sangat baik untuk kasus penggunaan SaaS. Izin Terverifikasi juga mendukung kemampuan untuk menulis kebijakan yang berlaku untuk semua penyewa, sehingga sangat mudah untuk menerapkan kebijakan pagar pembatas untuk mencegah akses tidak sah sebagai penyedia SaaS.



Mengapa menggunakan Izin Terverifikasi?

Gunakan Izin Terverifikasi dengan penyedia identitas seperti Amazon Cognito untuk solusi manajemen akses berbasis kebijakan yang lebih dinamis untuk aplikasi Anda. Anda dapat membangun aplikasi yang membantu pengguna berbagi informasi dan berkolaborasi sambil menjaga keamanan, kerahasiaan, dan privasi data mereka. Izin Terverifikasi membantu mengurangi biaya operasional dengan memberi Anda sistem otorisasi berbutir halus untuk menegakkan akses

berdasarkan peran dan atribut identitas dan sumber daya Anda. Anda dapat menentukan model kebijakan, membuat dan menyimpan kebijakan di lokasi pusat, dan mengevaluasi permintaan akses dalam milidetik.

Di Izin Terverifikasi, Anda dapat mengekspresikan izin dengan menggunakan bahasa deklaratif sederhana yang dapat dibaca manusia yang disebut Cedar. Kebijakan yang ditulis dalam Cedar dapat dibagikan di seluruh tim terlepas dari bahasa pemrograman yang digunakan oleh aplikasi masing-masing tim.

Apa yang harus dipertimbangkan saat Anda menggunakan Izin Terverifikasi

Di Izin Terverifikasi, Anda dapat membuat kebijakan dan mengotomatiskannya sebagai bagian dari penyediaan. Anda juga dapat membuat kebijakan saat runtime sebagai bagian dari logika aplikasi. Sebagai praktik terbaik, Anda harus menggunakan pipeline integrasi berkelanjutan dan penerapan berkelanjutan (CI/CD) untuk mengelola, memodifikasi, dan melacak versi kebijakan saat membuat kebijakan sebagai bagian dari orientasi dan penyediaan penyewa. Atau, aplikasi dapat mengelola, memodifikasi, dan melacak versi kebijakan; namun, logika aplikasi tidak secara inheren menjalankan fungsi ini. Untuk mendukung kemampuan ini dalam aplikasi Anda, Anda harus secara eksplisit merancang aplikasi Anda untuk mengimplementasikan fungsi ini.

Jika perlu menyediakan data eksternal dari sumber lain untuk mencapai keputusan otorisasi, data ini harus diambil dan diberikan kepada Izin Terverifikasi sebagai bagian dari permintaan otorisasi. Konteks, entitas, dan atribut tambahan tidak diambil secara default dengan layanan ini.

Ikhtisar cedar

Cedar adalah bahasa kontrol akses berbasis kebijakan yang fleksibel, dapat diperluas, dan dapat diskalakan yang membantu pengembang mengekspresikan izin aplikasi sebagai kebijakan. Administrator dan pengembang dapat menentukan kebijakan yang mengizinkan atau melarang pengguna untuk bertindak atas sumber daya aplikasi. Beberapa kebijakan dapat dilampirkan ke satu sumber daya. Saat pengguna aplikasi Anda mencoba melakukan tindakan pada sumber daya, aplikasi Anda meminta otorisasi dari mesin kebijakan Cedar. Cedar mengevaluasi kebijakan yang berlaku dan mengembalikan keputusan `ALLOW` atau`DENY`. Cedar mendukung aturan otorisasi untuk semua jenis prinsipal dan sumber daya, memungkinkan kontrol akses berbasis peran (RBAC) dan kontrol akses berbasis atribut (ABAC), dan mendukung analisis melalui alat penalaran otomatis.

Cedar memungkinkan Anda memisahkan logika bisnis Anda dari logika otorisasi. Ketika Anda membuat permintaan dari kode aplikasi Anda, Anda memanggil mesin otorisasi Cedar untuk

menentukan apakah permintaan tersebut diotorisasi. Jika diotorisasi (keputusannya`ALLOW`), aplikasi Anda dapat melakukan operasi yang diminta. Jika tidak diotorisasi (keputusannya`DENY`), aplikasi Anda dapat mengembalikan pesan kesalahan. Fitur utama Cedar meliputi:

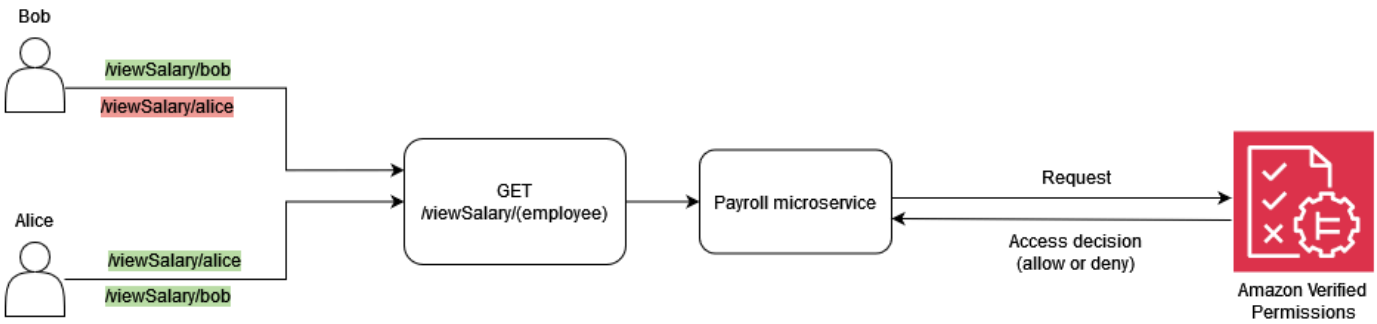
- Ekspresif - Cedar dibuat khusus untuk mendukung kasus penggunaan otorisasi dan dikembangkan dengan mempertimbangkan keterbacaan manusia.
- Kinerja — Cedar mendukung kebijakan pengindeksan untuk pengambilan cepat, dan menyediakan evaluasi real-time yang cepat dan terukur dengan latensi terbatas.
- Analisis — Cedar mendukung alat analisis yang dapat mengoptimalkan kebijakan Anda dan memverifikasi model keamanan Anda.

Untuk informasi lebih lanjut, lihat situs web Cedar.

Contoh 1: ABAC Dasar dengan Izin Terverifikasi dan Cedar

Dalam skenario contoh ini, Izin Terverifikasi Amazon digunakan untuk menentukan pengguna mana yang diizinkan mengakses informasi dalam layanan mikro Payroll fiksi. Bagian ini mencakup cuplikan kode Cedar untuk menunjukkan bagaimana Anda dapat menggunakan Cedar untuk membuat keputusan kontrol akses. Contoh-contoh ini tidak dimaksudkan untuk memberikan eksplorasi penuh atas kemampuan yang disediakan oleh Cedar dan Izin Terverifikasi. Untuk ikhtisar Cedar yang lebih menyeluruh, lihat dokumentasi Cedar.

Dalam diagram berikut, kami ingin menegakkan dua aturan bisnis umum yang terkait dengan `viewSalary` `GET` metode ini: Karyawan dapat melihat gaji mereka sendiri dan Karyawan dapat melihat gaji siapa saja yang melapor kepada mereka. Anda dapat menerapkan aturan bisnis ini dengan menggunakan kebijakan Izin Terverifikasi.



Karyawan dapat melihat gaji mereka sendiri.

Dalam Cedar, konstruksi dasar adalah entitas, yang mewakili prinsip, tindakan, atau sumber daya. Untuk membuat permintaan otorisasi dan memulai evaluasi dengan kebijakan Izin Terverifikasi, Anda perlu memberikan prinsipal, tindakan, sumber daya, dan daftar entitas.

- Principal (`principal`) adalah pengguna atau peran yang masuk.
- Action (`action`) adalah operasi yang dievaluasi oleh permintaan.
- Resource (`resource`) adalah komponen yang diakses oleh tindakan.
- Daftar entitas (`entityList`) berisi semua entitas yang diperlukan untuk mengevaluasi permintaan.

Untuk memenuhi aturan bisnis Karyawan dapat melihat gaji mereka sendiri, Anda dapat memberikan kebijakan Izin Terverifikasi seperti berikut ini.

```
permit (
    principal,
    action == Action::"viewSalary",
    resource
)
when {
    principal == resource.owner
};
```

Kebijakan ini mengevaluasi `ALLOW` apakah `Action` is `viewSalary` dan sumber daya dalam permintaan memiliki pemilik atribut yang sama dengan prinsipal. Misalnya, jika Bob adalah pengguna yang masuk yang meminta laporan gaji dan juga pemilik laporan gaji, kebijakan akan mengevaluasi`ALLOW`.

Permintaan otorisasi berikut diserahkan ke Izin Terverifikasi untuk dievaluasi oleh kebijakan sampel. Dalam contoh ini, Bob adalah pengguna yang masuk yang membuat `viewSalary` permintaan. Oleh karena itu, Bob adalah prinsipal dari tipe entitas`Employee`. Tindakan yang coba dilakukan Bob adalah `viewSalary`, dan sumber daya yang `viewSalary` akan ditampilkan adalah `Salary-Bob` dengan jenisnya`Salary`. Untuk mengevaluasi apakah Bob dapat melihat `Salary-Bob` sumber daya, Anda perlu menyediakan struktur entitas yang menghubungkan tipe `Employee` dengan nilai `Bob` (prinsipal) ke atribut pemilik sumber daya yang memiliki tipe`Salary`. Anda menyediakan struktur ini di`entityList`, di mana atribut yang terkait dengan `Salary` menyertakan pemilik, yang menentukan `entityIdentifier` yang berisi tipe `Employee` dan nilai`Bob`. Izin Terverifikasi membandingkan

yang `principal` disediakan dalam permintaan otorisasi dengan `owner` atribut yang terkait dengan `Salary` sumber daya untuk membuat keputusan.

```
{
  "policyStoreId": "PAYROLLAPP_POLICYSTOREID",
  "principal": {
    "entityType": "PayrollApp::Employee",
    "entityId": "Bob"
  },
  "action": {
    "actionType": "PayrollApp::Action",
    "actionId": "viewSalary"
  },
  "resource": {
    "entityType": "PayrollApp::Salary",
    "entityId": "Salary-Bob"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
          "entityType": "PayrollApp::Salary",
          "entityId": "Salary-Bob"
        },
        "attributes": {
          "owner": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "Bob"
            }
          }
        }
      },
      {
        "identifier": {
          "entityType": "PayrollApp::Employee",
          "entityId": "Bob"
        },
        "attributes": {}
      }
    ]
  }
```

```
}
```

Permintaan otorisasi untuk Izin Terverifikasi mengembalikan berikut ini sebagai output, di mana atributnya `decision` adalah salah satu atau`ALLOW`. `DENY`

```
{
    "determiningPolicies":
        [
            {
                "determiningPolicyId": "PAYROLLAPP_POLICYSTOREID"
            }
        ],
    "decision": "ALLOW",
    "errors": []
}
```

Dalam hal ini, karena Bob mencoba melihat gajinya sendiri, permintaan otorisasi yang dikirim ke Izin Terverifikasi mengevaluasi. `ALLOW` Namun, tujuan kami adalah menggunakan Izin Terverifikasi untuk menegakkan dua aturan bisnis. Aturan bisnis yang menyatakan hal-hal berikut juga harus benar:

Karyawan dapat melihat gaji siapa saja yang melapor kepada mereka.

Untuk memenuhi aturan bisnis ini, Anda dapat memberikan kebijakan lain. Kebijakan berikut mengevaluasi `ALLOW` apakah tindakan tersebut `viewSalary` dan sumber daya dalam permintaan memiliki atribut `owner.manager` yang sama dengan prinsipal. Misalnya, jika Alice adalah pengguna yang masuk yang meminta laporan gaji dan Alice adalah manajer pemilik laporan, kebijakan tersebut akan dievaluasi. `ALLOW`

```
permit (
    principal,
    action == Action::"viewSalary",
    resource
)
when {
    principal == resource.owner.manager
};
```

Permintaan otorisasi berikut diserahkan ke Izin Terverifikasi untuk dievaluasi oleh kebijakan sampel. Dalam contoh ini, Alice adalah pengguna yang masuk yang membuat `viewSalary` permintaan. Oleh karena itu Alice adalah kepala sekolah dan entitas adalah tipe`Employee`. Tindakan yang Alice

coba lakukan adalah`viewSalary`, dan sumber daya yang `viewSalary` akan ditampilkan adalah tipe `Salary` dengan nilai. `Salary-Bob` Untuk mengevaluasi apakah Alice dapat melihat `Salary-Bob` sumber daya, Anda perlu menyediakan struktur entitas yang menghubungkan tipe `Employee` dengan nilai `manager` atribut, yang kemudian harus dikaitkan dengan `owner` atribut tipe `Salary` dengan nilai`Salary-Bob`. `Alice` Anda menyediakan struktur ini di`entityList`, di mana atribut yang terkait dengan `Salary` menyertakan pemilik, yang menentukan `entityIdentifier` yang berisi tipe `Employee` dan nilai`Bob`. Izin Terverifikasi terlebih dahulu memeriksa `owner` atribut, yang mengevaluasi tipe `Employee` dan nilainya. `Bob` Kemudian, Izin Terverifikasi mengevaluasi `manager` atribut yang terkait dengan `Employee` dan membandingkannya dengan prinsipal yang disediakan untuk membuat keputusan otorisasi. Dalam hal ini, keputusannya adalah `ALLOW` karena `resource.owner.manager` atribut `principal` dan setara.

```
{
  "policyStoreId": "PAYROLLAPP_POLICYSTOREID",
  "principal": {
    "entityType": "PayrollApp::Employee",
    "entityId": "Alice"
  },
  "action": {
    "actionType": "PayrollApp::Action",
    "actionId": "viewSalary"
  },
  "resource": {
    "entityType": "PayrollApp::Salary",
    "entityId": "Salary-Bob"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
          "entityType": "PayrollApp::Employee",
          "entityId": "Alice"
        },
        "attributes": {
          "manager": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "None"
            }
          }
        },
```

```
        "parents": []
      },
      {
        "identifier": {
          "entityType": "PayrollApp::Salary",
          "entityId": "Salary-Bob"
        },
        "attributes": {
          "owner": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "Bob"
            }
          }
        },
        "parents": []
      },
      {
        "identifier": {
          "entityType": "PayrollApp::Employee",
          "entityId": "Bob"
        },
        "attributes": {
          "manager": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "Alice"
            }
          }
        },
       "parents": []
      }
    ]
  }
}
```

Sejauh ini dalam contoh ini, kami memberikan dua aturan bisnis yang terkait dengan `viewSalary` metode ini, Karyawan dapat melihat gaji mereka sendiri dan Karyawan dapat melihat gaji siapa saja yang melapor kepada mereka, ke Izin Terverifikasi sebagai kebijakan untuk memenuhi ketentuan masing-masing aturan bisnis secara independen. Anda juga dapat menggunakan satu kebijakan Izin Terverifikasi untuk memenuhi ketentuan kedua aturan bisnis:

Karyawan dapat melihat gaji mereka sendiri dan gaji siapa saja yang melapor kepada mereka.

Bila Anda menggunakan permintaan otorisasi sebelumnya, kebijakan berikut akan mengevaluasi `ALLOW` apakah tindakan tersebut `viewSalary` dan sumber daya dalam permintaan memiliki atribut `owner.manager` yang sama dengan`principal`, atau atribut `owner` yang sama dengan. `principal`

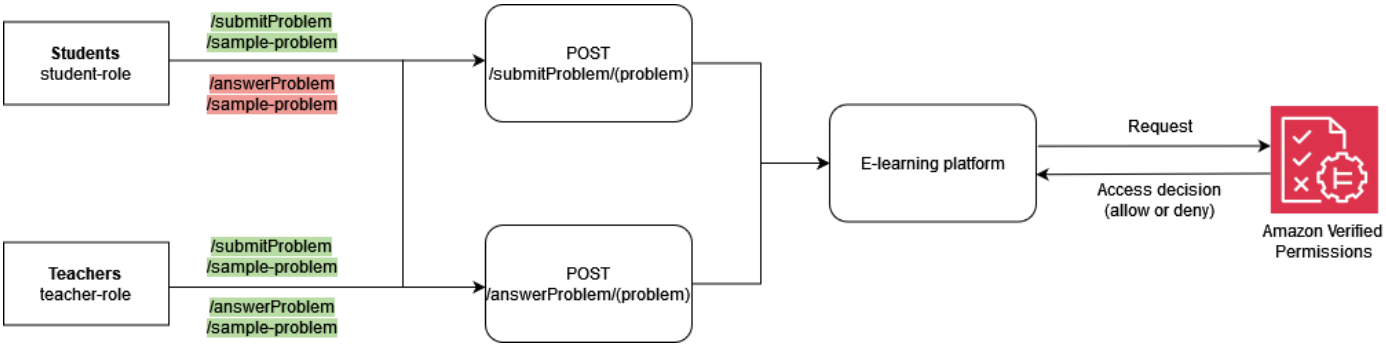
```
permit (
    principal,
    action == PayrollApp::Action::"viewSalary",
    resource
)
when {
    principal == resource.owner.manager ||
    principal == resource.owner
};
```

Misalnya, jika Alice adalah pengguna yang masuk yang meminta laporan gaji, dan jika Alice adalah manajer pemilik atau pemilik laporan, maka kebijakan tersebut akan dievaluasi. `ALLOW`

Untuk informasi selengkapnya tentang menggunakan operator logis dengan kebijakan Cedar, lihat dokumentasi Cedar.

Contoh 2: RBAC Dasar dengan Izin Terverifikasi dan Cedar

Contoh ini menggunakan Izin Terverifikasi dan Cedar untuk mendemonstrasikan RBAC dasar. Seperti disebutkan sebelumnya, konstruksi dasar Cedar adalah entitas. Pengembang mendefinisikan entitas mereka sendiri dan secara opsional dapat membuat hubungan antar entitas. Contoh berikut mencakup tiga jenis entitas:`Users`,`Roles`, dan`Problems`. `Students`dan `Teachers` dapat dianggap entitas dari jenis `Role`, dan masing-masing `User` dapat dikaitkan dengan nol atau salah satu`Roles`.



Di Cedar, hubungan ini diekspresikan dengan menghubungkan `Role Student` ke `User Bob` sebagai induknya. Asosiasi ini secara logis mengelompokkan semua pengguna siswa dalam satu kelompok. Untuk informasi lebih lanjut tentang pengelompokan di Cedar, lihat dokumentasi Cedar.

Kebijakan berikut mengevaluasi keputusan untuk tindakan `ALLOW submitProblem,` untuk semua kepala sekolah yang terkait dengan kelompok `Students` logis dari jenis tersebut. `Role`

```
permit (
    principal in ElearningApp::Role::"Students",
    action == ElearningApp::Action::"submitProblem",
    resource
);
```

Kebijakan berikut mengevaluasi keputusan `ALLOW` untuk tindakan `submitProblem` atau`answerProblem`, untuk semua prinsip yang terkait dengan kelompok `Teachers` logis dari jenis tersebut. `Role`

```
permit (
    principal in ElearningApp::Role::"Teachers",
    action in [
        ElearningApp::Action::"submitProblem",
        ElearningApp::Action::"answerProblem"
    ],
    resource
);
```

Untuk mengevaluasi permintaan dengan kebijakan ini, mesin evaluasi perlu mengetahui apakah prinsipal yang dirujuk dalam permintaan otorisasi adalah anggota kelompok yang sesuai. Oleh karena itu, aplikasi harus meneruskan informasi keanggotaan grup yang relevan ke mesin evaluasi sebagai bagian dari permintaan otorisasi. Ini dilakukan melalui `entities` properti, yang memungkinkan Anda untuk menyediakan mesin evaluasi Cedar dengan atribut dan data keanggotaan grup untuk kepala sekolah dan sumber daya yang terlibat dalam panggilan otorisasi. Dalam kode berikut, keanggotaan grup ditunjukkan dengan mendefinisikan `User::"Bob"` sebagai orang tua yang dipanggil`Role::"Students"`.

```
{
  "policyStoreId": "ELEARNING_POLICYSTOREID",
  "principal": {
    "entityType": "ElearningApp::User",
```

```
    "entityId": "Bob"
  },
  "action": {
    "actionType": "ElearningApp::Action",
    "actionId": "answerProblem"
  },
  "resource": {
    "entityType": "ElearningApp::Problem",
    "entityId": "SomeProblem"
  },
  "entities": {
    "entityList": [
        {
            "identifier": {
                "entityType": "ElearningApp::User",
                "entityId": "Bob"
            },
            "attributes": {},
            "parents": [
                {
                    "entityType": "ElearningApp::Role",
                    "entityId": "Students"
                }
            ]
        },
        {
          "identifier": {
            "entityType": "ElearningApp::Problem",
            "entityId": "SomeProblem"
          },
          "attributes": {},
          "parents": []
        }
      ]
  }
}
```

Dalam contoh ini, Bob adalah pengguna yang masuk yang membuat `answerProblem` permintaan. Oleh karena itu, Bob adalah prinsipal dan entitas adalah tipe`User`. Tindakan yang Bob coba lakukan adalah`answerProblem`. Untuk mengevaluasi apakah Bob dapat melakukan `answerProblem` tindakan, Anda perlu menyediakan struktur entitas yang menghubungkan entitas `User` dengan nilai `Bob` dan menetapkan keanggotaan grupnya dengan mencantumkan entitas induk

sebagai`Role::"Students"`. Karena entitas dalam grup pengguna hanya `Role::"Students"` diizinkan untuk melakukan tindakan`submitProblem`, permintaan otorisasi ini akan dievaluasi. `DENY`

Di sisi lain, jika tipe `User` yang memiliki nilai `Alice` dan merupakan bagian dari grup `Role::"Teachers"` mencoba melakukan `answerProblem` tindakan, permintaan otorisasi mengevaluasi`ALLOW`, karena kebijakan menentukan bahwa kepala sekolah dalam grup `Role::"Teachers"` diizinkan untuk melakukan tindakan pada semua sumber daya. `answerProblem` Kode berikut menunjukkan jenis permintaan otorisasi yang mengevaluasi. `ALLOW`

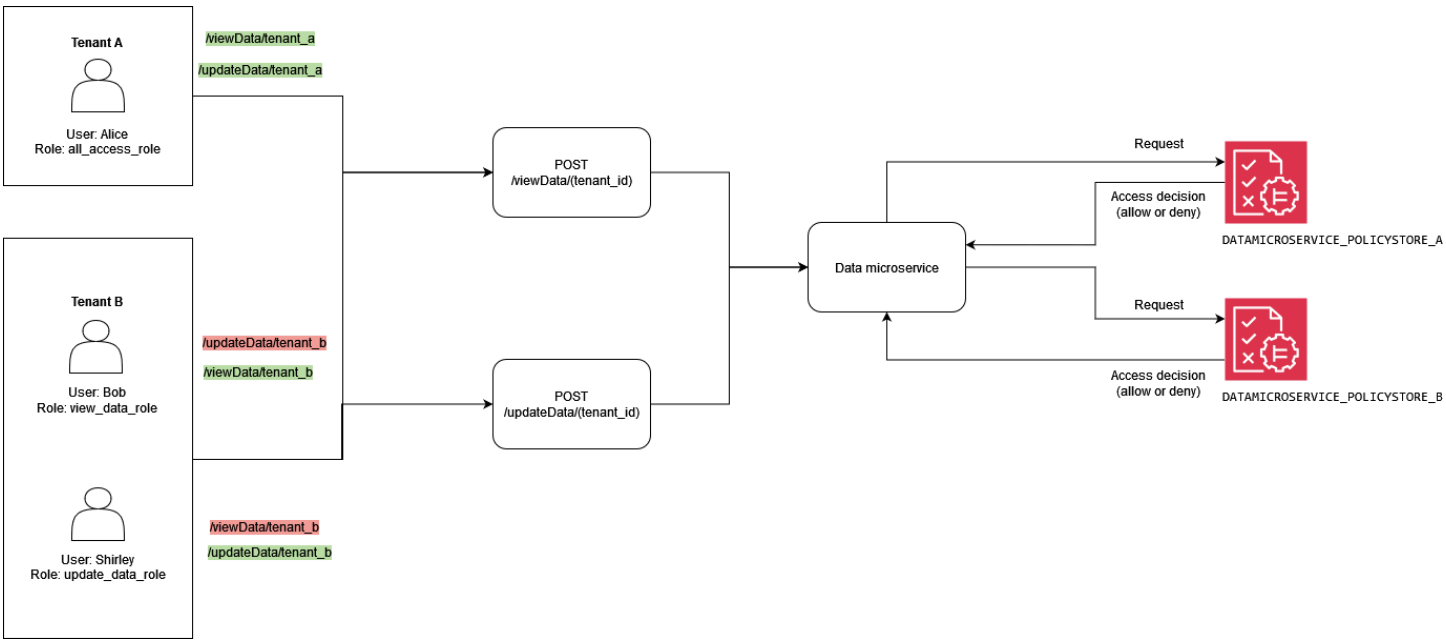
```
{
  "policyStoreId": "ELEARNING_POLICYSTOREID",
  "principal": {
    "entityType": "ElearningApp::User",
    "entityId": "Alice"
  },
  "action": {
    "actionType": "ElearningApp::Action",
    "actionId": "answerProblem"
  },
  "resource": {
    "entityType": "ElearningApp::Problem",
    "entityId": "SomeProblem"
  },
  "entities": {
    "entityList": [
        {
            "identifier": {
                "entityType": "ElearningApp::User",
                "entityId": "Alice"
            },
            "attributes": {},
            "parents": [
                {
                    "entityType": "ElearningApp::Role",
                    "entityId": "Teachers"
                }
            ]
        },
        {
            "identifier": {
                "entityType": "ElearningApp::Problem",
                "entityId": "SomeProblem"
```

```
            },
            "attributes": {},
            "parents": []
        }
      ]
  }
}
```

Contoh 3: Kontrol akses multi-penyewa dengan RBAC

Untuk menguraikan contoh RBAC sebelumnya, Anda dapat memperluas persyaratan Anda untuk memasukkan multi-tenancy SaaS, yang merupakan persyaratan umum untuk penyedia SaaS. Dalam solusi multi-penyewa, akses sumber daya selalu disediakan atas nama penyewa tertentu. Artinya, pengguna Tenant A tidak dapat melihat data Tenant B, bahkan jika data tersebut secara logis atau fisik ditempatkan dalam suatu sistem. Contoh berikut menggambarkan bagaimana Anda dapat menerapkan isolasi penyewa dengan menggunakan beberapa penyimpanan kebijakan Izin Terverifikasi, dan bagaimana Anda dapat menggunakan peran pengguna untuk menentukan izin dalam penyewa.

Menggunakan pola desain Toko Kebijakan Per Penyewa adalah praktik terbaik untuk mempertahankan isolasi penyewa sambil menerapkan kontrol akses dengan Izin Terverifikasi. Dalam skenario ini, permintaan pengguna Penyewa A dan Penyewa B diverifikasi terhadap penyimpanan kebijakan terpisah, `DATAMICROSERVICE_POLICYSTORE_A` dan`DATAMICROSERVICE_POLICYSTORE_B`, masing-masing. Untuk informasi selengkapnya tentang pertimbangan desain Izin Terverifikasi untuk aplikasi SaaS multi-penyewa, lihat bagian pertimbangan desain multi-penyewa Izin Terverifikasi.



Kebijakan berikut berada di toko `DATAMICROSERVICE_POLICYSTORE_A` kebijakan. Ini memverifikasi bahwa kepala sekolah akan menjadi bagian dari kelompok `allAccessRole` tipe`Role`. Dalam hal ini, kepala sekolah akan diizinkan untuk melakukan `viewData` dan `updateData` tindakan pada semua sumber daya yang terkait dengan Penyewa A.

```
permit (
    principal in MultitenantApp::Role::"allAccessRole",
    action in [
        MultitenantApp::Action::"viewData",
        MultitenantApp::Action::"updateData"
    ],
    resource
);
```

Kebijakan berikut berada di toko `DATAMICROSERVICE_POLICYSTORE_B` kebijakan. Kebijakan pertama memverifikasi bahwa prinsipal adalah bagian dari `updateDataRole` kelompok tipe`Role`. Dengan asumsi demikian, ia memberikan izin kepada kepala sekolah untuk melakukan `updateData` tindakan pada sumber daya yang terkait dengan Penyewa B.

```
permit (
    principal in MultitenantApp::Role::"updateDataRole",
    action == MultitenantApp::Action::"updateData",
    resource
);
```

Kebijakan kedua ini mengamanatkan bahwa kepala sekolah yang merupakan bagian dari `viewDataRole` kelompok tipe `Role` harus diizinkan untuk melakukan `viewData` tindakan terhadap sumber daya yang terkait dengan Penyewa B.

```
permit (
    principal in MultitenantApp::Role::"viewDataRole",
    action == MultitenantApp::Action::"viewData",
    resource
);
```

Permintaan otorisasi yang dibuat dari Penyewa A perlu dikirim ke toko `DATAMICROSERVICE_POLICYSTORE_A` kebijakan dan diverifikasi oleh kebijakan milik toko tersebut. Dalam hal ini, ini diverifikasi oleh kebijakan pertama yang dibahas sebelumnya sebagai bagian dari contoh ini. Dalam permintaan otorisasi ini, prinsipal tipe `User` dengan nilai `Alice` meminta untuk melakukan tindakan. `viewData` Kepala sekolah termasuk dalam kelompok `allAccessRole` tipe`Role`. Alice mencoba melakukan `viewData` aksi pada `SampleData` sumber daya. Karena Alice memiliki `allAccessRole` peran, evaluasi ini menghasilkan `ALLOW` keputusan.

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE_A",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "viewData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
      "entityId": "SampleData"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
            "entityType": "MultitenantApp::User",
            "entityId": "Alice"
        },
        "attributes": {},
        "parents": [
```

```
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "allAccessRole"
            }
        ]
      },
      {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": []
      }
    ]
  }
}
```

Jika, sebaliknya, Anda melihat permintaan yang dibuat dari Penyewa B oleh`User Bob`, Anda akan melihat sesuatu seperti permintaan otorisasi berikut. Permintaan dikirim ke toko `DATAMICROSERVICE_POLICYSTORE_B` kebijakan karena berasal dari Penyewa B. Dalam permintaan ini, kepala sekolah `Bob` ingin melakukan tindakan `updateData` pada sumber daya. `SampleData` Namun, Bob bukan bagian dari grup yang memiliki akses ke tindakan `updateData` pada sumber daya itu. Oleh karena itu, permintaan menghasilkan `DENY` keputusan.

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE_B",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Bob"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "updateData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
      "entityId": "SampleData"
  },
  "entities": {
    "entityList": [
      {
```

```
        "identifier": {
            "entityType": "MultitenantApp::User",
            "entityId": "Bob"
        },
        "attributes": {},
        "parents": [
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "viewDataRole"
            }
        ]
      },
      {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": []
      }
    ]
  }
}
```

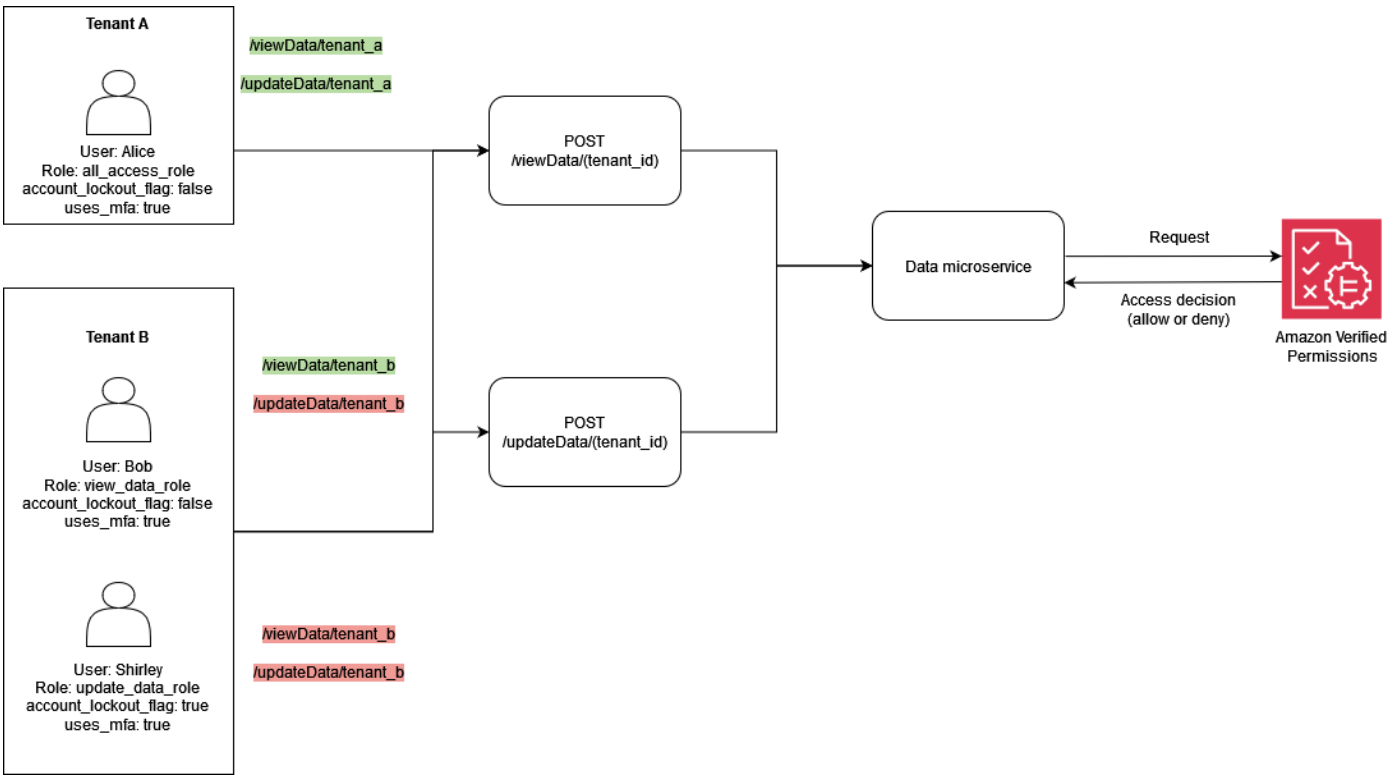
Dalam contoh ketiga ini, `User Alice` cobalah untuk melakukan `viewData` tindakan pada sumber dayaSampleData. Permintaan ini diarahkan ke toko DATAMICROSERVICE_POLICYSTORE_A kebijakan karena kepala sekolah `Alice` milik Penyewa A. `Alice` adalah bagian `allAccessRole` dari kelompok tipeRole, yang memungkinkannya untuk melakukan `viewData` tindakan pada sumber daya. Dengan demikian, permintaan menghasilkan `ALLOW` keputusan.

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE_A",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "viewData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
```

```
      "entityId": "SampleData"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
            "entityType": "MultitenantApp::User",
            "entityId": "Alice"
        },
        "attributes": {},
        "parents": [
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "allAccessRole"
            }
        ]
      },
      {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": []
      }
    ]
  }
}
```

Contoh 4: Kontrol akses multi-tenant dengan RBAC dan ABAC

Untuk menyempurnakan contoh RBAC di bagian sebelumnya, Anda dapat menambahkan atribut ke pengguna untuk membuat pendekatan hybrid RBAC-ABAC untuk kontrol akses multi-tenant. Contoh ini mencakup peran yang sama dari contoh sebelumnya, tetapi menambahkan atribut pengguna `account_lockout_flag` dan parameter konteks`uses_mfa`. Contoh ini juga mengambil pendekatan yang berbeda untuk menerapkan kontrol akses multi-penyewa dengan menggunakan RBAC dan ABAC, dan menggunakan satu penyimpanan kebijakan bersama alih-alih penyimpanan kebijakan yang berbeda untuk setiap penyewa.



Contoh ini mewakili solusi SaaS multi-penyewa di mana Anda perlu memberikan keputusan otorisasi untuk Penyewa A dan Penyewa B, mirip dengan contoh sebelumnya.

Untuk mengimplementasikan fitur kunci pengguna, contoh menambahkan atribut `account_lockout_flag` ke prinsipal `User` entitas dalam permintaan otorisasi. Bendera ini mengunci akses pengguna ke sistem dan akan `DENY` semua hak istimewa bagi pengguna yang terkunci. `account_lockout_flag`Atribut dikaitkan dengan `User` entitas dan berlaku untuk `User` sampai flag dicabut secara aktif di beberapa sesi. Contoh menggunakan `when` kondisi untuk mengevaluasi`account_lockout_flag`.

Contoh ini juga menambahkan detail tentang permintaan dan sesi. Informasi konteks menentukan bahwa sesi telah diautentikasi dengan menggunakan otentikasi multi-faktor. Untuk mengimplementasikan validasi ini, contoh menggunakan `when` kondisi untuk mengevaluasi `uses_mfa` bendera sebagai bagian dari bidang konteks. Untuk informasi selengkapnya tentang praktik terbaik untuk menambahkan konteks, lihat dokumentasi Cedar.

```
permit (
    principal in MultitenantApp::Role::"allAccessRole",
    action in [
        MultitenantApp::Action::"viewData",
        MultitenantApp::Action::"updateData"
```

```
    ],
    resource
)
when {
    principal.account_lockout_flag == false &&
    context.uses_mfa == true &&
    resource in principal.Tenant
};
```

Kebijakan ini mencegah akses ke sumber daya kecuali sumber daya berada dalam grup yang sama dengan `Tenant` atribut prinsipal yang meminta. Pendekatan untuk mempertahankan isolasi penyewa ini disebut sebagai pendekatan One Shared Multi-Tenant Policy Store. Untuk informasi selengkapnya tentang pertimbangan desain Izin Terverifikasi untuk aplikasi SaaS multi-penyewa, lihat bagian pertimbangan desain multi-penyewa Izin Terverifikasi.

Kebijakan ini juga memastikan bahwa kepala sekolah adalah anggota `allAccessRole` dan membatasi tindakan untuk `viewData` dan`updateData`. Selain itu, kebijakan ini memverifikasi `account_lockout_flag` itu `false` dan bahwa nilai konteks untuk `uses_mfa` mengevaluasi. `true`

Demikian pula, kebijakan berikut memastikan bahwa prinsipal dan sumber daya terkait dengan penyewa yang sama, yang mencegah akses lintas penyewa. Kebijakan ini juga memastikan bahwa kepala sekolah adalah anggota `viewDataRole` dan membatasi tindakan untuk`viewData`. Selain itu, ini memverifikasi bahwa `account_lockout_flag` adalah `false` dan bahwa nilai konteks untuk `uses_mfa` dievaluasi. `true`

```
permit (
    principal in MultitenantApp::Role::"viewDataRole",
    action == MultitenantApp::Action::"viewData",
    resource
)
when {
    principal.account_lockout_flag == false &&
    context.uses_mfa == true &&
    resource in principal.Tenant
};
```

Kebijakan ketiga mirip dengan yang sebelumnya. Kebijakan mengharuskan sumber daya untuk menjadi anggota grup yang sesuai dengan entitas yang diwakili oleh`principal.Tenant`. Ini memastikan bahwa prinsipal dan sumber daya terkait dengan Penyewa B, yang mencegah akses lintas penyewa. Kebijakan ini memastikan bahwa kepala sekolah adalah anggota `updateDataRole`

dan membatasi tindakan untuk`updateData`. Selain itu, kebijakan ini memverifikasi bahwa `account_lockout_flag` adalah `false` dan bahwa nilai konteks untuk `uses_mfa` dievaluasi. `true`

```
permit (
    principal in MultitenantApp::Role::"updateDataRole",
    action == MultitenantApp::Action::"updateData",
    resource
)
when {
    principal.account_lockout_flag == false &&
    context.uses_mfa == true &&
    resource in principal.Tenant
};
```

Permintaan otorisasi berikut dievaluasi oleh tiga kebijakan yang dibahas sebelumnya di bagian ini. Dalam permintaan otorisasi ini, kepala sekolah tipe `User` dan dengan nilai `Alice` membuat `updateData` permintaan dengan peran `allAccessRole` tersebut. `Alice`memiliki atribut `Tenant` yang nilainya`Tenant::"TenantA"`. Tindakan `Alice` yang coba dilakukan adalah `updateData`, dan sumber daya yang akan diterapkan adalah `SampleData` dari jenisnya`Data`. `SampleData`memiliki `TenantA` sebagai entitas induk.

Menurut kebijakan pertama di toko <DATAMICROSERVICE_POLICYSTOREID> kebijakan, `Alice` dapat melakukan `updateData` tindakan pada sumber daya, dengan asumsi bahwa kondisi dalam `when` klausul kebijakan terpenuhi. Kondisi pertama membutuhkan `principal.Tenant` atribut untuk dievaluasi`TenantA`. Kondisi kedua membutuhkan atribut kepala sekolah `account_lockout_flag` untuk menjadi`false`. Kondisi akhir membutuhkan konteksnya `uses_mfa``true`. Karena ketiga kondisi terpenuhi, permintaan mengembalikan `ALLOW` keputusan.

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "updateData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
```

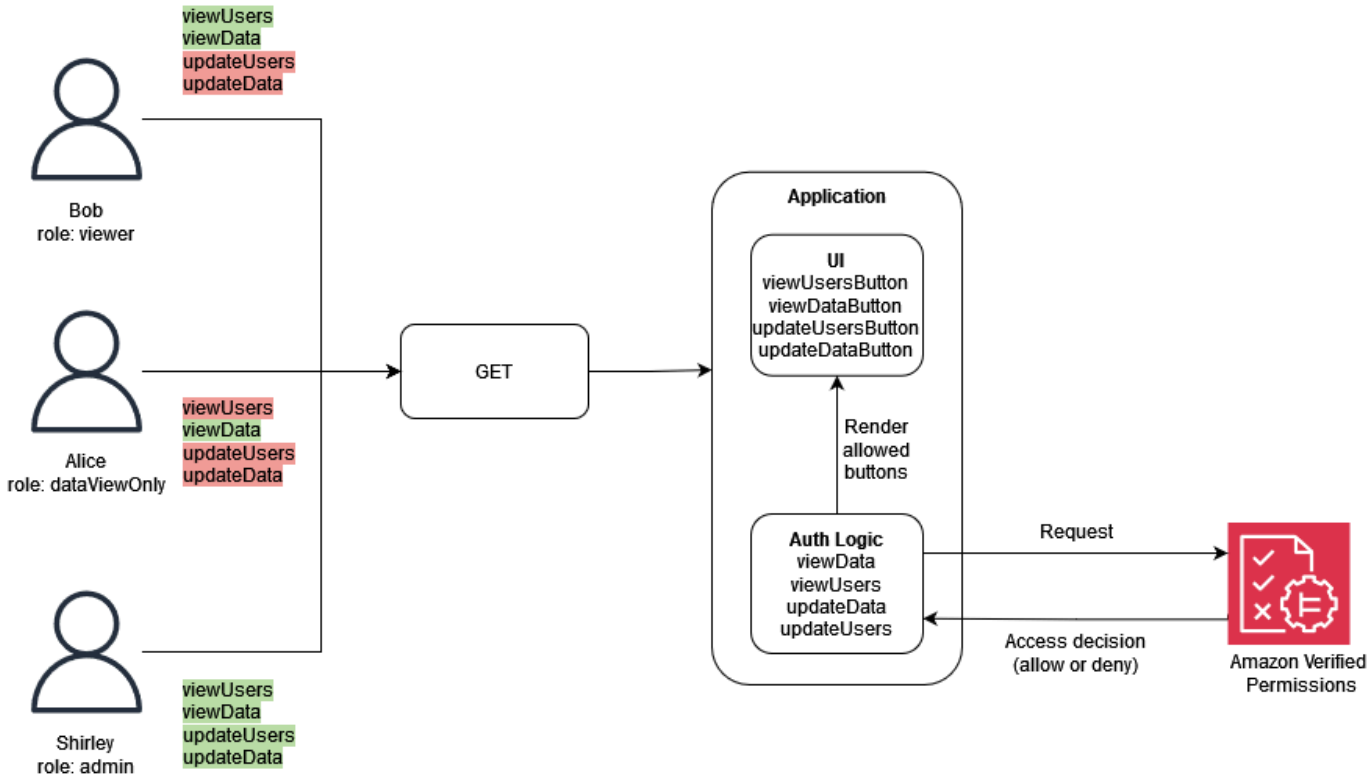
```
      "entityId": "SampleData"
  },
  "context": {
    "contextMap": {
        "uses_mfa": {
            "boolean": true
        }
    }
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
            "entityType": "MultitenantApp::User",
            "entityId": "Alice"
        },
        "attributes": {
            {
                "account_lockout_flag": {
                    "boolean": false
                },
                "Tenant": {
                   "entityIdentifier": {
                        "entityType":"MultitenantApp::Tenant",
                        "entityId":"TenantA"
                   }
                }
            }
        },
        "parents": [
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "allAccessRole"
            }
        ]
        },
     {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": [
```

```
            {
                "entityType": "MultitenantApp::Tenant",
                "entityId": "TenantA"
            }
        ]
      }
    ]
  }
}
```

Contoh 5: Pemfilteran UI dengan Izin Terverifikasi dan Cedar

Anda juga dapat menggunakan Izin Terverifikasi untuk menerapkan pemfilteran RBAC elemen UI berdasarkan tindakan yang diotorisasi. Ini sangat berharga untuk aplikasi yang memiliki elemen UI peka konteks yang mungkin terkait dengan pengguna atau penyewa tertentu dalam kasus aplikasi SaaS multi-penyewa.

Dalam contoh berikut, `Users` dari tidak `Role` `viewer` diizinkan untuk melakukan pembaruan. Untuk pengguna ini, UI tidak boleh membuat tombol pembaruan apa pun.



Dalam contoh ini, aplikasi web satu halaman memiliki empat tombol. Tombol mana yang terlihat tergantung pada `Role` pengguna yang saat ini masuk ke aplikasi. Saat aplikasi web satu halaman

merender UI, ia menanyakan Izin Terverifikasi untuk menentukan tindakan mana yang diizinkan oleh pengguna untuk dilakukan, dan kemudian menghasilkan tombol berdasarkan keputusan otorisasi.

Kebijakan berikut menetapkan bahwa tipe `Role` dengan nilai `viewer` dapat menampilkan pengguna dan data. Keputusan `ALLOW` otorisasi untuk kebijakan ini memerlukan `viewUsers` tindakan `viewData` atau, dan juga memerlukan sumber daya untuk dikaitkan dengan jenis `Data` atau`Users`. `ALLOW`Keputusan memungkinkan UI untuk membuat dua tombol: `viewDataButton` dan`viewUsersButton`.

```
permit (
    principal in GuiAPP::Role::"viewer",
    action in [GuiAPP::Action::"viewData", GuiAPP::Action::"viewUsers"],
    resource
)
when {
   resource in [GuiAPP::Type::"Data", GuiAPP::Type::"Users"]
};
```

Kebijakan berikut menetapkan bahwa tipe `Role` dengan nilai hanya `viewerDataOnly` dapat menampilkan data. Keputusan `ALLOW` otorisasi untuk kebijakan ini memerlukan `viewData` tindakan, dan juga memerlukan sumber daya untuk dikaitkan dengan jenisnya`Data`. `ALLOW`Keputusan memungkinkan UI untuk merender tombol`viewDataButton`.

```
permit (
    principal in GuiApp::Role::"viewerDataOnly",
    action in [GuiApp::Action::"viewData"],
    resource in [GuiApp::Type::"Data"]
);
```

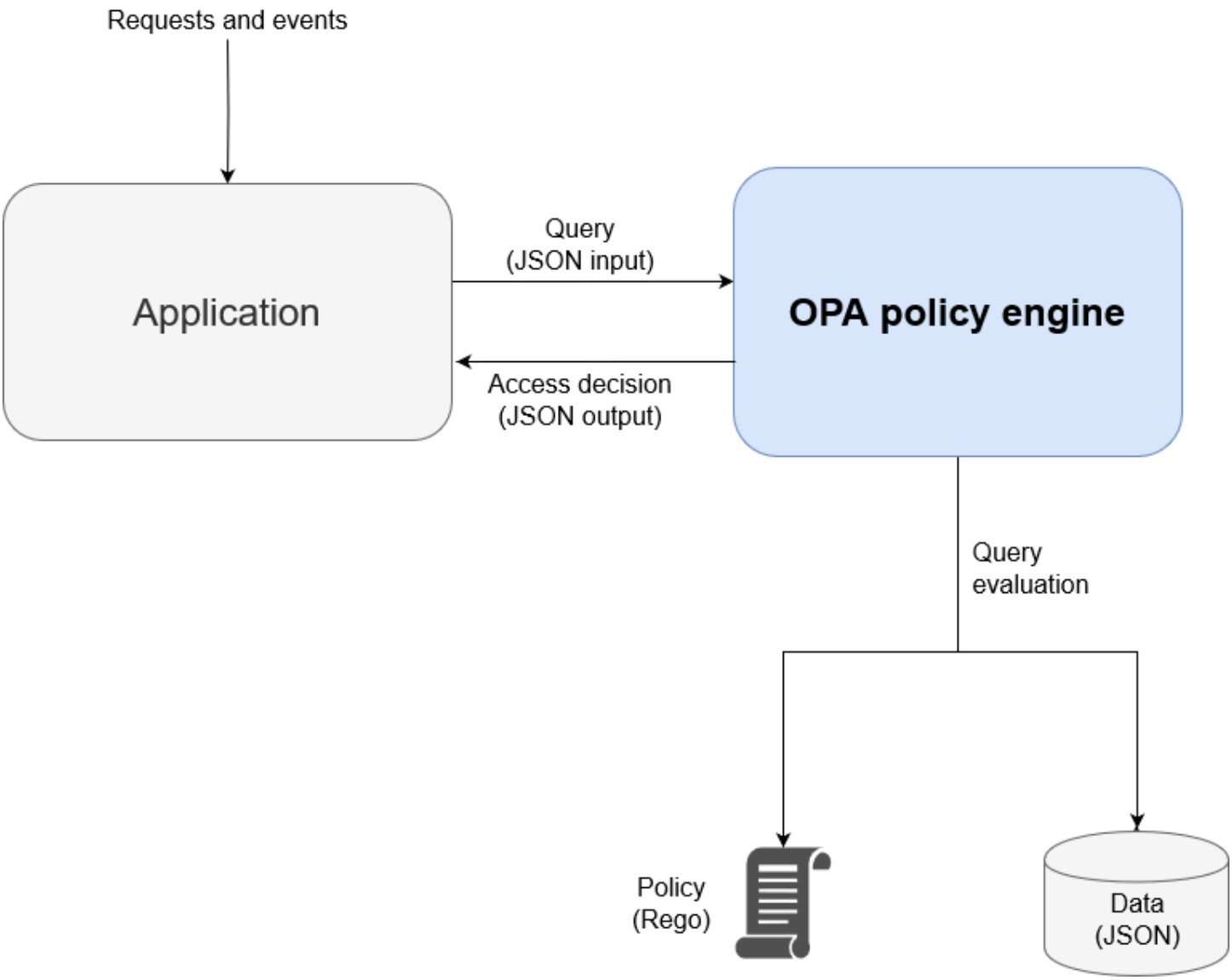
Kebijakan berikut menetapkan bahwa tipe `Role` dengan nilai `admin` dapat mengedit dan menampilkan data dan pengguna. Keputusan `ALLOW` otorisasi untuk kebijakan ini memerlukan tindakan`updateData`,, `viewData`, atau `updateUsers``viewUsers`, dan juga memerlukan sumber daya untuk dikaitkan dengan jenis `Data` atau`Users`. `ALLOW`Keputusan memungkinkan UI untuk merender keempat tombol:`updateDataButton`,, `updateUsersButton``viewDataButton`, dan`viewUsersButton`.

```
permit (
    principal in GuiApp::Role::"admin",
    action in [
```

```
        GuiApp::Action::"updateData",
        GuiApp::Action::"updateUsers",
        GuiApp::Action::"viewData",
        GuiApp::Action::"viewUsers"
       ],
    resource
)
when {
   resource in [GuiApp::Type::"Data", GuiApp::Type::"Users"]
};
```

Menerapkan PDP dengan menggunakan OPA

Open Policy Agent (OPA) adalah mesin kebijakan open-source dan tujuan umum. OPA memiliki banyak kasus penggunaan, tetapi kasus penggunaan yang relevan untuk implementasi PDP adalah kemampuannya untuk memisahkan logika otorisasi dari aplikasi. Ini disebut decoupling kebijakan. OPA berguna dalam menerapkan PDP karena beberapa alasan. Ini menggunakan bahasa deklaratif tingkat tinggi yang disebut Rego untuk menyusun kebijakan dan aturan. Kebijakan dan aturan ini ada secara terpisah dari aplikasi dan dapat membuat keputusan otorisasi tanpa logika khusus aplikasi. OPA juga mengekspos RESTful API untuk membuat pengambilan keputusan otorisasi menjadi sederhana dan mudah. Untuk membuat keputusan otorisasi, aplikasi menanyakan OPA dengan input JSON, dan OPA mengevaluasi input terhadap kebijakan yang ditentukan untuk mengembalikan keputusan akses di JSON. OPA juga mampu mengimpor data eksternal yang mungkin relevan dalam membuat keputusan otorisasi.



OPA memiliki beberapa keunggulan dibandingkan mesin kebijakan khusus:

- OPA dan evaluasi kebijakannya dengan Rego menyediakan mesin kebijakan pra-bangun yang fleksibel yang hanya memerlukan penyisipan kebijakan dan data apa pun yang diperlukan untuk membuat keputusan otorisasi. Logika evaluasi kebijakan ini harus dibuat ulang dalam solusi mesin kebijakan khusus.
- OPA menyederhanakan logika otorisasi dengan memiliki kebijakan yang ditulis dalam bahasa deklaratif. Anda dapat memodifikasi dan mengelola kebijakan dan aturan ini secara independen dari kode aplikasi apa pun, tanpa keterampilan pengembangan aplikasi.
- OPA mengekspos RESTful API, yang menyederhanakan integrasi dengan poin penegakan kebijakan (PEP).

- OPA menyediakan dukungan bawaan untuk memvalidasi dan mendekode Token Web JSON (JWT).
- OPA adalah standar otorisasi yang diakui, yang berarti bahwa dokumentasi dan contoh berlimpah jika Anda memerlukan bantuan atau penelitian untuk memecahkan masalah tertentu.
- Mengadopsi standar otorisasi seperti OPA memungkinkan kebijakan yang ditulis dalam Rego untuk dibagikan di seluruh tim terlepas dari bahasa pemrograman yang digunakan oleh aplikasi tim.

Ada dua hal yang tidak disediakan OPA secara otomatis:

- OPA tidak memiliki bidang kontrol yang kuat untuk memperbarui dan mengelola kebijakan. OPA memang menyediakan beberapa pola dasar untuk menerapkan pembaruan kebijakan, pemantauan, dan agregasi log dengan mengekspos API manajemen, tetapi integrasi dengan API ini harus ditangani oleh pengguna OPA. Sebagai praktik terbaik, Anda harus menggunakan pipeline integrasi berkelanjutan dan penerapan berkelanjutan (CI/CD) untuk mengelola, memodifikasi, dan melacak versi kebijakan serta mengelola kebijakan di OPA.
- OPA tidak dapat mengambil data dari sumber eksternal secara default. Sumber data eksternal untuk keputusan otorisasi dapat berupa database yang menyimpan atribut pengguna. Ada beberapa fleksibilitas dalam bagaimana data eksternal diberikan kepada OPA - dapat di-cache secara lokal terlebih dahulu atau diambil secara dinamis dari API ketika keputusan otorisasi diminta - tetapi mendapatkan informasi ini bukanlah sesuatu yang dapat dilakukan OPA atas nama Anda.

Ikhtisar Rego

Rego adalah bahasa kebijakan tujuan umum, yang berarti ia berfungsi untuk setiap lapisan tumpukan dan domain apa pun. Tujuan utama Rego adalah untuk menerima input dan data JSON/YAMB yang dievaluasi untuk membuat keputusan yang memungkinkan kebijakan tentang sumber daya infrastruktur, identitas, dan operasi. Rego memungkinkan Anda untuk menulis kebijakan tentang setiap lapisan tumpukan atau domain tanpa memerlukan perubahan atau ekstensi bahasa. Berikut adalah beberapa contoh keputusan yang dapat dibuat oleh Rego:

- Apakah permintaan API ini diizinkan atau ditolak?
- Apa nama host dari server cadangan untuk aplikasi ini?
- Berapa skor risiko untuk perubahan infrastruktur yang diusulkan ini?
- Kluster mana yang harus digunakan wadah ini untuk ketersediaan tinggi?
- Informasi perutean apa yang harus digunakan untuk layanan mikro ini?

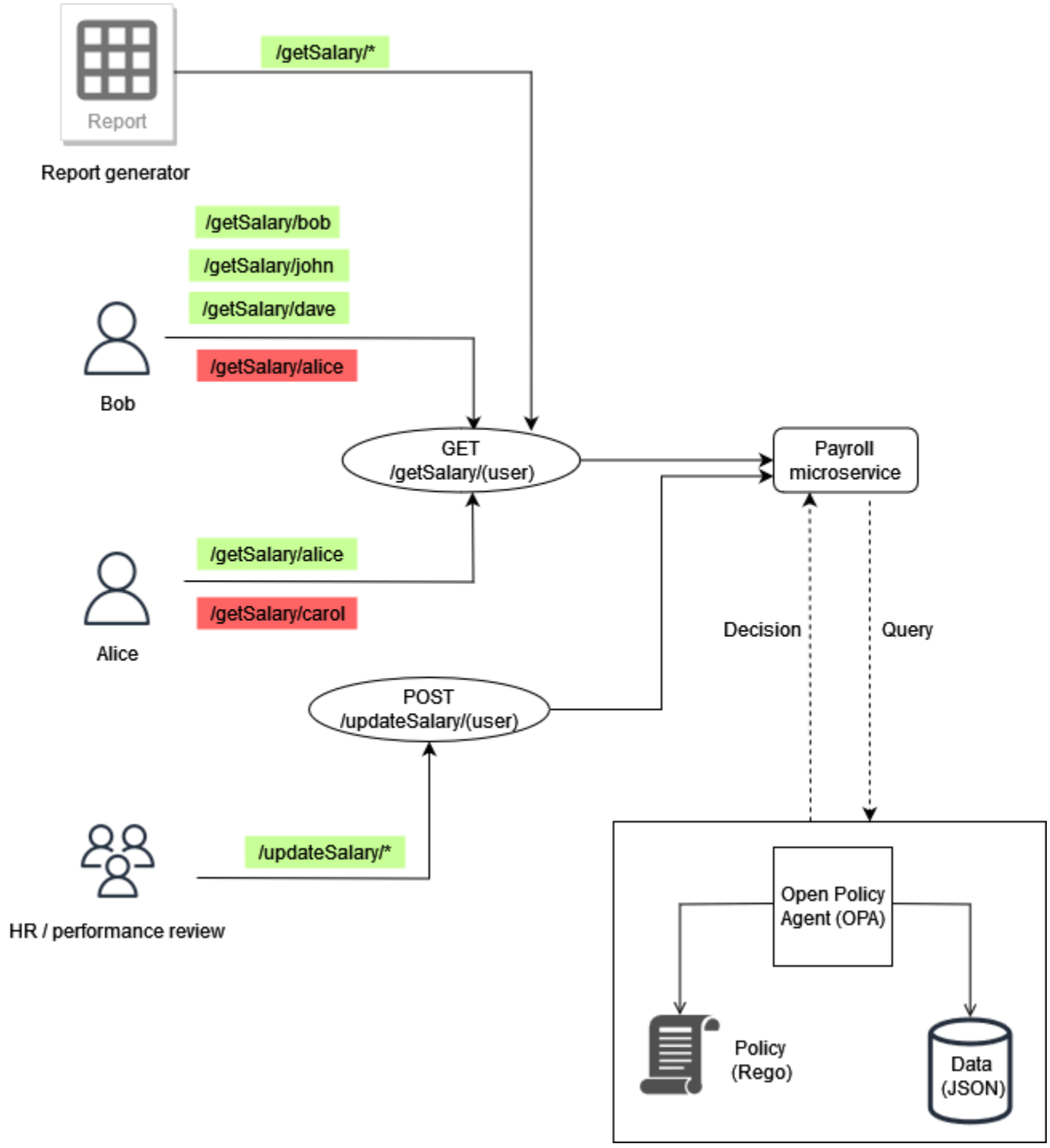
Untuk menjawab pertanyaan-pertanyaan ini, Rego menggunakan filosofi dasar tentang bagaimana keputusan ini dapat dibuat. Dua prinsip utama saat menyusun kebijakan di Rego adalah:

- Setiap sumber daya, identitas, atau operasi dapat direpresentasikan sebagai data JSON atau YAMM.
- Kebijakan adalah logika yang diterapkan pada data.

Rego membantu sistem perangkat lunak membuat keputusan otorisasi dengan mendefinisikan logika tentang bagaimana input data JSON/YAMB dievaluasi. Bahasa pemrograman seperti C, Java, Go, dan Python adalah solusi biasa untuk masalah ini, tetapi Rego dirancang untuk fokus pada data dan input yang mewakili sistem Anda, dan logika untuk membuat keputusan kebijakan dengan informasi ini.

Contoh 1: ABAC Dasar dengan OPA dan Rego

Bagian ini menjelaskan skenario di mana OPA digunakan untuk membuat keputusan akses tentang pengguna mana yang diizinkan mengakses informasi dalam layanan mikro Payroll fiksi. Cuplikan kode Rego disediakan untuk menunjukkan bagaimana Anda dapat menggunakan Rego untuk membuat keputusan kontrol akses. Contoh-contoh ini tidak lengkap atau eksplorasi penuh kemampuan Rego dan OPA. Untuk ikhtisar Rego yang lebih menyeluruh, kami sarankan Anda berkonsultasi dengan dokumentasi Rego di situs web OPA.



Contoh aturan OPA dasar

Pada diagram sebelumnya, salah satu aturan kontrol akses yang diberlakukan oleh OPA untuk layanan mikro Payroll adalah:

Karyawan dapat membaca gaji mereka sendiri.

Jika Bob mencoba mengakses layanan mikro Payroll untuk melihat gajinya sendiri, layanan mikro Payroll dapat mengarahkan panggilan API ke OPA RESTful API untuk membuat keputusan akses. Layanan Payroll menanyakan OPA untuk keputusan dengan masukan JSON berikut:

```
{
    "user": "bob",
    "method": "GET",
    "path": ["getSalary", "bob"]
}
```

OPA memilih kebijakan atau kebijakan berdasarkan kueri. Dalam hal ini, kebijakan berikut, yang ditulis dalam Rego, mengevaluasi input JSON.

```
default allow = false
allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    input.user == user
}
```

Kebijakan ini menolak akses secara default. Kemudian mengevaluasi input dalam kueri dengan mengikatnya ke variabel `input` global. Operator titik digunakan dengan variabel ini untuk mengakses nilai variabel. Aturan Rego `allow` mengembalikan nilai true jika ekspresi dalam aturan juga benar. Aturan Rego memverifikasi bahwa input `method` dalam sama dengan GET. Kemudian memverifikasi bahwa elemen pertama dalam daftar `path` adalah `getSalary` sebelum menetapkan elemen kedua dalam daftar ke variabel. `user` Terakhir, ia memeriksa bahwa jalur yang diakses adalah `/getSalary/bob` dengan memeriksa apakah `user` membuat permintaan,`input.user`, cocok dengan `user` variabel. Aturan `allow` menerapkan logika jika-maka untuk mengembalikan nilai Boolean, seperti yang ditunjukkan pada output:

```
{
    "allow": true
}
```

Aturan sebagian menggunakan data eksternal

Untuk menunjukkan kemampuan OPA tambahan, Anda dapat menambahkan persyaratan ke aturan akses yang Anda terapkan. Mari kita asumsikan bahwa Anda ingin menerapkan persyaratan kontrol akses ini dalam konteks ilustrasi sebelumnya:

Karyawan dapat membaca gaji siapa saja yang melapor kepada mereka.

Dalam contoh ini, OPA memiliki akses ke data eksternal yang dapat diimpor untuk membantu membuat keputusan akses:

```
"managers": {
        "bob": ["dave", "john"],
        "carol": ["alice"]
}
```

Anda dapat menghasilkan respons JSON arbitrer dengan membuat aturan parsi di OPA, yang mengembalikan sekumpulan nilai alih-alih respons tetap. Ini adalah contoh aturan sebagian:

```
direct_report[user_ids] {
    user_ids = data.managers[input.user][_]
}
```

Aturan ini mengembalikan satu set semua pengguna yang melaporkan ke nilai`input.user`, yang, dalam hal ini, adalah`bob`. `[_]`Konstruk dalam aturan digunakan untuk mengulangi nilai-nilai himpunan. Ini adalah output dari aturan:

```
{
    "direct_report": [
      "dave",
      "john"
    ]
}
```

Mengambil informasi ini dapat membantu menentukan apakah pengguna adalah laporan langsung dari seorang manajer. Untuk beberapa aplikasi, mengembalikan JSON dinamis lebih disukai daripada mengembalikan respons Boolean sederhana.

Menyatukan semuanya

Persyaratan akses terakhir lebih kompleks daripada dua yang pertama karena menggabungkan kondisi yang ditentukan dalam kedua persyaratan:

Karyawan dapat membaca gaji mereka sendiri dan gaji siapa saja yang melapor kepada mereka.

Untuk memenuhi persyaratan ini, Anda dapat menggunakan kebijakan Rego ini:

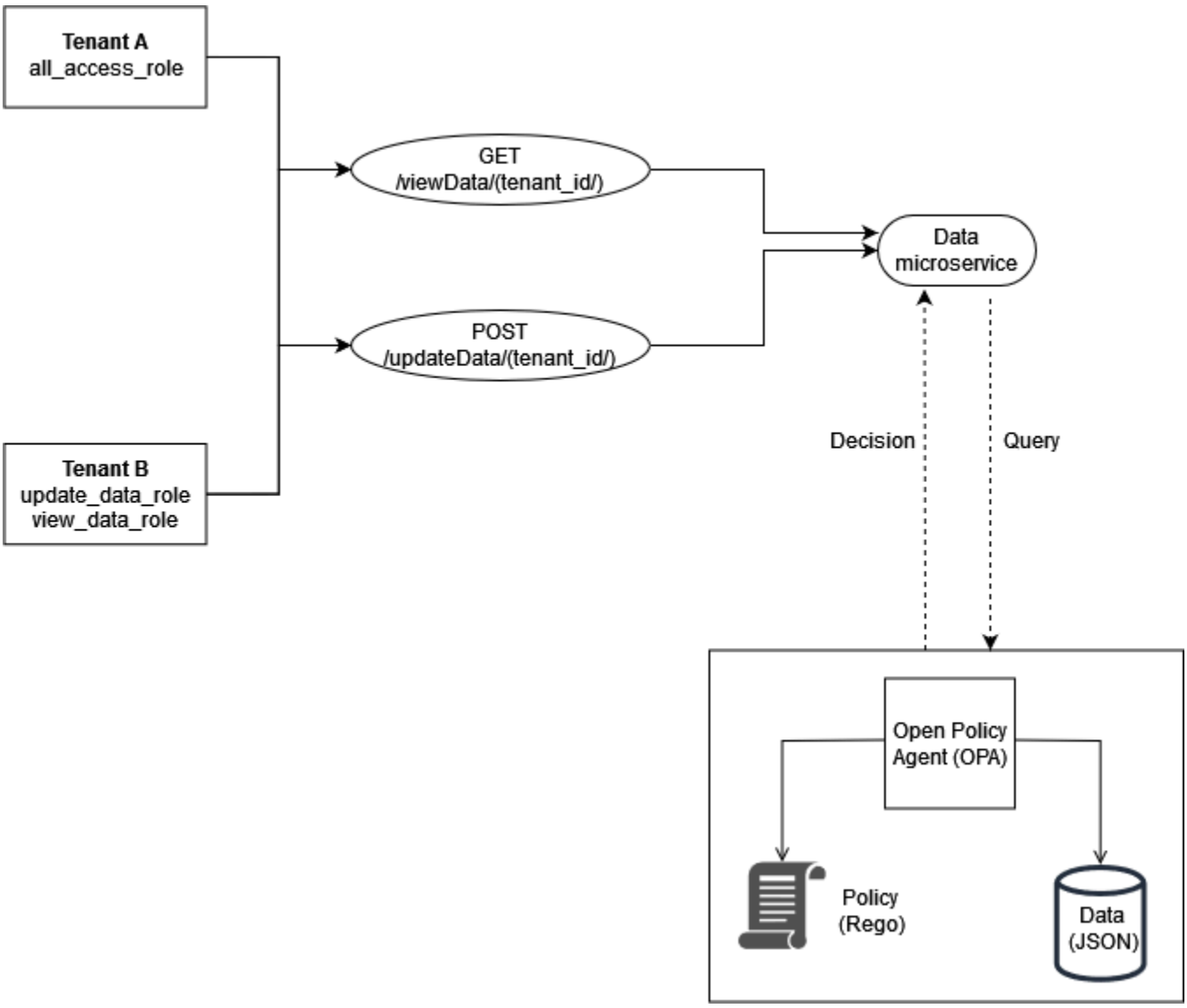
```
default allow = false

allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    input.user == user
}

allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    managers := data.managers[input.user][_]
    contains(managers, user)
}
```

Aturan pertama dalam kebijakan memungkinkan akses untuk setiap pengguna yang mencoba melihat informasi gaji mereka sendiri, seperti yang dibahas sebelumnya. Memiliki dua aturan dengan nama yang sama,`allow`, berfungsi sebagai logika atau operator di Rego. Aturan kedua mengambil daftar semua laporan langsung yang terkait dengan `input.user` (dari data dalam diagram sebelumnya) dan menetapkan daftar ini ke variabel. `managers` Terakhir, aturan memeriksa apakah pengguna yang mencoba melihat gaji mereka adalah laporan langsung `input.user` dengan memverifikasi bahwa nama mereka terkandung dalam `managers` variabel.

Contoh-contoh di bagian ini sangat mendasar dan tidak memberikan eksplorasi lengkap atau menyeluruh tentang kemampuan Rego dan OPA. Untuk informasi lebih lanjut, tinjau dokumentasi OPA, lihat file OPA GitHub README, dan bereksperimen di taman bermain Rego.

Contoh 2: Kontrol akses multi-penyewa dan RBAC yang ditentukan pengguna dengan OPA dan Rego

Contoh ini menggunakan OPA dan Rego untuk mendemonstrasikan bagaimana kontrol akses dapat diimplementasikan pada API untuk aplikasi multi-penyewa dengan peran khusus yang ditentukan oleh pengguna penyewa. Ini juga menunjukkan bagaimana akses dapat dibatasi berdasarkan penyewa. Model ini menunjukkan bagaimana OPA dapat membuat keputusan izin terperinci berdasarkan informasi yang diberikan dalam peran tingkat tinggi.



Peran untuk penyewa disimpan dalam data eksternal (data RBAC) yang digunakan untuk membuat keputusan akses untuk OPA:

```
{
    "roles": {
        "tenant_a": {
            "all_access_role": ["viewData", "updateData"]
        },
        "tenant_b": {
            "update_data_role": ["updateData"],
            "view_data_role": ["viewData"]
        }
    }
```

```
}
```

Peran ini, ketika didefinisikan oleh pengguna penyewa, harus disimpan dalam sumber data eksternal atau penyedia identitas (idP) yang dapat bertindak sebagai sumber kebenaran saat memetakan peran yang ditentukan penyewa ke izin dan penyewa itu sendiri.

Contoh ini menggunakan dua kebijakan dalam OPA untuk membuat keputusan otorisasi dan untuk memeriksa bagaimana kebijakan ini menegakkan isolasi penyewa. Kebijakan ini menggunakan data RBAC yang ditentukan sebelumnya.

```
default allowViewData = false
allowViewData = true {
    input.method == "GET"
    input.path = ["viewData", tenant_id]
    input.tenant_id == tenant_id
    role_permissions := data.roles[input.tenant_id][input.role][_]
    contains(role_permissions, "viewData")
}
```

Untuk menunjukkan bagaimana aturan ini akan berfungsi, pertimbangkan kueri OPA yang memiliki masukan berikut:

```
{
    "tenant_id": "tenant_a",
    "role": "all_access_role",
    "path": ["viewData", "tenant_a"],
    "method": "GET"
}
```

Keputusan otorisasi untuk panggilan API ini dibuat sebagai berikut, dengan menggabungkan data RBAC, kebijakan OPA, dan input kueri OPA:

1. Seorang pengguna dari `Tenant A` membuat panggilan API ke`/viewData/tenant_a`.
2. Layanan mikro Data menerima panggilan dan menanyakan `allowViewData` aturan, meneruskan input yang ditunjukkan dalam contoh input kueri OPA.
3. OPA menggunakan aturan kueri dalam kebijakan OPA untuk mengevaluasi input yang diberikan. OPA juga menggunakan data dari data RBAC untuk mengevaluasi input. OPA melakukan hal berikut:

a. Memverifikasi bahwa metode yang digunakan untuk melakukan panggilan API adalah`GET`.
b. Memverifikasi bahwa jalur yang diminta adalah`viewData`.
c. Memeriksa bahwa `tenant_id` di jalur sama dengan yang `input.tenant_id` terkait dengan pengguna. Ini memastikan bahwa isolasi penyewa dipertahankan. Penyewa lain, bahkan dengan peran yang identik, tidak dapat diotorisasi dalam melakukan panggilan API ini.
d. Menarik daftar izin peran dari data eksternal peran dan menetapkannya ke variabel. `role_permissions` Daftar ini diambil dengan menggunakan peran yang ditentukan penyewa yang dikaitkan dengan pengguna di `input.role.`
e. Memeriksa `role_permissions` untuk melihat apakah itu berisi izin `viewData.`

4. OPA mengembalikan keputusan berikut ke layanan mikro Data:

```
{
    "allowViewData": true
}
```

Proses ini menunjukkan bagaimana RBAC dan kesadaran penyewa dapat berkontribusi untuk membuat keputusan otorisasi dengan OPA. Untuk mengilustrasikan poin ini lebih lanjut, pertimbangkan panggilan API `/viewData/tenant_b` dengan input kueri berikut:

```
{
    "tenant_id": "tenant_b",
    "role": "view_data_role",
    "path": ["viewData", "tenant_b"],
    "method": "GET"
}
```

Aturan ini akan mengembalikan output yang sama dengan input kueri OPA meskipun untuk penyewa berbeda yang memiliki peran berbeda. Ini karena panggilan ini untuk `/tenant_b` dan data `view_data_role` dalam RBAC masih memiliki `viewData` izin yang terkait dengannya. Untuk menerapkan jenis kontrol akses yang sama`/updateData`, Anda dapat menggunakan aturan OPA yang serupa:

```
default allowUpdateData = false
allowUpdateData = true {
    input.method == "POST"
    input.path = ["updateData", tenant_id]
```

```
    input.tenant_id == tenant_id
    role_permissions := data.roles[input.tenant_id][input.role][_]
    contains(role_permissions, "updateData")
}
```

Aturan ini secara fungsional sama dengan `allowViewData` aturan, tetapi memverifikasi jalur dan metode input yang berbeda. Aturan masih memastikan isolasi penyewa dan memeriksa apakah peran yang ditentukan penyewa memberikan izin pemanggil API. Untuk melihat bagaimana hal ini dapat diterapkan, periksa input kueri berikut untuk panggilan API ke`/updateData/tenant_b`:

```
{
    "tenant_id": "tenant_b",
    "role": "view_data_role",
    "path": ["updateData", "tenant_b"],
    "method": "POST"
}
```

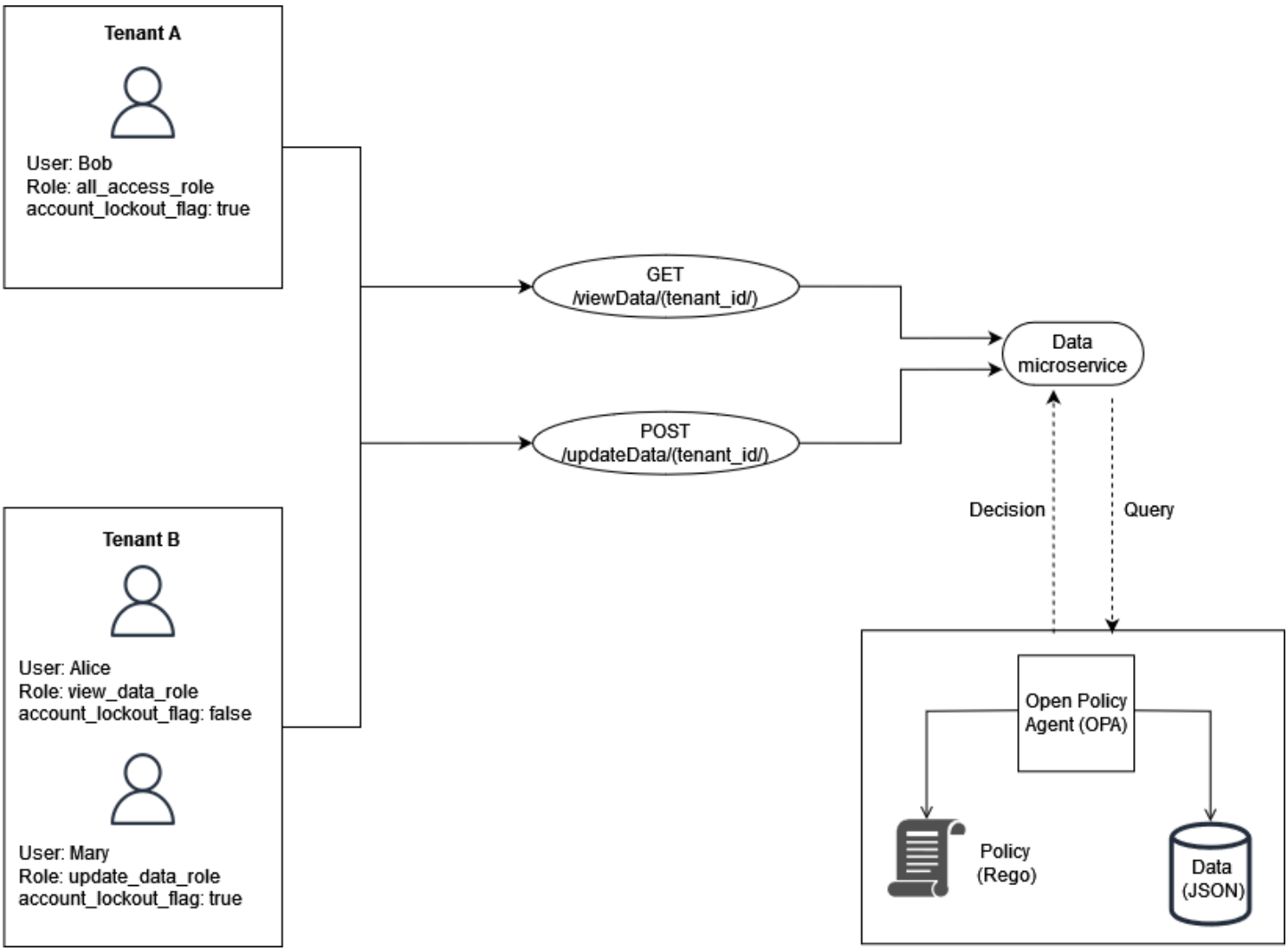
Input kueri ini, ketika dievaluasi dengan `allowUpdateData` aturan, mengembalikan keputusan otorisasi berikut:

```
{
    "allowUpdateData": false
}
```

Panggilan ini tidak akan diizinkan. Meskipun pemanggil API dikaitkan dengan yang benar `tenant_id` dan memanggil API dengan menggunakan metode yang disetujui, pemanggil `input.role` adalah yang ditentukan oleh penyewa`view_data_role`. `view_data_role`Tidak memiliki `updateData` izin; oleh karena itu, panggilan ke `/updateData` tidak sah. Panggilan ini akan berhasil bagi `tenant_b` pengguna yang memilikiupdate_data_role.

Contoh 3: Kontrol akses multi-tenant untuk RBAC dan ABAC dengan OPA dan Rego

Untuk meningkatkan contoh RBAC di bagian sebelumnya, Anda dapat menambahkan atribut ke pengguna.



Contoh ini mencakup peran yang sama dari contoh sebelumnya, tetapi menambahkan atribut penggunaaccount_lockout_flag. Ini adalah atribut khusus pengguna yang tidak terkait dengan peran tertentu. Anda dapat menggunakan data eksternal RBAC yang sama dengan yang Anda gunakan sebelumnya untuk contoh ini:

```
{
   "roles": {
       "tenant_a": {
           "all_access_role": ["viewData", "updateData"]
       },
       "tenant_b": {
           "update_data_role": ["updateData"],
           "view_data_role": ["viewData"]
       }
   }
```

```
}
```

Atribut `account_lockout_flag` pengguna dapat diteruskan ke layanan Data sebagai bagian dari input ke kueri OPA `/viewData/tenant_a` untuk pengguna Bob:

```
 {
    "tenant_id": "tenant_a",
    "role": "all_access_role",
    "path": ["viewData", "tenant_a"],
    "method": "GET",
    "account_lockout_flag": "true"
}
```

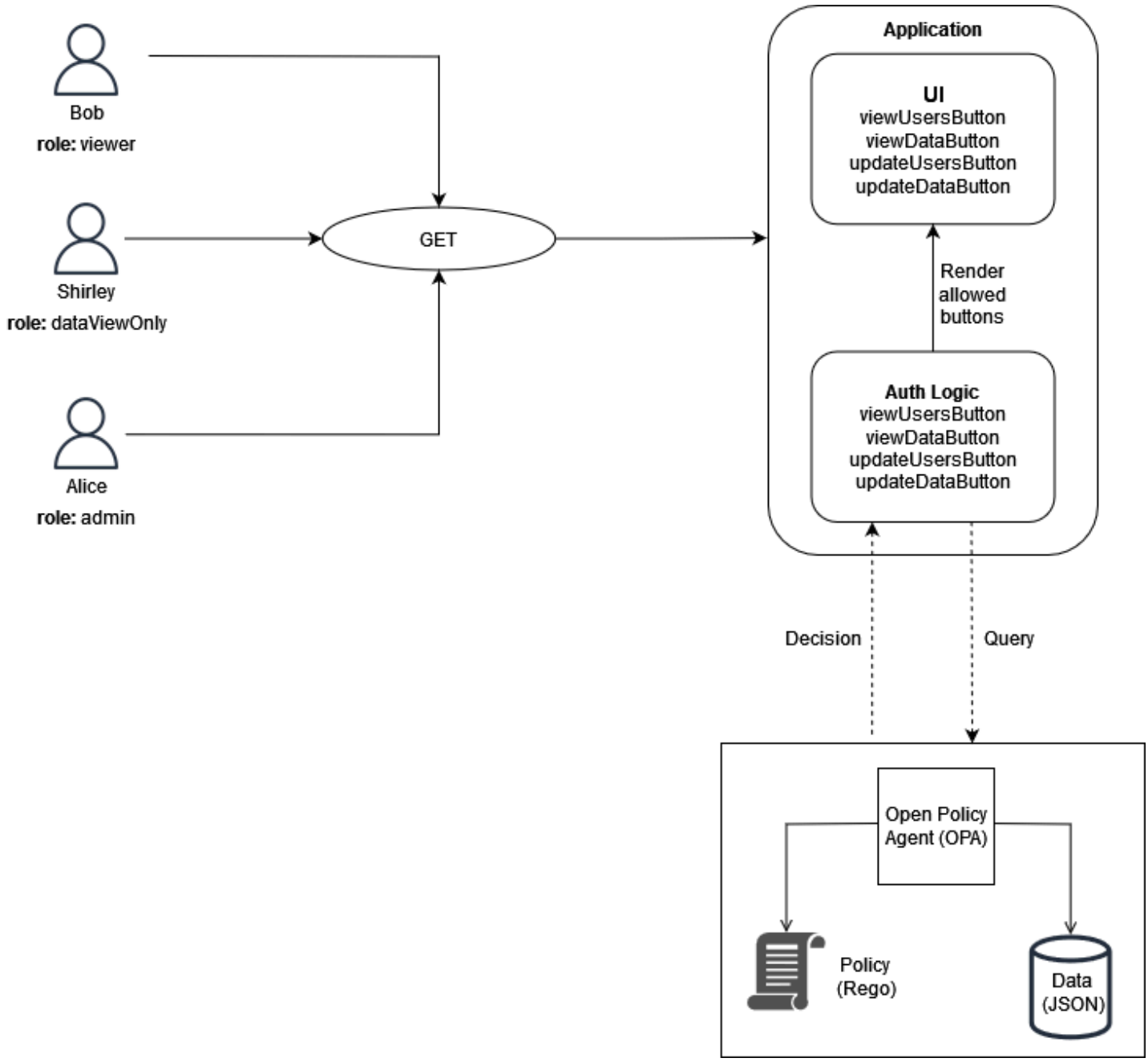
Aturan yang ditanyakan untuk keputusan akses mirip dengan contoh sebelumnya, tetapi menyertakan baris tambahan untuk memeriksa `account_lockout_flag` atribut:

```
default allowViewData = false
allowViewData = true {
    input.method == "GET"
    input.path = ["viewData", tenant_id]
    input.tenant_id == tenant_id
    role_permissions := data.roles[input.tenant_id][input.role][_]
    contains(role_permissions, "viewData")
    input.account_lockout_flag == "false"
}
```

Query ini mengembalikan keputusan otorisasi. `false` Ini karena `true` untuk Bob, dan aturan Rego `allowViewData` menolak akses meskipun Bob memiliki peran dan penyewa yang benar. `account_lockout_flag attribute`

Contoh 4: Pemfilteran UI dengan OPA dan Rego

Fleksibilitas OPA dan Rego mendukung kemampuan untuk memfilter elemen UI. Contoh berikut menunjukkan bagaimana aturan sebagian OPA dapat membuat keputusan otorisasi tentang elemen mana yang harus ditampilkan di UI dengan RBAC. Metode ini adalah salah satu dari banyak cara berbeda Anda dapat memfilter elemen UI dengan OPA.



Dalam contoh ini, aplikasi web satu halaman memiliki empat tombol. Katakanlah Anda ingin memfilter UI Bob, Shirley, dan Alice sehingga mereka hanya dapat melihat tombol yang sesuai dengan peran mereka. Ketika UI menerima permintaan dari pengguna, itu menanyakan aturan paral OPA untuk menentukan tombol mana yang harus ditampilkan di UI. Kueri meneruskan yang berikut ini sebagai input ke OPA saat Bob (dengan peran`viewer`) membuat permintaan ke UI:

```
{
    "role": "viewer"
```

```
}
```

OPA menggunakan data eksternal terstruktur untuk RBAC untuk membuat keputusan akses:

```
{
    "roles": {
        "viewer": ["viewUsers", "viewData"],
        "dataViewOnly": ["viewData"],
        "admin": ["viewUsers", "viewData", "updateUsers", "updateData"]
    }
}
```

Aturan paral OPA menggunakan data eksternal dan input untuk menghasilkan daftar tindakan yang diizinkan:

```
user_permissions[permissions] {
    permissions := data.roles[input.role][_]
}
```

Dalam aturan sebagian, OPA menggunakan yang `input.role` ditentukan sebagai bagian dari kueri untuk menentukan tombol mana yang harus ditampilkan. Bob memiliki peran`viewer`, dan data eksternal menentukan bahwa pemirsa memiliki dua izin: `viewUsers` dan. `viewData` Oleh karena itu, output dari aturan ini untuk Bob (dan untuk pengguna lain yang memiliki peran pemirsa) adalah sebagai berikut:

```
{
    "user_permissions": [
        "viewData",
        "viewUsers"
    ]
}
```

Output untuk Shirley, yang memiliki `dataViewOnly` peran, akan berisi tombol izin:. `viewData` Output untuk Alice, yang memiliki `admin` peran, akan berisi semua izin ini. Tanggapan ini dikembalikan ke UI saat OPA ditanyakan. `user_permissions` Aplikasi kemudian dapat menggunakan respons ini untuk menyembunyikan atau menampilkan`viewUsersButton`, `viewDataButton``updateUsersButton`,, dan`updateDataButton`.

Menggunakan mesin kebijakan khusus

Metode alternatif untuk menerapkan PDP adalah dengan membuat mesin kebijakan khusus. Tujuan dari mesin kebijakan ini adalah untuk memisahkan logika otorisasi dari aplikasi. Mesin kebijakan khusus bertanggung jawab untuk membuat keputusan otorisasi, mirip dengan Izin Terverifikasi atau OPA, untuk mencapai pemisahan kebijakan. Perbedaan utama antara solusi ini dan penggunaan Izin Terverifikasi atau OPA adalah logika untuk menulis dan mengevaluasi kebijakan dibuat khusus untuk mesin kebijakan khusus. Setiap interaksi dengan mesin harus diekspos melalui API atau metode lain untuk memungkinkan keputusan otorisasi untuk mencapai aplikasi. Anda dapat menulis mesin kebijakan khusus dalam bahasa pemrograman apa pun atau menggunakan mekanisme lain untuk evaluasi kebijakan, seperti Common Expression Language (CEL).

Menerapkan PEP

Policy enforcement point (PEP) bertanggung jawab untuk menerima permintaan otorisasi yang dikirim ke Policy Decision Point (PDP) untuk evaluasi. PEP dapat berada di mana saja dalam aplikasi di mana data dan sumber daya harus dilindungi, atau di mana logika otorisasi diterapkan. PEP relatif sederhana dibandingkan dengan PDP. PEP bertanggung jawab hanya untuk meminta dan mengevaluasi keputusan otorisasi dan tidak memerlukan logika otorisasi apa pun. PEP, tidak seperti PDP, tidak dapat dipusatkan dalam aplikasi SaaS. Ini karena otorisasi dan kontrol akses diperlukan untuk diimplementasikan di seluruh aplikasi dan titik aksesnya. PEP dapat diterapkan ke API, layanan mikro, lapisan Backend for Frontend (BFF), atau titik mana pun dalam aplikasi di mana kontrol akses diinginkan atau diperlukan. Membuat PEP meresap dalam aplikasi memastikan bahwa otorisasi sering diverifikasi dan independen di beberapa titik.

Untuk menerapkan PEP, langkah pertama adalah menentukan di mana penegakan kontrol akses harus terjadi dalam suatu aplikasi. Pertimbangkan prinsip ini ketika memutuskan di mana PEP harus diintegrasikan ke dalam aplikasi Anda:

Jika aplikasi mengekspos API, harus ada otorisasi dan kontrol akses pada API itu.

Ini karena dalam arsitektur berorientasi layanan mikro atau berorientasi layanan, API berfungsi sebagai pemisah antara fungsi aplikasi yang berbeda. Masuk akal untuk memasukkan kontrol akses sebagai pos pemeriksaan logis antara fungsi aplikasi. Kami sangat menyarankan agar Anda menyertakan PEP sebagai prasyarat untuk akses ke setiap API dalam aplikasi SaaS. Dimungkinkan juga untuk mengintegrasikan otorisasi di titik lain dalam suatu aplikasi. Dalam aplikasi monolitik, mungkin perlu memiliki PEP terintegrasi dalam logika aplikasi itu sendiri. Tidak ada lokasi tunggal di mana PEP harus disertakan, tetapi pertimbangkan untuk menggunakan prinsip API sebagai titik awal.

Meminta keputusan otorisasi

PEP harus meminta keputusan otorisasi dari PDP. Permintaan dapat mengambil beberapa bentuk. Metode termudah dan paling mudah diakses untuk meminta keputusan otorisasi adalah mengirim permintaan otorisasi atau kueri ke RESTful API yang diekspos oleh PDP (OPA atau Izin Terverifikasi). Jika Anda menggunakan Izin Terverifikasi, Anda juga dapat memanggil IsAuthorizedmetode dengan menggunakan AWS SDK untuk mengambil keputusan otorisasi. Satu-satunya fungsi PEP dalam pola ini adalah untuk meneruskan informasi yang dibutuhkan oleh permintaan otorisasi atau kueri. Ini bisa sesederhana meneruskan permintaan yang diterima oleh API sebagai input ke PDP. Ada metode lain untuk membuat PEP. Misalnya, Anda dapat

mengintegrasikan PDP OPA secara lokal dengan aplikasi yang ditulis dalam bahasa pemrograman Go sebagai pustaka alih-alih menggunakan API.

Mengevaluasi keputusan otorisasi

PEP perlu menyertakan logika untuk mengevaluasi hasil keputusan otorisasi. Ketika PDP diekspos sebagai API, keputusan otorisasi kemungkinan dalam format JSON dan dikembalikan oleh panggilan API. PEP harus mengevaluasi kode JSON ini untuk menentukan apakah tindakan yang diambil diotorisasi. Misalnya, jika PDP dirancang untuk memberikan Boolean mengizinkan atau menolak keputusan otorisasi, PEP mungkin hanya memeriksa nilai ini, dan kemudian mengembalikan kode status HTTP 200 untuk mengizinkan dan kode status HTTP 403 untuk penolakan. Pola menggabungkan PEP sebagai prasyarat untuk mengakses API adalah pola yang mudah diimplementasikan dan sangat efektif untuk menerapkan kontrol akses di seluruh aplikasi SaaS. Dalam skenario yang lebih rumit, PEP mungkin bertanggung jawab untuk mengevaluasi kode JSON arbitrer yang dikembalikan oleh PDP. PEP harus ditulis untuk memasukkan logika apa pun yang diperlukan untuk menafsirkan keputusan otorisasi yang dikembalikan PDP. Karena PEP kemungkinan akan diimplementasikan di banyak tempat berbeda dalam aplikasi Anda, kami sarankan Anda mengemas kode PEP Anda sebagai pustaka atau artefak yang dapat digunakan kembali dalam bahasa pemrograman pilihan Anda. Dengan cara ini, PEP Anda dapat dengan mudah diintegrasikan kapan saja dalam aplikasi Anda dengan pengerjaan ulang minimal.

Model desain untuk arsitektur SaaS multi-penyewa

Ada banyak cara untuk menerapkan kontrol akses API dan otorisasi. Panduan ini berfokus pada tiga model desain yang efektif untuk arsitektur SaaS multi-penyewa. Desain ini berfungsi sebagai referensi tingkat tinggi untuk implementasi titik keputusan kebijakan (PDP) dan poin penegakan kebijakan (PEP), untuk membentuk model otorisasi yang kohesif dan ada di mana-mana untuk aplikasi.

Model desain:

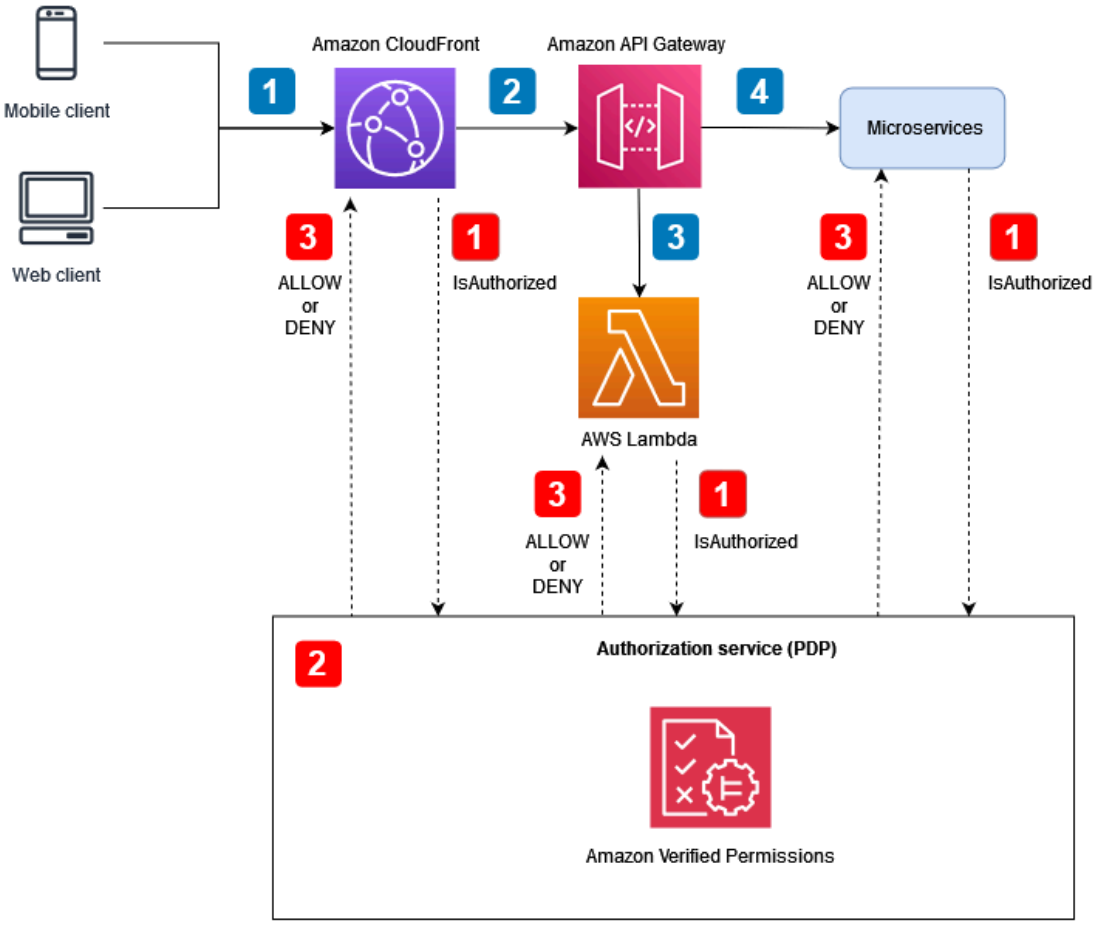
- Model desain untuk Izin Terverifikasi Amazon
- Model desain untuk OPA

Model desain untuk Izin Terverifikasi Amazon

Menggunakan PDP terpusat dengan PEP pada API

Titik keputusan kebijakan terpusat (PDP) dengan poin penegakan kebijakan (PEP) pada model API mengikuti praktik terbaik industri untuk menciptakan sistem yang efektif dan mudah dipelihara untuk kontrol akses dan otorisasi API. Pendekatan ini mendukung beberapa prinsip utama:

- Otorisasi dan kontrol akses API diterapkan di beberapa titik dalam aplikasi.
- Logika otorisasi tidak tergantung pada aplikasi.
- Keputusan kontrol akses terpusat.



Alur aplikasi (diilustrasikan dengan callout bernomor biru dalam diagram):

1. Pengguna yang diautentikasi dengan JSON Web Token (JWT) menghasilkan permintaan HTTP ke Amazon. CloudFront
2. CloudFront meneruskan permintaan ke Amazon API Gateway, yang dikonfigurasi sebagai CloudFront asal.
3. Authorizer kustom API Gateway dipanggil untuk memverifikasi JWT.
4. Layanan mikro menanggapi permintaan tersebut.

Alur kontrol akses otorisasi dan API (diilustrasikan dengan callout bernomor merah dalam diagram):

1. PEP memanggil layanan otorisasi dan meneruskan data permintaan, termasuk JWT apa pun.
2. Layanan otorisasi (PDP), dalam hal ini Izin Terverifikasi, menggunakan data permintaan sebagai input kueri dan mengevaluasinya berdasarkan kebijakan relevan yang ditentukan oleh kueri.
3. Keputusan otorisasi dikembalikan ke PEP dan dievaluasi.

Model ini menggunakan PDP terpusat untuk membuat keputusan otorisasi. PEP diimplementasikan pada titik yang berbeda untuk membuat permintaan otorisasi ke PDP. Diagram berikut menunjukkan bagaimana Anda dapat menerapkan model ini dalam aplikasi SaaS multi-penyewa hipotetis.

Dalam arsitektur ini, PEP meminta keputusan otorisasi di titik akhir layanan untuk Amazon CloudFront dan Amazon API Gateway dan untuk setiap layanan mikro. Keputusan otorisasi dibuat oleh layanan otorisasi, Izin Terverifikasi Amazon (PDP). Karena Izin Terverifikasi adalah layanan yang dikelola sepenuhnya, Anda tidak perlu mengelola infrastruktur yang mendasarinya. Anda dapat berinteraksi dengan Izin Terverifikasi menggunakan RESTful API atau SDK AWS .

Anda juga dapat menggunakan arsitektur ini dengan mesin kebijakan khusus. Namun, keuntungan apa pun yang diperoleh dari Izin Terverifikasi harus diganti dengan logika yang disediakan oleh mesin kebijakan khusus.

PDP terpusat dengan PEP pada API menyediakan opsi mudah untuk membuat sistem otorisasi yang kuat untuk API. Ini menyederhanakan proses otorisasi dan juga menyediakan antarmuka yang dapat diulang untuk membuat keputusan otorisasi untuk API easy-to-use, layanan mikro, lapisan Backend for Frontend (BFF), atau komponen aplikasi lainnya.

Menggunakan SDK Cedar

Izin Terverifikasi Amazon menggunakan bahasa Cedar untuk mengelola izin berbutir halus di aplikasi kustom Anda. Dengan Izin Terverifikasi, Anda dapat menyimpan kebijakan Cedar di lokasi pusat, memanfaatkan latensi rendah dengan pemrosesan milidetik, dan izin audit di berbagai aplikasi. Anda juga dapat secara opsional mengintegrasikan SDK Cedar langsung ke aplikasi Anda untuk memberikan keputusan otorisasi tanpa menggunakan Izin Terverifikasi. Opsi ini memerlukan pengembangan aplikasi khusus tambahan untuk mengelola dan menyimpan kebijakan untuk kasus penggunaan Anda. Namun, ini bisa menjadi alternatif yang layak, terutama dalam kasus di mana akses ke Izin Terverifikasi terputus-putus atau tidak mungkin karena konektivitas internet yang tidak konsisten.

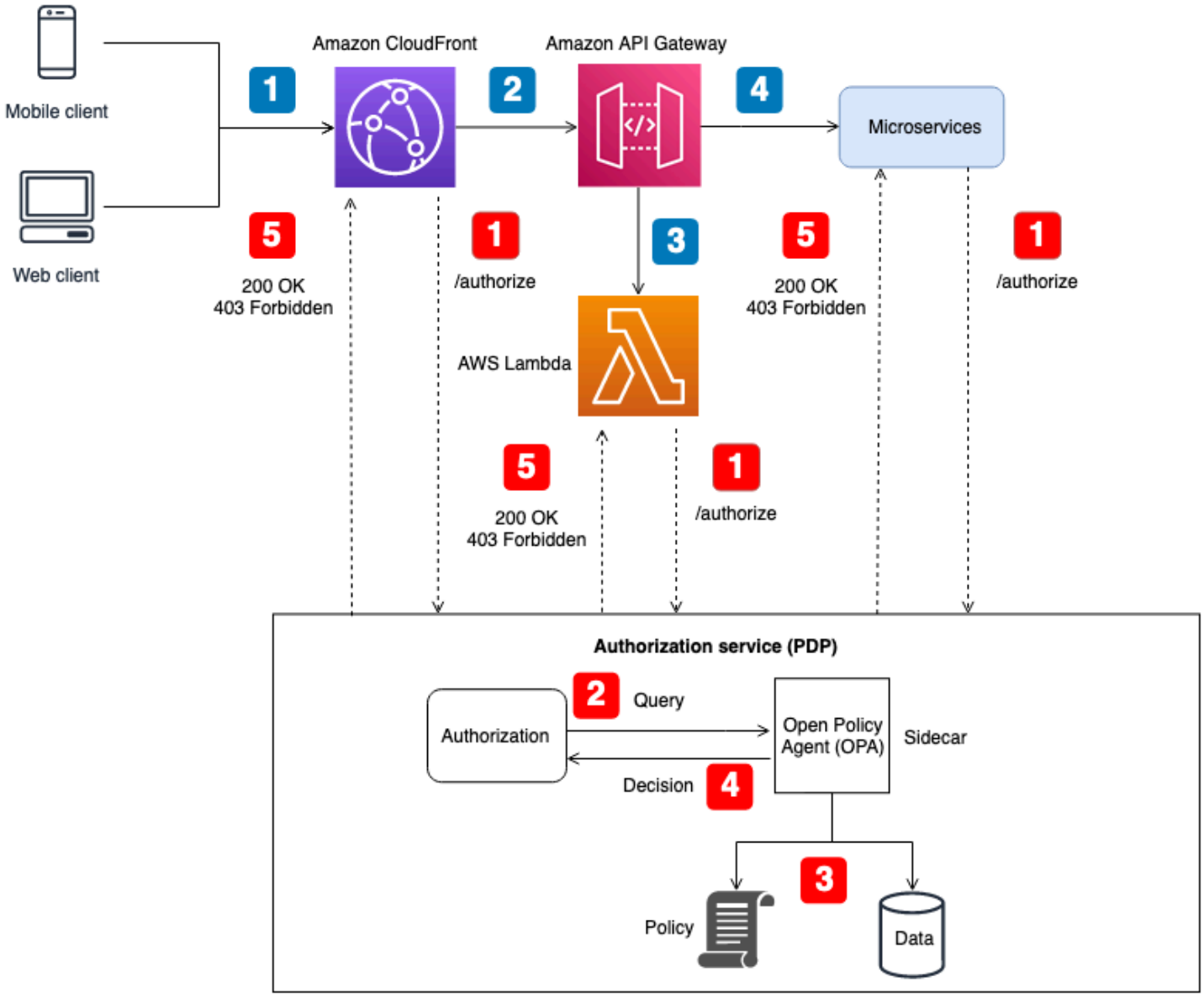
Model desain untuk OPA

Menggunakan PDP terpusat dengan PEP pada API

Titik keputusan kebijakan terpusat (PDP) dengan poin penegakan kebijakan (PEP) pada model API mengikuti praktik terbaik industri untuk menciptakan sistem yang efektif dan mudah dipelihara untuk kontrol akses dan otorisasi API. Pendekatan ini mendukung beberapa prinsip utama:

- Otorisasi dan kontrol akses API diterapkan di beberapa titik dalam aplikasi.
- Logika otorisasi tidak tergantung pada aplikasi.
- Keputusan kontrol akses terpusat.

Model ini menggunakan PDP terpusat untuk membuat keputusan otorisasi. PEP diimplementasikan di semua API untuk membuat permintaan otorisasi ke PDP. Diagram berikut menunjukkan bagaimana Anda dapat menerapkan model ini dalam aplikasi SaaS multi-penyewa hipotetis.



Alur aplikasi (diilustrasikan dengan callout bernomor biru dalam diagram):

1. Pengguna yang diautentikasi dengan JWT menghasilkan permintaan HTTP ke Amazon. CloudFront
2. CloudFront meneruskan permintaan ke Amazon API Gateway, yang dikonfigurasi sebagai CloudFront asal.
3. Authorizer kustom API Gateway dipanggil untuk memverifikasi JWT.
4. Layanan mikro menanggapi permintaan tersebut.

Alur kontrol akses otorisasi dan API (diilustrasikan dengan callout bernomor merah dalam diagram):

1. PEP memanggil layanan otorisasi dan meneruskan data permintaan, termasuk JWT apa pun.
2. Layanan otorisasi (PDP) mengambil data permintaan dan menanyakan API REST agen OPA, yang berjalan sebagai sespan. Data permintaan berfungsi sebagai masukan ke kueri.
3. OPA mengevaluasi input berdasarkan kebijakan relevan yang ditentukan oleh kueri. Data diimpor untuk membuat keputusan otorisasi jika perlu.
4. OPA mengembalikan keputusan ke layanan otorisasi.
5. Keputusan otorisasi dikembalikan ke PEP dan dievaluasi.

Dalam arsitektur ini, PEP meminta keputusan otorisasi di titik akhir layanan untuk Amazon CloudFront dan Amazon API Gateway, dan untuk setiap layanan mikro. Keputusan otorisasi dibuat oleh layanan otorisasi (PDP) dengan sespan OPA. Anda dapat mengoperasikan layanan otorisasi ini sebagai wadah atau sebagai contoh server tradisional. Sidecar OPA mengekspos RESTful API secara lokal sehingga API hanya dapat diakses oleh layanan otorisasi. Layanan otorisasi mengekspos API terpisah yang tersedia untuk PEP. Memiliki layanan otorisasi bertindak sebagai perantara antara PEP dan OPA memungkinkan penyisipan logika transformasi apa pun antara PEP dan OPA yang mungkin diperlukan—misalnya, ketika permintaan otorisasi dari PEP tidak sesuai dengan input kueri yang diharapkan oleh OPA.

Anda juga dapat menggunakan arsitektur ini dengan mesin kebijakan khusus. Namun, keuntungan apa pun yang diperoleh dari OPA harus diganti dengan logika yang disediakan oleh mesin kebijakan khusus.

PDP terpusat dengan PEP pada API menyediakan opsi mudah untuk membuat sistem otorisasi yang kuat untuk API. Mudah diimplementasikan dan juga menyediakan antarmuka yang easy-to-use dapat diulang untuk membuat keputusan otorisasi untuk API, layanan mikro, lapisan Backend for Frontend (BFF), atau komponen aplikasi lainnya. Namun, pendekatan ini mungkin membuat terlalu banyak

latensi dalam aplikasi Anda, karena keputusan otorisasi memerlukan pemanggilan API terpisah. Jika latensi jaringan menjadi masalah, Anda dapat mempertimbangkan PDP terdistribusi.

Menggunakan PDP terdistribusi dengan PEP pada API

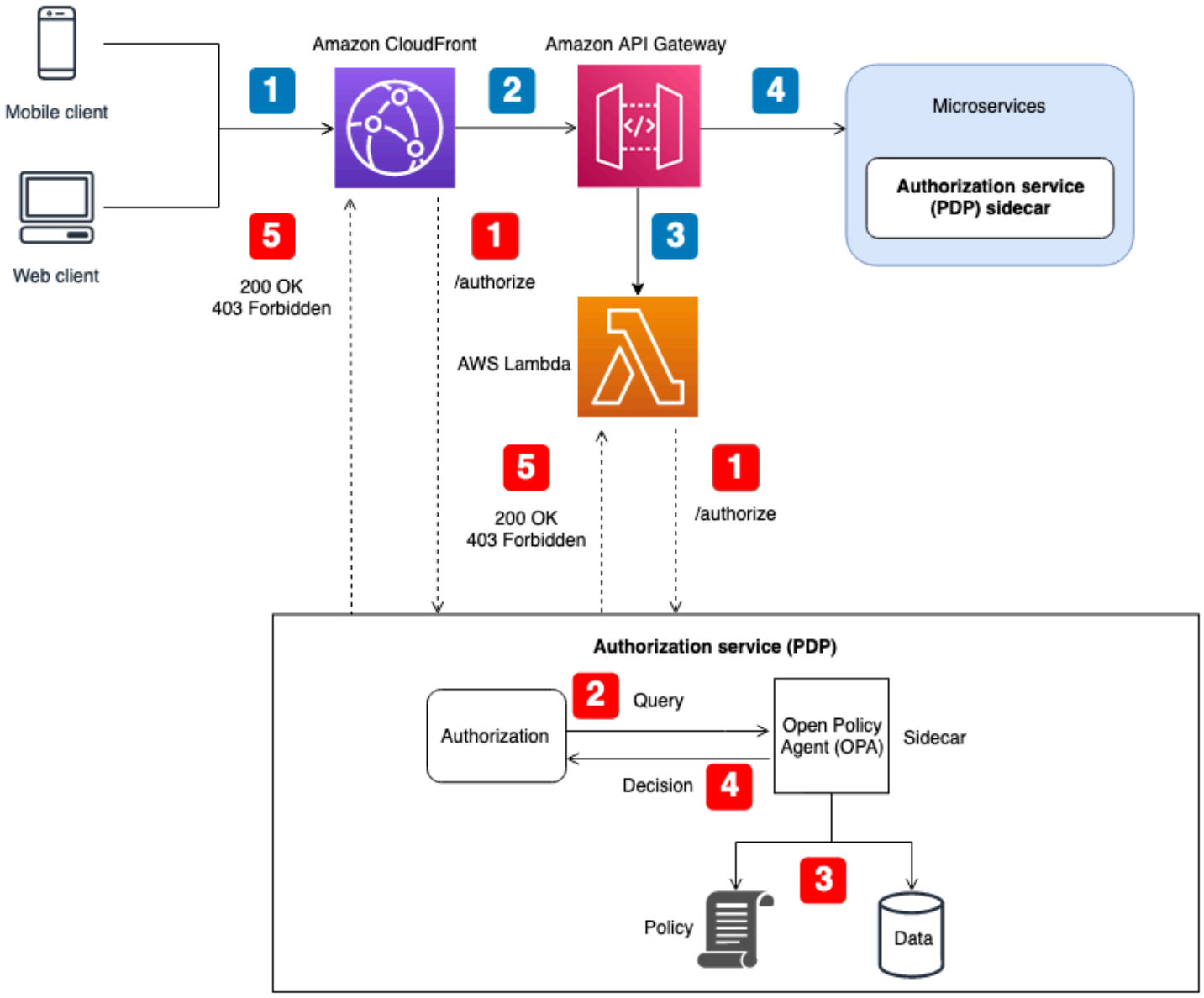
Titik keputusan kebijakan terdistribusi (PDP) dengan poin penegakan kebijakan (PEP) pada model API mengikuti praktik terbaik industri untuk menciptakan sistem yang efektif untuk kontrol akses dan otorisasi API. Seperti PDP terpusat dengan PEP pada model API, pendekatan ini mendukung prinsip-prinsip utama berikut:

- Otorisasi dan kontrol akses API diterapkan di beberapa titik dalam aplikasi.
- Logika otorisasi tidak tergantung pada aplikasi.
- Keputusan kontrol akses terpusat.

Anda mungkin bertanya-tanya mengapa keputusan kontrol akses terpusat ketika PDP didistribusikan. Meskipun PDP mungkin ada di beberapa tempat dalam aplikasi, PDP harus menggunakan logika otorisasi yang sama untuk membuat keputusan kontrol akses. Semua PDP memberikan keputusan kontrol akses yang sama dengan masukan yang sama. PEP diimplementasikan di semua API untuk membuat permintaan otorisasi ke PDP. Gambar berikut menunjukkan bagaimana model terdistribusi ini dapat diimplementasikan dalam aplikasi SaaS multi-penyewa hipotetis.

Dalam pendekatan ini, PDP diimplementasikan di banyak tempat dalam aplikasi. Untuk komponen aplikasi yang memiliki kemampuan komputasi onboard yang dapat menjalankan OPA dan mendukung PDP, seperti layanan kontainer dengan sespan atau Amazon Elastic Compute Cloud (Amazon EC2) Elastic Cloud (Amazon EC2), keputusan PDP dapat diintegrasikan langsung ke dalam komponen aplikasi tanpa harus melakukan panggilan RESTful API ke layanan PDP terpusat. Ini memiliki manfaat mengurangi latensi yang mungkin Anda temui dalam model PDP terpusat, karena tidak setiap komponen aplikasi harus melakukan panggilan API tambahan untuk mendapatkan keputusan otorisasi. Namun, PDP terpusat masih diperlukan dalam model ini untuk komponen aplikasi yang tidak memiliki kemampuan komputasi onboard yang memungkinkan integrasi langsung PDP—seperti layanan Amazon atau Amazon CloudFront API Gateway.

Diagram berikut menunjukkan bagaimana kombinasi PDP terpusat dan PDP terdistribusi ini dapat diimplementasikan dalam aplikasi SaaS multi-penyewa hipotetis.



Alur aplikasi (diilustrasikan dengan callout bernomor biru dalam diagram):

1. Pengguna yang diautentikasi dengan JWT menghasilkan permintaan HTTP ke Amazon. CloudFront
2. CloudFront meneruskan permintaan ke Amazon API Gateway, yang dikonfigurasi sebagai CloudFront asal.
3. Authorizer kustom API Gateway dipanggil untuk memverifikasi JWT.
4. Layanan mikro menanggapi permintaan tersebut.

Alur kontrol akses otorisasi dan API (diilustrasikan dengan callout bernomor merah dalam diagram):

1. PEP memanggil layanan otorisasi dan meneruskan data permintaan, termasuk JWT apa pun.
2. Layanan otorisasi (PDP) mengambil data permintaan dan menanyakan API REST agen OPA, yang berjalan sebagai sespan. Data permintaan berfungsi sebagai masukan ke kueri.
3. OPA mengevaluasi input berdasarkan kebijakan relevan yang ditentukan oleh kueri. Data diimpor untuk membuat keputusan otorisasi jika perlu.
4. OPA mengembalikan keputusan ke layanan otorisasi.
5. Keputusan otorisasi dikembalikan ke PEP dan dievaluasi.

Dalam arsitektur ini, PEP meminta keputusan otorisasi di titik akhir layanan untuk dan API CloudFront Gateway, dan untuk setiap layanan mikro. Keputusan otorisasi untuk layanan mikro dibuat oleh layanan otorisasi (PDP) yang beroperasi sebagai sespan dengan komponen aplikasi. Model ini dimungkinkan untuk layanan mikro (atau layanan) yang berjalan pada container atau instans Amazon Elastic Compute Cloud (Amazon EC2). Keputusan otorisasi untuk layanan seperti API Gateway dan CloudFront masih perlu menghubungi layanan otorisasi eksternal. Terlepas dari itu, layanan otorisasi mengekspos API yang tersedia untuk PEP. Memiliki layanan otorisasi bertindak sebagai perantara antara PEP dan OPA memungkinkan penyisipan logika transformasi apa pun antara PEP dan OPA yang mungkin diperlukan—misalnya, ketika permintaan otorisasi dari PEP tidak sesuai dengan input kueri yang diharapkan oleh OPA.

Anda juga dapat menggunakan arsitektur ini dengan mesin kebijakan khusus. Namun, keuntungan apa pun yang diperoleh dari OPA harus diganti dengan logika yang disediakan oleh mesin kebijakan khusus.

PDP terdistribusi dengan PEP pada API menyediakan opsi untuk membuat sistem otorisasi yang kuat untuk API. Sangat mudah untuk menerapkan dan menyediakan antarmuka yang easy-to-use dapat diulang untuk membuat keputusan otorisasi untuk API, layanan mikro, lapisan Backend for Frontend (BFF), atau komponen aplikasi lainnya. Pendekatan ini juga memiliki keuntungan mengurangi latensi yang mungkin Anda temui dalam model PDP terpusat.

Menggunakan PDP terdistribusi sebagai pustaka

Anda juga dapat meminta keputusan otorisasi dari PDP yang tersedia sebagai pustaka atau paket untuk digunakan dalam aplikasi. OPA dapat digunakan sebagai pustaka pihak ketiga Go. Untuk bahasa pemrograman lain, mengadopsi model ini umumnya berarti Anda harus membuat mesin kebijakan khusus.

Pertimbangan desain multi-penyewa Izin Terverifikasi Amazon

Ada beberapa opsi desain yang perlu dipertimbangkan saat Anda menerapkan otorisasi dengan menggunakan Izin Terverifikasi Amazon dalam solusi SaaS multi-penyewa. Sebelum menjelajahi opsi ini, mari kita perjelas perbedaan antara isolasi dan otorisasi dalam konteks SaaS multi-penyewa. Mengisolasi penyewa mencegah data masuk dan keluar terpapar ke penyewa yang salah. Otorisasi memastikan bahwa pengguna memiliki izin untuk mengakses penyewa.

Di Izin Terverifikasi, kebijakan disimpan di toko kebijakan. Seperti yang dijelaskan dalam dokumentasi Izin Terverifikasi, Anda dapat mengisolasi kebijakan penyewa dengan menggunakan penyimpanan kebijakan terpisah untuk setiap penyewa, atau mengizinkan penyewa untuk berbagi kebijakan dengan menggunakan satu penyimpanan kebijakan untuk semua penyewa. Bagian ini membahas keuntungan dan kerugian dari dua strategi isolasi ini, dan menjelaskan bagaimana mereka dapat digunakan dengan menggunakan model penyebaran berjenjang. Untuk konteks tambahan, lihat dokumentasi Izin Terverifikasi.

Meskipun kritisi yang dibahas di bagian ini berfokus pada Izin Terverifikasi, konsep umum berakar pada pola pikir isolasi dan panduan yang diberikannya. Aplikasi SaaS harus selalu mempertimbangkan isolasi penyewa sebagai bagian dari desain mereka, dan prinsip umum isolasi ini mencakup Izin Terverifikasi dalam aplikasi SaaS. Bagian ini juga mereferensikan model isolasi SaaS inti seperti model SaaS silo dan model SaaS gabungan. Untuk informasi tambahan, lihat konsep isolasi inti dalam Kerangka AWS Well-Architected, SaaS Lens.

Pertimbangan utama saat merancang solusi SaaS multi-penyewa adalah isolasi penyewa dan orientasi penyewa. Isolasi penyewa berdampak pada keamanan, privasi, ketahanan, dan kinerja. Orientasi penyewa memengaruhi proses operasional Anda karena berkaitan dengan overhead operasional dan observabilitas. Organizations yang melalui perjalanan SaaS atau menerapkan solusi multi-tenant harus selalu memprioritaskan bagaimana penyewaan akan ditangani oleh aplikasi SaaS. Meskipun solusi SaaS mungkin condong ke model isolasi tertentu, konsistensi tidak selalu diperlukan di seluruh solusi SaaS. Misalnya, model isolasi yang Anda pilih untuk komponen frontend aplikasi Anda mungkin tidak sama dengan model isolasi yang Anda pilih untuk layanan microservice atau otorisasi.

Pertimbangan desain:

- Orientasi penyewa dan pendaftaran penyewa pengguna

- Toko kebijakan per penyewa
- Satu toko kebijakan multi-penyewa bersama
- Model penyebaran berjenjang

Orientasi penyewa dan pendaftaran penyewa pengguna

Aplikasi SaaS mengamati konsep identitas SaaS dan mengikuti praktik terbaik umum untuk mengikat identitas pengguna ke identitas penyewa. Binding melibatkan penyimpanan pengenal penyewa sebagai klaim atau atribut untuk pengguna di penyedia identitas. Ini mengalihkan tanggung jawab pemetaan identitas kepada penyewa dari setiap aplikasi ke proses pendaftaran pengguna. Setiap pengguna yang diautentikasi kemudian memiliki identitas penyewa yang benar sebagai bagian dari JSON Web Token (JWT).

Demikian pula, pemilihan penyimpanan kebijakan yang benar untuk permintaan otorisasi tidak boleh ditentukan oleh logika aplikasi. Untuk menentukan penyimpanan kebijakan mana yang harus digunakan permintaan otorisasi tertentu, pertahankan pemetaan pengguna ke toko kebijakan, atau penyewa ke toko kebijakan. Pemetaan ini biasanya disimpan di penyimpanan data seperti Amazon DynamoDB atau Amazon Relational Database Service (Amazon RDS) yang menjadi referensi aplikasi Anda. Anda juga dapat memberikan atau melengkapi pemetaan ini dengan data di penyedia identitas (iDP). Hubungan antara penyewa, pengguna, dan toko kebijakan kemudian biasanya diberikan kepada pengguna melalui JWT yang berisi semua hubungan yang diperlukan untuk permintaan otorisasi.

Contoh ini menunjukkan bagaimana JWT mungkin muncul untuk penggunaAlice, yang termasuk dalam penyewa TenantA dan menggunakan penyimpanan kebijakan dengan ID ps-43214321 penyimpanan kebijakan untuk otorisasi.

```
{
   "sub":"1234567890",
   "name":"Alice",
   "tenant":"TenantA",
   "policyStoreId":"ps-43214321"
}
```

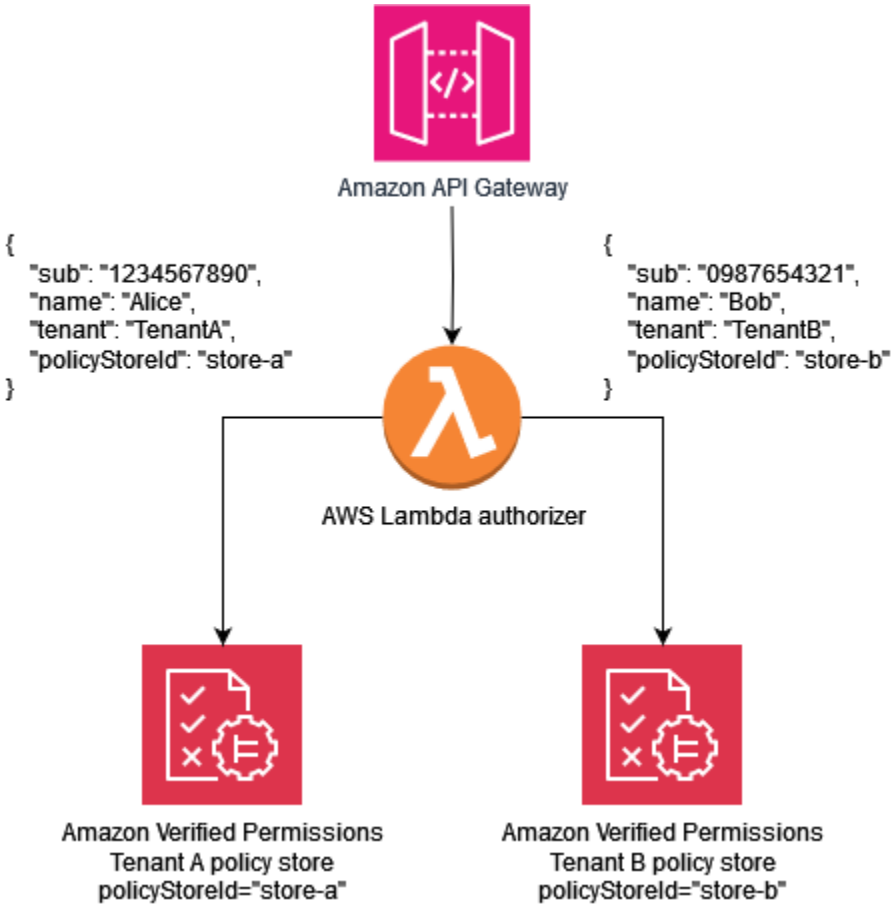
Toko kebijakan per penyewa

Model desain toko kebijakan per-penyewa di Izin Terverifikasi Amazon mengaitkan setiap penyewa dalam aplikasi SaaS dengan toko kebijakannya sendiri. Model ini mirip dengan model isolasi silo SaaS. Kedua model mengamanatkan penciptaan infrastruktur khusus penyewa dan memiliki manfaat dan kerugian yang sama. Manfaat utama dari pendekatan ini adalah isolasi penyewa yang diberlakukan infrastruktur, dukungan untuk model otorisasi unik berdasarkan per-penyewa, penghapusan masalah tetangga yang bising, dan pengurangan cakupan dampak kegagalan dalam pembaruan atau penerapan kebijakan. Kerugian dari pendekatan ini termasuk proses orientasi penyewa yang lebih kompleks, penerapan, dan operasi. Toko kebijakan per penyewa adalah pendekatan yang disarankan jika solusi memiliki kebijakan unik per penyewa.

Model toko kebijakan per-penyewa dapat memberikan pendekatan yang sangat silo untuk isolasi penyewa, jika aplikasi SaaS Anda memerlukannya. Anda juga dapat menggunakan model ini dengan isolasi kolam, tetapi implementasi Izin Terverifikasi Anda tidak akan berbagi manfaat standar dari model isolasi kolam yang lebih luas seperti manajemen dan operasi yang disederhanakan.

Di toko kebijakan per-penyewa, isolasi penyewa dicapai dengan memetakan pengenal toko kebijakan penyewa ke Identitas SaaS pengguna selama proses pendaftaran pengguna, seperti yang dibahas sebelumnya. Pendekatan ini sangat mengikat toko kebijakan penyewa dengan prinsipal pengguna dan menyediakan cara yang konsisten untuk berbagi pemetaan di seluruh solusi SaaS. Anda dapat memberikan pemetaan ke aplikasi SaaS dengan mempertahankannya sebagai bagian dari IDP atau dalam sumber data eksternal seperti DynamoDB. Ini juga memastikan bahwa kepala sekolah adalah bagian dari penyewa dan bahwa penyimpanan kebijakan penyewa digunakan.

Contoh ini menunjukkan bagaimana JWT yang berisi `policyStoreId` dan `tenant` pengguna diteruskan dari titik akhir API ke titik evaluasi kebijakan di AWS Lambda otorisasi, yang merutekan permintaan ke penyimpanan kebijakan yang benar.



Contoh kebijakan berikut menggambarkan paradigma desain toko kebijakan per penyewa. Pengguna `Alice` milik `TenantA`. The juga policyStoreId `store-a` dipetakan ke identitas penyewa `Alice`, dan memberlakukan penggunaan toko kebijakan yang benar. Ini memastikan bahwa kebijakan `TenantA` digunakan.

> **Note**
>
> Model toko kebijakan per-penyewa mengisolasi kebijakan penyewa. Otorisasi memberlakukan tindakan yang diizinkan dilakukan pengguna pada data mereka. Sumber daya yang terlibat dalam aplikasi hipotetis apa pun yang menggunakan model ini harus diisolasi dengan menggunakan mekanisme isolasi lain, sebagaimana didefinisikan dalam Kerangka AWS Well-Architected, dokumentasi Lensa SaaS.

Dalam kebijakan ini, `Alice` memiliki izin untuk melihat data semua sumber daya.

```
permit (
```

```
    principal == MultiTenantApp::User::"Alice",
    action == MultiTenantApp::Action::"viewData",
    resource
);
```

Untuk membuat permintaan otorisasi dan memulai evaluasi dengan kebijakan Izin Terverifikasi, Anda harus memberikan ID penyimpanan kebijakan yang sesuai dengan ID unik yang dipetakan ke penyewa. `store-a`

```
{
   "policyStoreId":"store-a",
   "principal":{
      "entityType":"MultiTenantApp::User",
      "entityId":"Alice"
   },
   "action":{
      "actionType":"MultiTenantApp::Action",
      "actionId":"viewData"
   },
   "resource":{
      "entityType":"MultiTenantApp::Data",
      "entityId":"my_example_data"
   },
   "entities":{
      "entityList":[
         [
            {
               "identifier":{
                  "entityType":"MultiTenantApp::User",
                  "entityId":"Alice"
               },
               "attributes":{},
               "parents":[]
            },
            {
               "identifier":{
                  "entityType":"MultiTenantApp::Data",
                  "entityId":"my_example_data"
               },
               "attributes":{},
               "parents":[]
            }
```

```
         ]
      ]
   }
}
```

Pengguna Bob milik Penyewa B, dan juga policyStoreId `store-b` dipetakan ke identitas penyewaBob, yang memberlakukan penggunaan penyimpanan kebijakan yang benar. Ini memastikan bahwa kebijakan Penyewa B digunakan.

Dalam kebijakan ini, Bob memiliki izin untuk menyesuaikan data semua sumber daya. Dalam contoh ini, `customizeData` mungkin tindakan yang khusus hanya untuk Penyewa B, sehingga kebijakan akan unik untuk Penyewa B. Model penyimpanan kebijakan per-penyewa secara inheren mendukung kebijakan khusus berdasarkan per-penyewa.

```
permit (
    principal == MultiTenantApp::User::"Bob",
    action == MultiTenantApp::Action::"customizeData",
    resource
);
```

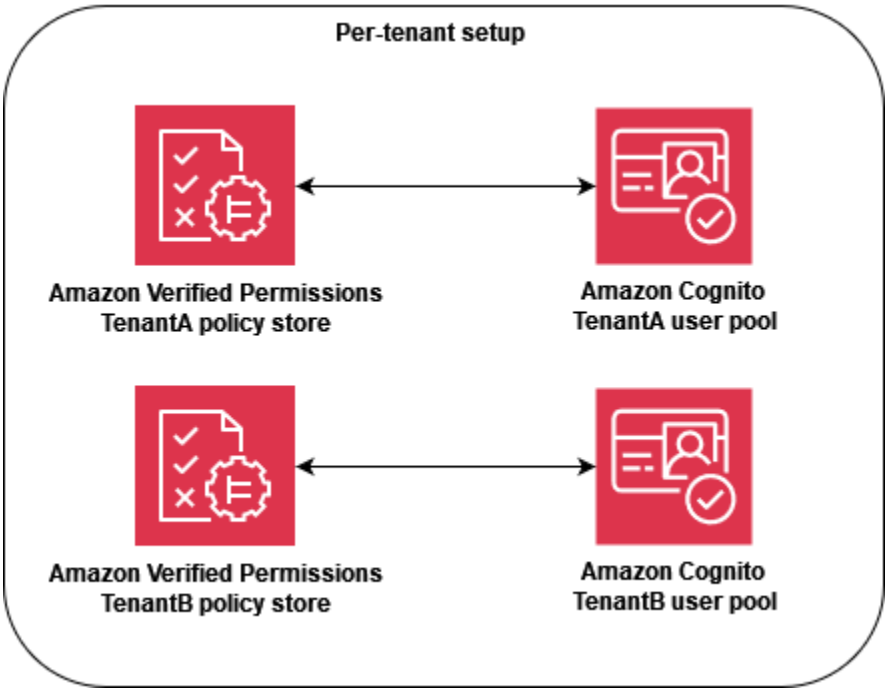
Untuk membuat permintaan otorisasi dan memulai evaluasi dengan kebijakan Izin Terverifikasi, Anda harus memberikan ID penyimpanan kebijakan yang sesuai dengan ID unik yang dipetakan ke penyewa. `store-b`

```
{
   "policyStoreId":"store-b",
   "principal":{
      "entityType":"MultiTenantApp::User",
      "entityId":"Bob"
   },
   "action":{
      "actionType":"MultiTenantApp::Action",
      "actionId":"customizeData"
   },
   "resource":{
      "entityType":"MultiTenantApp::Data",
      "entityId":"my_example_data"
   },
   "entities":{
      "entityList":[
         [
```

```
            {
               "identifier":{
                  "entityType":"MultiTenantApp::User",
                  "entityId":"Bob"
               },
               "attributes":{},
               "parents":[]
            },
            {
               "identifier":{
                  "entityType":"MultiTenantApp::Data",
                  "entityId":"my_example_data"
               },
               "attributes":{},
               "parents":[]
            }
         ]
      ]
   }
}
```

Dengan Izin Terverifikasi, dimungkinkan, tetapi tidak diperlukan, untuk mengintegrasikan iDP dengan toko kebijakan. Integrasi ini memungkinkan kebijakan untuk secara eksplisit merujuk prinsipal di toko identitas sebagai prinsipal kebijakan. Untuk informasi selengkapnya tentang cara mengintegrasikan dengan Amazon Cognito sebagai iDP untuk Izin Terverifikasi, lihat dokumentasi Izin Terverifikasi dan dokumentasi Amazon Cognito.

Saat mengintegrasikan penyimpanan kebijakan dengan iDP, Anda hanya dapat menggunakan satu sumber identitas per toko kebijakan. Misalnya, jika Anda memilih untuk mengintegrasikan Izin Terverifikasi dengan Amazon Cognito, Anda harus mencerminkan strategi yang digunakan untuk isolasi penyewa dari toko kebijakan Izin Terverifikasi dan kumpulan pengguna Amazon Cognito. Toko kebijakan dan kumpulan pengguna juga harus sama Akun AWS.



Pada tingkat operasional, toko kebijakan per penyewa memiliki keunggulan audit, karena Anda dapat dengan mudah menanyakan aktivitas yang dicatat secaraAWS CloudTrail independen untuk setiap penyewa. Namun, kami tetap menyarankan agar Anda mencatat metrik khusus tambahan pada dimensi per-penyewa ke Amazon. CloudWatch

Pendekatan penyimpanan kebijakan per-penyewa juga memerlukan perhatian yang cermat terhadap dua kuota Izin Terverifikasi untuk memastikan bahwa kuota tersebut tidak mengganggu pengoperasian solusi SaaS Anda. Kuota ini adalah toko Kebijakan per Wilayah per akun dan IsAuthorized permintaan per detik per Wilayah per akun. Anda dapat meminta kenaikan untuk kedua kuota.

Untuk contoh lebih rinci tentang cara menerapkan model penyimpanan kebijakan per penyewa, lihat kontrol akses SaaS AWS posting blog menggunakan Izin Terverifikasi Amazon dengan penyimpanan kebijakan per-penyewa.

Satu toko kebijakan multi-penyewa bersama

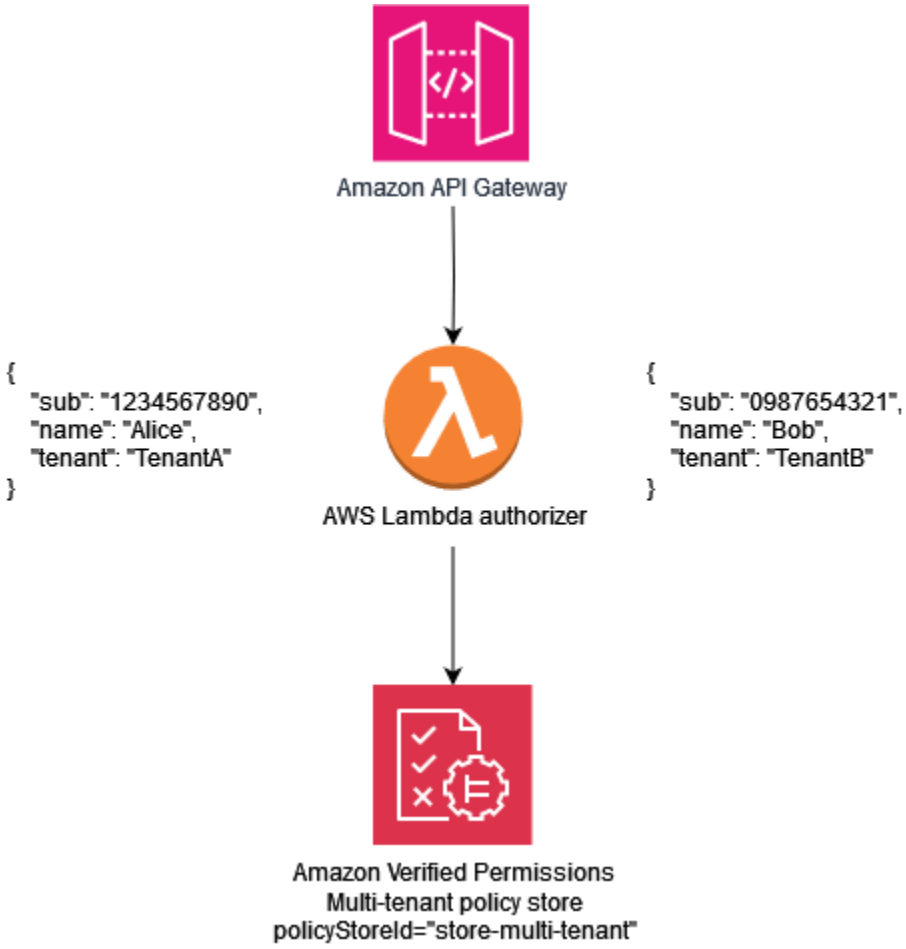
Model desain toko kebijakan multi-penyewa bersama menggunakan penyimpanan kebijakan multi-penyewa tunggal di Izin Terverifikasi Amazon untuk semua penyewa dalam solusi SaaS. Manfaat utama dari pendekatan ini adalah manajemen dan operasi yang disederhanakan, terutama karena Anda tidak perlu membuat toko kebijakan tambahan selama orientasi penyewa. Kerugian dari

pendekatan ini termasuk peningkatan cakupan dampak dari kegagalan atau kesalahan dalam pembaruan atau penerapan kebijakan, dan paparan yang lebih besar terhadap efek tetangga yang bising. Selain itu, kami tidak merekomendasikan pendekatan ini jika solusi Anda memerlukan kebijakan unik untuk setiap penyewa. Dalam hal ini, gunakan model penyimpanan kebijakan per-penyewa sebagai gantinya untuk menjamin bahwa kebijakan penyewa yang benar digunakan.

Pendekatan penyimpanan kebijakan multi-penyewa bersama mirip dengan model isolasi gabungan SaaS. Ini dapat memberikan pendekatan gabungan untuk isolasi penyewa, jika aplikasi SaaS Anda membutuhkannya. Anda juga dapat menggunakan model ini jika solusi SaaS Anda menerapkan isolasi silo ke layanan mikro-nya. Ketika Anda memilih model, Anda harus mengevaluasi persyaratan untuk isolasi data penyewa dan struktur kebijakan Izin Terverifikasi yang diperlukan untuk aplikasi SaaS secara independen.

Untuk menerapkan cara yang konsisten dalam membagikan pengenal penyewa di seluruh solusi SaaS Anda, adalah praktik yang baik untuk memetakan pengenal ke identitas SaaS pengguna selama pendaftaran pengguna, seperti yang dibahas sebelumnya. Anda dapat memberikan pemetaan ini ke aplikasi SaaS dengan mempertahankannya sebagai bagian dari IDP atau dalam sumber data eksternal seperti DynamoDB. Kami juga menyarankan Anda memetakan ID penyimpanan kebijakan bersama kepada pengguna. Meskipun ID tidak digunakan sebagai bagian dari isolasi penyewa, ini adalah praktik yang baik karena memfasilitasi perubahan di masa depan.

Contoh berikut menunjukkan bagaimana titik akhir API mengirimkan JWT untuk pengguna `Alice` dan`Bob`, yang termasuk dalam penyewa yang berbeda tetapi membagikan penyimpanan kebijakan dengan ID penyimpanan kebijakan untuk otorisasi. `store-multi-tenant` Karena semua penyewa berbagi satu toko kebijakan, Anda tidak perlu mempertahankan ID penyimpanan kebijakan dalam token atau database. Karena semua penyewa berbagi satu ID penyimpanan kebijakan, Anda dapat memberikan ID sebagai variabel lingkungan yang dapat digunakan aplikasi Anda untuk melakukan panggilan ke penyimpanan kebijakan.



Contoh kebijakan berikut menggambarkan paradigma desain kebijakan multi-penyewa bersama. Dalam kebijakan ini, prinsipal `MultiTenantApp::User` yang memiliki induk `MultiTenantApp::Role` `Admin` memiliki izin untuk melihat data semua sumber daya.

```
permit (
    principal in MultiTenantApp::Role::"Admin",
    action == MultiTenantApp::Action::"viewData",
    resource
);
```

Karena penyimpanan kebijakan tunggal sedang digunakan, penyimpanan kebijakan Izin Terverifikasi harus memastikan bahwa atribut penyewaan yang terkait dengan prinsipal cocok dengan atribut penyewaan yang terkait dengan sumber daya. Ini dapat dicapai dengan menyertakan kebijakan berikut di toko kebijakan, untuk memastikan bahwa semua permintaan otorisasi yang tidak memiliki atribut penyewaan yang cocok pada sumber daya dan prinsipal ditolak.

```
forbid(
```

```
    principal,
    action,
    resource
)
unless {
    resource.Tenant == principal.Tenant
};
```

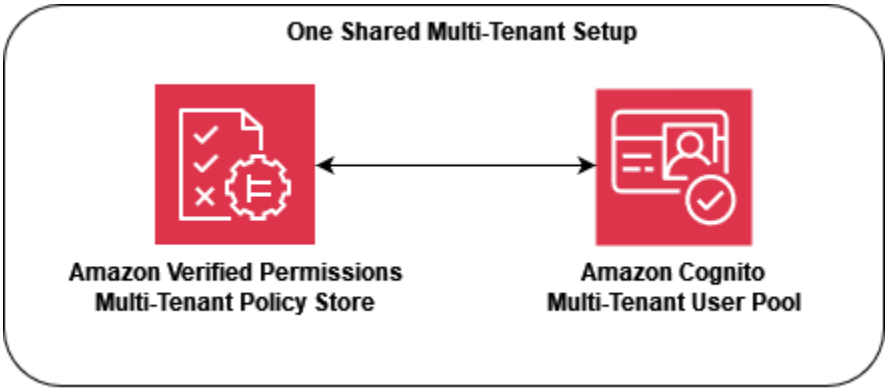
Untuk permintaan otorisasi yang menggunakan model penyimpanan kebijakan multi-penyewa bersama, ID penyimpanan kebijakan adalah pengenal penyimpanan kebijakan bersama. Dalam permintaan berikut, akses `User Alice` diizinkan karena dia memiliki `Role` dari`Admin`, dan `Tenant` atribut yang terkait dengan sumber daya dan prinsipal keduanya`TenantA`.

```
{
   "policyStoreId":"store-multi-tenant",
   "principal":{
      "entityType":"MultiTenantApp::User",
      "entityId":"Alice"
   },
   "action":{
      "actionType":"MultiTenantApp::Action",
      "actionId":"viewData"
   },
   "resource":{
      "entityType":"MultiTenantApp::Data",
      "entityId":"my_example_data"
   },
   "entities":{
      "entityList":[
         {
            "identifier":{
               "entityType":"MultiTenantApp::User",
               "entityId":"Alice"
            },
            "attributes": {
                {
                    "Tenant": {
                        "entityIdentifier": {
                            "entityType":"MultitenantApp::Tenant",
                            "entityId":"TenantA"
                        }
                    }
```

```
                }
            },
            "parents":[
               {
                  "entityType":"MultiTenantApp::Role",
                  "entityId":"Admin"
               }
            ]
         },
         {
            "identifier":{
               "entityType":"MultiTenantApp::Data",
               "entityId":"my_example_data"
            },
            "attributes": {
                {
                    "Tenant": {
                        "entityIdentifier": {
                            "entityType":"MultitenantApp::Tenant",
                            "entityId":"TenantA"
                        }
                    }
                }
            },
            "parents":[]
         }
      ]
   }
}
```

Dengan Izin Terverifikasi, dimungkinkan, tetapi tidak diperlukan, untuk mengintegrasikan iDP dengan toko kebijakan. Integrasi ini memungkinkan kebijakan untuk secara eksplisit merujuk prinsipal di toko identitas sebagai prinsipal kebijakan. Untuk informasi selengkapnya tentang cara mengintegrasikan dengan Amazon Cognito sebagai iDP untuk Izin Terverifikasi, lihat dokumentasi Izin Terverifikasi dan dokumentasi Amazon Cognito.

Saat mengintegrasikan penyimpanan kebijakan dengan iDP, Anda hanya dapat menggunakan satu sumber identitas per toko kebijakan. Misalnya, jika Anda memilih untuk mengintegrasikan Izin Terverifikasi dengan Amazon Cognito, Anda harus mencerminkan strategi yang digunakan untuk isolasi penyewa dari toko kebijakan Izin Terverifikasi dan kumpulan pengguna Amazon Cognito. Toko kebijakan dan kumpulan pengguna juga harus sama Akun AWS.



Dari perspektif operasional dan audit, model penyimpanan kebijakan multi-penyewa bersama memiliki kelemahan karena aktivitas yang dicatat AWS CloudTrail memerlukan lebih banyak kueri yang terlibat untuk menyaring aktivitas individu pada penyewa, karena setiap CloudTrail panggilan yang dicatat menggunakan penyimpanan kebijakan yang sama. Dalam skenario ini, akan sangat membantu untuk mencatat metrik khusus tambahan pada dimensi per-penyewa ke Amazon CloudWatch untuk memastikan tingkat pengamatan dan kemampuan audit yang sesuai.

Pendekatan penyimpanan kebijakan multi-penyewa bersama juga memerlukan perhatian yang cermat terhadap kuota Izin Terverifikasi untuk memastikan bahwa kuota tersebut tidak mengganggu pengoperasian solusi SaaS Anda. Secara khusus, kami menyarankan Anda memantau IsAuthorized permintaan per detik per Wilayah per kuota akun untuk memastikan bahwa batasannya tidak terlampaui. Anda dapat meminta kenaikan kuota ini.

Model penyebaran berjenjang

Dengan membuat model penerapan berjenjang, Anda dapat mengisolasi penyewa “Tingkat Perusahaan” prioritas tinggi dari volume pelanggan “Tingkat Standar” yang berpotensi lebih tinggi. Dalam model ini, Anda dapat meluncurkan perubahan apa pun yang diterapkan pada kebijakan di toko kebijakan secara terpisah untuk setiap tingkat, yang mengisolasi setiap tingkat pelanggan dari perubahan yang dilakukan di luar tingkat mereka. Dalam model penerapan berjenjang, penyimpanan kebijakan biasanya dibuat sebagai bagian dari penyediaan infrastruktur awal untuk setiap tingkatan alih-alih diterapkan saat penyewa di-onboard.

Jika solusi Anda terutama menggunakan model isolasi gabungan, Anda mungkin memerlukan isolasi atau penyesuaian tambahan. Misalnya, Anda dapat membuat “Tingkat Premium” di mana setiap penyewa akan mendapatkan infrastruktur tingkat penyewa mereka sendiri, yang membuat model siloed dengan menerapkan instance gabungan dengan hanya satu penyewa. Ini bisa berupa

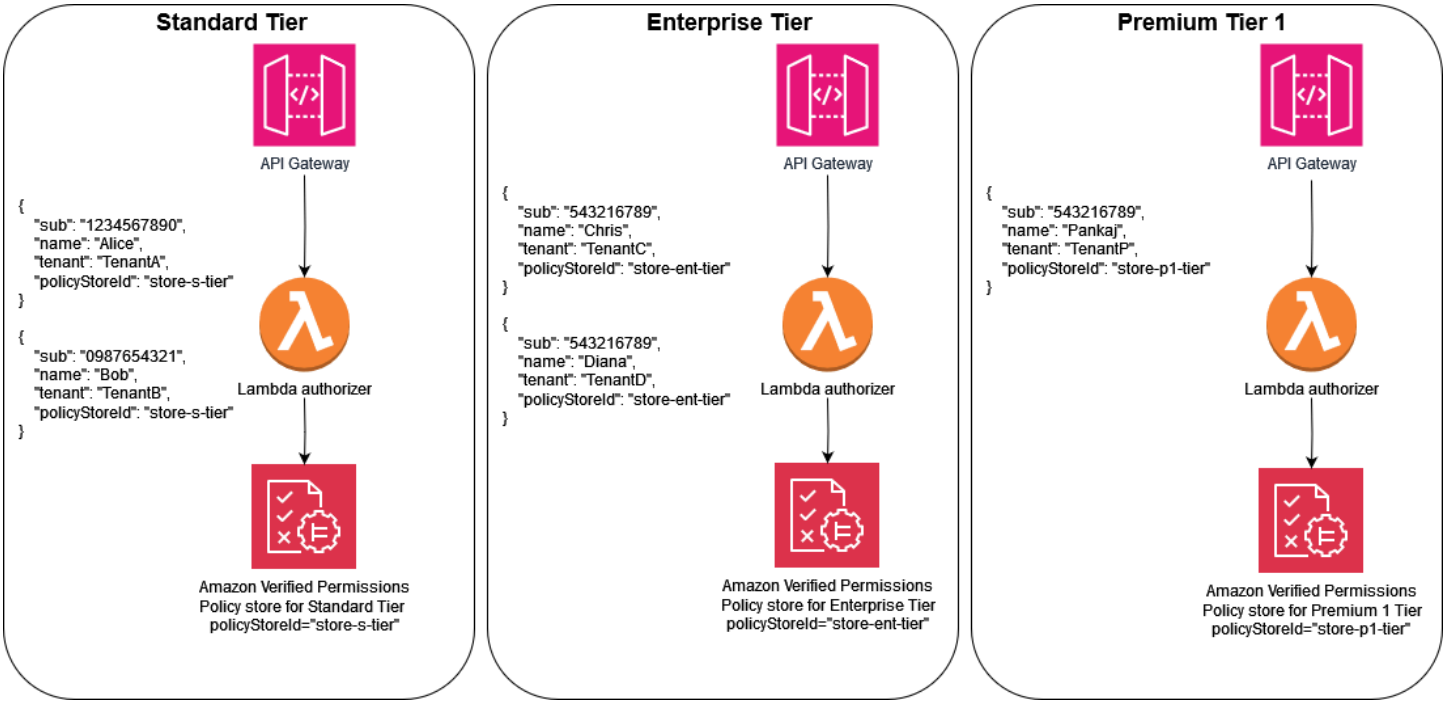
infrastruktur “Premium Tier Tenant A” dan “Premium Tier Tenant B” yang benar-benar terpisah, termasuk toko kebijakan. Pendekatan ini menghasilkan model isolasi tersilo untuk pelanggan tingkat tertinggi.

Dalam model penerapan berjenjang, setiap toko kebijakan harus mengikuti model isolasi yang sama, meskipun diterapkan secara terpisah. Karena ada beberapa toko kebijakan yang digunakan, Anda perlu menerapkan cara yang konsisten untuk membagikan pengenal toko kebijakan yang terkait dengan penyewa di seluruh solusi SaaS. Seperti halnya model penyimpanan kebijakan per-penyewa, merupakan praktik yang baik untuk memetakan pengenal penyewa ke identitas SaaS pengguna selama pendaftaran pengguna.

Diagram berikut menunjukkan tiga tingkatan:`Standard Tier`,`Enterprise Tier`, dan`Premium Tier 1`. Setiap tingkatan dikerahkan secara terpisah di infrastrukturnya sendiri dan menggunakan satu penyimpanan kebijakan bersama di dalam tingkatan. Tingkatan Standar dan Perusahaan berisi banyak penyewa. `TenantA`dan `TenantB` berada di`Standard Tier`, dan `TenantC` dan `TenantD` berada di Tingkat Perusahaan.

`Premium Tier 1`hanya berisi`TenantP`, sehingga Anda dapat melayani penyewa premium seolah-olah solusinya memiliki model isolasi yang sepenuhnya tertutup dan menyediakan fitur seperti kebijakan yang disesuaikan. Orientasi pelanggan tingkat premium baru akan menghasilkan penciptaan `Premium Tier 2` infrastruktur.

> **Note**
>
> Aplikasi, penerapan, dan orientasi penyewa di tingkat premium identik dengan tingkatan standar dan perusahaan. Satu-satunya perbedaan adalah alur kerja orientasi tingkat premium dimulai dengan penyediaan infrastruktur tingkat baru.


Standard Tier
API Gateway
{
"sub": "1234567890",
"name": "Alice",
"tenant": "TenantA",
"policyStoreId": "store-s-tier"
}
{
"sub": "0987654321",
"name": "Bob",
"tenant": "TenantB",
"policyStoreId": "store-s-tier"
}
Lambda authorizer
Amazon Verified Permissions
Policy store for Standard Tier
policyStoreId="store-s-tier"
Enterprise Tier
API Gateway
{
"sub": "543216789",
"name": "Chris",
"tenant": "TenantC",
"policyStoreId": "store-ent-tier"
}
{
"sub": "543216789",
"name": "Diana",
"tenant": "TenantD",
"policyStoreId": "store-ent-tier"
}
Lambda authorizer
Amazon Verified Permissions
Policy store for Enterprise Tier
policyStoreId="store-ent-tier"
Premium Tier 1
API Gateway
{
"sub": "543216789",
"name": "Pankaj",
"tenant": "TenantP",
"policyStoreId": "store-p1-tier"
}
Lambda authorizer
Amazon Verified Permissions
Policy store for Premium 1 Tier
policyStoreId="store-p1-tier"

Pertimbangan desain multi-penyewa OPA

Open Policy Agent (OPA) adalah layanan fleksibel yang dapat diterapkan pada berbagai kasus penggunaan di mana aplikasi diperlukan untuk membuat keputusan kebijakan dan otorisasi. Menggunakan OPA dengan aplikasi SaaS multi-penyewa memerlukan pertimbangan kriteria unik untuk memastikan bahwa praktik terbaik SaaS utama seperti isolasi penyewa tetap menjadi bagian dari implementasi OPA. Kriteria ini termasuk pola penerapan OPA, isolasi penyewa dan model dokumen OPA, dan orientasi penyewa. Masing-masing mempengaruhi desain optimal untuk OPA karena berkaitan dengan aplikasi multi-tenant.

Meskipun diskusi di bagian ini berfokus pada OPA, konsep umum berakar pada pola pikir isolasi dan panduan yang diberikannya. Aplikasi SaaS harus selalu mempertimbangkan isolasi penyewa sebagai bagian dari desain mereka, dan prinsip umum isolasi ini meluas ke memasukkan OPA dalam aplikasi SaaS. OPA, jika digunakan dengan tepat, dapat menjadi bagian penting dari bagaimana isolasi diberlakukan dalam aplikasi SaaS. Bagian ini juga mereferensikan model isolasi SaaS inti seperti model SaaS silo dan model SaaS gabungan. Untuk informasi tambahan, lihat konsep isolasi inti dalam Kerangka AWS Well-Architected, SaaS Lens.

Pertimbangan desain:

- Membandingkan pola penyebaran terpusat dan terdistribusi
- Isolasi penyewa dengan model dokumen OPA
- Orientasi penyewa

Membandingkan pola penyebaran terpusat dan terdistribusi

Anda dapat menerapkan OPA dalam pola penyebaran terpusat atau terdistribusi, dan metode ideal untuk aplikasi multi-penyewa bergantung pada kasus penggunaan. Untuk contoh pola ini, lihat bagian Menggunakan PDP terpusat dengan PEP pada API dan Menggunakan PDP dan PEP terdistribusi pada API sebelumnya dalam panduan ini. Karena OPA dapat digunakan sebagai daemon dalam sistem operasi atau wadah, OPA dapat diimplementasikan dalam berbagai cara untuk mendukung aplikasi multi-tenant.

Dalam pola penyebaran terpusat, OPA digunakan sebagai wadah atau daemon dengan RESTful API yang tersedia untuk layanan lain dalam aplikasi. Ketika layanan memerlukan keputusan dari OPA, OPA RESTful API pusat dipanggil untuk menghasilkan keputusan ini. Pendekatan ini sederhana

untuk diterapkan dan dipelihara, karena hanya ada satu penyebaran OPA. Kelemahan dari pendekatan ini adalah tidak menyediakan mekanisme apa pun untuk mempertahankan pemisahan data penyewa. Karena hanya ada satu penyebaran OPA, semua data penyewa yang digunakan dalam keputusan OPA, termasuk data eksternal apa pun yang direferensikan oleh OPA, harus tersedia untuk OPA. Anda dapat mempertahankan isolasi data penyewa dengan pendekatan ini, tetapi harus ditegakkan oleh kebijakan dan struktur dokumen OPA atau akses ke data eksternal. Pola penerapan terpusat juga memerlukan latensi yang lebih tinggi, karena setiap keputusan otorisasi harus membuat panggilan RESTful API ke layanan lain.

Dalam pola penyebaran terdistribusi, OPA digunakan sebagai wadah atau daemon di samping layanan aplikasi multi-penyewa. Itu bisa digunakan sebagai wadah sespan atau sebagai daemon yang berjalan secara lokal pada sistem operasi. Untuk mengambil keputusan dari OPA, layanan membuat panggilan RESTful API ke penerapan OPA lokal. (Karena OPA dapat digunakan sebagai paket Go, Anda dapat menggunakan Go secara native untuk mengambil keputusan alih-alih menggunakan panggilan RESTful API.) Tidak seperti pola penyebaran terpusat, pola terdistribusi membutuhkan upaya yang jauh lebih kuat untuk menyebarkan, memelihara, dan memperbarui karena ada di beberapa area aplikasi. Manfaat dari pola penyebaran terdistribusi adalah kemampuan untuk mempertahankan isolasi data penyewa, terutama untuk aplikasi yang menggunakan model SaaS silo. Data khusus penyewa dapat diisolasi dalam penerapan OPA yang khusus untuk penyewa tersebut, karena OPA dalam model terdistribusi digunakan bersama penyewa. Selain itu, pola penyebaran terdistribusi memiliki latensi yang jauh lebih rendah daripada pola penerapan terpusat, karena setiap keputusan otorisasi dapat dibuat secara lokal.

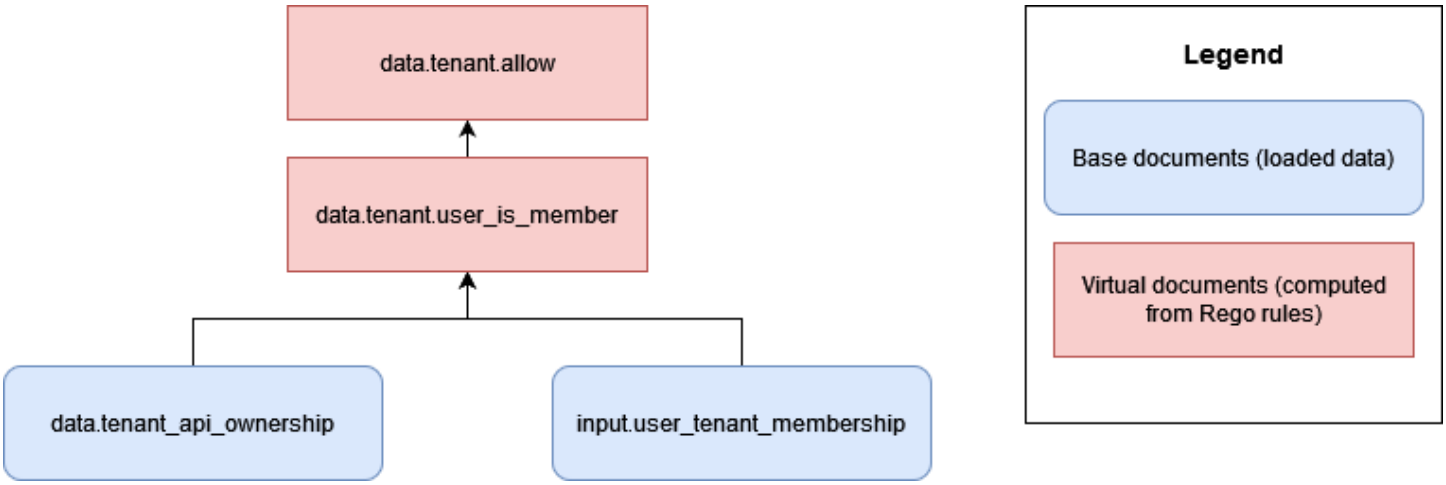
Saat Anda memilih pola penerapan OPA di aplikasi multi-tenant Anda, pastikan untuk mengevaluasi kriteria yang paling penting untuk aplikasi Anda. Jika aplikasi multi-tenant Anda sensitif terhadap latensi, pola penyebaran terdistribusi menawarkan kinerja yang lebih baik dengan mengorbankan penerapan dan pemeliharaan yang lebih kompleks. Meskipun Anda dapat mengelola beberapa kompleksitas ini melalui DevOps dan otomatisasi, masih membutuhkan upaya tambahan jika dibandingkan dengan pola penyebaran terpusat.

Jika aplikasi multi-tenant Anda menggunakan model SaaS siloed, Anda dapat menggunakan pola penyebaran OPA terdistribusi untuk meniru pendekatan siloed untuk isolasi data penyewa. Ini karena ketika OPA berjalan bersama setiap layanan aplikasi khusus penyewa, Anda dapat menyesuaikan setiap penyebaran OPA agar hanya berisi data yang terkait dengan penyewa tersebut. Siloing data OPA dalam pola penerapan OPA terpusat tidak dimungkinkan. Jika Anda menggunakan pola penyebaran terpusat atau pola terdistribusi bersama dengan model SaaS yang dikumpulkan, isolasi data penyewa harus dipertahankan dalam model dokumen OPA.

Isolasi penyewa dengan model dokumen OPA

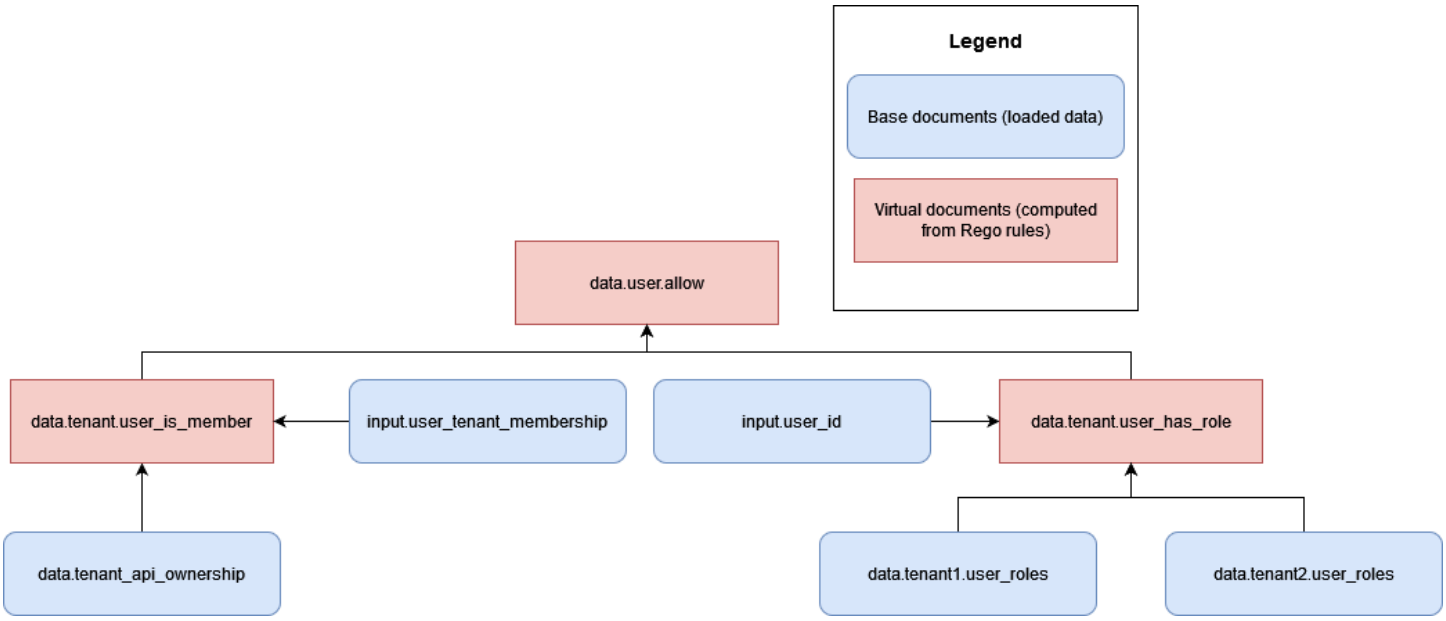
OPA menggunakan dokumen untuk membuat keputusan. Dokumen-dokumen ini dapat berisi data khusus penyewa, jadi Anda harus mempertimbangkan cara mempertahankan isolasi data penyewa. Dokumen OPA terdiri dari dokumen dasar dan dokumen virtual. Dokumen dasar berisi data dari dunia luar. Ini termasuk data yang diberikan kepada OPA secara langsung, data tentang permintaan OPA, dan data yang mungkin diteruskan ke OPA sebagai input. Dokumen virtual dihitung berdasarkan kebijakan dan mencakup kebijakan dan aturan OPA. Untuk informasi selengkapnya, lihat dokumentasi OPA.

Untuk merancang model dokumen di OPA untuk aplikasi multi-penyewa, Anda harus terlebih dahulu mempertimbangkan jenis dokumen dasar apa yang Anda perlukan untuk membuat keputusan di OPA. Jika dokumen dasar ini berisi data khusus penyewa, Anda harus mengambil tindakan untuk memastikan bahwa data ini tidak terpapar secara tidak sengaja ke akses lintas penyewa. Untungnya, dalam banyak kasus, data khusus penyewa tidak diperlukan untuk membuat keputusan di OPA. Contoh berikut menunjukkan model dokumen OPA hipotetis yang memungkinkan akses ke API berdasarkan penyewa mana yang memiliki API, dan apakah pengguna adalah anggota penyewa, seperti yang ditunjukkan dalam dokumen input.



Dalam pendekatan ini, OPA tidak memiliki akses ke data khusus penyewa apa pun kecuali untuk informasi tentang penyewa mana yang memiliki API. Dalam hal ini, tidak ada kekhawatiran atas OPA memfasilitasi akses lintas penyewa, karena satu-satunya informasi yang digunakan OPA untuk membuat keputusan akses adalah asosiasi pengguna dengan penyewa dan asosiasi penyewa dengan API. Anda dapat menerapkan jenis model dokumen OPA ini ke model SaaS silo, karena setiap penyewa akan memiliki kepemilikan sumber daya independen.

Namun, dalam banyak pendekatan otorisasi RBAC, ada potensi paparan informasi lintas penyewa. Dalam contoh berikut, model dokumen OPA hipotetis memungkinkan akses ke API berdasarkan apakah pengguna adalah anggota penyewa, dan apakah pengguna memiliki peran yang benar untuk mengakses API.



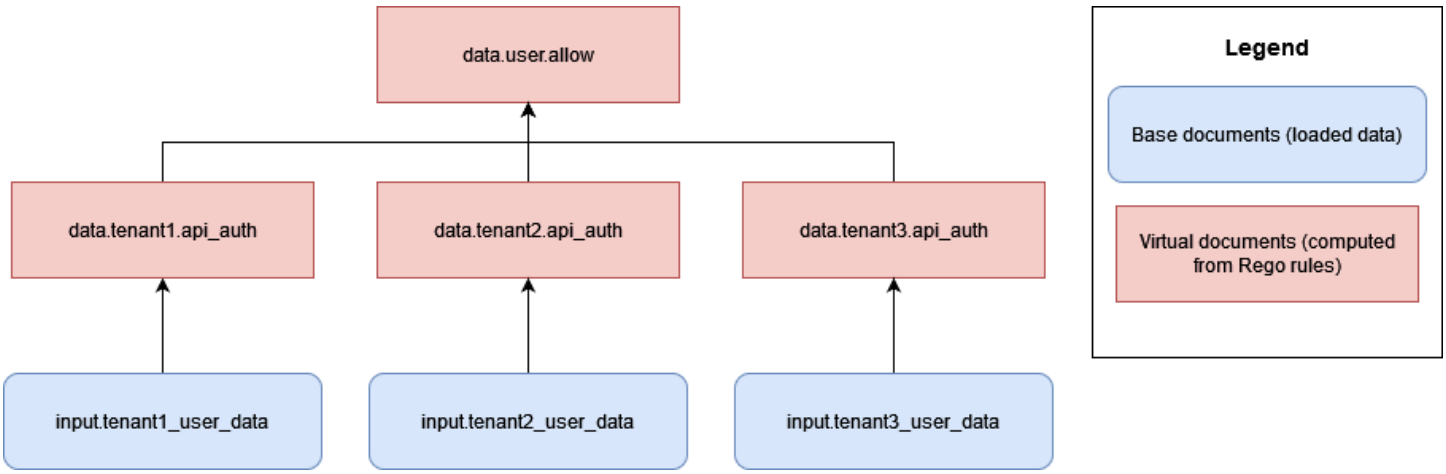
Model ini memperkenalkan risiko akses lintas penyewa, karena peran dan izin beberapa penyewa di `data.tenant1.user_roles` dan sekarang `data.tenant2.user_roles` harus dapat diakses oleh OPA untuk membuat keputusan otorisasi. Untuk menjaga isolasi penyewa dan privasi pemetaan peran, data ini tidak boleh berada dalam OPA. Data RBAC harus berada di sumber data eksternal seperti database. Selain itu, OPA tidak boleh digunakan untuk memetakan peran yang telah ditentukan ke izin tertentu, karena ini menyulitkan penyewa untuk menentukan peran dan izin mereka sendiri. Itu juga membuat logika otorisasi Anda kaku dan membutuhkan pembaruan konstan. Untuk panduan tentang cara memasukkan data RBAC dengan aman ke dalam proses pengambilan keputusan OPA, lihat bagian Rekomendasi untuk isolasi penyewa dan privasi data nanti dalam panduan ini.

Anda dapat dengan mudah mempertahankan isolasi penyewa di OPA dengan tidak menyimpan data khusus penyewa sebagai dokumen dasar asinkron. Dokumen dasar asinkron adalah data yang disimpan dalam memori dan dapat diperbarui secara berkala, di OPA. Dokumen dasar lainnya, seperti input OPA, diteruskan secara serempak dan hanya tersedia pada waktu pengambilan keputusan. Misalnya, memberikan data khusus penyewa sebagai bagian dari input OPA ke kueri tidak akan merupakan pelanggaran isolasi penyewa, karena data tersebut hanya tersedia secara sinkron selama proses pengambilan keputusan.

Orientasi penyewa

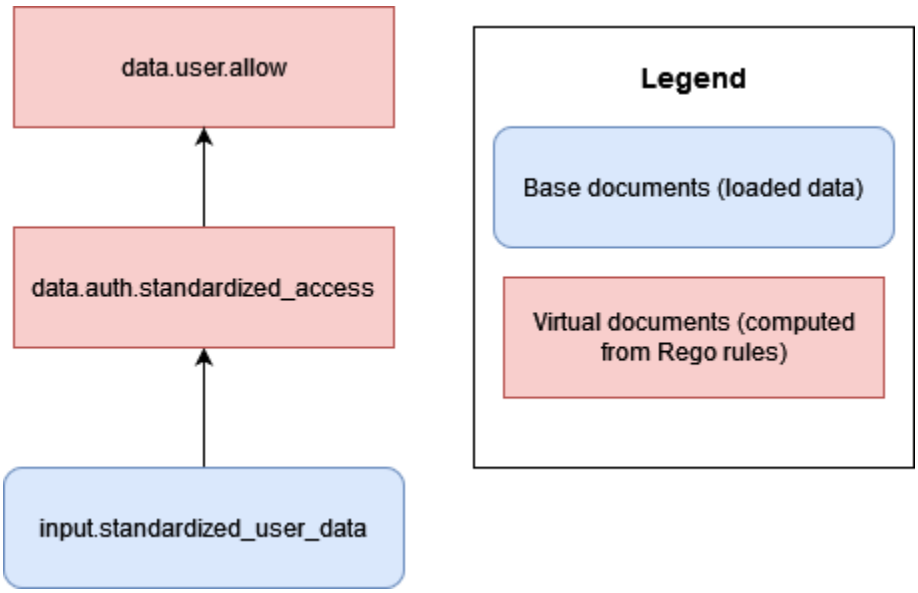
Struktur dokumen OPA harus memungkinkan orientasi penyewa langsung tanpa memperkenalkan persyaratan yang rumit. Anda dapat mengatur dokumen virtual dalam hierarki model dokumen OPA dengan paket, dan paket ini dapat berisi banyak aturan. Saat Anda merencanakan model dokumen OPA untuk aplikasi multi-penyewa, pertama-tama tentukan data mana yang diperlukan untuk OPA untuk membuat keputusan. Anda dapat memberikan data sebagai input, memuatnya terlebih dahulu ke OPA, atau menyediakannya dari sumber data eksternal pada waktu pengambilan keputusan atau secara berkala. Untuk informasi selengkapnya tentang penggunaan data eksternal dengan OPA, lihat bagian Mengambil data eksternal untuk PDP di OPA nanti dalam panduan ini.

Setelah Anda menentukan data yang diperlukan untuk membuat keputusan dalam OPA, pertimbangkan bagaimana menerapkan aturan OPA yang diatur sebagai paket, untuk membuat keputusan dengan data tersebut. Misalnya, dalam model SaaS silo di mana setiap penyewa mungkin memiliki persyaratan unik tentang bagaimana keputusan otorisasi dibuat, Anda dapat menerapkan paket aturan OPA khusus penyewa.



Kelemahan dari pendekatan ini adalah Anda harus menambahkan seperangkat aturan OPA baru, khusus untuk setiap penyewa, untuk setiap penyewa yang Anda tambahkan ke aplikasi SaaS Anda. Ini rumit dan sulit untuk skala, tetapi mungkin tidak dapat dihindari tergantung pada kebutuhan penyewa Anda.

Atau, dalam model SaaS yang dikumpulkan, jika semua penyewa membuat keputusan otorisasi berdasarkan aturan yang sama dan menggunakan struktur data yang sama, Anda dapat menggunakan paket OPA standar yang memiliki aturan yang berlaku umum untuk mempermudah penyewa onboard dan menskalakan implementasi OPA Anda.



Jika memungkinkan, kami menyarankan Anda menggunakan aturan dan paket OPA umum (atau dokumen virtual) untuk membuat keputusan berdasarkan data standar yang disediakan oleh setiap penyewa. Pendekatan ini membuat OPA mudah diskalakan, karena Anda hanya mengubah data yang diberikan ke OPA untuk setiap penyewa–bukan bagaimana OPA memberikan keputusannya melalui aturannya. Anda hanya perlu memperkenalkan rules-per-tenant model ketika penyewa individu memerlukan keputusan unik atau harus memberikan OPA data yang berbeda dari penyewa lainnya.

DevOps, memantau, mencatat, dan mengambil data untuk PDP

Dalam paradigma otorisasi yang diusulkan ini, kebijakan dipusatkan dalam layanan otorisasi. Sentralisasi ini disengaja karena salah satu tujuan dari model desain yang dibahas dalam panduan ini adalah untuk mencapai decoupling kebijakan, atau penghapusan logika otorisasi dari komponen lain dalam aplikasi. Izin Terverifikasi Amazon dan Agen Kebijakan Terbuka (OPA) menyediakan mekanisme untuk memperbarui kebijakan saat perubahan logika otorisasi diperlukan.

Dalam kasus Izin Terverifikasi, mekanisme untuk memperbarui kebijakan secara terprogram ditawarkan oleh AWS SDK (lihat Panduan Referensi API Izin Terverifikasi Amazon). Dengan menggunakan SDK, Anda dapat mendorong kebijakan baru sesuai permintaan. Selain itu, karena Izin Terverifikasi adalah layanan terkelola, Anda tidak perlu mengelola, mengonfigurasi, atau memelihara bidang kontrol atau agen untuk melakukan pembaruan. Namun, kami menyarankan Anda menggunakan pipeline integrasi berkelanjutan dan penerapan berkelanjutan (CI/CD) untuk mengelola penyebaran penyimpanan kebijakan Izin Terverifikasi dan pembaruan kebijakan menggunakan SDK. AWS

Izin Terverifikasi menyediakan akses mudah ke fitur observabilitas. Ini dapat dikonfigurasi untuk mencatat semua upaya akses ke AWS CloudTrail, grup CloudWatch log Amazon, bucket Amazon Simple Storage Service (Amazon S3), atau aliran pengiriman Amazon Data Firehose untuk memungkinkan respons cepat terhadap insiden keamanan dan permintaan audit. Selain itu, Anda dapat memantau kesehatan layanan Izin Terverifikasi melalui. AWS Health Dashboard Karena Izin Terverifikasi adalah layanan terkelola, kesehatannya dijaga oleh AWS, dan Anda dapat mengonfigurasi fitur observabilitas dengan menggunakan layanan AWS terkelola lainnya.

Dalam kasus OPA, REST API menawarkan cara untuk memperbarui kebijakan secara terprogram. Anda dapat mengonfigurasi API untuk menarik versi paket kebijakan baru dari lokasi yang ditetapkan atau untuk mendorong kebijakan sesuai permintaan. Selain itu, OPA menawarkan layanan penemuan dasar di mana agen baru dapat dikonfigurasi secara dinamis dan dikelola secara terpusat oleh pesawat kontrol yang mendistribusikan bundel penemuan. (Bidang kontrol untuk OPA harus diatur dan dikonfigurasi oleh operator OPA.) Sebaiknya Anda membuat pipeline CI/CD yang kuat untuk membuat versi, memverifikasi, dan memperbarui kebijakan, baik mesin kebijakan Izin Terverifikasi, OPA, atau solusi lain.

Untuk OPA, bidang kontrol juga menyediakan opsi untuk pemantauan dan audit. Anda dapat mengekspor log yang berisi keputusan otorisasi OPA ke server HTTP jarak jauh untuk agregasi log. Log keputusan ini sangat berharga untuk tujuan audit.

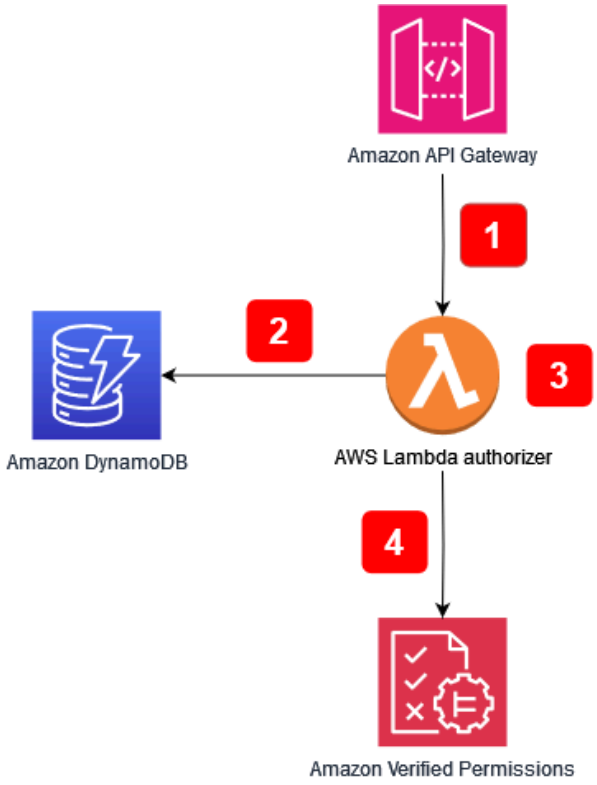
Jika Anda mempertimbangkan untuk mengadopsi model otorisasi di mana keputusan kontrol akses dipisahkan dari aplikasi Anda, pastikan bahwa layanan otorisasi Anda memiliki kemampuan pemantauan, pencatatan, dan manajemen CI/CD yang efektif untuk melakukan orientasi pada PDP baru atau memperbarui kebijakan.

Topik

Mengambil data eksternal untuk PDP di Izin Terverifikasi Amazon

Izin Terverifikasi Amazon tidak mendukung pengambilan data eksternal untuk PDP, tetapi dapat menyimpan data yang disediakan pengguna sebagai bagian dari skemanya. Seperti dalam OPA, jika semua data untuk keputusan otorisasi dapat diberikan sebagai bagian dari permintaan otorisasi atau sebagai bagian dari Token Web JSON (JWT) yang diteruskan sebagai bagian dari permintaan, tidak diperlukan konfigurasi tambahan. Namun, Anda dapat memberikan data tambahan dari sumber eksternal ke Izin Terverifikasi melalui permintaan otorisasi sebagai bagian dari layanan otorisasi aplikasi yang memanggil Izin Terverifikasi. Misalnya, layanan otorisasi aplikasi dapat meminta sumber eksternal seperti DynamoDB atau Amazon RDS untuk data, dan layanan ini kemudian dapat menyertakan data yang disediakan secara eksternal sebagai bagian dari permintaan otorisasi.

Diagram berikut menunjukkan contoh bagaimana data tambahan dapat diambil dan dimasukkan ke dalam permintaan otorisasi Izin Terverifikasi. Mungkin perlu menggunakan metode ini untuk mengambil data seperti pemetaan peran RBAC, untuk mengambil atribut tambahan yang relevan dengan sumber daya atau prinsipal, atau dalam kasus di mana data berada di berbagai bagian aplikasi dan tidak dapat disediakan melalui token penyedia identitas (IDP).



Aliran aplikasi:

1. Aplikasi menerima panggilan API ke Amazon API Gateway dan meneruskan panggilan ke AWS Lambda otorisasi.
2. Authorizer Lambda memanggil Amazon DynamoDB untuk mengambil data tambahan tentang prinsipal yang membuat permintaan.
3. Authorizer Lambda memasukkan data tambahan ke dalam permintaan otorisasi yang dibuat untuk Izin Terverifikasi.
4. Otorisasi Lambda membuat permintaan otorisasi ke Izin Terverifikasi dan menerima keputusan otorisasi.

Diagram mencakup fitur Amazon API Gateway yang disebut Lambda authorizer. Meskipun fitur ini mungkin tidak tersedia untuk API yang disediakan oleh layanan atau aplikasi lain, Anda dapat mereplikasi model umum menggunakan otorisasi untuk mengambil data tambahan untuk dimasukkan ke dalam permintaan otorisasi Izin Terverifikasi di banyak kasus penggunaan.

Mengambil data eksternal untuk PDP di OPA

Untuk OPA, jika semua data yang diperlukan untuk keputusan otorisasi dapat diberikan sebagai input atau sebagai bagian dari Token Web JSON (JWT) yang diteruskan sebagai komponen kueri, tidak diperlukan konfigurasi tambahan. (Relatif mudah untuk meneruskan data konteks JWT dan SaaS ke OPA sebagai bagian dari input kueri.) OPA dapat menerima input JSON arbitrer dalam apa yang disebut pendekatan input overload. Jika PDP memerlukan data di luar apa yang dapat dimasukkan sebagai input atau JWT, OPA menyediakan beberapa opsi untuk mengambil data ini. Ini termasuk bundling, mendorong data (replikasi), dan pengambilan data dinamis.

OPA bundling

Fitur bundling OPA mendukung proses berikut untuk pengambilan data eksternal:

1. Poin penegakan kebijakan (PEP) meminta keputusan otorisasi.
2. OPA mengunduh bundel kebijakan baru, termasuk data eksternal.
3. Layanan bundling mereplikasi data dari sumber data.

Saat Anda menggunakan fitur bundling, OPA secara berkala mengunduh kebijakan dan bundel data dari layanan bundel terpusat. (OPA tidak menyediakan implementasi dan pengaturan layanan bundel.) Semua kebijakan dan data eksternal yang ditarik dari layanan bundel disimpan dalam memori. Opsi ini tidak akan berfungsi jika ukuran data eksternal terlalu besar untuk disimpan dalam memori, atau jika data berubah terlalu sering.

Untuk informasi selengkapnya tentang fitur bundling, lihat dokumentasi OPA.

Replikasi OPA (mendorong data)

Pendekatan replikasi OPA mendukung proses berikut untuk pengambilan data eksternal:

1. PEP meminta keputusan otorisasi.
2. Replikator data mendorong data ke OPA.
3. Replikator data mereplikasi data dari sumber data.

Dalam alternatif pendekatan bundling ini, data didorong ke, alih-alih ditarik secara berkala oleh, OPA. (OPA tidak menyediakan implementasi dan pengaturan replikator.) Pendekatan push memiliki batasan ukuran data yang sama dengan pendekatan bundling, karena OPA menyimpan semua

data dalam memori. Keuntungan utama dari opsi push adalah Anda dapat memperbarui data di OPA dengan delta alih-alih mengganti semua data eksternal setiap kali. Ini membuat opsi push lebih sesuai untuk kumpulan data yang sering berubah.

Untuk informasi selengkapnya tentang opsi replikasi, lihat dokumentasi OPA.

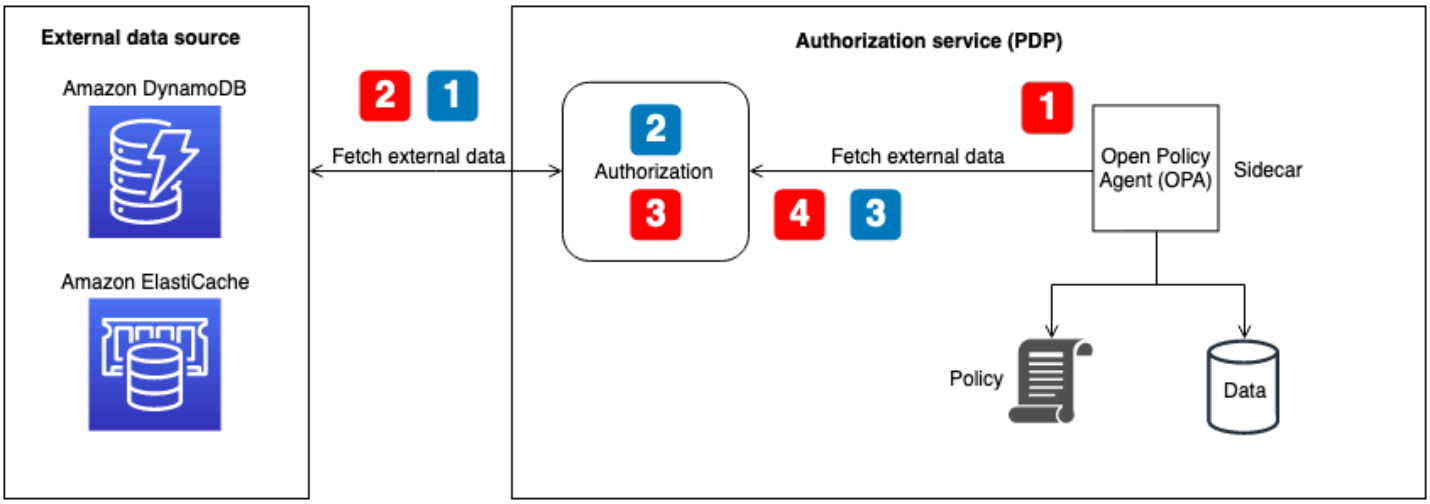
Pengambilan data dinamis OPA

Jika data eksternal yang akan diambil terlalu besar untuk di-cache dalam memori OPA, data dapat ditarik secara dinamis dari sumber eksternal selama evaluasi keputusan otorisasi. Ketika Anda menggunakan pendekatan ini, data selalu up to date. Pendekatan ini memiliki dua kelemahan: latensi jaringan dan aksesibilitas. Saat ini, OPA dapat mengambil data saat runtime hanya melalui permintaan HTTP. Jika panggilan yang masuk ke sumber data eksternal tidak dapat mengembalikan data sebagai respons HTTP, panggilan tersebut memerlukan API khusus atau mekanisme lain untuk menyediakan data ini ke OPA. Karena OPA dapat mengambil data hanya melalui permintaan HTTP, dan kecepatan pengambilan data sangat penting, kami sarankan Anda menggunakan seperti Layanan AWS Amazon DynamoDB untuk menyimpan data eksternal bila memungkinkan.

Untuk informasi lebih lanjut tentang pendekatan tarik, lihat dokumentasi OPA.

Menggunakan layanan otorisasi untuk implementasi dengan OPA

Saat Anda mengambil data eksternal dengan menggunakan bundling, replikasi, atau pendekatan tarik dinamis, kami menyarankan agar layanan otorisasi memfasilitasi interaksi ini. Ini karena layanan otorisasi dapat mengambil data eksternal dan mengubahnya menjadi JSON untuk OPA untuk membuat keputusan otorisasi. Diagram berikut menunjukkan bagaimana layanan otorisasi dapat berfungsi dengan tiga pendekatan pengambilan data eksternal ini.



Mengambil data eksternal untuk aliran OPA — bundel atau pengambilan data dinamis pada waktu keputusan (diilustrasikan dengan callout bernomor merah dalam diagram):

1. OPA memanggil titik akhir API lokal untuk layanan otorisasi, yang dikonfigurasi sebagai titik akhir bundel atau titik akhir untuk pengambilan data dinamis selama keputusan otorisasi.
2. Layanan otorisasi meminta atau memanggil sumber data eksternal untuk mengambil data eksternal. (Untuk titik akhir bundel, data ini juga harus berisi kebijakan dan aturan OPA. Pembaruan bundel menggantikan semuanya—baik data maupun kebijakan—dalam cache OPA.)
3. Layanan otorisasi melakukan transformasi apa pun yang diperlukan pada data yang dikembalikan untuk mengubahnya menjadi input JSON yang diharapkan.
4. Data dikembalikan ke OPA. Ini di-cache dalam memori untuk konfigurasi bundel dan digunakan segera untuk keputusan otorisasi dinamis.

Mengambil data eksternal untuk aliran OPA — replikator (diilustrasikan dengan callout bernomor biru dalam diagram):

1. Replikator (bagian dari layanan otorisasi) memanggil sumber data eksternal dan mengambil data apa pun yang akan diperbarui di OPA. Ini dapat mencakup kebijakan, aturan, dan data eksternal. Panggilan ini dapat berada pada irama yang ditetapkan, atau dapat terjadi sebagai respons terhadap pembaruan data di sumber eksternal.
2. Layanan otorisasi melakukan transformasi apa pun yang diperlukan pada data yang dikembalikan untuk mengubahnya menjadi input JSON yang diharapkan.
3. Layanan otorisasi memanggil OPA dan menyimpan data dalam memori. Layanan otorisasi dapat secara selektif memperbarui data, kebijakan, dan aturan.

Rekomendasi untuk isolasi penyewa dan privasi data

Bagian sebelumnya menyediakan beberapa pendekatan untuk menggunakan data eksternal dengan Izin Terverifikasi OPA dan Amazon untuk membantu dalam membuat keputusan otorisasi. Jika memungkinkan, kami menyarankan Anda menggunakan pendekatan input kelebihan beban untuk meneruskan data konteks SaaS ke OPA untuk membuat keputusan otorisasi alih-alih menyimpan data dalam memori OPA. Kasus penggunaan ini tidak berlaku untuk AWS Cloud Map, karena tidak mendukung penyimpanan data eksternal dalam layanan.

Dalam model hibrida kontrol akses berbasis peran (RBAC) atau RBAC dan kontrol akses berbasis atribut (ABAC), data yang disediakan semata-mata oleh permintaan otorisasi atau kueri mungkin

tidak cukup, karena peran dan izin harus direferensikan untuk membuat keputusan otorisasi. Untuk menjaga isolasi penyewa dan privasi pemetaan peran, data ini tidak boleh berada dalam OPA. Data RBAC harus berada di sumber data eksternal seperti database atau harus diteruskan sebagai bagian dari klaim dalam JWT dari iDP. Dalam Izin Terverifikasi, data RBAC dapat dipertahankan sebagai bagian dari kebijakan dan skema dalam model penyimpanan kebijakan per penyewa, karena setiap penyewa memiliki penyimpanan kebijakan terpisah secara logis. Namun, dalam satu model penyimpanan kebijakan multi-penyewa bersama, data pemetaan peran tidak boleh berada dalam Izin Terverifikasi untuk mempertahankan isolasi penyewa.

Selain itu, OPA dan Izin Terverifikasi tidak boleh digunakan untuk memetakan peran yang telah ditentukan sebelumnya ke izin tertentu, karena ini menyulitkan penyewa untuk menentukan peran dan izin mereka sendiri. Itu juga membuat logika otorisasi Anda kaku dan membutuhkan pembaruan konstan. Pengecualian untuk pedoman ini adalah model penyimpanan kebijakan per penyewa di Izin Terverifikasi, karena model ini memungkinkan setiap penyewa memiliki kebijakan sendiri yang dapat dievaluasi secara independen berdasarkan per-penyewa.

Izin Terverifikasi Amazon

Satu-satunya tempat di mana Izin Terverifikasi dapat menyimpan data RBAC yang berpotensi pribadi adalah dalam skema. Ini dapat diterima dalam model toko kebijakan per penyewa, karena setiap penyewa memiliki toko kebijakan terpisah secara logis. Namun, itu dapat membahayakan isolasi penyewa dalam model toko kebijakan multi-penyewa bersama. Dalam kasus di mana data ini diperlukan untuk membuat keputusan otorisasi, data tersebut harus diambil dari sumber eksternal seperti DynamoDB atau Amazon RDS dan dimasukkan ke dalam permintaan otorisasi Izin Terverifikasi.

OPA

Pendekatan aman dengan OPA untuk menjaga privasi dan isolasi penyewa data RBAC termasuk menggunakan pengambilan atau replikasi data dinamis untuk mendapatkan data eksternal. Ini karena Anda dapat menggunakan layanan otorisasi yang diilustrasikan dalam diagram sebelumnya untuk hanya menyediakan data eksternal khusus penyewa atau khusus pengguna untuk membuat keputusan otorisasi. Misalnya, Anda dapat menggunakan replikator untuk menyediakan data RBAC atau matriks izin ke cache OPA saat pengguna masuk, dan meminta data direferensikan berdasarkan pengguna yang disediakan dalam data input. Anda dapat menggunakan pendekatan serupa dengan data yang ditarik secara dinamis untuk mengambil hanya data yang relevan untuk membuat keputusan otorisasi. Selanjutnya, dalam pendekatan pengambilan data dinamis, data ini tidak harus di-cache di OPA. Pendekatan bundling tidak seefektif pendekatan pengambilan dinamis

dalam mempertahankan isolasi penyewa, karena memperbarui semua yang ada di cache OPA dan tidak dapat memproses pembaruan yang tepat. Model bundling masih merupakan pendekatan yang baik untuk memperbarui kebijakan OPA dan data non-RBAC.

Praktik terbaik

Bagian ini mencantumkan beberapa takeaways tingkat tinggi dari panduan ini. Untuk diskusi terperinci tentang setiap poin, ikuti tautan ke bagian yang sesuai.

Pilih model kontrol akses yang berfungsi untuk aplikasi Anda

Panduan ini membahas beberapa model kontrol akses. Bergantung pada aplikasi dan persyaratan bisnis Anda, Anda harus memilih model yang sesuai untuk Anda. Pertimbangkan bagaimana Anda dapat menggunakan model ini untuk memenuhi kebutuhan kontrol akses Anda, dan bagaimana kebutuhan kontrol akses Anda mungkin berkembang, yang memerlukan perubahan pada pendekatan yang Anda pilih.

Menerapkan PDP

Policy decision point (PDP) dapat dicirikan sebagai mesin kebijakan atau aturan. Komponen ini bertanggung jawab untuk menerapkan kebijakan atau aturan dan mengembalikan keputusan apakah akses tertentu diizinkan. PDP memungkinkan logika otorisasi dalam kode aplikasi untuk diturunkan ke sistem terpisah. Ini dapat menyederhanakan kode aplikasi. Ini juga menyediakan antarmuka easy-to-use idempoten untuk membuat keputusan otorisasi untuk API, microservices, Backend for Frontend (BFF) layer, atau komponen aplikasi lainnya. PDP dapat digunakan untuk menegakkan persyaratan sewa secara konsisten di seluruh aplikasi.

Menerapkan PEP untuk setiap API di aplikasi Anda

Implementasi titik penegakan kebijakan (PEP) memerlukan penentuan di mana penegakan kontrol akses harus terjadi dalam suatu aplikasi. Sebagai langkah pertama, temukan titik-titik dalam aplikasi Anda di mana Anda dapat memasukkan PEP. Pertimbangkan prinsip ini saat memutuskan di mana menambahkan PEP:

Jika aplikasi mengekspos API, harus ada otorisasi dan kontrol akses pada API itu.

Pertimbangkan untuk menggunakan Izin Terverifikasi Amazon atau OPA sebagai mesin kebijakan untuk PDP Anda

Izin Terverifikasi Amazon memiliki keunggulan dibandingkan mesin kebijakan khusus. Ini adalah layanan manajemen izin dan otorisasi yang dapat diskalakan dan berbutir halus untuk aplikasi yang Anda buat. Ini mendukung kebijakan penulisan dalam bahasa open-source deklaratif tingkat tinggi Cedar. Akibatnya, menerapkan mesin kebijakan dengan menggunakan Izin Terverifikasi membutuhkan lebih sedikit upaya pengembangan daripada menerapkan solusi Anda sendiri. Selain itu, Izin Terverifikasi dikelola sepenuhnya, sehingga Anda tidak perlu mengelola infrastruktur yang mendasarinya.

Open Policy Agent (OPA) memiliki keunggulan dibandingkan mesin kebijakan khusus. OPA dan evaluasi kebijakannya dengan Rego menyediakan mesin kebijakan pra-bangun yang fleksibel yang mendukung penulisan kebijakan dalam bahasa deklaratif tingkat tinggi. Hal ini membuat tingkat upaya yang diperlukan untuk menerapkan mesin kebijakan jauh lebih sedikit daripada membangun solusi Anda sendiri. Selain itu, OPA dengan cepat menjadi standar otorisasi yang didukung dengan baik.

Menerapkan bidang kontrol untuk OPA untuk DevOps, pemantauan, dan pencatatan

Karena OPA tidak menyediakan sarana untuk memperbarui dan melacak perubahan logika otorisasi melalui kontrol sumber, kami menyarankan Anda menerapkan bidang kontrol untuk menjalankan fungsi-fungsi ini. Ini akan memungkinkan pembaruan untuk lebih mudah didistribusikan ke agen OPA, terutama jika OPA beroperasi dalam sistem terdistribusi, yang akan mengurangi beban administrasi menggunakan OPA. Selain itu, pesawat kontrol dapat digunakan untuk mengumpulkan log untuk agregasi dan untuk memantau status agen OPA.

Konfigurasikan fitur logging dan observabilitas di Izin Terverifikasi

Izin Terverifikasi menyediakan akses mudah ke fitur observabilitas. Anda dapat mengonfigurasi layanan untuk mencatat semua upaya akses ke AWS CloudTrail, grup CloudWatch log Amazon, bucket S3, atau aliran pengiriman Amazon Data Firehose untuk mengaktifkan respons cepat terhadap insiden keamanan dan permintaan audit. Selain itu, Anda dapat memantau kesehatan layanan melalui AWS Health Dashboard. Karena Izin Terverifikasi adalah layanan terkelola,

kesehatannya dipertahankan oleh AWS, dan Anda dapat mengonfigurasi fitur observabilitasnya dengan menggunakan layanan AWS terkelola lainnya.

Menggunakan pipeline CI/CD untuk menyediakan dan memperbarui penyimpanan kebijakan dan kebijakan di Izin Terverifikasi

Izin Terverifikasi adalah layanan terkelola, jadi Anda tidak perlu mengelola, mengonfigurasi, atau memelihara bidang kontrol atau agen untuk melakukan pembaruan. Namun, kami tetap menyarankan agar Anda menggunakan pipeline integrasi berkelanjutan dan penerapan berkelanjutan (CI/CD) untuk mengelola penyebaran penyimpanan kebijakan Izin Terverifikasi dan pembaruan kebijakan dengan menggunakan SDK. AWS Upaya ini dapat menghapus upaya manual dan mengurangi kemungkinan kesalahan operator saat Anda membuat perubahan pada sumber daya Izin Terverifikasi.

Tentukan apakah data eksternal diperlukan untuk keputusan otorisasi, dan pilih model untuk mengakomodasinya

Jika PDP dapat membuat keputusan otorisasi hanya berdasarkan data yang terkandung dalam JSON Web Token (JWT), biasanya tidak perlu mengimpor data eksternal untuk membantu dalam membuat keputusan ini. Jika Anda menggunakan Izin Terverifikasi atau OPA sebagai PDP, itu juga dapat menerima masukan tambahan yang diteruskan sebagai bagian dari permintaan, bahkan jika data ini tidak disertakan dalam JWT. Untuk Izin Terverifikasi, Anda dapat menggunakan parameter konteks untuk data tambahan. Untuk OPA, Anda dapat menggunakan data JSON sebagai input overload. Jika Anda menggunakan JWT, konteks atau metode input kelebihan beban umumnya jauh lebih mudah daripada mempertahankan data eksternal di sumber lain. Jika data eksternal yang lebih kompleks diperlukan untuk membuat keputusan otorisasi, OPA menawarkan beberapa model untuk mengambil data eksternal, dan Izin Terverifikasi dapat melengkapi data dalam permintaan otorisasi dengan merujuk sumber eksternal dengan layanan otorisasi.

Pertanyaan yang Sering Diajukan

Bagian ini memberikan jawaban atas pertanyaan yang sering diajukan tentang penerapan kontrol akses API dan otorisasi dalam aplikasi SaaS multi-penyewa.

Q. Apa perbedaan antara otorisasi dan otentikasi?

A. Otentikasi adalah proses verifikasi siapa pengguna. Otorisasi memberikan izin kepada pengguna untuk mengakses sumber daya tertentu.

T. Apa perbedaan antara otorisasi dan isolasi penyewa dalam aplikasi SaaS?

Isolasi penyewa mengacu pada mekanisme eksplisit yang digunakan dalam sistem SaaS untuk memastikan bahwa sumber daya setiap penyewa, bahkan ketika beroperasi pada infrastruktur bersama, terisolasi. Otorisasi multi-penyewa mengacu pada otorisasi tindakan masuk dan mencegahnya diterapkan pada penyewa yang salah. Pengguna hipotetis dapat diautentikasi dan diotorisasi, tetapi mungkin masih dapat mengakses sumber daya penyewa lain. Tidak ada tentang otentikasi dan otorisasi yang menghalangi akses ini, tetapi isolasi penyewa diperlukan untuk mencapai tujuan ini. Untuk informasi lebih lanjut tentang kedua konsep ini, lihat diskusi isolasi penyewa di whitepaper SaaS Architecture AWS Fundamentals.

T. Mengapa saya perlu mempertimbangkan isolasi penyewa untuk aplikasi SaaS saya?

A. Aplikasi SaaS memiliki banyak penyewa. Penyewa dapat berupa organisasi pelanggan atau entitas eksternal apa pun yang menggunakan aplikasi SaaS itu. Bergantung pada bagaimana aplikasi dirancang, ini berarti penyewa dapat mengakses API bersama, database, atau sumber daya lainnya. Penting untuk mempertahankan isolasi penyewa — yaitu, konstruksi yang mengontrol akses ke sumber daya secara ketat, dan memblokir setiap upaya untuk mengakses sumber daya penyewa lain — untuk mencegah pengguna dari satu penyewa mengakses informasi pribadi penyewa lain. Aplikasi SaaS sering dirancang untuk memastikan bahwa isolasi penyewa dipertahankan di seluruh aplikasi dan penyewa hanya dapat mengakses sumber daya mereka sendiri.

T. Mengapa saya memerlukan model kontrol akses?

Model kontrol akses digunakan untuk membuat metode yang konsisten untuk menentukan bagaimana memberikan akses ke sumber daya dalam aplikasi. Ini dapat dilakukan dengan menetapkan peran kepada pengguna yang selaras dengan logika bisnis, atau dapat didasarkan pada atribut kontekstual lainnya seperti waktu hari atau apakah pengguna memenuhi kondisi yang telah

ditentukan sebelumnya. Model kontrol akses membentuk logika dasar yang digunakan aplikasi Anda saat membuat keputusan otorisasi untuk menentukan izin pengguna.

Q. Apakah kontrol akses API diperlukan untuk aplikasi saya?

A. Ya. API harus selalu memverifikasi bahwa penelepon memiliki akses yang sesuai. Kontrol akses API yang meresap juga memastikan bahwa akses hanya diberikan berdasarkan penyewa sehingga isolasi yang tepat dapat dipertahankan.

T. Mengapa mesin kebijakan atau PDP direkomendasikan untuk otorisasi?

A. Policy Decision Point (PDP) memungkinkan logika otorisasi dalam kode aplikasi diturunkan ke sistem yang terpisah. Ini dapat menyederhanakan kode aplikasi. Ini juga menyediakan antarmuka easy-to-use idempoten untuk membuat keputusan otorisasi untuk API, microservices, Backend for Frontend (BFF) layer, atau komponen aplikasi lainnya.

Q. Apa itu PEP?

A. Policy Enforcement Point (PEP) bertanggung jawab untuk menerima permintaan otorisasi yang dikirim ke PDP untuk evaluasi. PEP dapat berada di mana saja dalam aplikasi di mana data dan sumber daya harus dilindungi, atau di mana logika otorisasi diterapkan. PEP relatif sederhana dibandingkan dengan PDP. PEP bertanggung jawab hanya untuk meminta dan mengevaluasi keputusan otorisasi dan tidak memerlukan logika otorisasi apa pun untuk dimasukkan ke dalamnya.

T. Bagaimana saya harus memilih antara Izin Terverifikasi Amazon dan OPA?

A. Untuk memilih antara Izin Terverifikasi dan Agen Kebijakan Terbuka (OPA), selalu ingat kasus penggunaan dan persyaratan unik Anda. Izin Terverifikasi menyediakan cara yang dikelola sepenuhnya untuk menentukan izin berbutir halus, izin audit di seluruh aplikasi, dan memusatkan sistem administrasi kebijakan untuk aplikasi Anda sambil memenuhi persyaratan latensi aplikasi Anda dengan pemrosesan milidetik. OPA adalah open source, mesin kebijakan tujuan umum yang juga dapat membantu Anda menyatukan kebijakan di seluruh tumpukan aplikasi Anda. Untuk menjalankan OPA, Anda perlu meng-hostingnya di AWS lingkungan Anda, biasanya dengan wadah atau AWS Lambda fungsi.

Izin Terverifikasi menggunakan bahasa kebijakan Cedar open source, sedangkan OPA menggunakan Rego. Oleh karena itu, keakraban dengan salah satu bahasa ini mungkin mempengaruhi Anda untuk memilih solusi itu. Namun, kami menyarankan Anda membaca tentang kedua bahasa dan kemudian bekerja kembali dari masalah yang Anda coba selesaikan untuk menemukan solusi terbaik untuk kasus penggunaan Anda.

T. Apakah ada alternatif sumber terbuka untuk Izin Terverifikasi dan OPA?

Ada beberapa sistem open-source yang mirip dengan Verified Permissions dan Open Policy Agent (OPA), seperti Common Expression Language (CEL). Panduan ini berfokus pada Izin Terverifikasi, sebagai manajemen izin yang dapat diskalakan dan layanan otorisasi berbutir halus, dan OPA, yang diadopsi secara luas, didokumentasikan, dan dapat disesuaikan dengan berbagai jenis aplikasi dan persyaratan otorisasi.

T. Apakah saya perlu menulis layanan otorisasi untuk menggunakan OPA, atau dapatkah saya berinteraksi dengan OPA secara langsung?

A. Anda dapat berinteraksi dengan OPA secara langsung. Layanan otorisasi dalam konteks panduan ini mengacu pada layanan yang menerjemahkan permintaan keputusan otorisasi ke dalam kueri OPA, dan sebaliknya. Jika aplikasi Anda dapat menanyakan dan menerima tanggapan OPA secara langsung, tidak perlu memperkenalkan kompleksitas tambahan ini.

T. Bagaimana cara memantau agen OPA saya untuk tujuan uptime dan audit?

A. OPA menyediakan pencatatan dan pemantauan uptime dasar, meskipun konfigurasi defaultnya kemungkinan tidak akan cukup untuk penerapan perusahaan. Untuk informasi selengkapnya, lihat bagian DevOps, pemantauan, dan pencatatan sebelumnya dalam panduan ini.

T. Bagaimana cara memantau Izin Terverifikasi untuk tujuan uptime dan audit?

A. Izin Terverifikasi adalah layanan AWS terkelola, dan dapat dipantau ketersediaannya melalui. AWS Health Dashboard Selain itu, Izin Terverifikasi mampu masuk ke AWS CloudTrail, Amazon CloudWatch Log, Amazon S3, dan Amazon Data Firehose.

T. Sistem operasi dan layanan AWS mana yang dapat saya gunakan untuk menjalankan OPA?

A. Anda dapat menjalankan OPA di macOS, Windows, dan Linux. Agen OPA dapat dikonfigurasi pada agen Amazon Elastic Compute Cloud (Amazon EC2) serta layanan container seperti Amazon Elastic Container Service (Amazon ECS) dan Amazon Elastic Kubernetes Service (Amazon EKS) dan Amazon Elastic Kubernetes Service (Amazon EKS).

T. Sistem operasi dan AWS layanan apa yang dapat saya gunakan untuk menjalankan Izin Terverifikasi?

A. Izin Terverifikasi adalah layanan AWS terkelola dan dioperasikan oleh AWS. Tidak diperlukan konfigurasi, instalasi, atau hosting tambahan untuk menggunakan Izin Terverifikasi kecuali kemampuan untuk membuat permintaan otorisasi ke layanan.

T. Dapatkah saya menjalankan OPA? AWS Lambda

A. Anda dapat menjalankan OPA di Lambda sebagai pustaka Go. Untuk informasi tentang bagaimana Anda dapat melakukan ini untuk otorisasi Lambda API Gateway, lihat posting AWS blog Membuat otorisasiLambda kustom menggunakan Agen Kebijakan Terbuka.

T. Bagaimana saya harus memutuskan antara PDP terdistribusi dan pendekatan PDP terpusat?

A. Hal ini tergantung pada aplikasi Anda. Kemungkinan besar akan ditentukan berdasarkan perbedaan latensi antara model PDP terdistribusi dan terpusat. Kami menyarankan Anda membuat bukti konsep dan menguji kinerja aplikasi Anda untuk memverifikasi solusi Anda.

T. Dapatkah saya menggunakan OPA untuk kasus penggunaan selain API?

A. Ya. Dokumentasi OPA memberikan contoh untuk Kubernetes, Envoy, Docker, Kafka, SSHdan sudo, dan Terraform. Selain itu, OPA dapat mengembalikan respons JSON sewenang-wenang ke kueri dengan menggunakan aturan paral Rego. Bergantung pada kueri, OPA dapat digunakan untuk menjawab banyak pertanyaan dengan tanggapan JSON.

T. Dapatkah saya menggunakan Izin Terverifikasi untuk kasus penggunaan selain API?

A. Ya. Izin Terverifikasi dapat memberikan `ALLOW` atau `DENY` tanggapan untuk setiap permintaan otorisasi yang diterimanya. Izin Terverifikasi dapat memberikan tanggapan otorisasi untuk aplikasi atau layanan apa pun yang memerlukan keputusan otorisasi.

T. Dapatkah saya membuat kebijakan dalam Izin Terverifikasi dengan menggunakan bahasa kebijakan IAM?

A. Tidak. Anda harus menggunakan bahasa kebijakan Cedar untuk membuat kebijakan. Cedar dirancang untuk mendukung manajemen izin untuk sumber daya aplikasi pelanggan, sedangkan bahasa kebijakan AWS Identity and Access Management (IAM) berkembang untuk mendukung kontrol akses untuk sumber daya. AWS

Langkah selanjutnya

Kompleksitas otorisasi dan kontrol akses API untuk aplikasi SaaS multi-penyewa dapat diatasi dengan mengadopsi pendekatan agnostik bahasa standar untuk membuat keputusan otorisasi. Pendekatan ini menggabungkan poin keputusan kebijakan (PDP) dan poin penegakan kebijakan (PDP) yang menegakkan akses secara fleksibel dan meresap. Berbagai pendekatan untuk kontrol akses — seperti kontrol akses berbasis peran (RBAC), kontrol akses berbasis atribut (ABAC), atau kombinasi keduanya — dapat dimasukkan ke dalam strategi kontrol akses yang kohesif. Menghapus logika otorisasi dari aplikasi menghilangkan overhead termasuk solusi ad hoc dalam kode aplikasi untuk mengatasi kontrol akses. Implementasi dan praktik terbaik yang dibahas dalam panduan ini dimaksudkan untuk menginformasikan dan membakukan pendekatan penerapan otorisasi dan kontrol akses API dalam aplikasi SaaS multi-penyewa. Anda dapat menggunakan panduan ini sebagai langkah pertama dalam mengumpulkan informasi dan merancang kontrol akses dan sistem otorisasi yang kuat untuk aplikasi Anda. Langkah selanjutnya:

- Tinjau kebutuhan otorisasi dan isolasi penyewa Anda, dan pilih model kontrol akses untuk aplikasi Anda.
- Buat bukti konsep untuk pengujian dengan menggunakan Izin Terverifikasi Amazon atau Agen Kebijakan Terbuka (OPA), atau dengan menulis mesin kebijakan kustom Anda sendiri.
- Identifikasi API dan lokasi dalam aplikasi Anda di mana PEP harus diimplementasikan.

Sumber daya

Referensi

- Dokumentasi Izin Terverifikasi Amazon (AWS dokumentasi)
- Cara menggunakan Izin Terverifikasi Amazon untuk otorisasi (AWS posting blog)
- Menerapkan Penyedia Kebijakan Otorisasi Kustom untuk Aplikasi Inti ASP.NET menggunakan Izin Terverifikasi Amazon (posting blog)AWS
- Kelola Peran dan hak dengan PBAC menggunakan AWS Izin Terverifikasi Amazon (posting blog)
- Kontrol akses SaaS menggunakan Izin Terverifikasi Amazon dengan toko kebijakan per penyewa (posting blog)AWS
- Dokumentasi resmi OPA
- Mengapa Perusahaan Harus Merangkul Proyek CNCF Yang Baru Lulus — Agen Kebijakan Terbuka (artikel Forbes oleh Janakiram MSV, 8 Februari 2021)
- Membuat otorisasi Lambda kustom menggunakan Open Policy Agent (posting blog)AWS
- Wujudkan kebijakan sebagai kode dengan AWS Cloud Development Kit melalui Open Policy Agent (posting AWS blog)
- Tata kelola cloud dan AWS kepatuhan terhadap kebijakan sebagai kode (posting AWS blog)
- Menggunakan Agen Kebijakan Terbuka di Amazon EKS (posting AWS blog)
- Kepatuhan sebagai Kode untuk Amazon ECS menggunakan Agen Kebijakan Terbuka, Amazon EventBridge, dan AWS Lambda (posting AWS blog)
- Penanggulangan berbasis kebijakan untuk Kubernetes - Bagian 1 (posting blog)AWS
- Menggunakan otorisasi API Gateway Lambda (dokumentasi)AWS

Alat

- The Cedar Playground (untuk menguji Cedar di browser)
- Repositori Cedar Github
- Referensi Bahasa Cedar
- Rego Playground (untuk menguji Rego di browser)
- Repositori OPA GitHub

Mitra

- Mitra Identitas and Access Management
- Mitra Keamanan Aplikasi
- Mitra Tata Kelola Cloud
- Mitra Keamanan dan Kepatuhan
- Mitra Operasi dan Otomasi Keamanan
- Mitra Teknik Keamanan

Riwayat dokumen

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

Perubahan	Deskripsi	Tanggal
Menambahkan detail dan contoh untuk Izin Terverifikasi Amazon	Menambahkan diskusi terperinci, contoh, dan kode untuk menggunakan Izin Terverifikasi Amazon untuk mengimplementasikan PDP. Bagian baru meliputi: • Menerapkan PDP dengan menggunakan Izin Terverifikasi Amazon • Model desain untuk Izin Terverifikasi Amazon • Pertimbangan desain multi-penyewa Izin Terverifikasi Amazon • Mengambil data eksternal untuk PDP di Izin Terverifikasi Amazon	28 Mei 2024
Informasi yang diklarifikasi	Mengklarifikasi PDP terdistribusi dengan PEP pada model desain API.	10 Januari 2024
Menambahkan informasi singkat tentang AWS layanan baru	Menambahkan informasi tentang Izin Terverifikasi Amazon, yang menyediakan fungsionalitas dan manfaat yang sama dengan OPA.	22 Mei 2023

—	Publikasi awal	17 Agustus 2021

AWS Glosarium Panduan Preskriptif

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

Nomor

7 Rs

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

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

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

A

ABAC

Lihat kontrol akses berbasis atribut.

layanan abstrak

Lihat layanan terkelola.

ASAM

Lihat atomisitas, konsistensi, isolasi, daya tahan.

migrasi aktif-aktif

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

migrasi aktif-pasif

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

fungsi agregat

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

AI

Lihat kecerdasan buatan.

AIOps

Lihat operasi kecerdasan buatan.

anonimisasi

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

anti-pola

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

kontrol aplikasi

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

portofolio aplikasi

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

kecerdasan buatan (AI)

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

operasi kecerdasan buatan (AIOps)

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

enkripsi asimetris

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

atomisitas, konsistensi, isolasi, daya tahan (ACID)

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

kontrol akses berbasis atribut (ABAC)

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

sumber data otoritatif

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

Zona Ketersediaan

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

AWS Kerangka Adopsi Cloud (AWS CAF)

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

AWS Kerangka Kualifikasi Beban Kerja (AWS WQF)

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

B

bot buruk

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

BCP

Lihat perencanaan kontinuitas bisnis.

grafik perilaku

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

sistem big-endian

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

klasifikasi biner

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

filter mekar

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

deployment biru/hijau

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

bot

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

botnet

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

cabang

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

akses break-glass

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

strategi brownfield

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

cache penyangga

Area memori tempat data yang paling sering diakses disimpan.

kemampuan bisnis

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

perencanaan kelangsungan bisnis (BCP)

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

C

KAFE

Lihat Kerangka Adopsi AWS Cloud.

penyebaran kenari

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

CCoE

Lihat Cloud Center of Excellence.

CDC

Lihat mengubah pengambilan data.

ubah pengambilan data (CDC)

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

rekayasa kekacauan

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

CI/CD

Lihat integrasi berkelanjutan dan pengiriman berkelanjutan.

klasifikasi

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

Enkripsi sisi klien

Enkripsi data secara lokal, sebelum target Layanan AWS menerimanya.

Cloud Center of Excellence (CCoE)

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

komputasi cloud

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

model operasi cloud

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

tahap adopsi cloud

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

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

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

CMDB

Lihat database manajemen konfigurasi.

repositori kode

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

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

cache dingin

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

data dingin

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

visi komputer (CV)

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

konfigurasi drift

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

database manajemen konfigurasi (CMDB)

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

paket kesesuaian

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

integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD)

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

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

CV

Lihat visi komputer.

D

data saat istirahat

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

klasifikasi data

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

penyimpangan data

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

data dalam transit

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

jala data

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

minimalisasi data

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

perimeter data

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

prapemrosesan data

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

asal data

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

subjek data

Individu yang datanya dikumpulkan dan diproses.

gudang data

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

bahasa definisi database (DDL)

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

bahasa manipulasi basis data (DHTML)

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

DDL

Lihat bahasa definisi database.

ansambel yang dalam

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

pembelajaran mendalam

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

defense-in-depth

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

administrator yang didelegasikan

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

deployment

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

lingkungan pengembangan

Lihat lingkungan.

kontrol detektif

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

pemetaan aliran nilai pengembangan (DVSM)

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

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

kembar digital

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

tabel dimensi

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

musibah

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

pemulihan bencana (DR)

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

DML~

Lihat bahasa manipulasi database.

desain berbasis domain

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

DR

Lihat pemulihan bencana.

deteksi drift

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

DVSM

Lihat pemetaan aliran nilai pengembangan.

E

EDA

Lihat analisis data eksplorasi.

komputasi tepi

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

enkripsi

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

kunci enkripsi

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

endianness

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

titik akhir

Lihat titik akhir layanan.

layanan endpoint

Layanan yang dapat Anda host di cloud pribadi virtual (VPC) untuk dibagikan dengan pengguna lain. Anda dapat membuat layanan endpoint dengan AWS PrivateLink dan memberikan izin kepada prinsipal lain Akun AWS atau ke AWS Identity and Access Management (IAM). Akun atau prinsipal ini dapat terhubung ke layanan endpoint Anda secara pribadi dengan membuat titik akhir VPC antarmuka. Untuk informasi selengkapnya, lihat Membuat layanan titik akhir di dokumentasi Amazon Virtual Private Cloud (Amazon VPC).

perencanaan sumber daya perusahaan (ERP)

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

enkripsi amplop

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

lingkungan

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

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

epik

Dalam metodologi tangkas, kategori fungsional yang membantu mengatur dan memprioritaskan pekerjaan Anda. Epik memberikan deskripsi tingkat tinggi tentang persyaratan dan tugas

implementasi. Misalnya, epos keamanan AWS CAF mencakup manajemen identitas dan akses, kontrol detektif, keamanan infrastruktur, perlindungan data, dan respons insiden. Untuk informasi selengkapnya tentang epos dalam strategi AWS migrasi, lihat panduan implementasi program.

ERP

Lihat perencanaan sumber daya perusahaan.

analisis data eksplorasi (EDA)

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

F

tabel fakta

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

gagal cepat

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

batas isolasi kesalahan

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

cabang fitur

Lihat cabang.

fitur

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

pentingnya fitur

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

transformasi fitur

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

FGAC

Lihat kontrol akses berbutir halus.

kontrol akses berbutir halus (FGAC)

Penggunaan beberapa kondisi untuk mengizinkan atau menolak permintaan akses.

migrasi flash-cut

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

G

pemblokiran geografis

Lihat pembatasan geografis.

pembatasan geografis (pemblokiran geografis)

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

Alur kerja Gitflow

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

strategi greenfield

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

pagar pembatas

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

H

HA

Lihat ketersediaan tinggi.

migrasi database heterogen

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

ketersediaan tinggi (HA)

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

modernisasi sejarawan

Pendekatan yang digunakan untuk memodernisasi dan meningkatkan sistem teknologi operasional (OT) untuk melayani kebutuhan industri manufaktur dengan lebih baik. Sejarawan adalah jenis database yang digunakan untuk mengumpulkan dan menyimpan data dari berbagai sumber di pabrik.

migrasi database homogen

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

data panas

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

perbaikan terbaru

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

periode hypercare

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

I

IAc

Lihat infrastruktur sebagai kode.

kebijakan berbasis identitas

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

aplikasi idle

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

IIoT

Lihat Internet of Things industri.

infrastruktur yang tidak dapat diubah

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

masuk (masuknya) VPC

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

migrasi inkremental

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

Industri 4.0

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

infrastruktur

Semua sumber daya dan aset yang terkandung dalam lingkungan aplikasi.

infrastruktur sebagai kode (IAc)

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

Internet of Things industri (IIoT)

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

inspeksi VPC

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

Internet of Things (IoT)

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

interpretabilitas

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

IoT

Lihat Internet of Things.

Perpustakaan informasi TI (ITIL)

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

Manajemen layanan TI (ITSM)

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

ITIL

Lihat perpustakaan informasi TI.

ITSM

Lihat manajemen layanan TI.

L

kontrol akses berbasis label (LBAC)

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

landing zone

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

migrasi besar

Migrasi 300 atau lebih server.

LBAC

Lihat kontrol akses berbasis label.

hak istimewa paling sedikit

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

angkat dan geser

Lihat 7 Rs.

sistem endian kecil

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

lingkungan yang lebih rendah

Lihat lingkungan.

M

pembelajaran mesin (ML)

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

cabang utama

Lihat cabang.

malware

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

layanan terkelola

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

sistem eksekusi manufaktur (MES)

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

PETA

Lihat Program Percepatan Migrasi.

mekanisme

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

akun anggota

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

MES

Lihat sistem eksekusi manufaktur.

Transportasi Telemetri Antrian Pesan (MQTT)

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

layanan mikro

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

arsitektur microservices

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

Program Percepatan Migrasi (MAP)

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

migrasi dalam skala

Proses memindahkan sebagian besar portofolio aplikasi ke cloud dalam gelombang, dengan lebih banyak aplikasi bergerak pada tingkat yang lebih cepat di setiap gelombang. Fase ini menggunakan praktik terbaik dan pelajaran yang dipetik dari fase sebelumnya untuk mengimplementasikan pabrik migrasi tim, alat, dan proses untuk merampingkan migrasi beban kerja melalui otomatisasi dan pengiriman tangkas. Ini adalah fase ketiga dari strategi AWS migrasi.

pabrik migrasi

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

metadata migrasi

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

pola migrasi

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

Penilaian Portofolio Migrasi (MPA)

Alat online yang menyediakan informasi untuk memvalidasi kasus bisnis untuk bermigrasi ke. AWS Cloud MPA menyediakan penilaian portofolio terperinci (ukuran kanan server, harga,

perbandingan TCO, analisis biaya migrasi) serta perencanaan migrasi (analisis data aplikasi dan pengumpulan data, pengelompokan aplikasi, prioritas migrasi, dan perencanaan gelombang). Alat MPA (memerlukan login) tersedia gratis untuk semua AWS konsultan dan konsultan APN Partner.

Penilaian Kesiapan Migrasi (MRA)

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

strategi migrasi

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

ML

Lihat pembelajaran mesin.

modernisasi

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

penilaian kesiapan modernisasi

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

aplikasi monolitik (monolit)

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

Anda dapat menggunakan arsitektur microservices. Untuk informasi lebih lanjut, lihat Mengurai monolit menjadi layanan mikro.

MPA

Lihat Penilaian Portofolio Migrasi.

MQTT

Lihat Transportasi Telemetri Antrian Pesan.

klasifikasi multiclass

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

infrastruktur yang bisa berubah

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

O

OAC

Lihat kontrol akses asal.

OAI

Lihat identitas akses asal.

OCM

Lihat manajemen perubahan organisasi.

migrasi offline

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

OI

Lihat integrasi operasi.

OLA

Lihat perjanjian tingkat operasional.

migrasi online

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

OPC-UA

Lihat Komunikasi Proses Terbuka - Arsitektur Terpadu.

Komunikasi Proses Terbuka - Arsitektur Terpadu (OPC-UA)

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

perjanjian tingkat operasional (OLA)

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

Tinjauan Kesiapan Operasional (ORR)

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

teknologi operasional (OT)

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

integrasi operasi (OI)

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

jejak organisasi

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

manajemen perubahan organisasi (OCM)

Kerangka kerja untuk mengelola transformasi bisnis utama yang mengganggu dari perspektif orang, budaya, dan kepemimpinan. OCM membantu organisasi mempersiapkan, dan transisi ke, sistem dan strategi baru dengan mempercepat adopsi perubahan, mengatasi masalah transisi, dan mendorong perubahan budaya dan organisasi. Dalam strategi AWS migrasi, kerangka kerja ini disebut percepatan orang, karena kecepatan perubahan yang diperlukan dalam proyek adopsi cloud. Untuk informasi lebih lanjut, lihat panduan OCM.

kontrol akses asal (OAC)

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

identitas akses asal (OAI)

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

ORR

Lihat tinjauan kesiapan operasional.

OT

Lihat teknologi operasional.

keluar (jalan keluar) VPC

Dalam arsitektur AWS multi-akun, VPC yang menangani koneksi jaringan yang dimulai dari dalam aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan

Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

P

batas izin

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

Informasi Identifikasi Pribadi (PII)

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

PII

Lihat informasi yang dapat diidentifikasi secara pribadi.

buku pedoman

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

PLC

Lihat pengontrol logika yang dapat diprogram.

PLM

Lihat manajemen siklus hidup produk.

kebijakan

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

persistensi poliglot

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

penilaian portofolio

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

predikat

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

predikat pushdown

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

kontrol preventif

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

principal

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

Privasi oleh Desain

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

zona host pribadi

Container yang menyimpan informasi tentang bagaimana Anda ingin Amazon Route 53 merespons kueri DNS untuk domain dan subdomainnya dalam satu atau beberapa VPC. Untuk

informasi selengkapnya, lihat Bekerja dengan zona yang dihosting pribadi di dokumentasi Route 53.

kontrol proaktif

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

manajemen siklus hidup produk (PLM)

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

lingkungan produksi

Lihat lingkungan.

pengontrol logika yang dapat diprogram (PLC)

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

pseudonimisasi

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

terbitkan/berlangganan (pub/sub)

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

Q

rencana kueri

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

regresi rencana kueri

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

R

Matriks RACI

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

ransomware

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

Matriks RASCI

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

RCAC

Lihat kontrol akses baris dan kolom.

replika baca

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

arsitek ulang

Lihat 7 Rs.

tujuan titik pemulihan (RPO)

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

tujuan waktu pemulihan (RTO)

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

refactor

Lihat 7 Rs.

Wilayah

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

regresi

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

rehost

Lihat 7 Rs.

melepaskan

Dalam proses penyebaran, tindakan mempromosikan perubahan pada lingkungan produksi.

memindahkan

Lihat 7 Rs.

memplatform ulang

Lihat 7 Rs.

pembelian kembali

Lihat 7 Rs.

ketahanan

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

kebijakan berbasis sumber daya

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

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

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

kontrol responsif

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

melestarikan

Lihat 7 Rs.

pensiun

Lihat 7 Rs.

rotasi

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

kontrol akses baris dan kolom (RCAC)

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

RPO

Lihat tujuan titik pemulihan.

RTO

Lihat tujuan waktu pemulihan.

buku runbook

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

D

SAML 2.0

Standar terbuka yang digunakan oleh banyak penyedia identitas (IdPs). Fitur ini memungkinkan sistem masuk tunggal gabungan (SSO), sehingga pengguna dapat masuk ke AWS Management Console atau memanggil operasi AWS API tanpa Anda harus membuat pengguna di IAM untuk semua orang di organisasi Anda. Untuk informasi lebih lanjut tentang federasi berbasis SAMP 2.0, lihat Tentang federasi berbasis SAMP 2.0 dalam dokumentasi IAM.

PENIPUAN

Lihat kontrol pengawasan dan akuisisi data.

SCP

Lihat kebijakan kontrol layanan.

Rahasia

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

kontrol keamanan

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

pengerasan keamanan

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

sistem informasi keamanan dan manajemen acara (SIEM)

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

otomatisasi respons keamanan

Tindakan yang telah ditentukan dan diprogram yang dirancang untuk secara otomatis merespons atau memulihkan peristiwa keamanan. Otomatisasi ini berfungsi sebagai kontrol keamanan detektif atau responsif yang membantu Anda menerapkan praktik terbaik AWS keamanan. Contoh tindakan respons otomatis termasuk memodifikasi grup keamanan VPC, menambal instans Amazon EC2, atau memutar kredensil.

enkripsi sisi server

Enkripsi data di tujuannya, oleh Layanan AWS yang menerimanya.

kebijakan kontrol layanan (SCP)

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

titik akhir layanan

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

perjanjian tingkat layanan (SLA)

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

indikator tingkat layanan (SLI)

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

tujuan tingkat layanan (SLO)

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

model tanggung jawab bersama

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

SIEM

Lihat informasi keamanan dan sistem manajemen acara.

titik kegagalan tunggal (SPOF)

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

SLA

Lihat perjanjian tingkat layanan.

SLI

Lihat indikator tingkat layanan.

SLO

Lihat tujuan tingkat layanan.

split-and-seed model

Pola untuk menskalakan dan mempercepat proyek modernisasi. Ketika fitur baru dan rilis produk didefinisikan, tim inti berpisah untuk membuat tim produk baru. Ini membantu meningkatkan kemampuan dan layanan organisasi Anda, meningkatkan produktivitas pengembang, dan

mendukung inovasi yang cepat. Untuk informasi lebih lanjut, lihat Pendekatan bertahap untuk memodernisasi aplikasi di. AWS Cloud

SPOF

Lihat satu titik kegagalan.

skema bintang

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

pola ara pencekik

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

subnet

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

kontrol pengawasan dan akuisisi data (SCADA)

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

enkripsi simetris

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

pengujian sintetis

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

T

tag

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

variabel target

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

daftar tugas

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

lingkungan uji

Lihat lingkungan.

pelatihan

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

gerbang transit

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

alur kerja berbasis batang

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

akses tepercaya

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

penyetelan

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

tim dua pizza

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

U

waswas

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

tugas yang tidak terdiferensiasi

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

lingkungan atas

Lihat lingkungan.

V

menyedot debu

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

kendali versi

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

Peering VPC

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

kerentanan

Kelemahan perangkat lunak atau perangkat keras yang membahayakan keamanan sistem.

W

cache hangat

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

data hangat

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

fungsi jendela

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

beban kerja

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

aliran kerja

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

CACING

Lihat menulis sekali, baca banyak.

WQF

Lihat Kerangka Kualifikasi Beban Kerja AWS.

tulis sekali, baca banyak (WORM)

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

Z

eksploitasi zero-day

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

kerentanan zero-day

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

aplikasi zombie

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

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