Panduan Developerr

AWS AppSync

AWS AppSync: Panduan Developerr

Table of Contents

Apakah AWS AppSync itu?

AWSAppSyncmemungkinkan pengembang untuk menghubungkan aplikasi dan layanan mereka ke data dan peristiwa dengan GraphQL dan Pub/Sub API yang aman, tanpa server, dan berkinerja tinggi. Anda dapat melakukan hal berikut dengan AWSAppSync:

- Akses data dari satu atau lebih sumber data dari satu titik akhir API GraphQL.
- Gabungkan beberapa API GraphQL sumber menjadi satu API GraphQL yang digabungkan.
- Publikasikan pembaruan data real-time ke aplikasi Anda.
- Manfaatkan keamanan, pemantauan, pencatatan, dan pelacakan bawaan, dengan caching opsional untuk latensi rendah.
- Hanya membayar permintaan API dan pesan real-time yang dikirimkan.

Topik

AWSAppSyncfitur

- Akses dan kueri data yang disederhanakan, didukung oleh GraphQL
- Tanpa server WebSockets untuk langganan GraphQL dan saluran pub/sub
- Caching sisi server untuk membuat data tersedia dalam cache dalam memori berkecepatan tinggi untuk latensi rendah
- JavaScriptdan TypeScript dukungan untuk menulis logika bisnis
- Keamanan perusahaan dengan API Pribadi untuk membatasi akses dan integrasi API AWS WAF
- Kontrol otorisasi bawaan, dengan dukungan untuk kunci API, IAM, Amazon Cognito, penyedia OpenID Connect, dan otorisasi Lambda untuk logika kustom.
- API gabungan untuk mendukung kasus penggunaan federasi

Untuk detail selengkapnya tentang masing-masing kemampuan ini, lihat AWSAppSyncfitur.

Apakah Anda pengguna AWS AppSync baru?

Kami menyarankan AWS AppSync pengguna baru, mulailah dengan membaca bagian berikut:

- Jika Anda tidak terbiasa dengan GraphQL, lihat. Memulai: Membuat API GraphQL pertama Anda
- Jika Anda membangun aplikasi yang menggunakan API GraphQL, lihat Membangun aplikasi klien dan. the section called “Data waktu nyata”
- Jika Anda mencari informasi resolver GraphQL, lihat yang berikut ini:

  JavaScript/TypeScript

  - Tutorial Resolver () JavaScript
  - Referensi Resolver () JavaScript

  VTL

  - Tutorial Resolver (VTL)
  - Referensi template pemetaan resolver (VTL)
- Jika Anda mencari AWS AppSync contoh proyek, pembaruan, dan lainnya, lihat AppSyncblognya.

Layanan terkait

Jika Anda membangun web atau aplikasi seluler dari bawah ke atas, pertimbangkan untuk menggunakannya AWS Amplify. Perkuat Amplify AWS AppSync dan AWS layanan lainnya untuk membantu Anda membangun aplikasi seluler dan web yang lebih tangguh dan andal dengan lebih sedikit pekerjaan.

Harga untuk AWS AppSync

AWS AppSyncdibanderol berdasarkan jutaan permintaan dan pembaruan. Caching dikenakan biaya tambahan. Untuk informasi selengkapnya, lihat Harga AWS AppSync.

Berikut ini mencantumkan pengecualian untuk AWS AppSync harga umum:

- Caching API AWS AppSync tidak memenuhi syarat untuk Tingkat AWS Gratis.
- Permintaan tidak dikenakan biaya untuk kegagalan otorisasi dan otentikasi.
- Panggilan ke metode yang memerlukan kunci API tidak dikenakan biaya saat kunci API hilang atau tidak valid.

GraphQL dan arsitektur AWS AppSync

> **Note**
>
> Panduan ini mengasumsikan pengguna memiliki pengetahuan tentang gaya arsitektur REST. Kami merekomendasikan untuk meninjau ini dan topik front-end lainnya sebelum bekerja dengan GraphQL dan. AWS AppSync

GraphQL adalah bahasa query dan manipulasi untuk API. GraphQL menyediakan sintaks yang fleksibel dan intuitif untuk menggambarkan kebutuhan dan interaksi data. Ini memungkinkan pengembang untuk meminta apa yang dibutuhkan dan mendapatkan kembali hasil yang dapat diprediksi. Ini juga memungkinkan untuk mengakses banyak sumber dalam satu permintaan, mengurangi jumlah panggilan jaringan dan persyaratan bandwidth, sehingga menghemat masa pakai baterai dan siklus CPU yang dikonsumsi oleh aplikasi.

Membuat pembaruan data dibuat sederhana dengan mutasi, memungkinkan pengembang untuk menjelaskan bagaimana data harus berubah. GraphQL juga memfasilitasi pengaturan cepat solusi real-time melalui langganan. Semua fitur ini digabungkan, ditambah dengan alat pengembang yang kuat, menjadikan GraphQL penting untuk mengelola data aplikasi.

GraphQL adalah alternatif untuk REST. Arsitektur RESTful saat ini merupakan salah satu solusi yang lebih populer untuk komunikasi client-server. Ini berpusat pada konsep sumber daya Anda (data) yang diekspos oleh URL. URL ini dapat digunakan untuk mengakses dan memanipulasi data melalui operasi CRUD (membuat, membaca, memperbarui, menghapus) dalam bentuk metode HTTP sepertiGET,, POST dan. DELETE Keuntungan REST adalah relatif sederhana untuk dipelajari dan diterapkan. Anda dapat dengan cepat mengatur RESTful API untuk memanggil berbagai layanan.

Namun, teknologi semakin rumit. Ketika aplikasi, alat, dan layanan mulai berskala untuk audiens di seluruh dunia, kebutuhan akan arsitektur yang cepat dan terukur sangat penting. REST memiliki banyak kekurangan ketika berhadapan dengan operasi yang dapat diskalakan. Lihat kasus penggunaan ini sebagai contoh.

Pada bagian berikut, kita akan meninjau beberapa konsep seputar RESTful API. Kami kemudian akan memperkenalkan GraphQL dan cara kerjanya.

Untuk informasi selengkapnya tentang GraphQL dan manfaat migrasi ke, lihat panduan Keputusan AWS untuk implementasi GraphQL.

Topik

Apa itu API?

Antarmuka pemrograman aplikasi (API) mendefinisikan aturan yang harus Anda ikuti untuk berkomunikasi dengan sistem perangkat lunak lain. Pengembang mengekspos atau membuat API sehingga aplikasi lain dapat berkomunikasi dengan aplikasi mereka secara terprogram. Misalnya, aplikasi lembar waktu mengekspos API yang meminta nama lengkap karyawan dan rentang tanggal. Ketika menerima informasi ini, secara internal memproses lembar waktu karyawan dan mengembalikan jumlah jam kerja dalam rentang tanggal tersebut.

Anda dapat menganggap API web sebagai gateway antara klien dan sumber daya di web.

Klien

Klien adalah pengguna yang ingin mengakses informasi dari web. Klien dapat berupa orang atau sistem perangkat lunak yang menggunakan API. Misalnya, pengembang dapat menulis program yang mengakses data cuaca dari sistem cuaca. Atau Anda dapat mengakses data yang sama dari browser Anda ketika Anda mengunjungi situs web cuaca secara langsung.

Sumber daya

Sumber daya adalah informasi yang diberikan oleh berbagai aplikasi kepada klien mereka. Sumber daya dapat berupa gambar, video, teks, angka, atau jenis data apa pun. Mesin yang memberikan sumber daya ke klien juga disebut server. Organizations menggunakan API untuk berbagi sumber daya dan menyediakan layanan web sambil menjaga keamanan, kontrol, dan otentikasi. Selain itu, API membantu mereka menentukan klien mana yang mendapatkan akses ke sumber daya internal tertentu.

Apa itu REST?

Pada tingkat tinggi, representational State Transfer (REST) adalah arsitektur perangkat lunak yang memaksakan kondisi pada bagaimana API harus bekerja. REST awalnya dibuat sebagai pedoman untuk mengelola komunikasi pada jaringan yang kompleks seperti internet. Anda dapat menggunakan arsitektur berbasis REST untuk mendukung komunikasi berkinerja tinggi dan andal dalam skala besar. Anda dapat dengan mudah menerapkan dan memodifikasinya, membawa visibilitas dan portabilitas lintas platform ke sistem API apa pun.

Pengembang API dapat mendesain API menggunakan beberapa arsitektur yang berbeda. API yang mengikuti gaya arsitektur REST disebut REST API. Layanan web yang menerapkan arsitektur REST disebut layanan web RESTful. Istilah RESTful API umumnya mengacu pada API web RESTful. Namun, Anda dapat menggunakan istilah REST API dan RESTful API secara bergantian.

Berikut ini adalah beberapa prinsip gaya arsitektur REST:

Antarmuka seragam

Antarmuka yang seragam sangat penting untuk desain layanan web RESTful apa pun. Ini menunjukkan bahwa server mentransfer informasi dalam format standar. Sumber daya yang diformat disebut representasi dalam REST. Format ini dapat berbeda dari representasi internal sumber daya pada aplikasi server. Misalnya, server dapat menyimpan data sebagai teks tetapi mengirimkannya dalam format representasi HTML.

Antarmuka seragam memaksakan empat kendala arsitektur:

1. Permintaan harus mengidentifikasi sumber daya. Mereka melakukannya dengan menggunakan pengenal sumber daya seragam.
2. Klien memiliki informasi yang cukup dalam representasi sumber daya untuk memodifikasi atau menghapus sumber daya jika mereka mau. Server memenuhi kondisi ini dengan mengirimkan metadata yang menjelaskan sumber daya lebih lanjut.
3. Klien menerima informasi tentang cara memproses representasi lebih lanjut. Server mencapai ini dengan mengirimkan pesan deskriptif diri yang berisi metadata tentang bagaimana klien dapat menggunakannya dengan sebaik-baiknya.
4. Klien menerima informasi tentang semua sumber daya terkait lainnya yang mereka butuhkan untuk menyelesaikan tugas. Server mencapai ini dengan mengirimkan hyperlink dalam representasi sehingga klien dapat secara dinamis menemukan lebih banyak sumber daya.

Tanpa kewarganegaraan

Dalam arsitektur REST, statelessness mengacu pada metode komunikasi di mana server menyelesaikan setiap permintaan klien secara independen dari semua permintaan sebelumnya. Klien dapat meminta sumber daya dalam urutan apa pun, dan setiap permintaan tidak memiliki kewarganegaraan atau terisolasi dari permintaan lain. Kendala desain REST API ini menyiratkan bahwa server dapat sepenuhnya memahami dan memenuhi permintaan setiap saat.

Sistem berlapis

Dalam arsitektur sistem berlapis, klien dapat terhubung ke perantara resmi lainnya antara klien dan server, dan masih akan menerima tanggapan dari server. Server juga dapat meneruskan permintaan ke server lain. Anda dapat merancang layanan web RESTful Anda untuk berjalan di beberapa server dengan beberapa lapisan seperti keamanan, aplikasi, dan logika bisnis, bekerja sama untuk memenuhi permintaan klien. Lapisan-lapisan ini tetap tidak terlihat oleh klien.

Cacheability

Layanan web RESTful mendukung caching, yang merupakan proses menyimpan beberapa tanggapan pada klien atau pada perantara untuk meningkatkan waktu respons server. Misalnya, Anda mengunjungi situs web yang memiliki gambar header dan footer umum di setiap halaman. Setiap kali Anda mengunjungi halaman situs web baru, server harus mengirim ulang gambar yang sama. Untuk menghindari hal ini, klien menyimpan atau menyimpan gambar-gambar ini setelah respons pertama dan kemudian menggunakan gambar langsung dari cache. Layanan web RESTful mengontrol caching dengan menggunakan respons API yang mendefinisikan diri mereka sebagai cacheable atau noncacheable.

Apa itu RESTful API?

RESTful API adalah antarmuka yang digunakan dua sistem komputer untuk bertukar informasi dengan aman melalui internet. Sebagian besar aplikasi bisnis harus berkomunikasi dengan aplikasi internal dan pihak ketiga lainnya untuk melakukan berbagai tugas. Misalnya, untuk menghasilkan slip gaji bulanan, sistem akun internal Anda harus berbagi data dengan sistem perbankan pelanggan Anda untuk mengotomatiskan faktur dan berkomunikasi dengan aplikasi lembar waktu internal. RESTful API mendukung pertukaran informasi ini karena mereka mengikuti standar komunikasi perangkat lunak yang aman, andal, dan efisien.

Bagaimana cara kerja RESTful API?

Fungsi dasar RESTful API sama dengan browsing internet. Klien menghubungi server dengan menggunakan API saat membutuhkan sumber daya. Pengembang API menjelaskan bagaimana klien harus menggunakan REST API dalam dokumentasi API aplikasi server. Ini adalah langkah-langkah umum untuk setiap panggilan REST API:

1. Klien mengirimkan permintaan ke server. Klien mengikuti dokumentasi API untuk memformat permintaan dengan cara yang dipahami server.
2. Server mengotentikasi klien dan mengonfirmasi bahwa klien memiliki hak untuk membuat permintaan itu.
3. Server menerima permintaan dan memprosesnya secara internal.
4. Server mengembalikan respons ke klien. Respons berisi informasi yang memberi tahu klien apakah permintaan berhasil. Tanggapan juga mencakup informasi apa pun yang diminta klien.

Permintaan REST API dan detail respons sedikit berbeda tergantung pada bagaimana pengembang API mendesain API.

Mengapa Menggunakan GraphQL di atas REST?

REST adalah salah satu gaya arsitektur landasan API web. Namun, ketika dunia menjadi lebih saling berhubungan, kebutuhan untuk mengembangkan aplikasi yang kuat dan terukur akan menjadi masalah yang lebih mendesak. Sementara REST saat ini merupakan standar industri untuk membangun API web, ada beberapa kelemahan berulang untuk implementasi RESTful yang telah diidentifikasi:

1. Permintaan data: Menggunakan RESTful API, Anda biasanya akan meminta data yang Anda butuhkan melalui titik akhir. Masalah muncul ketika Anda memiliki data yang mungkin tidak begitu rapi dikemas. Data yang Anda butuhkan mungkin berada di belakang beberapa lapisan abstraksi, dan satu-satunya cara untuk mengambil data adalah dengan menggunakan beberapa titik akhir, yang berarti membuat beberapa permintaan untuk mengekstrak semua data.
2. Overfetching dan underfetching: Untuk menambah masalah beberapa permintaan, data dari setiap titik akhir didefinisikan secara ketat, artinya Anda akan mengembalikan data apa pun yang ditentukan untuk API itu, bahkan jika Anda tidak menginginkannya secara teknis.

   Hal ini dapat mengakibatkan pengambilan berlebihan, yang berarti permintaan kami mengembalikan data yang berlebihan. Misalnya, katakanlah Anda meminta data personel

perusahaan dan ingin mengetahui nama-nama karyawan di departemen tertentu. Titik akhir yang mengembalikan data akan berisi nama, tetapi mungkin juga berisi data lain seperti jabatan atau tanggal lahir. Karena API sudah diperbaiki, Anda tidak bisa hanya meminta nama saja; sisa data menyertainya.

Situasi sebaliknya di mana kita tidak mengembalikan cukup data disebut under-fetching. Untuk mendapatkan semua data yang diminta, Anda mungkin harus membuat beberapa permintaan ke layanan. Bergantung pada bagaimana data disusun, Anda dapat mengalami kueri yang tidak efisien yang mengakibatkan masalah seperti masalah n+1 yang ditakuti.

3. Iterasi pengembangan lambat: Banyak pengembang menyesuaikan API RESTful mereka agar sesuai dengan aliran aplikasi mereka. Namun, seiring berkembangnya aplikasi mereka, baik bagian depan maupun backend mungkin memerlukan perubahan ekstensif. Akibatnya, API mungkin tidak lagi sesuai dengan bentuk data dengan cara yang efisien atau berdampak. Ini menghasilkan iterasi produk yang lebih lambat karena kebutuhan akan modifikasi API.
4. Kinerja dalam skala: Karena masalah peracikan ini, ada banyak area di mana skalabilitas akan terpengaruh. Kinerja di sisi aplikasi mungkin terpengaruh karena permintaan Anda akan mengembalikan terlalu banyak data atau terlalu sedikit (menghasilkan lebih banyak permintaan). Kedua situasi menyebabkan ketegangan yang tidak perlu pada jaringan yang mengakibatkan kinerja yang buruk. Di sisi pengembang, kecepatan pengembangan dapat dikurangi karena API Anda sudah diperbaiki dan tidak lagi sesuai dengan data yang mereka minta.

Nilai jual GraphQL adalah untuk mengatasi kekurangan REST. Berikut adalah beberapa solusi utama yang ditawarkan GraphQL kepada pengembang:

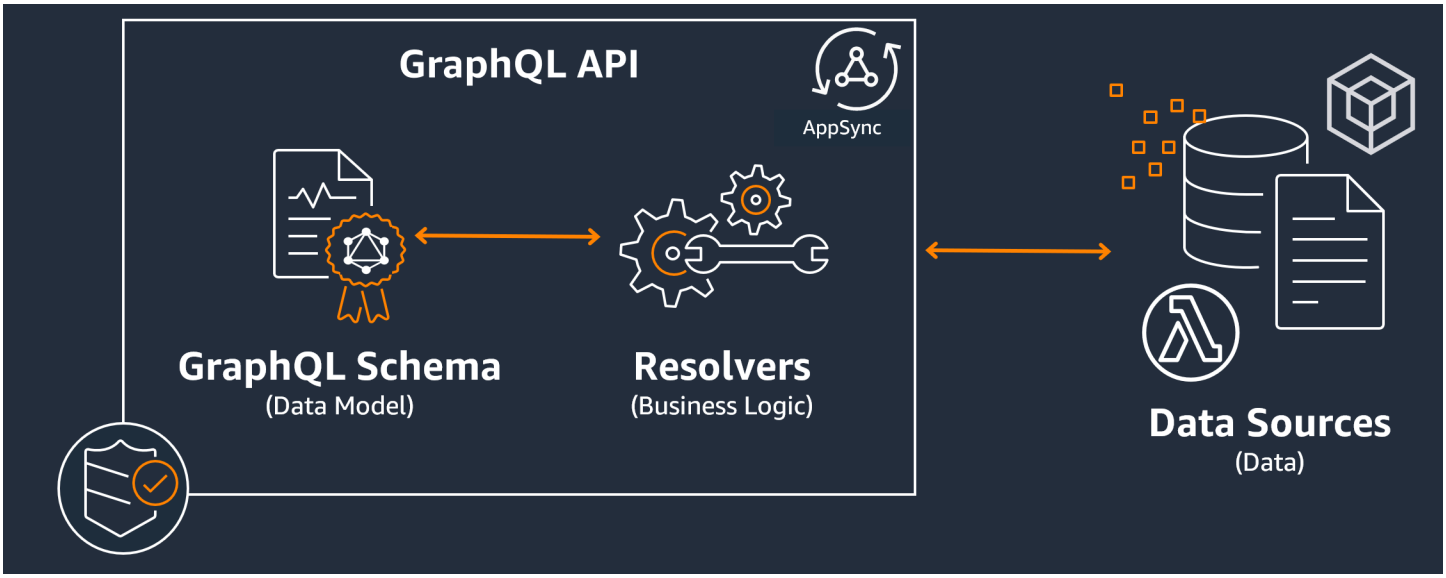
1. Titik akhir tunggal: GraphQL menggunakan titik akhir tunggal untuk menanyakan data. Tidak perlu membangun beberapa API agar sesuai dengan bentuk data. Ini menghasilkan lebih sedikit permintaan melalui jaringan.
2. Pengambilan: GraphQL memecahkan masalah abadi dari pengambilan berlebihan dan kurang hanya dengan mendefinisikan data yang Anda butuhkan. GraphQL memungkinkan Anda membentuk data agar sesuai dengan kebutuhan Anda sehingga Anda hanya menerima apa yang Anda minta.
3. Abstraksi: GraphQL API berisi beberapa komponen dan sistem yang menggambarkan data menggunakan standar agnostik bahasa. Dengan kata lain, bentuk dan struktur data distandarisasi sehingga bagian depan dan backend tahu bagaimana data akan dikirim melalui jaringan. Hal ini memungkinkan pengembang di kedua ujungnya untuk bekerja dengan sistem GraphQL dan tidak di sekitar mereka.

4. Iterasi cepat: Karena standarisasi data, perubahan pada satu ujung pengembangan mungkin tidak diperlukan di sisi lain. Misalnya, perubahan presentasi frontend mungkin tidak menghasilkan perubahan backend yang ekstensif karena GraphQL memungkinkan spesifikasi data untuk dimodifikasi dengan mudah. Anda cukup menentukan atau memodifikasi bentuk data agar sesuai dengan kebutuhan aplikasi saat tumbuh. Ini menghasilkan pekerjaan pengembangan yang kurang potensial.

Ini hanya beberapa manfaat dari GraphQL. Di beberapa bagian berikutnya, Anda akan mempelajari bagaimana GraphQL terstruktur dan properti yang menjadikannya alternatif unik untuk REST.

Komponen GraphQL API

GraphQL API standar terdiri dari skema tunggal yang menangani bentuk data yang akan ditanyakan. Skema Anda ditautkan ke satu atau beberapa sumber data Anda seperti database atau fungsi Lambda. Di antara keduanya duduk satu atau lebih resolver yang menangani logika bisnis untuk permintaan Anda. Setiap komponen memainkan peran penting dalam implementasi GraphQL Anda. Bagian berikut akan memperkenalkan ketiga komponen ini dan peran yang mereka mainkan dalam layanan GraphQL.



Topik

- Skema
- Sumber data
- Penyelesai

Skema

Skema GraphQL adalah dasar dari GraphQL API. Ini berfungsi sebagai cetak biru yang mendefinisikan bentuk data Anda. Ini juga merupakan kontrak antara klien dan server Anda yang menentukan bagaimana data Anda akan diambil dan/atau dimodifikasi.

Skema GraphQL ditulis dalam Schema Definition Language (SDL). SDL terdiri dari jenis dan bidang dengan struktur yang mapan:

- Jenis: Jenis adalah bagaimana GraphQL mendefinisikan bentuk dan perilaku data. GraphQL mendukung banyak jenis yang akan dijelaskan nanti di bagian ini. Setiap jenis yang didefinisikan dalam skema Anda akan berisi cakupannya sendiri. Di dalam ruang lingkup akan ada satu atau lebih bidang yang dapat berisi nilai atau logika yang akan digunakan dalam layanan GraphQL Anda. Jenis mengisi banyak peran yang berbeda, yang paling umum adalah objek atau skalar (tipe nilai primitif).
- Bidang: Bidang ada dalam lingkup tipe dan menyimpan nilai yang diminta dari layanan GraphQL. Ini sangat mirip dengan variabel dalam bahasa pemrograman lain. Bentuk data yang Anda tentukan di bidang Anda akan menentukan bagaimana data terstruktur dalam operasi permintaan/respons. Hal ini memungkinkan pengembang untuk memprediksi apa yang akan dikembalikan tanpa mengetahui bagaimana backend layanan diimplementasikan.

Untuk memvisualisasikan seperti apa skema, mari kita tinjau isi skema GraphQL sederhana. Dalam kode produksi, skema Anda biasanya akan berada dalam file bernama `schema.graphql` atau`schema.json`. Mari kita asumsikan bahwa kita mengintip ke dalam proyek yang mengimplementasikan layanan GraphQL. Proyek ini menyimpan data personel perusahaan, dan `schema.graphql` file tersebut digunakan untuk mengambil data personel dan menambahkan personel baru ke database. Kode mungkin terlihat seperti ini:

schema.graphql

```
type Person {
   id: ID!
   name: String
   age: Int
}
type Query {
  people: [Person]
}
type Mutation {
```

```
  addPerson(id: ID!, name: String, age: Int): Person
}
```

Kita dapat melihat bahwa ada tiga jenis yang didefinisikan dalam skema:`Person`,`Query`, dan`Mutation`. Melihat`Person`, kita dapat menebak bahwa ini adalah cetak biru untuk contoh karyawan perusahaan, yang akan membuat tipe ini menjadi objek. Di dalam ruang lingkupnya, kita lihat`id`,`name`, dan`age`. Ini adalah bidang yang menentukan properti dari a`Person`. Ini berarti sumber data kami menyimpan masing-masing `Person` `name` sebagai tipe `String` skalar (primitif) dan `age` sebagai tipe `Int` skalar (primitif). `id`Bertindak sebagai pengidentifikasi khusus dan unik untuk masing-masing`Person`. Ini juga merupakan nilai yang diperlukan seperti yang dilambangkan dengan simbol. `!`

Dua tipe objek berikutnya berperilaku berbeda. GraphQL menyimpan beberapa kata kunci untuk jenis objek khusus yang menentukan bagaimana data akan diisi dalam skema. `Query`Tipe akan mengambil data dari sumbernya. Dalam contoh kita, query kita mungkin mengambil `Person` objek dari database. Ini mungkin mengingatkan Anda tentang `GET` operasi dalam terminologi RESTful. A `Mutation` akan memodifikasi data. Dalam contoh kita, mutasi kita dapat menambahkan lebih banyak `Person` objek ke database. Ini mungkin mengingatkan Anda tentang operasi yang mengubah negara seperti `PUT` atau. `POST` Perilaku semua jenis objek khusus akan dijelaskan nanti di bagian ini.

Mari kita asumsikan `Query` dalam contoh kita akan mengambil sesuatu dari database. Jika kita melihat bidang`Query`, kita melihat satu bidang disebut`people`. Nilai bidangnya adalah`[Person]`. Ini berarti kita ingin mengambil beberapa contoh dari `Person` dalam database. Namun, penambahan tanda kurung berarti kita ingin mengembalikan daftar semua `Person` instance dan bukan hanya yang spesifik.

`Mutation`Tipe ini bertanggung jawab untuk melakukan operasi perubahan status seperti modifikasi data. Mutasi bertanggung jawab untuk melakukan beberapa operasi perubahan keadaan pada sumber data. Dalam contoh kita, mutasi kita berisi operasi yang disebut `addPerson` yang menambahkan `Person` objek baru ke database. Mutasi menggunakan a `Person` dan mengharapkan input untuk`id`,`name`, dan `age` bidang.

Pada titik ini, Anda mungkin bertanya-tanya bagaimana operasi seperti `addPerson` bekerja tanpa implementasi kode mengingat bahwa itu seharusnya melakukan beberapa perilaku dan terlihat sangat mirip fungsi dengan nama fungsi dan parameter. Saat ini, itu tidak akan berfungsi karena skema hanya berfungsi sebagai deklarasi. Untuk mengimplementasikan perilaku`addPerson`, kita harus menambahkan resolver ke dalamnya. Resolver adalah unit kode yang dieksekusi setiap kali bidang terkait (dalam hal ini, `addPerson` operasi) dipanggil. Jika Anda ingin menggunakan

operasi, Anda harus menambahkan implementasi resolver di beberapa titik. Di satu sisi, Anda dapat menganggap operasi skema sebagai deklarasi fungsi dan resolver sebagai definisi. Resolver akan dijelaskan di bagian yang berbeda.

Contoh ini hanya menunjukkan cara paling sederhana skema dapat memanipulasi data. Anda membangun aplikasi yang kompleks, kuat, dan dapat diskalakan dengan memanfaatkan fitur GraphQL dan. AWS AppSync Di bagian selanjutnya, kita akan mendefinisikan semua jenis dan perilaku lapangan yang berbeda yang dapat Anda manfaatkan dalam skema Anda.

Jenis GraphQL

GraphQL mendukung berbagai jenis. Seperti yang Anda lihat di bagian sebelumnya, tipe menentukan bentuk atau perilaku data Anda. Mereka adalah blok bangunan mendasar dari skema GraphQL.

Jenis dapat dikategorikan ke dalam input dan output. Input adalah tipe yang diizinkan untuk diteruskan sebagai argumen untuk tipe objek khusus (`Query`,, dll.)`Mutation`, sedangkan tipe output secara ketat digunakan untuk menyimpan dan mengembalikan data. Daftar jenis dan kategorisasi mereka tercantum di bawah ini:

- Objek: Objek berisi bidang yang menggambarkan entitas. Misalnya, objek bisa berupa sesuatu seperti a `book` dengan bidang yang menggambarkan karakteristiknya sepertiauthorName,`publishingYear`, dll. Mereka adalah tipe keluaran yang ketat.
- Skalar: Ini adalah tipe primitif seperti int, string, dll. Mereka biasanya ditugaskan ke bidang. Menggunakan `authorName` bidang sebagai contoh, dapat diberikan `String` skalar untuk menyimpan nama seperti “John Smith”. Skalar dapat berupa tipe input dan output.
- Input: Input memungkinkan Anda untuk melewati sekelompok bidang sebagai argumen. Mereka terstruktur sangat mirip dengan objek, tetapi mereka dapat diteruskan sebagai argumen ke objek khusus. Input memungkinkan Anda untuk menentukan skalar, enum, dan input lain dalam cakupannya. Input hanya bisa berupa tipe input.
- Objek khusus: Objek khusus melakukan operasi yang mengubah keadaan dan melakukan sebagian besar pengangkatan berat layanan. Ada tiga jenis objek khusus: kueri, mutasi, dan langganan. Kueri biasanya mengambil data; mutasi memanipulasi data; langganan membuka dan memelihara koneksi dua arah antara klien dan server untuk komunikasi konstan. Objek khusus bukanlah input atau output mengingat fungsinya.
- Enum: Enum adalah daftar nilai hukum yang telah ditentukan sebelumnya. Jika Anda memanggil enum, nilainya hanya dapat ditentukan dalam cakupannya. Misalnya, jika Anda memiliki enum yang disebut `trafficLights` menggambarkan daftar sinyal lalu lintas, itu bisa

memiliki nilai seperti `redLight` dan `greenLight` tetapi tidak. `purpleLight` Lampu lalu lintas nyata hanya akan memiliki begitu banyak sinyal, sehingga Anda dapat menggunakan enum untuk mendefinisikannya dan memaksanya menjadi satu-satunya nilai hukum saat mereferensikan`trafficLight`. Enum dapat berupa tipe input dan output.

- Serikat/Antarmuka: Serikat memungkinkan Anda mengembalikan satu atau lebih hal dalam permintaan tergantung pada data yang diminta oleh klien. Misalnya, jika Anda memiliki `Book` tipe dengan `title` bidang dan `Author` tipe dengan `name` bidang, Anda dapat membuat gabungan antara kedua jenis tersebut. Jika klien Anda ingin menanyakan database untuk frasa “Julius Caesar”, serikat pekerja dapat mengembalikan Julius Caesar (drama oleh William Shakespeare) dari *`Book title`* dan Julius Caesar (penulis Commentarii de Bello Gallico) dari. `Author name` Serikat pekerja hanya bisa menjadi tipe keluaran.

  Antarmuka adalah kumpulan bidang yang harus diimplementasikan objek. Ini sedikit mirip dengan antarmuka dalam bahasa pemrograman seperti Java di mana Anda harus mengimplementasikan bidang yang ditentukan dalam antarmuka. Misalnya, katakanlah Anda membuat antarmuka yang disebut `Book` yang berisi `title` bidang. Katakanlah Anda kemudian membuat tipe yang disebut `Novel` yang diimplementasikan`Book`. Anda `Novel` harus memasukkan `title` bidang. Namun, Anda juga `Novel` bisa menyertakan bidang lain yang tidak ada di antarmuka seperti `pageCountISBN`. Antarmuka hanya dapat berupa tipe output.

Bagian berikut akan menjelaskan bagaimana setiap jenis bekerja di GraphQL.

Objek

Objek GraphQL adalah tipe utama yang akan Anda lihat dalam kode produksi. Dalam GraphQL, Anda dapat menganggap objek sebagai pengelompokan bidang yang berbeda (mirip dengan variabel dalam bahasa lain), dengan setiap bidang ditentukan oleh tipe (biasanya skalar atau objek lain) yang dapat menyimpan nilai. Objek mewakili unit data yang dapat diambil/dimanipulasi dari implementasi layanan Anda.

Jenis objek dideklarasikan menggunakan `Type` kata kunci. Mari kita memodifikasi contoh skema kita sedikit:

```
type Person {
  id: ID!
  name: String
  age: Int
  occupation: Occupation
}
```

```
type Occupation {
  title: String
}
```

Jenis objek di sini adalah `Person` dan`Occupation`. Setiap objek memiliki bidangnya sendiri dengan tipenya sendiri. Salah satu fitur GraphQL adalah kemampuan untuk mengatur bidang ke jenis lain. Anda dapat melihat `occupation` bidang `Person` berisi jenis `Occupation` objek. Kita dapat membuat asosiasi ini karena GraphQL hanya mendeskripsikan data dan bukan implementasi layanan.

Skalar

Skalar pada dasarnya adalah tipe primitif yang menyimpan nilai. DalamAWS AppSync, ada dua jenis skalar: skalar GraphQL default dan skalar. AWS AppSync Skalar biasanya digunakan untuk menyimpan nilai bidang dalam tipe objek. Jenis GraphQL default `Int` meliputi`Float`,,,`String`, `Boolean` dan. `ID` Mari kita gunakan contoh sebelumnya lagi:

```
type Person {
  id: ID!
  name: String
  age: Int
  occupation: Occupation
}

type Occupation {
  title: String
}
```

Memilih `title` bidang `name` dan, keduanya memegang `String` skalar. `Name`dapat mengembalikan nilai string seperti "`John  Smith`" dan judulnya dapat mengembalikan sesuatu seperti "`firefighter`”. Beberapa implementasi GraphQL juga mendukung skalar kustom menggunakan `Scalar` kata kunci dan menerapkan perilaku tipe. Namun, AWS AppSync saat ini tidak mendukung skalar khusus. Untuk daftar skalar, lihat Jenis skalar di. AWS AppSync

Masukan

Karena konsep tipe input dan output, ada batasan tertentu saat meneruskan argumen. Jenis yang biasanya perlu diteruskan, terutama objek, dibatasi. Anda dapat menggunakan tipe input untuk melewati aturan ini. Input adalah jenis yang berisi skalar, enum, dan jenis input lainnya.

Input didefinisikan menggunakan `input` kata kunci:

```
type Person {
  id: ID!
  name: String
  age: Int
  occupation: Occupation
}

type Occupation {
  title: String
}

input personInput {
  id: ID!
  name: String
  age: Int
  occupation: occupationInput
}

input occupationInput {
  title: String
}
```

Seperti yang Anda lihat, kita dapat memiliki input terpisah yang meniru tipe aslinya. Masukan ini akan sering digunakan dalam operasi lapangan Anda seperti ini:

```
type Person {
  id: ID!
  name: String
  age: Int
  occupation: Occupation
}

type Occupation {
  title: String
}

input occupationInput {
  title: String
}

type Mutation {
```

```
  addPerson(id: ID!, name: String, age: Int, occupation: occupationInput): Person
}
```

Perhatikan bagaimana kita masih meneruskan `occupationInput` di tempat `Occupation` untuk membuat`Person`.

Ini hanyalah satu skenario untuk input. Mereka tidak perlu menyalin objek 1:1, dan dalam kode produksi, kemungkinan besar Anda tidak akan menggunakannya seperti ini. Ini adalah praktik yang baik untuk memanfaatkan skema GraphQL dengan mendefinisikan hanya apa yang perlu Anda masukan sebagai argumen.

Juga, input yang sama dapat digunakan dalam beberapa operasi, tetapi kami tidak menyarankan melakukan ini. Setiap operasi idealnya harus berisi salinan input uniknya sendiri jika persyaratan skema berubah.

Benda khusus

GraphQL menyimpan beberapa kata kunci untuk objek khusus yang mendefinisikan beberapa logika bisnis untuk bagaimana skema Anda akan mengambil/memanipulasi data. Paling-paling, bisa ada salah satu dari masing-masing kata kunci ini dalam skema. Mereka bertindak sebagai titik masuk untuk semua data yang diminta yang dijalankan klien Anda terhadap layanan GraphQL Anda.

Objek khusus juga didefinisikan menggunakan `type` kata kunci. Meskipun mereka digunakan secara berbeda dari tipe objek biasa, implementasinya sangat mirip.

Queries

Kueri sangat mirip dengan `GET` operasi karena mereka melakukan pengambilan hanya-baca untuk mendapatkan data dari sumber Anda. Dalam GraphQL, `Query` mendefinisikan semua titik masuk untuk klien yang membuat permintaan terhadap server Anda. Akan selalu ada `Query` dalam implementasi GraphQL Anda.

Berikut adalah jenis objek `Query` dan modifikasi yang kami gunakan dalam contoh skema kami sebelumnya:

```
type Person {
  id: ID!
  name: String
  age: Int
  occupation: Occupation
}
```

```
type Occupation {
  title: String
}
type Query {
  people: [Person]
}
```

Kami `Query` berisi bidang `people` yang disebut yang mengembalikan daftar `Person` contoh dari sumber data. Katakanlah kita perlu mengubah perilaku aplikasi kita, dan sekarang kita perlu mengembalikan daftar hanya `Occupation` instance untuk beberapa tujuan terpisah. Kami cukup menambahkannya ke kueri:

```
type Query {
  people: [Person]
  occupations: [Occupation]
}
```

Di GraphQL, kami dapat memperlakukan kueri kami sebagai sumber permintaan tunggal. Seperti yang Anda lihat, ini berpotensi jauh lebih sederhana daripada implementasi RESTful yang mungkin menggunakan titik akhir yang berbeda untuk mencapai hal yang sama (`.../api/1/people`dan). `.../api/1/occupations`

Dengan asumsi kita memiliki implementasi resolver untuk query ini, kita sekarang dapat melakukan query yang sebenarnya. Sementara `Query` jenisnya ada, kita harus secara eksplisit memanggilnya agar dapat dijalankan dalam kode aplikasi. Ini dapat dilakukan dengan menggunakan `query` kata kunci:

```
query getItems {
   people {
      name
   }
   occupations {
      title
   }
}
```

Seperti yang Anda lihat, kueri ini dipanggil `getItems` dan dikembalikan `people` (daftar `Person` objek) dan `occupations` (daftar `Occupation` objek). Di`people`, kami hanya mengembalikan `name` bidang masing-masing`Person`, sementara kami mengembalikan `title` bidang masing-masing`Occupation`. Responsnya mungkin terlihat seperti ini:

```
{
  "data": {
    "people": [
      {
        "name": "John Smith"
      },
      {
        "name": "Andrew Miller"
      },
      .
      .
      .
    ],
    "occupations": [
      {
        "title": "Firefighter"
      },
      {
        "title": "Bookkeeper"
      },
      .
      .
      .
    ]
  }
}
```

Contoh respon menunjukkan bagaimana data mengikuti bentuk query. Setiap entri yang diambil tercantum dalam lingkup bidang. `people`dan mengembalikan `occupations` hal-hal sebagai daftar terpisah. Meskipun berguna, mungkin lebih mudah untuk memodifikasi kueri untuk mengembalikan daftar nama dan pekerjaan orang:

```
query getItems {
   people {
      name
      occupation {
        title
      }
}
```

Ini adalah modifikasi hukum karena `Person` tipe kami berisi `occupation` bidang tipe`Occupation`. Ketika terdaftar dalam lingkup`people`, kami mengembalikan masing-masing

`Person` name bersama dengan yang terkait dengannya `Occupationtitle`. Responsnya mungkin terlihat seperti ini:

```
}
  "data": {
    "people": [
      {
        "name": "John Smith",
        "occupation": {
          "title": "Firefighter"
        }
      },
      {
        "name": "Andrew Miller",
        "occupation": {
          "title": "Bookkeeper"
        }
      },
      .
      .
      .
    ]
  }
}
```

Mutations

Mutasi mirip dengan operasi yang mengubah keadaan seperti atau. `PUT` `POST` Mereka melakukan operasi tulis untuk memodifikasi data di sumber, lalu mengambil responsnya. Mereka menentukan titik masuk Anda untuk permintaan modifikasi data. Tidak seperti kueri, mutasi mungkin atau mungkin tidak termasuk dalam skema tergantung pada kebutuhan proyek. Berikut mutasi dari contoh skema:

```
type Mutation {
  addPerson(id: ID!, name: String, age: Int): Person
}
```

`addPerson`Bidang mewakili satu titik masuk yang menambahkan a `Person` ke sumber data. `addPerson`adalah nama bidang;`id`,`name`, dan `age` merupakan parameter; dan `Person` merupakan tipe pengembalian. Melihat kembali `Person` tipenya:

```
type Person {
  id: ID!
  name: String
  age: Int
  occupation: Occupation
}
```

Kami menambahkan `occupation` bidang. Namun, kami tidak dapat mengatur bidang ini secara `Occupation` langsung karena objek tidak dapat diteruskan sebagai argumen; mereka benar-benar tipe keluaran. Sebagai gantinya, kita harus meneruskan input dengan bidang yang sama sebagai argumen:

```
input occupationInput {
  title: String
}
```

Kami juga dapat dengan mudah memperbarui kami `addPerson` untuk memasukkan ini sebagai parameter saat membuat `Person` instance baru:

```
type Mutation {
  addPerson(id: ID!, name: String, age: Int, occupation: occupationInput): Person
}
```

Berikut skema yang diperbarui:

```
type Person {
  id: ID!
  name: String
  age: Int
  occupation: Occupation
}

type Occupation {
  title: String
}

input occupationInput {
  title: String
}

type Mutation {
```

```
  addPerson(id: ID!, name: String, age: Int, occupation: occupationInput): Person
}
```

Perhatikan bahwa `occupation` akan lulus di `title` bidang dari `occupationInput` untuk menyelesaikan pembuatan `Person` bukan `Occupation` objek asli. Dengan asumsi kita memiliki implementasi resolver untuk`addPerson`, kita sekarang dapat melakukan mutasi yang sebenarnya. Sementara `Mutation` jenisnya ada, kita harus secara eksplisit memanggilnya agar dapat dijalankan dalam kode aplikasi. Ini dapat dilakukan dengan menggunakan `mutation` kata kunci:

```
mutation createPerson {
  addPerson(id: ID!, name: String, age: Int, occupation: occupationInput) {
    name
    age
    occupation {
      title
    }
  }
}
```

Mutasi ini disebut`createPerson`, dan `addPerson` merupakan operasi. Untuk membuat yang baru`Person`, kita bisa memasukkan argumen untuk`id`,`name`,`age`, dan`occupation`. Dalam lingkup`addPerson`, kita juga dapat melihat bidang lain seperti`name`,`age`, dll. Ini adalah tanggapan Anda; ini adalah bidang yang akan dikembalikan setelah `addPerson` operasi selesai. Berikut adalah bagian akhir dari contoh:

```
mutation createPerson {
  addPerson(id: "1", name: "Steve Powers", age: "50", occupation: "Miner") {
    id
    name
    age
    occupation {
      title
    }
  }
}
```

Dengan menggunakan mutasi ini, hasilnya mungkin terlihat seperti ini:

```
{
```

```
  "data": {
    "addPerson": {
      "id": "1",
      "name": "Steve Powers",
      "age": "50",
      "occupation": {
        "title": "Miner"
      }
    }
  }
}
```

Seperti yang Anda lihat, respons mengembalikan nilai yang kami minta dalam format yang sama yang ditentukan dalam mutasi kami. Merupakan praktik yang baik untuk mengembalikan semua nilai yang telah dimodifikasi untuk mengurangi kebingungan dan kebutuhan akan lebih banyak kueri di masa mendatang. Mutasi memungkinkan Anda untuk memasukkan beberapa operasi dalam cakupannya. Mereka akan dijalankan secara berurutan dalam urutan yang tercantum dalam mutasi. Misalnya, jika kita membuat operasi lain `add0ccupation` yang disebut yang menambahkan judul pekerjaan ke sumber data, kita dapat memanggil ini dalam mutasi setelahnya`addPerson`. `addPerson`akan ditangani terlebih dahulu diikuti oleh`add0ccupation`.

Subscriptions

Langganan digunakan WebSocketsuntuk membuka koneksi dua arah yang langgeng antara server dan kliennya. Biasanya, klien akan berlangganan, atau mendengarkan, ke server. Setiap kali server membuat perubahan sisi server atau melakukan acara, klien berlangganan akan menerima pembaruan. Jenis protokol ini berguna ketika beberapa klien berlangganan dan perlu diberitahu tentang perubahan yang terjadi di server atau klien lain. Misalnya, langganan dapat digunakan untuk memperbarui umpan media sosial. Mungkin ada dua pengguna, Pengguna A dan Pengguna B, yang keduanya berlangganan pembaruan notifikasi otomatis setiap kali mereka menerima pesan langsung. Pengguna A pada Klien A dapat mengirim pesan langsung ke Pengguna B pada Klien B. Klien Pengguna A akan mengirim pesan langsung, yang akan diproses oleh server. Server kemudian akan mengirim pesan langsung ke akun Pengguna B saat mengirim pemberitahuan otomatis ke Klien B.

Berikut adalah contoh `Subscription` yang bisa kita tambahkan ke contoh skema:

```
type Subscription {
  personAdded: Person
}
```

`personAdded`Bidang akan mengirim pesan ke klien berlangganan setiap kali baru `Person` ditambahkan ke sumber data. Dengan asumsi kami memiliki implementasi resolver untuk`personAdded`, kami sekarang dapat menggunakan langganan. Sementara `Subscription` jenisnya ada, kita harus secara eksplisit memanggilnya agar dapat dijalankan dalam kode aplikasi. Ini dapat dilakukan dengan menggunakan `subscription` kata kunci:

```
subscription personAddedOperation {
  personAdded {
    id
    name
  }
}
```

Langganan disebut`personAddedOperation`, dan operasinya`personAdded`. `personAdded`akan mengembalikan `id` dan `name` bidang `Person` instance baru. Melihat contoh mutasi, kami menambahkan `Person` menggunakan operasi ini:

```
addPerson(id: "1", name: "Steve Powers", age: "50", occupation: "Miner")
```

Jika klien kami berlangganan pembaruan ke yang baru ditambahkan`Person`, mereka mungkin melihat ini setelah `addPerson` dijalankan:

```
{
  "data": {
    "personAdded": {
      "id": "1",
      "name": "Steve Powers"
    }
  }
}
```

Di bawah ini adalah ringkasan dari apa yang ditawarkan langganan:

Langganan adalah saluran dua arah yang memungkinkan klien dan server menerima pembaruan yang cepat, tetapi stabil. Mereka biasanya menggunakan WebSocket protokol, yang menciptakan koneksi standar dan aman.

Langganan gesit karena mengurangi overhead pengaturan koneksi. Setelah berlangganan, klien dapat terus menjalankan langganan itu untuk jangka waktu yang lama. Mereka umumnya

menggunakan sumber daya komputasi secara efisien dengan memungkinkan pengembang untuk menyesuaikan masa pakai langganan dan untuk mengkonfigurasi informasi apa yang akan diminta.

Secara umum, langganan memungkinkan klien untuk membuat beberapa langganan sekaligus. Sehubungan dengan ituAWS AppSync, langganan hanya digunakan untuk menerima pembaruan waktu nyata dari layanan. AWS AppSync Mereka tidak dapat digunakan untuk melakukan kueri atau mutasi.

Alternatif utama untuk langganan adalah polling, yang mengirimkan kueri pada interval yang ditetapkan untuk meminta data. Proses ini biasanya kurang efisien daripada langganan dan menempatkan banyak tekanan pada klien dan backend.

Satu hal yang tidak disebutkan dalam contoh skema kami adalah fakta bahwa tipe objek khusus Anda juga harus didefinisikan dalam `schema` root. Jadi saat Anda mengekspor skemaAWS AppSync, mungkin terlihat seperti ini:

schema.graphql

```
schema {
  query: Query
  mutation: Mutation
  subscription: Subscription
}

.
.
.

type Query {
  # code goes here
}
type Mutation {
  # code goes here
}
type Subscription {
  # code goes here
}
```

Pencacahan

Enumerasi, atau enum, adalah skalar khusus yang membatasi argumen hukum yang mungkin dimiliki tipe atau bidang. Ini berarti bahwa setiap kali enum didefinisikan dalam skema, jenis atau bidang terkait akan terbatas pada nilai dalam enum. Enum diserialkan sebagai skalar string. Perhatikan bahwa bahasa pemrograman yang berbeda dapat menangani enum GraphQL secara berbeda. Misalnya, tidak JavaScript memiliki dukungan enum asli, sehingga nilai enum dapat dipetakan ke nilai int sebagai gantinya.

Enum didefinisikan menggunakan `enum` kata kunci. Inilah contohnya:

```
enum trafficSignals {
  solidRed
  solidYellow
  solidGreen
  greenArrowLeft
  ...
}
```

Saat memanggil `trafficLights` enum, argumen hanya bisa`solidRed`,, `solidYellow``solidGreen`, dll. Adalah umum untuk menggunakan enum untuk menggambarkan hal-hal yang memiliki jumlah pilihan yang berbeda tetapi terbatas.

Serikat Serikat/Antarmuka

Lihat Antarmuka dan serikat pekerja di GraphQL.

Bidang GraphQL

Bidang ada dalam lingkup tipe dan menyimpan nilai yang diminta dari layanan GraphQL. Ini sangat mirip dengan variabel dalam bahasa pemrograman lain. Sebagai contoh, berikut adalah jenis `Person` objek:

```
type Person {
   name: String
   age: Int
}
```

Bidang dalam hal ini adalah `name` `age` dan dan memegang `Int` nilai `String` dan, masing-masing. Bidang objek seperti yang ditunjukkan di atas dapat digunakan sebagai input di bidang (operasi) kueri dan mutasi Anda. Misalnya, lihat `Query` di bawah ini:

```
type Query {
  people: [Person]
}
```

`people`Bidang meminta semua contoh `Person` dari sumber data. Ketika Anda menambahkan atau mengambil `Person` di server GraphQL Anda, Anda dapat mengharapkan data mengikuti format tipe dan bidang Anda, yaitu, struktur data Anda dalam skema menentukan bagaimana itu akan disusun dalam respons Anda:

```
}
  "data": {
    "people": [
      {
        "name": "John Smith",
        "age": "50"
      },
      {
        "name": "Andrew Miller",
        "age": "60"
      },
      .
      .
      .
    ]
  }
}
```

Bidang memainkan peran penting dalam penataan data. Ada beberapa properti tambahan yang dijelaskan di bawah ini yang dapat diterapkan ke bidang untuk penyesuaian lebih lanjut.

Daftar

Daftar mengembalikan semua item dari jenis tertentu. Daftar dapat ditambahkan ke jenis bidang menggunakan tanda kurung`[]`:

```
type Person {
  name: String
  age: Int
}
type Query {
  people: [Person]
```

```
}
```

DalamQuery, tanda kurung di sekitarnya Person menunjukkan bahwa Anda ingin mengembalikan semua contoh dari Person dari sumber data sebagai array. Sebagai tanggapan, age nilai name dan masing-masing Person akan dikembalikan sebagai daftar tunggal yang dibatasi:

```
}
  "data": {
    "people": [
      {
        "name": "John Smith",         # Data of Person 1
        "age": "50"
      },
      {
        "name": "Andrew Miller",      # Data of Person 2
        "age": "60"
      },
      .                               # Data of Person N
      .
      .
    ]
  }
}
```

Anda tidak terbatas pada jenis objek khusus. Anda juga dapat menggunakan daftar di bidang jenis objek biasa.

Non-nol

Non-null menunjukkan bidang yang tidak bisa null dalam respons. Anda dapat mengatur bidang ke non-null dengan menggunakan simbol: !

```
type Person {
  name: String!
  age: Int
}
type Query {
  people: [Person]
}
```

nameBidang tidak bisa secara eksplisit nol. Jika Anda menanyakan sumber data dan memberikan input nol untuk bidang ini, kesalahan akan muncul.

Anda dapat menggabungkan daftar dan non-null. Bandingkan pertanyaan ini:

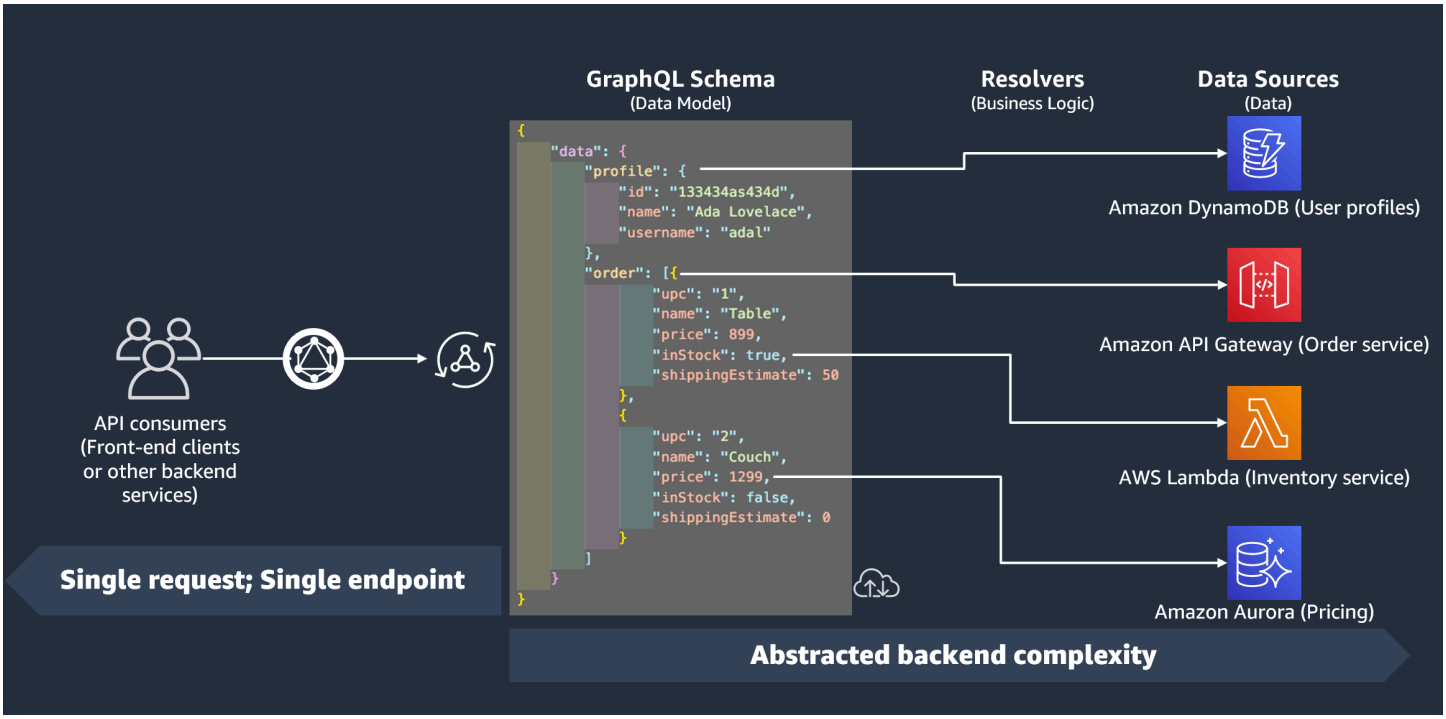
```
type Query {
  people: [Person!]      # Use case 1
}

.
.
.

type Query {
  people: [Person]!      # Use case 2
}

.
.
.

type Query {
  people: [Person!]!     # Use case 3
}
```

Dalam kasus penggunaan 1, daftar tidak dapat berisi item nol. Dalam kasus penggunaan 2, daftar itu sendiri tidak dapat diatur ke null. Dalam kasus penggunaan 3, daftar dan item-itemnya tidak bisa null. Namun, bagaimanapun, Anda masih dapat mengembalikan daftar kosong.

Seperti yang Anda lihat, ada banyak komponen bergerak di GraphQL. Pada bagian ini, kami menunjukkan struktur skema sederhana dan berbagai jenis dan bidang yang didukung skema. Di bagian berikut, Anda akan menemukan komponen lain dari GraphQL API dan bagaimana mereka bekerja dengan skema.

Sumber data

Pada bagian sebelumnya, kita belajar bahwa skema mendefinisikan bentuk data Anda. Namun, kami tidak pernah menjelaskan dari mana data itu berasal. Dalam proyek nyata, skema Anda seperti gateway yang menangani semua permintaan yang dibuat ke server. Ketika permintaan dibuat, skema bertindak sebagai titik akhir tunggal yang berinteraksi dengan klien. Skema akan mengakses, memproses, dan menyampaikan data dari sumber data kembali ke klien. Lihat infografis di bawah ini:



AWS AppSyncdan GraphQL mengimplementasikan solusi Backend For Frontend (BFF) dengan luar biasa. Mereka bekerja bersama-sama untuk mengurangi kompleksitas pada skala besar dengan mengabstraksi backend. Jika layanan Anda menggunakan sumber data dan/atau layanan mikro yang berbeda, pada dasarnya Anda dapat mengabstraksikan beberapa kompleksitas dengan mendefinisikan bentuk data dari setiap sumber (subgraf) dalam satu skema (supergraf). Ini berarti GraphQL API Anda tidak terbatas pada penggunaan satu sumber data. Anda dapat mengaitkan sejumlah sumber data dengan GraphQL API Anda dan menentukan dalam kode Anda bagaimana mereka akan berinteraksi dengan layanan.

Seperti yang Anda lihat di infografis, skema GraphQL berisi semua informasi yang dibutuhkan klien untuk meminta data. Ini berarti semuanya dapat diproses dalam satu permintaan daripada beberapa permintaan seperti halnya dengan REST. Permintaan ini melalui skema, yang merupakan satu-satunya titik akhir layanan. Ketika permintaan diproses, resolver (dijelaskan di bagian berikutnya) mengeksekusi kodenya untuk memproses data dari sumber data yang relevan. Ketika respons dikembalikan, subgraf yang terkait dengan sumber data akan diisi dengan data dalam skema.

AWS AppSyncmendukung banyak jenis sumber data yang berbeda. Pada tabel di bawah ini, kami akan menjelaskan setiap jenis, mencantumkan beberapa manfaat masing-masing, dan memberikan tautan yang berguna untuk konteks tambahan.

Sumber data	Deskripsi	Keuntungan	Informasi tambahan

Amazon DynamoDB	Amazon DynamoDB adalah layanan database NoSQL yang dikelola sepenuhnya yang memberikan kinerja yang cepat dan dapat diprediksi dengan skalabilitas yang mulus. DynamoDB memungkinkan Anda memindahkan beban administrasi untuk mengoperasikan dan menskalakan database terdistribusi sehingga Anda tidak perlu khawatir dengan penyediaan perangkat keras, penyetelan dan konfigurasi, replikasi , patching perangkat lunak, atau penskalaa n klaster. DynamoDB juga menawarkan enkripsi saat istirahat , yang menghilangkan beban operasional dan kompleksitas yang terlibat dalam melindungi data sensitif.	• Kinerja dalam skala: DynamoDB dirancang berdasarkan kinerja yang konsisten pada skala apa pun. Ini dimungkin kan melalui penggunaan partisi. DynamoDB akan secara otomatis mempartisi tabel Anda menjadi beberapa alokasi yang akan disimpan dalam beberapa SSD di beberapa node. Ini umumnya akan meningkatkan throughput jaringan dan mengurangi latensi.
		• Kapasitas dalam skala: DynamoDB memantau lalu lintas Anda dan memungkinkan Anda untuk secara otomatis menskalak an throughput Anda jika jaringan tetap kelebihan beban untuk waktu yang lama.
			• Dokumentasi resmi DynamoDB
			  • Partisi
			  • Penskalaan otomatis
			  • Toleransi kesalahan
			  • Pemantauan
			  • Keamanan
			• GraphQL dan DynamoDB
			• Operasi penyelesai untuk DynamoDB
			• Model harga

Sumber data	Deskripsi	Keuntungan	Informasi tambahan
		• Ketersediaan dan toleransi kesalahan : DynamoDB didukung oleh beberapa Wilayah yang terisolas i secara fisik, masing-masing berisi beberapa Availability Zone yang terisolas i secara fisik. DynamoDB akan secara otomatis beralih ke zona cadangan jika terjadi gangguan layanan. Anda juga dapat mencadang kan dan mereplika si data Anda secara manual untuk jaminan data. • Pencatatan dan pemantaua n: DynamoDB menyediakan beberapa alat analisis untuk tabel Anda. Anda dapat memantau kinerja tabel Anda dan membuat alarm untuk memberi tahu Anda tentang	

Sumber data	Deskripsi	Keuntungan	Informasi tambahan
		perubahan drastis pada layanan. • Keamanan: DynamoDB mengikuti protokol ketat untuk memastikan data Anda sesuai dengan persyarat an keamanan organisasi Anda. • Integrasi dengan AWS AppSync: DynamoDB terintegrasi mulus dengan layanan kami. Anda dapat membuat tabel DynamoDB baru dan secara otomatis menghasil kan skema dari mereka untuk merampingkan proses pengemban gan Anda. Kami juga menyediak an seluruh koleksi operasi untuk dengan mudah meminta data dari tabel DynamoDB yang ada di akun	

Sumber data	Deskripsi	Keuntungan	Informasi tambahan
		Anda di resolver Anda.	

Sumber data	Deskripsi	Keuntungan	Informasi tambahan

AWS Lambda	“AWS Lambdaadalah layanan komputasi yang memungkin kan Anda menjalank an kode tanpa menyediakan atau mengelola server.
	
	Lambda menjalank an kode Anda pada infrastruktur komputasi ketersedi aan tinggi dan melakukan semua administrasi sumber daya komputasi, termasuk pemelihar aan server dan sistem operasi, penyediaa n kapasitas dan penskalaan otomatis, dan pencatatan. Dengan Lambda, yang perlu Anda lakukan hanyalah menyediakan kode Anda di salah satu runtime bahasa yang didukung Lambda.
		• ay-as-you-use Model P: Lambda hanya menagih Anda saat Anda menggunakan sumber dayanya. Mereka juga memungkin kan Anda untuk mengukur jumlah sumber daya yang digunakan dengan kebutuhan aplikasi Anda.
		• Penskalaa n otomatis: Terkadang aplikasi Anda mungkin memerlukan daya komputasi ekstra untuk proses tertentu. Lambda memungkinkan Anda untuk secara otomatis menskalak an sumber daya komputasi agar sesuai dengan kebutuhan aplikasi Anda.
		• Waktu penerapan yang lebih cepat: Anda dapat merampingkan
			• Dokumentasi resmi
			  • Penskalaan
			  • penyebaran
			  • runtime
			• Tutorial penyelesai Lambda
			• Model harga

Sumber data	Deskripsi	Keuntungan	Informasi tambahan
		proses pengembangan Anda melalui paket penerapan . Gunakan paket untuk mengunggah kode fungsi Anda ke layanan Lambda. Anda kemudian dapat menggunakan lingkungan runtime mereka untuk menguji dan menjalankan fungsi Anda. • Keserbagunaan: Lambda dapat digunakan dalam banyak kasus penggunaan. Anda dapat menginteg rasikan Lambda dengan mulus dengan layanan AWS dan layanan pihak ketiga. Beberapa contoh termasuk pipa CI/ CD dan layanan surat massal. • Integrasi dengan AWS AppSync: Anda dapat dengan mudah	

Sumber data	Deskripsi	Keuntungan	Informasi tambahan
		memanggil fungsi Lambda Anda di resolver Anda untuk menangani permintaan. Layanan kami menyediakan operasi permintaa n yang efisien untuk melakukan panggilan Lambda. Kami mengizinkan panggilan tunggal dan batch.	

Sumber data	Deskripsi	Keuntungan	Informasi tambahan
OpenSearch	Amazon OpenSearch Service adalah layanan terkelola yang memudahkan penerapan, pengoperasian, dan skala OpenSearch cluster di AWS Cloud. Amazon OpenSearch Service mendukung OpenSearch dan warisan Elasticsearch OSS (hingga 7.10, versi open source terakhir dari perangkat lunak). Saat Anda membuat cluster, Anda memiliki opsi mesin pencari mana yang akan digunakan. OpenSearchadalah mesin pencari dan analitik sumber terbuka penuh untuk kasus penggunaan seperti analitik log, pemantauan aplikasi waktu nyata, dan analisis clickstream. Untuk informasi lebih lanjut, lihat dokumentasi OpenSearch.	• Penskalaan: Anda dapat dengan mudah menskalakan layanan agar sesuai dengan kebutuhan layanan Anda melalui Tanpa OpenSearch Server. • Penyerapan data: Anda dapat menggunakan OpenSearch Ingestion untuk mengimpor, memproses, dan menganalisis data. Ada banyak aplikasi untuk konsumsi data, yang dapat Anda temukan di sini. • Keamanan: OpenSearch dapat mengelola konfigurasi AWS keamanan Anda termasuk IAM,, VPC CloudTrail, otentikasi, dll. • Ketersediaan: OpenSearch juga mendukung berbagai Wilayah	• Dokumentasi resmi • Tanpa server • Model harga

Sumber data	Deskripsi	Keuntungan	Informasi tambahan
	OpenSearch Layanan Amazon menyediakan semua sumber daya untuk OpenSearch klaster Anda dan meluncurkannya. Ini juga secara otomatis mendeteksi dan mengganti node OpenSearch Layanan yang gagal, mengurangi overhead yang terkait dengan infrastruktur yang dikelola sendiri. Anda dapat menskalakan klaster Anda dengan satu panggilan API atau beberapa klik di konsol.”	dan Availability Zone dalam layanannya. • Integrasi dengan AWS AppSync: In AWS AppSync, Anda dapat menggunakan GraphQL API untuk menyimpan dan mengambil data dari domain Layanan OpenSearch yang ada di akun Anda.	

Sumber data	Deskripsi	Keuntungan	Informasi tambahan
Titik akhir HTTP	Anda dapat menggunakan titik akhir HTTP sebagai sumber data. AWS AppSync dapat mengirim permintaan ke titik akhir dengan informasi yang relevan seperti params dan payload. Respons HTTP akan diekspos ke resolver, yang akan mengembalikan respons akhir setelah selesai operasinya.	• Berguna untuk aplikasi sederhana yang tidak terintegrasi dengan layanan seperti Lambda.	• Referensi penyelesai

Sumber data	Deskripsi	Keuntungan	Informasi tambahan

Amazon EventBridge	“EventBridge adalah layanan tanpa server yang menggunak an peristiwa untuk menghubungkan komponen aplikasi bersama-sama, sehingga memudahka n Anda untuk membangun aplikasi berbasis peristiwa yang dapat diskalaka n. Gunakan untuk merutekan acara dari sumber seperti aplikasi rumahan, AWS layanan, dan perangkat lunak pihak ketiga ke aplikasi konsumen di seluruh organisas i Anda. EventBrid ge menyediakan cara sederhana dan konsisten untuk menelan, memfilter , mengubah, dan menyampaikan acara sehingga Anda dapat membangun aplikasi baru dengan cepat.	• Arsitektur berbasis peristiwa: Anda dapat memanfaat kan arsitektur berbasis peristiwa.
		• Penjadwalan: Anda dapat menggunak an EventBridge Scheduler untuk mengotomatiskan tugas dan aturan Anda menggunakan ekspresi cron atau mengatur interval waktu sebagai alternatif untuk pola peristiwa.
		• Pipa: Menggunak an EventBridge Pipa, Anda dapat mengganti bus acara dengan pipa yang mencakup pola acara penyaringan tambahan dan pengayaan melalui transformasi data sebelum mengirim acara ke target.
		• Integrasi dengan AWS AppSync: AWS AppSync memungkinkan
			• Dokumentasi resmi
			  • Pipa
			  • Penjadwal
			• Referensi penyelesai
			• Model harga

Sumber data	Deskripsi	Keuntungan	Informasi tambahan
		Anda mengirim acara ke bus acara menggunakan resolver Anda.	

Sumber data	Deskripsi	Keuntungan	Informasi tambahan

Basis data relasional	Amazon Relational Database Service (Amazon RDS) adalah layanan web yang membuatnya lebih mudah untuk mengatur, mengoperasikan, dan menskalakan database relasional di Cloud. AWS Ini menyediakan efisiensi biaya, kapasitas resizable untuk database relasional standar industri dan mengelola tugas-tugas administrasi database umum.	• Mengelola menjadi mudah: Secara berkala, RDS melakukan pemeliharaan pada sumber dayanya. Pemeliharaan paling sering melibatkan pembaruan ke perangkat keras yang mendasari instans DB, sistem operasi yang mendasari (OS), atau versi mesin database. Dalam keadaan normal, Anda dapat memutuskan kapan harus melakukan pembaruan (pengecualian termasuk patch keamanan).
		• Rekomendasi: Fitur rekomendasi RDS memberikan saran otomatis untuk memperbaiki potensi masalah dalam instans Anda.
			• Dokumentasi resmi
			  • Fitur
			  • Pemeliharaan
			  • Rekomendasi
			  • Opsi penyimpanan
			  • Ketersediaan
			  • Keamanan
			• Model harga

Sumber data	Deskripsi	Keuntungan	Informasi tambahan
		• Ketersediaan: RDS tersedia di berbagai Wilayah fisik di seluruh dunia. Anda dapat dengan mudah mendistri busikan kebutuhan database Anda di berbagai node untuk memberika n layanan yang lebih baik kepada pelanggan Anda. • Kustomisasi: RDS disesuaikan untuk memenuhi persyaratan perusahaan besar. RDS menyediak an berbagai opsi untuk komputasi, penyebaran cepat, skalabilitas, dan penyimpanan. • Keamanan: RDS terintegrasi dengan beberapa alat dan layanan untuk menjaga keamanan database pada tingkat pengguna, database, dan jaringan.	

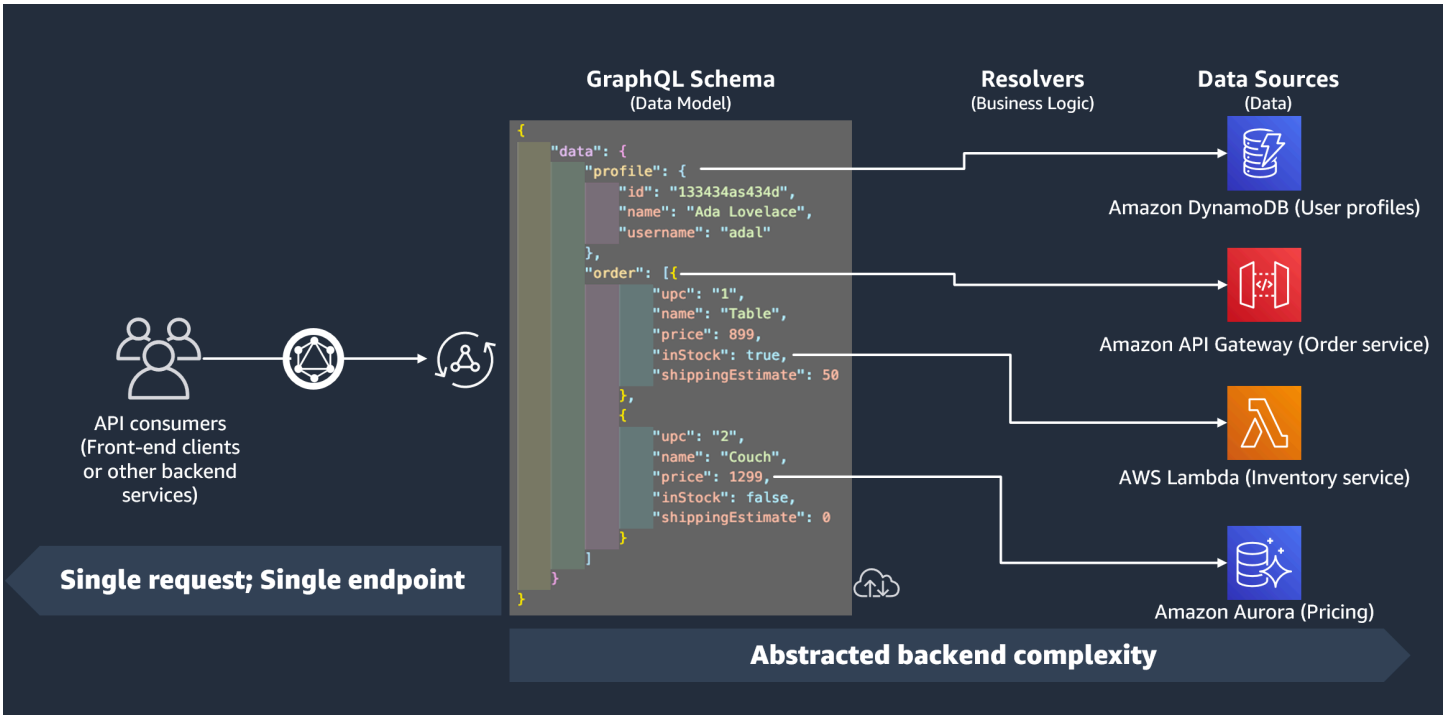
Sumber data	Deskripsi	Keuntungan	Informasi tambahan
		• Integrasi dengan AWS AppSync: Jika Anda mencari solusi backend yang matang, AWS AppSync memungkinkan Anda mengirim, memproses, menyimpan, dan mengembalikan data menggunak an instance Anda sebagai sumber data.	
Tidak ada sumber data	Jika Anda tidak berencana menggunakan layanan sumber data, Anda dapat mengaturnyanone. Sumber none data, meskipun masih secara eksplisit dikategorikan sebagai sumber data, bukanlah media penyimpanan. Meskipun demikian, ini masih berguna dalam kasus tertentu untuk manipulasi data dan pass-through.	• Berpotensi berguna untuk hal-hal seperti konversi data • Berguna saat menyelesaikan sesuatu secara lokal	• Referensi penyelesai

> **Tip**
>
> Untuk informasi selengkapnya tentang cara sumber data berinteraksiAWS AppSync, lihat Melampirkan sumber data.

Penyelesai

Dari bagian sebelumnya, Anda belajar tentang komponen skema dan sumber data. Sekarang, kita perlu membahas bagaimana skema dan sumber data berinteraksi. Semuanya dimulai dengan resolver.

Resolver adalah unit kode yang menangani bagaimana data bidang itu akan diselesaikan ketika permintaan dibuat ke layanan. Resolver dilampirkan ke bidang tertentu dalam tipe Anda dalam skema Anda. Mereka paling sering digunakan untuk mengimplementasikan operasi perubahan status untuk kueri, mutasi, dan operasi bidang langganan Anda. Penyelesai akan memproses permintaan klien, lalu mengembalikan hasilnya, yang dapat berupa sekelompok tipe keluaran seperti objek atau skalar:



Runtime penyelesai

DiAWS AppSync, Anda harus terlebih dahulu menentukan runtime untuk resolver Anda. Runtime resolver menunjukkan lingkungan di mana resolver dijalankan. Ini juga menentukan bahasa resolver

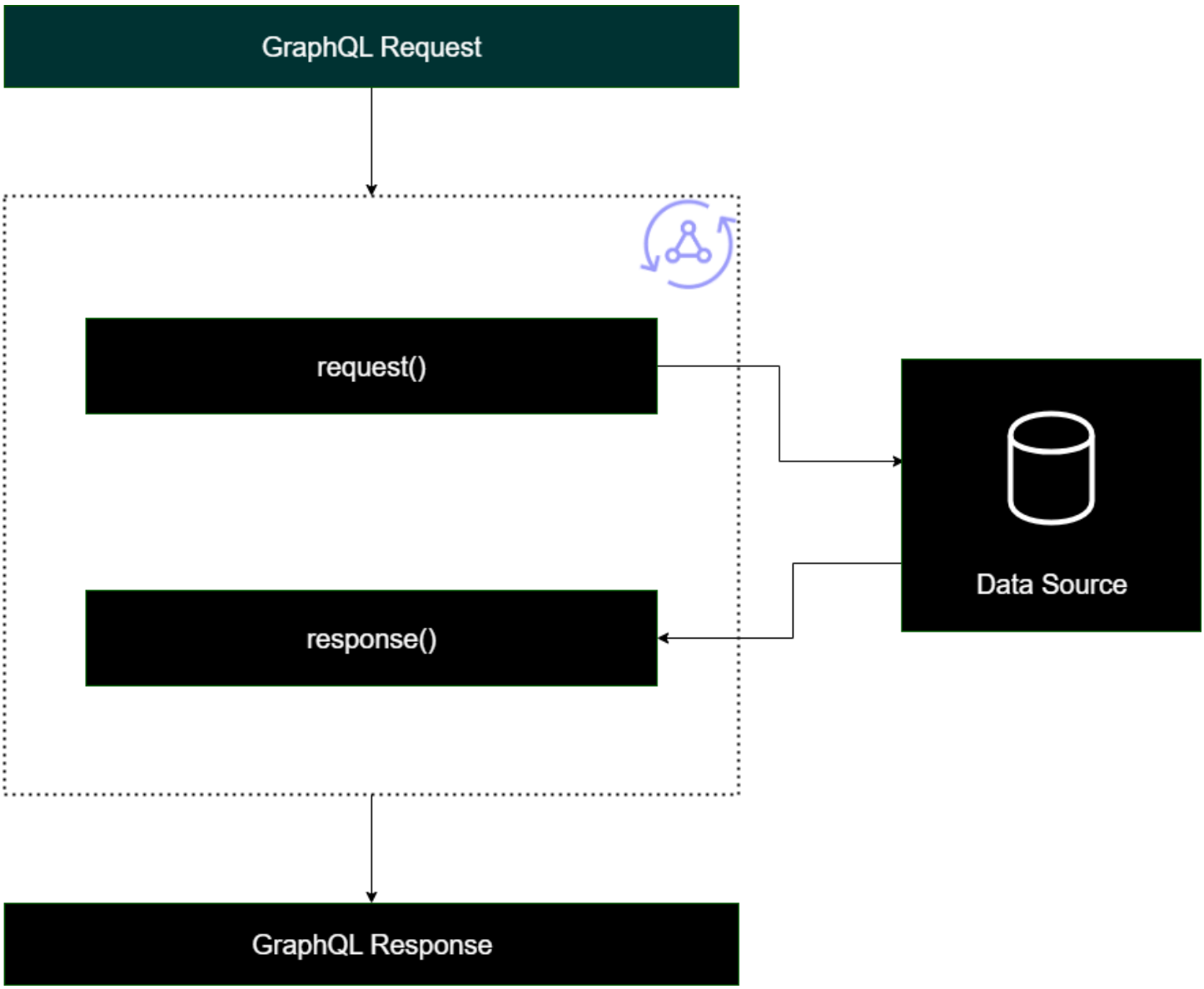
Anda akan ditulis. AWS AppSyncsaat ini mendukung APPSYNC_JS untuk JavaScript dan Velocity Template Language (VTL). Lihat fitur JavaScript runtime untuk resolver dan fungsi untuk JavaScript atau referensi utilitas template pemetaan Resolver untuk VTL.

Struktur penyelesai

Dari segi kode, resolver dapat disusun dalam beberapa cara. Ada resolver unit dan pipa.

Penyelesai unit

Unit resolver terdiri dari kode yang mendefinisikan permintaan tunggal dan penangan respons yang dijalankan terhadap sumber data. Handler permintaan mengambil objek konteks sebagai argumen dan mengembalikan payload permintaan yang digunakan untuk memanggil sumber data Anda. Response handler menerima payload kembali dari sumber data dengan hasil dari permintaan yang dieksekusi. Response handler mengubah payload menjadi respons GraphQL untuk menyelesaikan bidang GraphQL.



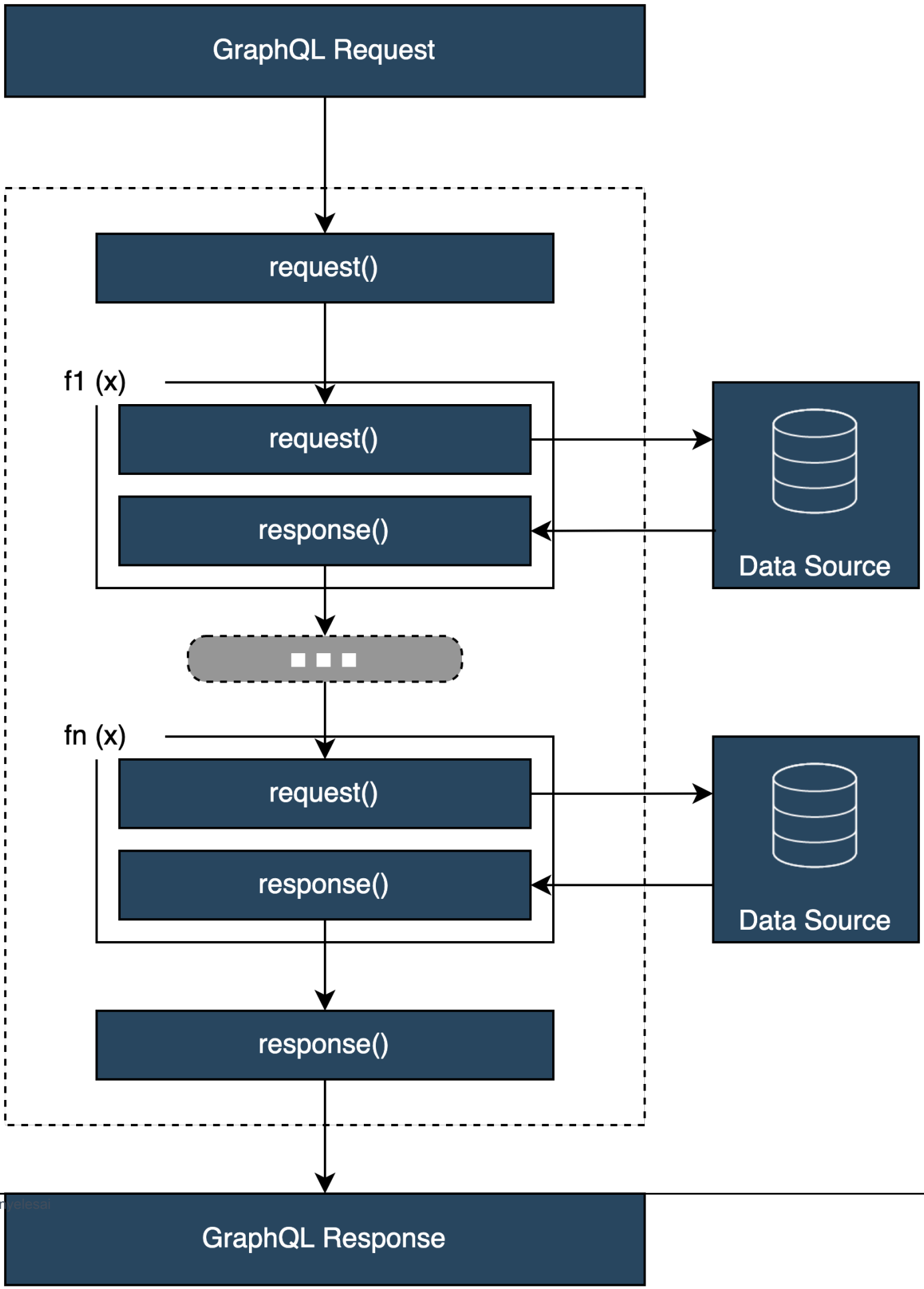
Penyelesai pipa

Saat menerapkan resolver pipa, ada struktur umum yang mereka ikuti:

- Sebelum langkah: Ketika permintaan dibuat oleh klien, resolver untuk bidang skema yang digunakan (biasanya kueri, mutasi, langganan Anda) diteruskan data permintaan. Penyelesai akan mulai memproses data permintaan dengan penangan langkah sebelum, yang memungkinkan beberapa operasi pra-pemrosesan dilakukan sebelum data bergerak melalui resolver.
- Fungsi: Setelah langkah sebelum berjalan, permintaan diteruskan ke daftar fungsi. Fungsi pertama dalam daftar akan dijalankan terhadap sumber data. Fungsi adalah subset dari kode resolver Anda yang berisi permintaan dan penangan responsnya sendiri. Seorang penangan permintaan akan

mengambil data permintaan dan melakukan operasi terhadap sumber data. Response handler akan memproses respon sumber data sebelum meneruskannya kembali ke daftar. Jika ada lebih dari satu fungsi, data permintaan akan dikirim ke fungsi berikutnya dalam daftar yang akan dieksekusi. Fungsi dalam daftar akan dijalankan secara serial dalam urutan yang ditentukan oleh pengembang. Setelah semua fungsi dieksekusi, hasil akhir diteruskan ke langkah setelahnya.

- Langkah setelah: Langkah setelah adalah fungsi handler yang memungkinkan Anda melakukan beberapa operasi akhir pada respons fungsi akhir sebelum meneruskannya ke respons GraphQL.


GraphQL Request
request()
f1 (x)
request()
response()
Data Source
fn (x)
request()
response()
Data Source
response()
GraphQL Response

Struktur penangan penyelesai

Handler biasanya fungsi yang disebut `Request` dan`Response`:

```
export function request(ctx) {
    // Code goes here
}

export function response(ctx) {
    // Code goes here
}
```

Dalam unit resolver, hanya akan ada satu set fungsi ini. Dalam resolver pipa, akan ada satu set ini untuk langkah sebelum dan sesudah dan satu set tambahan per fungsi. Untuk memvisualisasikan bagaimana ini bisa terlihat, mari kita tinjau `Query` jenis sederhana:

```
type Query {
 helloWorld: String!
}
```

Ini adalah kueri sederhana dengan satu bidang `helloWorld` yang disebut tipe`String`. Mari kita asumsikan kita selalu ingin bidang ini mengembalikan string “Hello World”. Untuk menerapkan perilaku ini, kita perlu menambahkan resolver ke bidang ini. Dalam unit resolver, kita bisa menambahkan sesuatu seperti ini:

```
export function request(ctx) {
    return {}
}

export function response(ctx) {
    return "Hello World"
}
```

Itu `request` bisa dibiarkan kosong karena kami tidak meminta atau memproses data. Kami juga dapat mengasumsikan sumber data kami`None`, menunjukkan kode ini tidak perlu melakukan pemanggilan apa pun. Responsnya hanya mengembalikan “Hello World”. Untuk menguji resolver ini, kita perlu membuat permintaan menggunakan tipe query:

```
query helloWorldTest {
```

```
  helloWorld
}
```

Ini adalah kueri `helloWorldTest` yang disebut yang mengembalikan `helloWorld` bidang. Saat dijalankan, penyelesai `helloWorld` bidang juga mengeksekusi dan mengembalikan respons:

```
{
  "data": {
    "helloWorld": "Hello World"
  }
}
```

Mengembalikan konstanta seperti ini adalah hal paling sederhana yang dapat Anda lakukan. Pada kenyataannya, Anda akan mengembalikan input, daftar, dan banyak lagi. Berikut adalah contoh yang lebih rumit:

```
type Book {
  id: ID!
  title: String
}

type Query {
  getBooks: [Book]
}
```

Di sini kami mengembalikan daftar`Books`. Mari kita asumsikan kita menggunakan tabel DynamoDB untuk menyimpan data buku. Penangan kami mungkin terlihat seperti ini:

```
/**
 * Performs a scan on the dynamodb data source
 */
export function request(ctx) {
  return { operation: 'Scan' };
}

/**
 * return a list of scanned post items
 */
export function response(ctx) {
  return ctx.result.items;
}
```

Permintaan kami menggunakan operasi pemindaian bawaan untuk mencari semua entri dalam tabel, menyimpan temuan dalam konteks, lalu meneruskannya ke respons. Tanggapan mengambil item hasil dan mengembalikannya sebagai tanggapan:

```
{
  "data": {
    "getBooks": {
      "items": [
        {
          "id": "abcdefgh-1234-1234-1234-abcdefghijkl",
          "title": "book1"
        },
        {
          "id": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
          "title": "book2"
        },

        ...

      ]
    }
  }
}
```

Konteks penyelesai

Dalam resolver, setiap langkah dalam rantai penangan harus menyadari status data dari langkah sebelumnya. Hasil dari satu handler dapat disimpan dan diteruskan ke yang lain sebagai argumen. GraphQL mendefinisikan empat argumen resolver dasar:

Argumen dasar penyelesai	Deskripsi
`obj`, `root`, `parent`, dll.	Hasil dari orang tua.
`args`	Argumen yang diberikan ke bidang dalam kueri GraphQL.
`context`	Nilai yang diberikan kepada setiap resolver dan menyimpan informasi kontekstual penting

Argumen dasar penyelesai	Deskripsi
	seperti pengguna yang saat ini masuk, atau akses ke database.
`info`	Nilai yang menyimpan informasi khusus bidang yang relevan dengan kueri saat ini serta detail skema.

DalamAWS AppSync, argumen `context` (ctx) dapat menampung semua data yang disebutkan di atas. Ini adalah objek yang dibuat per permintaan dan berisi data seperti kredensyal otorisasi, data hasil, kesalahan, metadata permintaan, dll. Konteksnya adalah cara mudah bagi programmer untuk memanipulasi data yang berasal dari bagian lain dari permintaan. Ambil cuplikan ini lagi:

```
/**
 * Performs a scan on the dynamodb data source
 */
export function request(ctx) {
  return { operation: 'Scan' };
}

/**
 * return a list of scanned post items
 */
export function response(ctx) {
  return ctx.result.items;
}
```

Permintaan diberikan konteks (ctx) sebagai argumen; ini adalah status permintaan. Ini melakukan pemindaian untuk semua item dalam tabel, kemudian menyimpan hasilnya kembali dalam konteks di`result`. Konteks kemudian diteruskan ke argumen respons, yang mengakses `result` dan mengembalikan isinya.

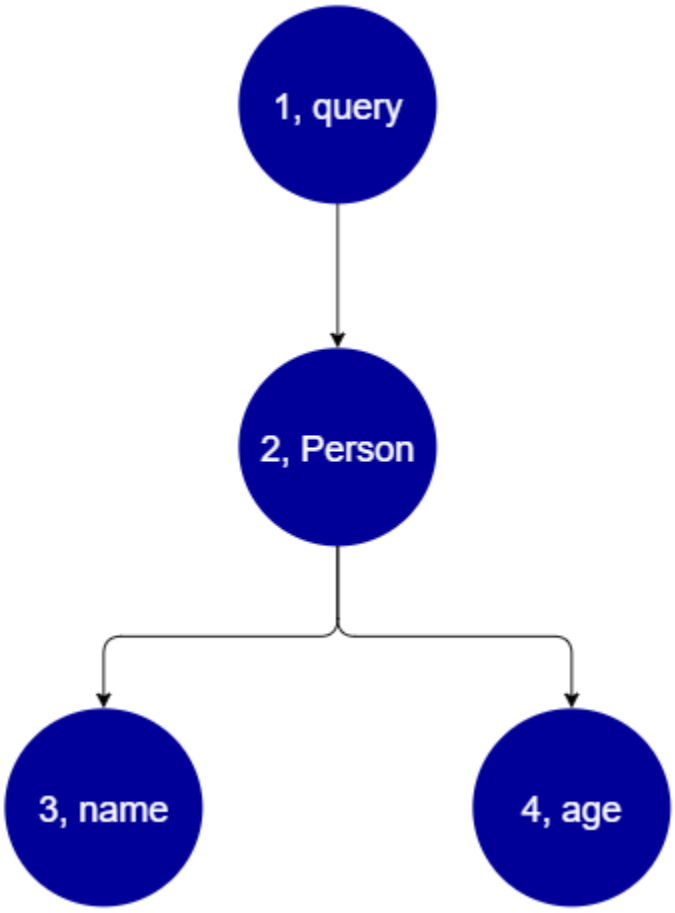
Permintaan dan Parsing

Saat Anda membuat kueri ke layanan GraphQL Anda, itu harus dijalankan melalui proses parsing dan validasi sebelum dieksekusi. Permintaan Anda akan diurai dan diterjemahkan ke dalam pohon sintaks abstrak. Isi pohon divalidasi dengan menjalankan beberapa algoritma validasi terhadap

skema Anda. Setelah langkah validasi, simpul pohon dilintasi dan diproses. Resolver dipanggil, hasilnya disimpan dalam konteks, dan respons dikembalikan. Misalnya, ambil kueri ini:

```
query {
  Person {  //object type
    name  //scalar
    age   //scalar
  }
}
```

Kami kembali `Person` dengan a `name` dan `age` ladang. Saat menjalankan kueri ini, pohon akan terlihat seperti ini:



Dari pohon, tampak bahwa permintaan ini akan mencari root untuk `Query` dalam skema. Di dalam kueri, `Person` bidang akan diselesaikan. Dari contoh sebelumnya, kita tahu bahwa ini bisa menjadi masukan dari pengguna, daftar nilai, dll `Person` kemungkinan besar terkait dengan tipe objek yang memegang bidang yang kita butuhkan (namedanage). Setelah dua bidang anak ini ditemukan, mereka diselesaikan dalam urutan yang diberikan (namediikuti olehage). Setelah pohon benar-benar diselesaikan, permintaan selesai dan akan dikirim kembali ke klien.

Properti tambahan dari GraphQL

GraphQL terdiri dari beberapa prinsip desain untuk menjaga kesederhanaan dan ketahanan dalam skala.

Deklaratif

GraphQL bersifat deklaratif, yang berarti pengguna akan mendeskripsikan (membentuk) data dengan hanya mendeklarasikan bidang yang ingin mereka kueri. Respons hanya akan mengembalikan data untuk properti ini. *Misalnya, berikut adalah operasi yang mengambil Book objek dalam tabel DynamoDB dengan nilai id ISBN 13 9780199536061:*

```
{
  getBook(id: "9780199536061") {
    name
    year
    author
  }
}
```

Respons akan mengembalikan bidang di payload (`name`,`year`, dan`author`) dan tidak ada yang lain:

```
{
  "data": {
    "getBook": {
      "name": "Anna Karenina",
      "year": "1878",
      "author": "Leo Tolstoy",
    }
  }
}
```

Karena prinsip desain ini, GraphQL menghilangkan masalah abadi dari pengambilan berlebihan dan kekurangan yang ditangani REST API dalam sistem yang kompleks. Ini menghasilkan pengumpulan data yang lebih efisien dan peningkatan kinerja jaringan.

Hirarkis

GraphQL fleksibel karena data yang diminta dapat dibentuk oleh pengguna agar sesuai dengan kebutuhan aplikasi. Data yang diminta selalu mengikuti jenis dan sintaks properti yang ditentukan

dalam GraphQL API Anda. *Misalnya, cuplikan berikut menunjukkan getBook operasi dengan lingkup bidang baru yang disebut yang mengembalikan semua string kutipan tersimpan dan halaman quotes yang ditautkan ke 9780199536061: Book*

```
{
  getBook(id: "9780199536061") {
    name
    year
    author
    quotes {
      description
      page
    }
  }
}
```

Menjalankan query ini mengembalikan hasil sebagai berikut:

```
{
  "data": {
    "getBook": {
      "name": "Anna Karenina",
      "year": "1878",
      "author": "Leo Tolstoy",
      "quotes": [
         {
            "description": "The highest Petersburg society is essentially one: in it
 everyone knows everyone else, everyone even visits everyone else.",
            "page": 135
         },
         {
            "description": "Happy families are all alike; every unhappy family is
 unhappy in its own way.",
            "page": 1
         },
         {
            "description": "To Konstantin, the peasant was simply the chief partner in
 their common labor.",
            "page": 251
         }
      ]
    }
  }
```

```
}
```

Seperti yang Anda lihat, `quotes` bidang yang ditautkan ke buku yang diminta dikembalikan sebagai array dalam format yang sama yang dijelaskan oleh kueri kami. Meskipun tidak ditampilkan di sini, GraphQL memiliki keuntungan tambahan karena tidak khusus tentang lokasi data yang diambilnya. `Books`dan `quotes` dapat disimpan secara terpisah, tetapi GraphQL akan tetap mengambil informasi selama asosiasi ada. Ini berarti kueri Anda dapat mengambil banyak data mandiri dalam satu permintaan.

Introspektif

GraphQL adalah mendokumentasikan diri, atau introspektif. Ini mendukung beberapa operasi bawaan yang memungkinkan pengguna untuk melihat jenis dan bidang yang mendasarinya dalam skema. Misalnya, berikut adalah `Foo` tipe dengan `description` bidang `date` dan:

```
type Foo {
 date: String
 description: String
}
```

Kita bisa menggunakan `__type` operasi untuk menemukan metadata pengetikan di bawah skema:

```
{
  __type(name: "Foo") {
    name                    # returns the name of the type
    fields {                # returns all fields in the type
      name                  # returns the name of each field
      type {                # returns all types for each field
        name                # returns the scalar type
      }
    }
  }
}
```

Ini akan mengembalikan respons:

```
{
  "__type": {
    "name": "Foo",                    # The type name
    "fields": [
```

```
      {
        "name": "date",                # The date field
        "type": { "name": "String" }   # The date's type
      },
      {
        "name": "description",         # The description field
        "type": { "name": "String" }   # The description's type
      },
    ]
  }
}
```

Fitur ini dapat digunakan untuk mengetahui jenis dan bidang apa yang didukung skema GraphQL tertentu. GraphQL mendukung berbagai macam operasi introspektif ini. Untuk informasi lebih lanjut, lihat Introspeksi.

Pengetikan yang kuat

GraphQL mendukung pengetikan yang kuat melalui jenis dan sistem bidangnya. Ketika Anda mendefinisikan sesuatu dalam skema Anda, itu harus memiliki tipe yang dapat divalidasi sebelum runtime. Itu juga harus mengikuti spesifikasi sintaks GraphQL. Konsep ini tidak berbeda dengan pemrograman dalam bahasa lain. Misalnya, inilah Foo tipe dari sebelumnya:

```
type Foo {
 date: String
 description: String
}
```

Kita bisa melihat bahwa Foo adalah objek yang akan dibuat. Di dalam instanceFoo, akan ada `date` dan `description` field, keduanya tipe `String` primitif (skalar). Secara sintaksis, kita melihat bahwa Foo dideklarasikan, dan bidangnya ada di dalam ruang lingkupnya. Kombinasi pemeriksaan tipe dan sintaks logis ini memastikan bahwa GraphQL API Anda ringkas dan terbukti dengan sendirinya. Spesifikasi pengetikan dan sintaks GraphQL dapat ditemukan di sini.

Memulai: Membuat API GraphQL pertama Anda

Anda dapat menggunakanAWS AppSynckonsol untuk mengonfigurasi dan meluncurkan API GraphQL. GraphQL API umumnya membutuhkan tiga komponen:

1. Skema GraphQL- Skema GraphQL Anda adalah cetak biru API. Ini mendefinisikan jenis dan bidang yang dapat Anda minta ketika operasi dijalankan. Untuk mengisi skema dengan data, Anda harus menghubungkan sumber data ke API GraphQL. Dalam panduan quickstart ini, kita akan membuat skema menggunakan model yang telah ditentukan.
2. Sumber data- Ini adalah sumber daya yang berisi data untuk mengisi API GraphQL Anda. Ini bisa berupa tabel DynamoDB, fungsi Lambda, dll.AWS AppSyncmendukung banyak sumber data untuk membangun API GraphQL yang kuat dan dapat diskalakan. Sumber data ditautkan ke bidang dalam skema. Setiap kali permintaan dilakukan pada bidang, data dari sumber mengisi bidang. Mekanisme ini dikendalikan oleh resolver. Dalam panduan quickstart ini, kita akan membuat sumber data menggunakan model yang telah ditentukan di samping skema.
3. Penyelesai- Resolver bertanggung jawab untuk menghubungkan bidang skema ke sumber data. Mereka mengambil data dari sumber, kemudian mengembalikan hasil berdasarkan apa yang didefinisikan oleh bidang.AWS AppSyncmendukung keduanyaJavaScriptdan VTL untuk menulis resolver untuk API GraphQL Anda. Dalam panduan mulai cepat ini, resolver akan dibuat secara otomatis berdasarkan skema dan sumber data. Kami tidak akan mempelajari ini di bagian ini.

AWS AppSyncmendukung pembuatan dan konfigurasi semua komponen GraphQL. Saat membuka konsol, Anda dapat menggunakan metode berikut untuk membuat API:

1. Merancang API GraphQL yang disesuaikan dengan menghasilkannya melalui model yang telah ditentukan dan menyiapkan tabel DynamoDB baru (sumber data) untuk mendukungnya.
2. Merancang API GraphQL dengan skema kosong dan tidak ada sumber data atau resolver.
3. Menggunakan tabel DynamoDB untuk mengimpor data dan menghasilkan jenis dan bidang skema Anda.
4. MenggunakanAWS AppSynciniWebSocketkemampuan dan arsitektur Pub/Sub untuk mengembangkan API real-time.
5. Menggunakan API GraphQL yang ada (API sumber) untuk menautkan ke API Gabungan.

> **Note**
>
> Kami merekomendasikan untuk meninjauMerancang skemabagian sebelum bekerja dengan alat yang lebih canggih. Panduan ini akan menjelaskan contoh sederhana yang dapat Anda gunakan secara konseptual untuk membangun aplikasi yang lebih kompleksAWS AppSync.

AWS AppSyncjuga mendukung beberapa opsi non-konsol untuk membuat API GraphQL. Ini termasuk:

1. AWS Amplify
2. AWS SAM
3. AWS CloudFormation
4. CDK

Contoh berikut akan menunjukkan cara membuat komponen dasar dari API GraphQL menggunakan model yang telah ditentukan dan DynamoDB.

Topik

Langkah 1: Luncurkan skema

Dalam contoh ini, Anda akan membuatTodoAPI yang memungkinkan pengguna untuk membuatTodoitem untuk pengingat tugas sehari-hari seperti`Selesaikan tugas`atau`Ambil bahan makanan`. API ini akan mendemonstrasikan cara menggunakan operasi GraphQL di mana status tetap ada dalam tabel DynamoDB.

Secara konseptual, ada tiga langkah utama untuk membuat API GraphQL pertama Anda. Anda harus menentukan skema (tipe dan bidang), melampirkan sumber data Anda ke bidang Anda, lalu menulis resolver yang menangani logika bisnis. Namun, pengalaman konsol mengubah urutan ini. Kita akan

mulai dengan mendefinisikan bagaimana kita ingin sumber data kita berinteraksi dengan skema kita, kemudian mendefinisikan skema dan resolver nanti.

Untuk membuat API GraphQL Anda

1. Masuk keAWS Management Consoledan bukaAppSynckonsol.
2. DiDasbor, pilihBuat API.
3. SementaraAPI GraphQLdipilih, pilihDesain dari awal. Lalu, pilih Selanjutnya.
4. UntukNama API, ubah nama yang telah diisi sebelumnya menjadi**`Todo API`**, lalu pilihBerikutnya.

   > **Note**
   >
   > Ada juga opsi lain yang ada di sini, tetapi kami tidak akan menggunakannya untuk contoh ini.

5. DiTentukan sumber daya GraphQLbagian, lakukan hal berikut:

   a. PilihBuat tipe yang didukung oleh tabel DynamoDB sekarang.

      > **Note**
      >
      > Ini berarti kita akan membuat tabel DynamoDB baru untuk dilampirkan sebagai sumber data.

   b. DiNama Modelbidang, masukkan**`Todo`**.

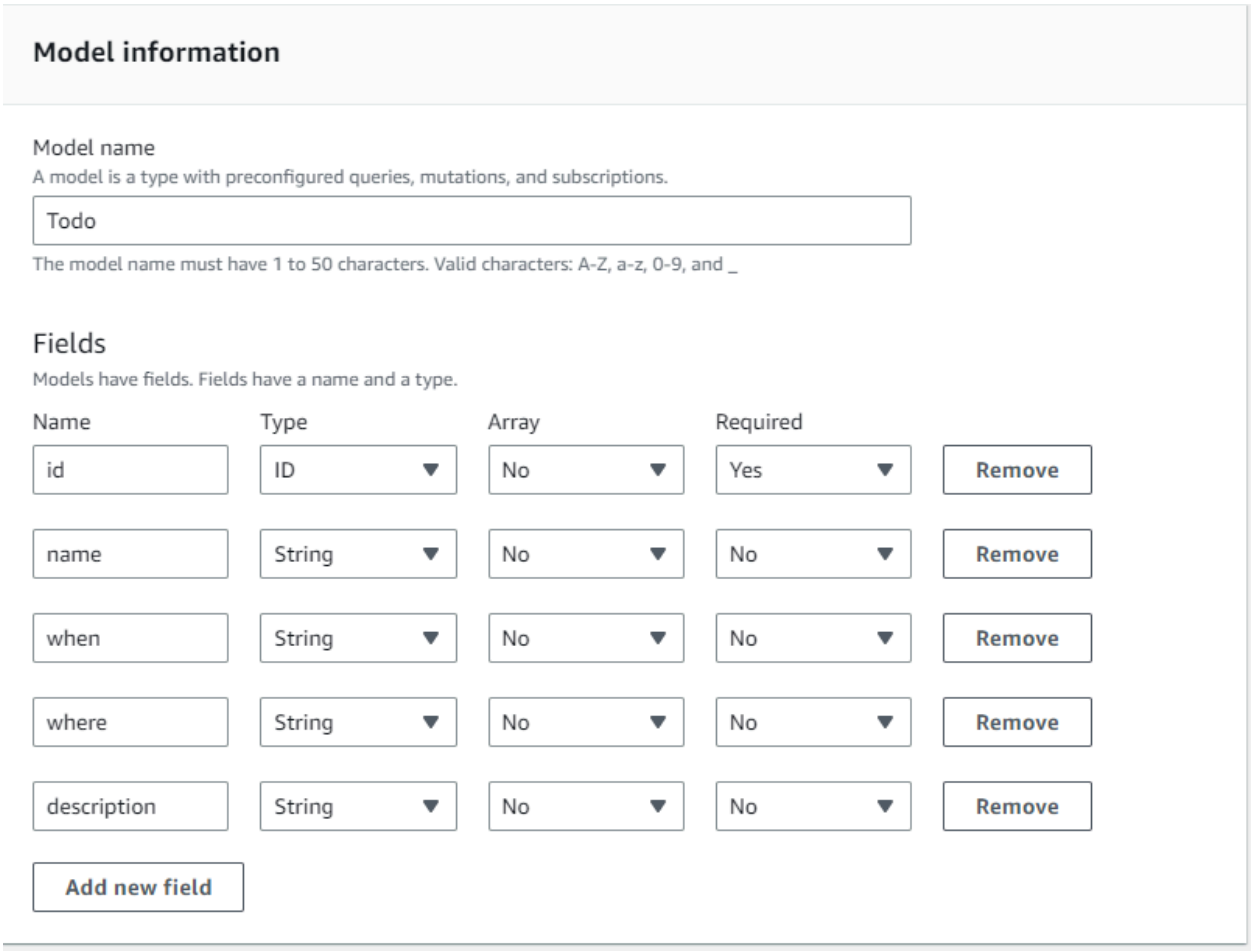
      > **Note**
      >
      > Persyaratan pertama kami adalah mendefinisikan skema kami. IniNama Modelakan menjadi nama tipe, jadi yang sebenarnya Anda lakukan adalah membuat`type`disebut`Todo`yang akan ada dalam skema:
      >
      > ```
      > type Todo {}
      > ```

   c. Di bawahBidang, lakukan hal berikut:

      i. Buat bidang bernama**`id`**, dengan tipe`ID`, dan diperlukan disetel ke`Yes`.

> **Note**
>
> Ini adalah bidang yang akan ada dalam ruang lingkup AndaTodojenis. Nama bidang Anda di sini akan dipanggiliddengan jenisID!:
>
> ```
> type Todo {
>  id: ID!
> }
> ```
>
> AWS AppSyncmendukung beberapa nilai skalar untuk kasus penggunaan yang berbeda.

ii. MenggunakanTambahkan bidang baru, buat empat bidang tambahan denganNamenilai diatur ke**`name`**,**`when`**,**`where`**, dan**`description`**. merekaTypenilai akanString, danArraydanRequirednilai keduanya akan diatur keNo. Itu akan terlihat seperti ini:



> **Note**
>
> Tipe lengkap dan bidangnya akan terlihat seperti ini:
>
> ```
> type Todo {
>  id: ID!
>  name: String
>  when: String
>  where: String
>  description: String
> }
> ```
>
> Karena kami membuat skema menggunakan model yang telah ditentukan ini, itu juga akan diisi dengan beberapa mutasi boilerplate berdasarkan tipe seperti`create`,`delete`, dan`update`untuk membantu Anda mengisi sumber data Anda dengan mudah.

d. Di bawahmengkonfigurasi tabel model, masukkan nama tabel, seperti**`TodoAPITable`**. MengaturKunci Utamakepada`id`.

> **Note**
>
> Kami pada dasarnya membuat tabel DynamoDB baru yang disebut*`TodoApitable`*yang akan dilampirkan ke API sebagai sumber data utama kami. Kunci utama kami diatur ke yang diperlukan`id`bidang yang kita definisikan sebelum ini. Perhatikan bahwa tabel baru ini kosong dan tidak berisi apa pun kecuali kunci partisi.

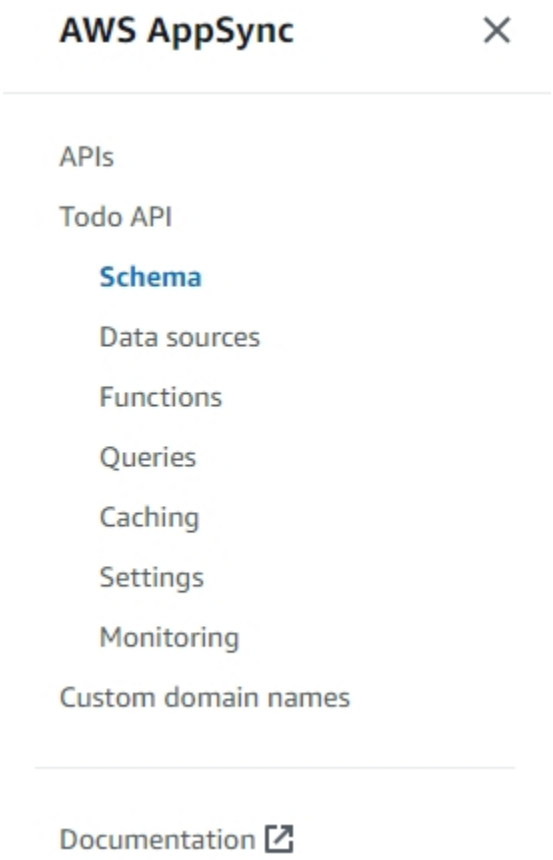
e. Pilih Selanjutnya.

6. Tinjau perubahan Anda dan pilihBuat API. Tunggu sebentar untuk membiarkanAWS AppSynclayanan selesai membuat API Anda.

Anda telah berhasil membuat API GraphQL dengan skema dan sumber data DynamoDB. Untuk meringkas langkah-langkah di atas, kami memilih untuk membuat API GraphQL yang benar-benar baru. Kami mendefinisikan nama API, lalu menambahkan definisi skema kami dengan menambahkan tipe pertama kami. Kami mendefinisikan jenis dan bidangnya, lalu memilih untuk melampirkan sumber data ke salah satu bidang dengan membuat tabel DynamoDB baru tanpa data di dalamnya.

Langkah 2: Ikuti tur konsol

Sebelum kita menambahkan data ke tabel DynamoDB kita, kita harus meninjau fitur dasarAWS AppSyncpengalaman konsol. TheAWS AppSynctab konsol di sisi kiri halaman memungkinkan pengguna untuk dengan mudah menavigasi ke salah satu komponen utama atau opsi konfigurasi yangAWS AppSyncmenyediakan:



Desainer skema

PilihSkemauntuk melihat skema yang baru saja Anda buat. Jika Anda meninjau konten skema, Anda akan melihat bahwa itu telah dimuat dengan banyak operasi pembantu untuk merampingkan proses pengembangan. DiSkemaeditor, jika Anda menggulir kode, Anda akhirnya akan mencapai model yang Anda tentukan di bagian sebelumnya:

```
type Todo {
 id: ID!
 name: String
 when: String
```

```
  where: String
  description: String
}
```

Model Anda menjadi tipe dasar yang digunakan di seluruh skema Anda. Kami akan mulai menambahkan data ke sumber data kami menggunakan mutasi yang secara otomatis dihasilkan dari jenis ini.

Berikut adalah beberapa tips dan fakta tambahan tentangSkemapenyunting:

1. Editor kode memiliki kemampuan linting dan pemeriksaan kesalahan yang dapat Anda gunakan saat menulis aplikasi Anda sendiri.
2. Sisi kanan konsol menunjukkan tipe GraphQL yang telah dibuat dan resolver pada berbagai jenis tingkat atas, seperti kueri.
3. Saat menambahkan tipe baru ke skema (misalnya,`type User {...}`), Anda dapat memilikiAWS AppSyncmenyediakan sumber daya DynamoDB untuk Anda. Ini termasuk kunci primer yang tepat, kunci sortir, dan desain indeks agar paling sesuai dengan pola akses data GraphQL Anda. Jika Anda memilihBuat Sumber Dayadi bagian atas dan pilih salah satu jenis yang ditentukan pengguna ini dari menu, Anda dapat memilih opsi bidang yang berbeda dalam desain skema. Kami akan membahas ini didesain skemabagian.

Konfigurasi penyelesai

Dalam perancang skema,Penyelesaibagian berisi semua jenis dan bidang dalam skema Anda. Jika Anda menggulir daftar bidang, Anda akan melihat bahwa Anda dapat melampirkan resolver ke bidang tertentu dengan memilihLampirkan. Ini akan membuka editor kode di mana Anda dapat menulis kode resolver Anda.AWS AppSyncmendukung VTL danJavaScriptruntime, yang dapat diubah di bagian atas halaman dengan memilihTindakan, makaPerbarui Runtime. Di bagian bawah halaman, Anda juga dapat membuat fungsi yang akan menjalankan beberapa operasi secara berurutan. Namun, resolver adalah topik lanjutan, dan kami tidak akan membahasnya di bagian ini.

Sumber data

PilihSumber datauntuk melihat tabel DynamoDB Anda. Dengan memilih`Resource`pilihan (jika tersedia), Anda dapat melihat konfigurasi sumber data Anda. Dalam contoh kita, ini mengarah ke konsol DynamoDB. Dari sana, Anda dapat mengedit data Anda. Anda juga dapat langsung mengedit beberapa data dengan memilih sumber data, lalu memilihSunting. Jika Anda perlu menghapus

sumber data Anda, Anda dapat memilih sumber data Anda, lalu pilihHapus. Terakhir, Anda dapat membuat sumber data baru dengan memilihBuat sumber data, kemudian mengkonfigurasi nama dan jenisnya. Perhatikan bahwa opsi ini untuk menautkanAWS AppSynclayanan ke sumber daya yang ada. Anda masih perlu membuat sumber daya di akun Anda menggunakan layanan yang relevan sebelumnyaAWS AppSyncmengenalinya.

Mengajukan Kueri

PilihPertanyaanuntuk melihat kueri dan mutasi Anda. Saat kami membuat API GraphQL kami menggunakan model kami,AWS AppSyncsecara otomatis menghasilkan beberapa mutasi dan kueri pembantu untuk tujuan pengujian. Di editor kueri, sisi kiri berisiPenjelajah. Ini adalah daftar yang menunjukkan semua mutasi dan kueri Anda. Anda dapat dengan mudah mengaktifkan operasi dan bidang yang ingin Anda gunakan di sini dengan mengklik nilai namanya. Ini akan menyebabkan kode muncul secara otomatis di bagian tengah editor. Di sini, Anda dapat mengedit mutasi dan kueri Anda dengan memodifikasi nilai. Di bagian bawah editor, Anda memilikiVariabel Kueriedit or yang memungkinkan Anda memasukkan nilai bidang untuk variabel input operasi Anda. MemilihJalankandi bagian atas editor akan memunculkan daftar drop-down untuk memilih kueri/mutasi yang akan dijalankan. Output untuk proses ini akan muncul di sisi kanan halaman. Kembali diPenjelajahbagian di bagian atas, Anda dapat memilih operasi (Query, Mutation, Subscription), lalu pilih+simbol untuk menambahkan contoh baru dari operasi tertentu. Di bagian atas halaman, akan ada daftar drop-down lain yang berisi mode otorisasi untuk kueri Anda berjalan. Namun, kami tidak akan membahas fitur itu di bagian ini (Untuk informasi lebih lanjut, lihatKeamanan.).

Pengaturan

PilihPengaturanuntuk melihat beberapa opsi konfigurasi untuk API GraphQL Anda. Di sini, Anda dapat mengaktifkan beberapa opsi seperti logging, tracing, dan fungsionalitas firewall aplikasi web. Anda juga dapat menambahkan mode otorisasi baru untuk melindungi data Anda dari kebocoran yang tidak diinginkan ke publik. Namun, opsi ini lebih maju dan tidak akan dibahas dalam bagian ini.

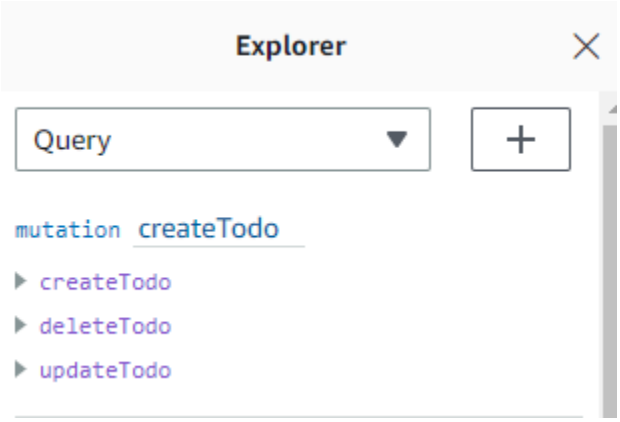
> **Note**
>
> Mode otorisasi default,`API_KEY`, menggunakan kunci API untuk menguji aplikasi. Ini adalah otorisasi dasar yang diberikan ke semua API GraphQL yang baru dibuat. Kami menyarankan Anda menggunakan metode yang berbeda untuk produksi. Demi contoh di bagian ini, kita hanya akan menggunakan kunci API. Untuk informasi selengkapnya tentang metode otorisasi yang didukung, lihatKeamanan.

Langkah 3: Tambahkan data dengan mutasi GraphQL

Langkah Anda selanjutnya adalah menambahkan data ke tabel DynamoDB kosong Anda menggunakan mutasi GraphQL. Mutasi adalah salah satu jenis operasi mendasar dalam GraphQL. Mereka didefinisikan dalam skema dan memungkinkan Anda untuk memanipulasi data dalam sumber data Anda. Dalam hal REST API, ini sangat mirip dengan operasi sepertiPUTatauPOST.

Untuk menambahkan data ke sumber data Anda

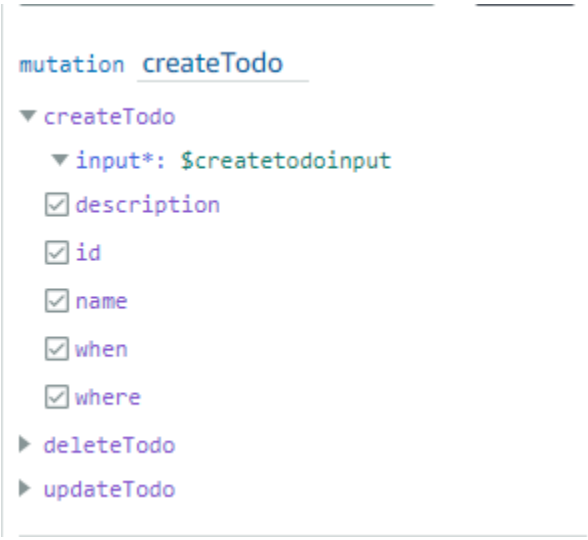
1. Jika Anda belum melakukannya, masuk keAWS Management Consoledan bukaAppSynckonsol.
2. Pilih API Anda dari tabel.
3. Di tab di sebelah kiri, pilihPertanyaan.
4. DiPenjelajahtab di sebelah kiri tabel, Anda mungkin melihat beberapa mutasi dan kueri yang sudah ditentukan dalam editor kueri:



> **Note**
>
> Mutasi ini sebenarnya duduk dalam skema Anda sebagaiMutationjenis. Ini memiliki kode:
>
> ```
> type Mutation {
>  createTodo(input: CreateTodoInput!): Todo
>  updateTodo(input: UpdateTodoInput!): Todo
>  deleteTodo(input: DeleteTodoInput!): Todo
> }
> ```
>
> Seperti yang Anda lihat, operasi di sini mirip dengan apa yang ada di dalam editor kueri.

AWS AppSyncsecara otomatis menghasilkan ini dari model yang kita definisikan sebelumnya. Contoh ini akan menggunakan`createTodo`omutasi untuk menambahkan entri ke kami*`TodoApitable`*meja.

5. Pilih`createTodo`ooperasi dengan memperluasnya di bawah`createTodo`omutasi:



Aktifkan kotak centang untuk semua bidang seperti gambar di atas.

> **Note**
>
> Atribut yang Anda lihat di sini adalah elemen mutasi yang dapat dimodifikasi yang berbeda. Anda`input`dapat dianggap sebagai parameter dari`createTodo`. Berbagai opsi dengan kotak centang adalah bidang yang akan dikembalikan dalam respons setelah operasi dilakukan.

6. Di editor kode di tengah layar, Anda akan melihat bahwa operasi muncul di bawah`createTodo`omutasi:

```
mutation createTodo($createtodoinput: CreateTodoInput!) {
  createTodo(input: $createtodoinput) {
    where
    when
    name
    id
    description
  }
```

```
}
```

> **Note**
>
> Untuk menjelaskan cuplikan ini dengan benar, kita juga harus melihat kode skema. Deklarasi`mutation createTodo($createtodoinput: CreateTodoInput!) {}`adalah mutasi dengan salah satu operasinya,`createTodo`. Mutasi penuh terletak di skema:
>
> ```
> type Mutation {
>  createTodo(input: CreateTodoInput!): Todo
>  updateTodo(input: UpdateTodoInput!): Todo
>  deleteTodo(input: DeleteTodoInput!): Todo
> }
> ```
>
> Kembali ke deklarasi mutasi dari editor, parameternya adalah objek yang disebut`$createtodoinput`dengan jenis input yang diperlukan dari`CreateTodoInput`. Perhatikan bahwa`CreateTodoInput`(dan semua input dalam mutasi) juga didefinisikan dalam skema. Misalnya, inilah kode boilerplate untuk`CreateTodoInput`:
>
> ```
> input CreateTodoInput {
>  name: String
>  when: String
>  where: String
>  description: String
> }
> ```
>
> Ini berisi bidang yang kami definisikan dalam model kami, yaitu`name`,`when`,`where`, dan`description`.
> Kembali ke kode editor, di`createTodo(input: $createtodoinput) {}`, kami mendeklarasikan masukan sebagai`$createtodoinput`, yang juga digunakan dalam deklarasi mutasi. Kami melakukan ini karena ini memungkinkan GraphQL untuk memvalidasi input kami terhadap jenis yang disediakan dan memastikan bahwa mereka digunakan dengan input yang benar.
> Bagian terakhir dari kode editor menunjukkan bidang yang akan dikembalikan dalam respons setelah operasi dilakukan:
>
> ```
> {
>     where
> ```

```
    when
    name
    id
    description
  }
```

DiVariabel kueritab di bawah editor ini, akan ada generik`createtodoinput`objek yang mungkin memiliki data berikut:

```
{
  "createtodoinput": {
    "name": "Hello, world!",
    "when": "Hello, world!",
    "where": "Hello, world!",
    "description": "Hello, world!"
  }
}
```

> **Note**
>
> Di sinilah kami mengalokasikan nilai untuk input yang disebutkan sebelumnya:
>
> ```
> input CreateTodoInput {
>  name: String
>  when: String
>  where: String
>  description: String
> }
> ```

Ubah`createtodoinput`dengan menambahkan informasi yang ingin kita masukkan ke dalam tabel DynamoDB kita. Dalam hal ini, kami ingin membuat beberapa`Todo`item sebagai pengingat:

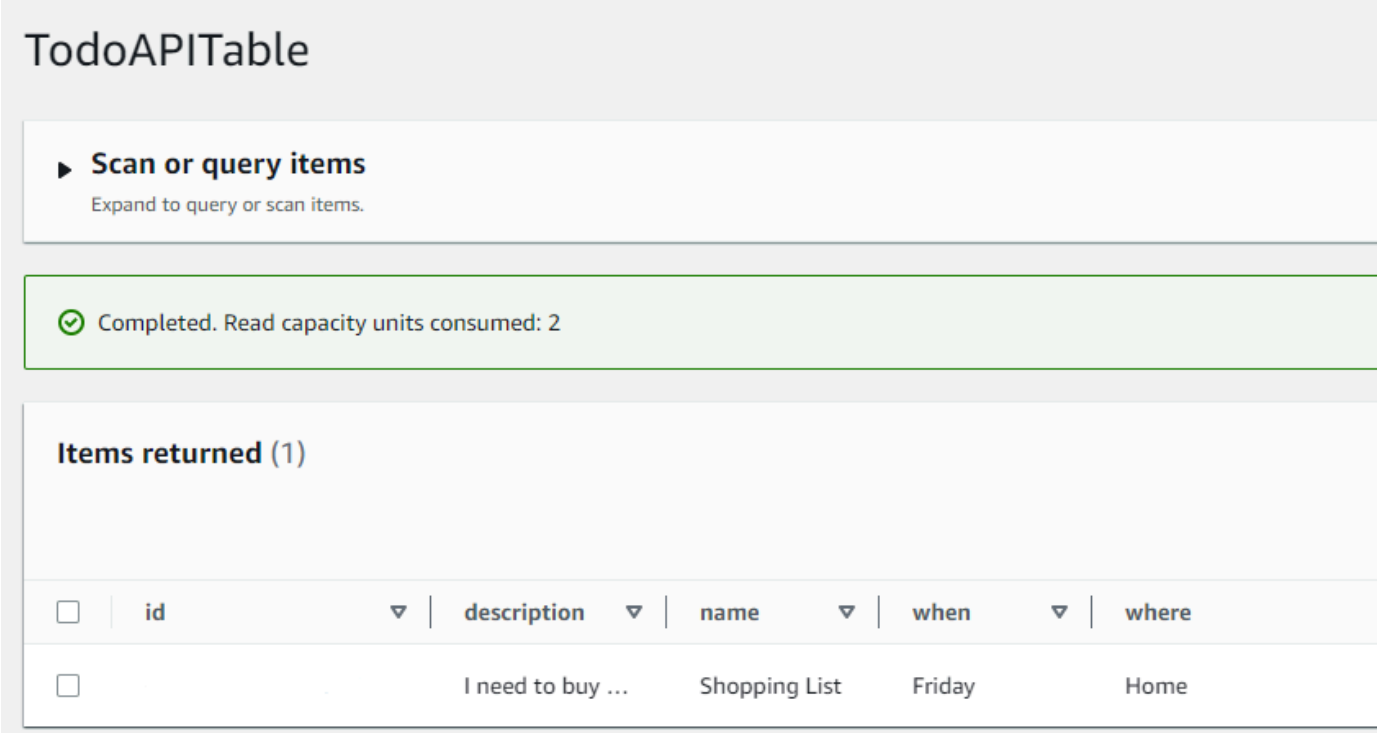
```
{
  "createtodoinput": {
    "name": "Shopping List",
    "when": "Friday",
    "where": "Home",
```

```
    "description": "I need to buy eggs"
  }
}
```

7. PilihJalankandi bagian atas editor. PilihCreateTododalam daftar drop-down. Di sisi kanan editor, Anda akan melihat responsnya. Mungkin terlihat seperti ini:

```
{
  "data": {
    "createTodo": {
      "where": "Home",
      "when": "Friday",
      "name": "Shopping List",
      "id": "abcdefgh-1234-1234-1234-abcdefghijkl",
      "description": "I need to buy eggs"
    }
  }
}
```

Jika Anda menavigasi ke layanan DynamoDB, Anda sekarang akan melihat entri di sumber data Anda dengan informasi ini:



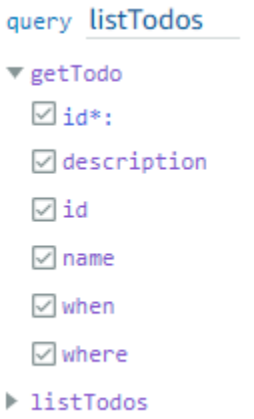
Untuk meringkas operasi, mesin GraphQL mengurai catatan, dan resolver memasukkannya ke dalam tabel Amazon DynamoDB Anda. Sekali lagi, Anda dapat memverifikasi ini di konsol DynamoDB. Perhatikan bahwa Anda tidak perlu melewati`id`nilai. Sebuah`id`dihasilkan dan dikembalikan dalam hasil. Hal ini karena contoh menggunakan`autoId()`berfungsi dalam resolver GraphQL untuk kunci partisi yang ditetapkan pada sumber daya DynamoDB Anda. Kami akan membahas bagaimana Anda dapat membangun resolver di bagian yang berbeda. Catat yang dikembalikan`id`nilai; Anda akan menggunakannya di bagian berikutnya untuk mengambil data dengan kueri GraphQL.

Langkah 4: Ambil data dengan kueri GraphQL

Sekarang catatan ada di database Anda, Anda akan mendapatkan hasil ketika Anda menjalankan kueri. Query adalah salah satu operasi fundamental lainnya dari GraphQL. Ini digunakan untuk mengurai dan mengambil informasi dari sumber data Anda. Dalam hal REST API, ini mirip dengan`GET`operasi. Keuntungan utama dari kueri GraphQL adalah kemampuan untuk menentukan persyaratan data yang tepat aplikasi Anda sehingga Anda mengambil data yang relevan pada waktu yang tepat.

Untuk menanyakan sumber data Anda

1. Jika Anda belum melakukannya, masuk keAWS Management Consoledan bukaAppSynckonsol.
2. Pilih API Anda dari tabel.
3. Di tab di sebelah kiri, pilihPertanyaan.
4. DiPenjelajahtab di sebelah kiri tabel, di bawah`query listTodos`, perluas`getTodo`operasi:



5. Di editor kode, Anda akan melihat kode operasi:

```
query listTodos {
  getTodo(id: "") {
```

```
    description
    id
    name
    when
    where
  }
```

Di`(id:"")`, isi nilai yang Anda simpan dalam hasil dari operasi mutasi. Dalam contoh kami, ini akan menjadi:

```
query listTodos {
  getTodo(id: "abcdefgh-1234-1234-1234-abcdefghijkl") {
    description
    id
    name
    when
    where
  }
```

6. PilihJalankan, makaListTodos. Hasilnya akan muncul di sebelah kanan editor. Contoh kami terlihat seperti ini:

```
{
  "data": {
    "getTodo": {
      "description": "I need to buy eggs",
      "id": "abcdefgh-1234-1234-1234-abcdefghijkl",
      "name": "Shopping List",
      "when": "Friday",
      "where": "Home"
    }
  }
}
```

> **Note**
>
> Kueri hanya mengembalikan bidang yang Anda tentukan. Anda dapat membatalkan pilihan bidang yang tidak Anda perlukan dengan menghapusnya dari bidang pengembalian:
>
> ```
> {
> ```

```
    description
    id
    name
    when
    where
  }
```

Anda juga dapat menghapus centang pada kotak diPenjelajahtab di sebelah bidang yang ingin Anda hapus.

7. Anda juga dapat mencoba`listTodos`operasi dengan mengulangi langkah-langkah untuk membuat entri di sumber data Anda, kemudian mengulangi langkah-langkah kueri dengan`listTodos`operasi. Berikut adalah contoh di mana kami menambahkan tugas kedua:

```
{
  "createtodoinput": {
    "name": "Second Task",
    "when": "Monday",
    "where": "Home",
    "description": "I need to mow the lawn"
  }
}
```

Dengan memanggil`listTodos`operasi, itu mengembalikan entri lama dan baru:

```
{
  "data": {
    "listTodos": {
      "items": [
        {
          "id": "abcdefgh-1234-1234-1234-abcdefghijkl",
          "name": "Shopping List",
          "when": "Friday",
          "where": "Home",
          "description": "I need to buy eggs"
        },
        {
          "id": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
          "name": "Second Task",
          "when": "Monday",
          "where": "Home",
          "description": "I need to mow the lawn"
```

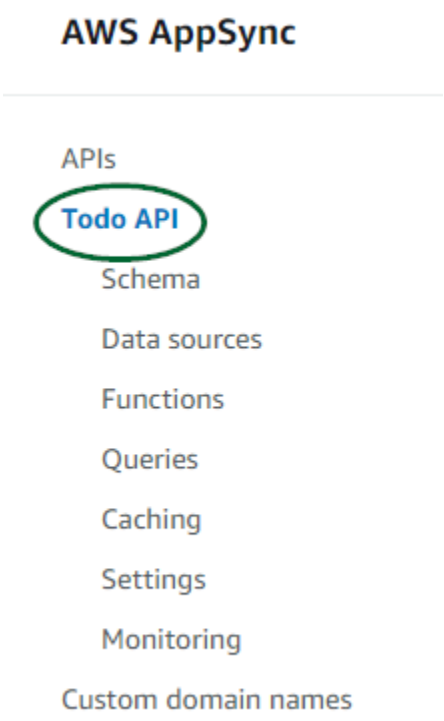
```
        }
      ]
    }
  }
}
```

Bagian tambahan

Bagian-bagian ini adalah referensi untuk yang lebih majuAWS AppSynctopik. Kami merekomendasikan mengikutiBacaan tambahanbagian sebelum melakukan hal lain.

Integrasi

Di tab konsol, jika Anda memilih nama API Anda,Integrasihalaman muncul:



Ini merangkum langkah-langkah untuk menyiapkan API Anda dan menguraikan langkah selanjutnya untuk membangun aplikasi klien. TheIntegrasikan dengan aplikasi Andabagian memberikan rincian untuk menggunakanAWSMemperkuat rantai alatuntuk mengotomatiskan proses menghubungkan API Anda dengan iOS, Android, danJavaScriptaplikasi melalui konfigurasi dan pembuatan kode. Toolchain Amplify menyediakan dukungan penuh untuk membangun proyek dari workstation lokal Anda termasuk penyediaan GraphQL dan alur kerja untuk CI/CD.

TheSampel Klienbagian juga mencantumkan contoh aplikasi klien (mis.,JavaScript, iOS, Android) untuk menguji pengalaman end-to-end. Anda dapat mengkloning dan mengunduh sampel ini, dan file konfigurasi memiliki informasi yang diperlukan (seperti URL titik akhir Anda) yang Anda butuhkan untuk memulai. Ikuti instruksi padaAWS Amplifyrantai alathalaman untuk menjalankan aplikasi Anda.

Bacaan tambahan

- Merancang GraphQL API- Ini adalah panduan komprehensif untuk membuat GraphQL Anda menggunakan skema kosong tanpa sumber data atau resolver.

Merancang GraphQL API

AWS AppSyncmemungkinkan Anda membuat GraphQL API menggunakan pengalaman konsol. Anda melihat sekilas ini di bagian Launching a sample schema. Namun, panduan itu tidak menunjukkan seluruh katalog opsi dan konfigurasi yang dapat Anda manfaatkan. AWS AppSync

Saat Anda memilih untuk membuat GraphQL API di konsol, ada beberapa opsi untuk dijelajahi. Jika Anda mengikuti panduan Peluncuran skema sampel kami, kami menunjukkan cara membuat API dari model yang telah ditentukan sebelumnya. Di bagian berikut, kami akan memandu Anda melalui sisa opsi dan konfigurasi untuk membuat GraphQL API di. AWS AppSync

Di bagian ini, Anda akan meninjau konsep-konsep berikut:

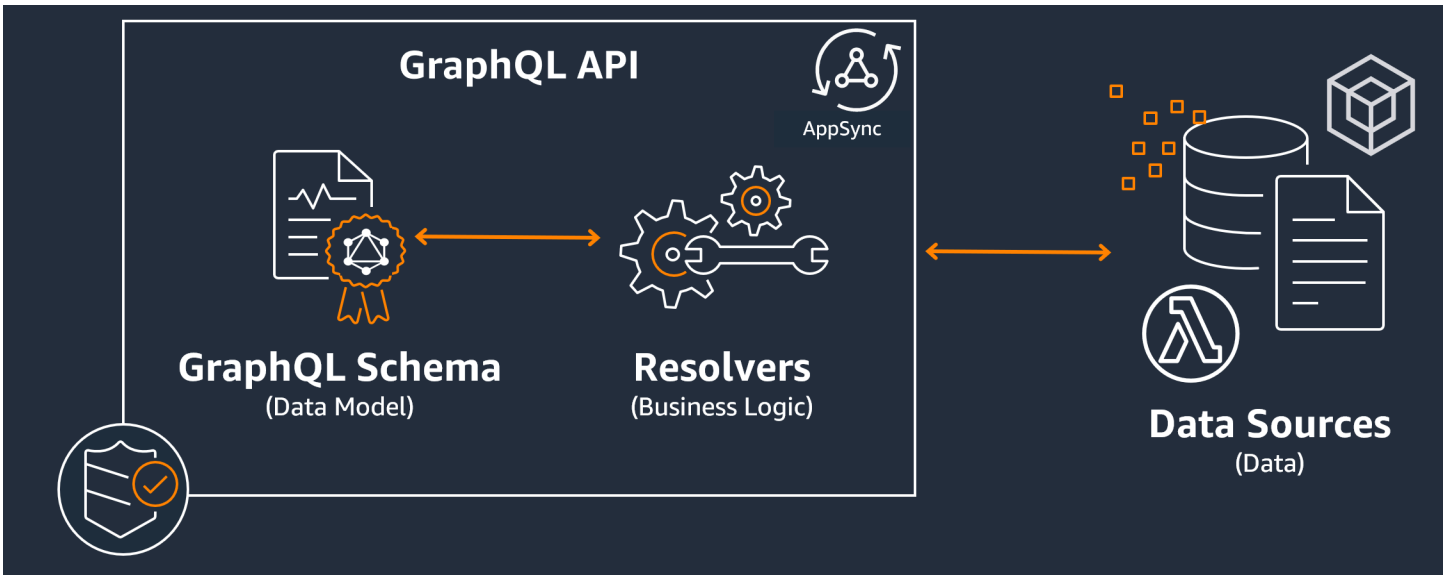
1. Blank APIs or imports: Panduan ini akan berjalan melalui seluruh proses pembuatan untuk membuat GraphQL API. Anda akan belajar cara membuat GraphQL dari template kosong tanpa model, mengonfigurasi sumber data untuk skema Anda, dan menambahkan resolver pertama Anda ke bidang.
2. Real-time data: Panduan ini akan menunjukkan kepada Anda opsi potensial untuk membuat API menggunakan AWS AppSync WebSocket mesin.
3. Merged APIs: Panduan ini akan menunjukkan cara membuat API GraphQL baru dengan mengaitkan dan menggabungkan data dari beberapa API GraphQL yang ada.
4. the section called “Introspeksi RDS”: Panduan ini akan menunjukkan cara mengintegrasikan tabel Amazon RDS Anda menggunakan API Data.

Menata API GraphQL (API kosong atau impor)

Sebelum Anda membuat API GraphQL Anda dari template kosong, akan membantu untuk meninjau konsep seputar GraphQL. Ada tiga komponen dasar dari API GraphQL:

1. Theskemaadalah file yang berisi bentuk dan definisi data Anda. Ketika permintaan dibuat oleh klien ke layanan GraphQL Anda, data yang dikembalikan akan mengikuti spesifikasi skema. Untuk informasi selengkapnya, lihat Skema.
2. Thesumber datamelekat pada skema Anda. Ketika permintaan dibuat, di sinilah data diambil dan dimodifikasi. Untuk informasi selengkapnya, lihat Data sources.

3. Thepenyelesaiberada di antara skema dan sumber data. Ketika permintaan dibuat, resolver melakukan operasi pada data dari sumber, kemudian mengembalikan hasilnya sebagai respons. Untuk informasi selengkapnya, lihat Resolvers.



AWS AppSyncmengelola API Anda dengan memungkinkan Anda membuat, mengedit, dan menyimpan kode untuk skema dan resolver Anda. Sumber data Anda akan berasal dari repositori eksternal seperti database, tabel DynamoDB, dan fungsi Lambda. Jika Anda menggunakanAWSlayanan untuk menyimpan data Anda atau berencana melakukannya,AWS AppSyncmemberikan pengalaman yang hampir mulus saat mengaitkan data dari AndaAWSakun ke API GraphQL Anda.

Di bagian selanjutnya, Anda akan belajar cara membuat masing-masing komponen ini menggunakanAWS AppSynclayanan.

Topik

- Langkah 1: Merancang skema Anda
- Langkah 2: Melampirkan sumber data
- Langkah 3: Mengkonfigurasi resolver
- Langkah 4: Menggunakan API: contoh CDK

Langkah 1: Merancang skema Anda

Skema GraphQL adalah dasar dari setiap implementasi server GraphQL. Setiap API GraphQL didefinisikan oleh abujangskema yang berisi jenis dan bidang yang menjelaskan bagaimana data dari permintaan akan diisi. Data yang mengalir melalui API Anda dan operasi yang dilakukan harus divalidasi terhadap skema.

Secara umum,Sistem tipe GraphQLmenjelaskan kemampuan server GraphQL dan digunakan untuk menentukan apakah kueri valid. Sistem tipe server sering disebut sebagai skema server dan dapat terdiri dari berbagai jenis objek, jenis skalar, tipe input, dan banyak lagi. GraphQL bersifat deklaratif dan diketik dengan kuat, artinya tipe akan didefinisikan dengan baik saat runtime dan hanya akan mengembalikan apa yang ditentukan.

AWS AppSyncmemungkinkan Anda untuk menentukan dan mengkonfigurasi skema GraphQL. Bagian berikut menjelaskan cara membuat skema GraphQL dari awal menggunakanAWS AppSynclayanan.

Penataan Skema GraphQL

> **Tip**
>
> Kami merekomendasikan untuk meninjauSkemabagian sebelum melanjutkan.

GraphQL adalah alat yang ampuh untuk mengimplementasikan layanan API. MenurutSitus web GraphQL, GraphQL adalah sebagai berikut:

“GraphQL adalah bahasa kueri untuk API dan runtime untuk memenuhi kueri tersebut dengan data Anda yang ada. GraphQL memberikan deskripsi data yang lengkap dan mudah dipahami di API Anda, memberi klien kekuatan untuk meminta apa yang mereka butuhkan dan tidak lebih, membuatnya lebih mudah untuk mengembangkan API dari waktu ke waktu, dan memungkinkan alat pengembang yang kuat.“

Bagian ini mencakup bagian pertama dari implementasi GraphQL Anda, skema. Menggunakan kutipan di atas, skema memainkan peran “memberikan deskripsi data yang lengkap dan dapat dimengerti di API Anda”. Dengan kata lain, skema GraphQL adalah representasi tekstual dari data, operasi, dan hubungan layanan Anda di antara mereka. Skema ini dianggap sebagai titik masuk utama untuk implementasi layanan GraphQL Anda. Tidak mengherankan, ini sering menjadi

salah satu hal pertama yang Anda buat dalam proyek Anda. Kami merekomendasikan untuk meninjauSkemabagian sebelum melanjutkan.

Untuk mengutipSkemabagian, skema GraphQL ditulis dalamSkema Definisi Bahasa(SDL). SDL terdiri dari jenis dan bidang dengan struktur yang mapan:

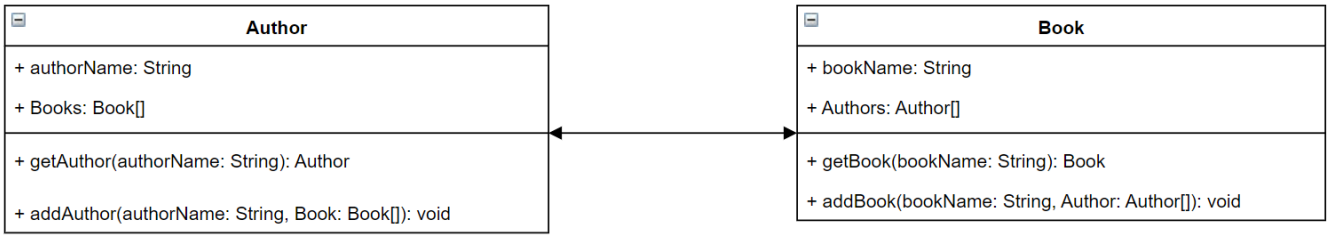
- JenisJenis adalah bagaimana GraphQL mendefinisikan bentuk dan perilaku data. GraphQL mendukung banyak jenis yang akan dijelaskan nanti di bagian ini. Setiap jenis yang didefinisikan dalam skema Anda akan berisi cakupannya sendiri. Di dalam lingkup akan ada satu atau lebih bidang yang dapat berisi nilai atau logika yang akan digunakan dalam layanan GraphQL Anda. Jenis mengisi banyak peran yang berbeda, yang paling umum adalah objek atau skalar (tipe nilai primitif).
- Bidang: Bidang ada dalam lingkup tipe dan menyimpan nilai yang diminta dari layanan GraphQL. Ini sangat mirip dengan variabel dalam bahasa pemrograman lain. Bentuk data yang Anda tentukan di bidang Anda akan menentukan bagaimana data terstruktur dalam operasi permintaan/respons. Hal ini memungkinkan pengembang untuk memprediksi apa yang akan dikembalikan tanpa mengetahui bagaimana backend layanan diimplementasikan.

Skema paling sederhana akan berisi tiga kategori data yang berbeda:

1. Akar skema: Roots menentukan titik masuk skema Anda. Ini menunjuk ke bidang yang akan melakukan beberapa operasi pada data seperti menambahkan, menghapus, atau memodifikasi sesuatu.
2. Jenis: Ini adalah tipe dasar yang digunakan untuk mewakili bentuk data. Anda hampir dapat menganggap ini sebagai objek atau representasi abstrak dari sesuatu dengan karakteristik yang ditentukan. Misalnya, Anda bisa membuat`Person`objek yang mewakili seseorang dalam database. Karakteristik setiap orang akan didefinisikan di dalam`Person`sebagai ladang. Mereka bisa berupa apa saja seperti nama orang tersebut, usia, pekerjaan, alamat, dll.
3. Jenis objek khusus: Ini adalah jenis yang menentukan perilaku operasi dalam skema Anda. Setiap jenis objek khusus didefinisikan sekali per skema. Mereka pertama kali ditempatkan di akar skema, kemudian didefinisikan dalam badan skema. Setiap bidang dalam jenis objek khusus mendefinisikan operasi tunggal yang akan diimplementasikan oleh resolver Anda.

Untuk menempatkan ini ke dalam perspektif, bayangkan Anda membuat layanan yang menyimpan penulis dan buku-buku yang telah mereka tulis. Setiap penulis memiliki nama dan serangkaian buku yang telah mereka tulis. Setiap buku memiliki nama dan daftar penulis terkait. Kami juga ingin

kemampuan untuk menambah atau mengambil buku dan penulis. Representasi UFL sederhana dari hubungan ini mungkin terlihat seperti ini:



Di GraphQL, entitas`Author`dan`Book`mewakili dua jenis objek yang berbeda dalam skema Anda:

```
type Author {
}

type Book {
}
```

`Author`mengandung`authorName`dan`Books`, sementara`Book`mengandung`bookName`dan`Authors`. Ini dapat direpresentasikan sebagai bidang dalam lingkup tipe Anda:

```
type Author {
  authorName: String
  Books: [Book]
}

type Book {
  bookName: String
  Authors: [Author]
}
```

Seperti yang Anda lihat, representasi tipe sangat dekat dengan diagram. Namun, metodenya adalah di mana ia menjadi sedikit lebih rumit. Ini akan ditempatkan di salah satu dari beberapa jenis objek khusus sebagai bidang. Kategorisasi objek khusus mereka tergantung pada perilaku mereka. GraphQL berisi tiga jenis objek khusus mendasar: kueri, mutasi, dan langganan. Untuk informasi lebih lanjut, lihatBenda khusus.

Karena`getAuthor`dan`getBook`keduanya meminta data, mereka akan ditempatkan di`Query`jenis objek khusus:

```
type Author {
  authorName: String
  Books: [Book]
}

type Book {
  bookName: String
  Authors: [Author]
}

type Query {
  getAuthor(authorName: String): Author
  getBook(bookName: String): Book
}
```

Operasi ditautkan ke kueri, yang itu sendiri terkait dengan skema. Menambahkan root skema akan menentukan jenis objek khusus (`Query`dalam hal ini) sebagai salah satu titik masuk Anda. Hal ini dapat dilakukan dengan menggunakan`schema`kata kunci:

```
schema {
  query: Query
}

type Author {
  authorName: String
  Books: [Book]
}

type Book {
  bookName: String
  Authors: [Author]
}

type Query {
  getAuthor(authorName: String): Author
  getBook(bookName: String): Book
}
```

Melihat dua metode terakhir,`addAuthor`dan`addBook`menambahkan data ke database Anda, sehingga mereka akan didefinisikan dalam`Mutation`jenis objek khusus. Namun, dariJenishalaman, kita juga tahu bahwa input yang secara langsung mereferensikan Objek

tidak diperbolehkan karena mereka benar-benar tipe keluaran. Dalam hal ini, kita tidak dapat menggunakan`Author`atau`Book`, jadi kita perlu membuat tipe input dengan bidang yang sama. Dalam contoh ini, kami menambahkan`AuthorInput`dan`BookInput`, keduanya menerima bidang yang sama dari jenisnya masing-masing. Kemudian, kami membuat mutasi kami menggunakan input sebagai parameter kami:

```
schema {
  query: Query
  mutation: Mutation
}

type Author {
  authorName: String
  Books: [Book]
}

input AuthorInput {
  authorName: String
  Books: [BookInput]
}

type Book {
  bookName: String
  Authors: [Author]
}

input BookInput {
  bookName: String
  Authors: [AuthorInput]
}

type Query {
  getAuthor(authorName: String): Author
  getBook(bookName: String): Book
}

type Mutation {
  addAuthor(input: [BookInput]): Author
  addBook(input: [AuthorInput]): Book
}
```

Mari kita tinjau apa yang baru saja kita lakukan:

1. Kami menciptakan skema dengan`Book`dan`Author`jenis untuk mewakili entitas kami.
2. Kami menambahkan bidang yang berisi karakteristik entitas kami.
3. Kami menambahkan query untuk mengambil informasi ini dari database.
4. Kami menambahkan mutasi untuk memanipulasi data dalam database.
5. Kami menambahkan tipe input untuk mengganti parameter objek kami dalam mutasi untuk mematuhi aturan GraphQL.
6. Kami menambahkan kueri dan mutasi ke skema root kami sehingga implementasi GraphQL memahami lokasi tipe root.

Seperti yang Anda lihat, proses pembuatan skema membutuhkan banyak konsep dari pemodelan data (terutama pemodelan basis data) secara umum. Anda dapat menganggap skema sesuai dengan bentuk data dari sumbernya. Ini juga berfungsi sebagai model yang akan diterapkan oleh resolver. Di bagian berikut, Anda akan belajar cara membuat skema menggunakan berbagaiAWSalat dan layanan yang didukung.

> **Note**
>
> Contoh di bagian berikut tidak dimaksudkan untuk berjalan dalam aplikasi nyata. Mereka hanya ada untuk menampilkan perintah sehingga Anda dapat membangun aplikasi Anda sendiri.

Membuat skema

Skema Anda akan berada dalam file bernama`schema.graphql`.AWS AppSyncmemungkinkan pengguna untuk membuat skema baru untuk API GraphQL mereka menggunakan berbagai metode. Dalam contoh ini, kita akan membuat API kosong bersama dengan skema kosong.

Console

1. Masuk keAWS Management Consoledan bukaAppSynckonsol.
   a. DiDasbor, pilihBuat API.
   b. Di bawahOpsi API, pilihAPI GraphQL,Desain dari awal, makaBerikutnya.
      i. UntukNama API, ubah nama yang telah diisi sebelumnya menjadi apa yang dibutuhkan aplikasi Anda.

ii. Untukrincian kontak, Anda dapat memasukkan titik kontak untuk mengidentifikasi manajer untuk API. Ini adalah bidang opsional.

iii. Di bawahKonfigurasi API pribadi, Anda dapat mengaktifkan fitur API pribadi. API pribadi hanya dapat diakses dari titik akhir VPC (VPCE) yang dikonfigurasi. Untuk informasi lebih lanjut, lihatAPI pribadi.

Kami tidak menyarankan mengaktifkan fitur ini untuk contoh ini. PilihBerikutnyasetelah meninjau input Anda.

c. Di bawahBuat tipe GraphQL, Anda dapat memilih untuk membuat tabel DynamoDB untuk digunakan sebagai sumber data atau melewatkan ini dan melakukannya nanti.

Untuk contoh ini, pilihBuat sumber daya GraphQL nanti. Kami akan membuat sumber daya di bagian terpisah.

d. Tinjau input Anda, lalu pilihBuat API.

2. Anda akan berada di dasbor API spesifik Anda. Anda bisa tahu karena nama API akan berada di bagian atas dasbor. Jika ini tidak terjadi, Anda dapat memilihAPIdiSidebar, lalu pilih API Anda diDasbor API.

- DiSidebardi bawah nama API Anda, pilihSkema.

3. DiEditor skema, Anda dapat mengonfigurasi`schema.graphql`berkas. Mungkin kosong atau diisi dengan tipe yang dihasilkan dari model. Di sebelah kanan, Anda memilikiPenyelesaibagian untuk melampirkan resolver ke bidang skema Anda. Kami tidak akan melihat resolver di bagian ini.

CLI

> **Note**
>
> Saat menggunakan CLI, pastikan Anda memiliki izin yang benar untuk mengakses dan membuat sumber daya dalam layanan. Anda mungkin ingin mengaturhak istimewa paling sedikitkebijakan untuk pengguna non-admin yang perlu mengakses layanan. Untuk informasi lebih lanjut tentangAWS AppSynckebijakan, lihatManajemen identitas dan akses untukAWS AppSync.
> Selain itu, kami sarankan membaca versi konsol terlebih dahulu jika Anda belum melakukannya.

1. Jika Anda belum melakukannya,menginstalsangAWSCLI, lalu tambahkankonfigurasi.
2. Buat objek API GraphQL dengan menjalankan`create-graphql-api`perintah.

   Anda harus mengetikkan dua parameter untuk perintah khusus ini:

   1. The`name`dari API Anda.
   2. The`authentication-type`, atau jenis kredensyal yang digunakan untuk mengakses API (IAM, OIDC, dll.).

   > **Note**
   >
   > Parameter lain seperti`Region`harus dikonfigurasi tetapi biasanya akan default ke nilai konfigurasi CLI Anda.

   Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-graphql-api --name testAPI123 --authentication-type API_KEY
```

   Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "graphqlApi": {
        "xrayEnabled": false,
        "name": "testAPI123",
        "authenticationType": "API_KEY",
        "tags": {},
        "apiId": "abcdefghijklmnopqrstuvwxyz",
        "uris": {
            "GRAPHQL": "https://zyxwvutsrqponmlkjihgfedcba.appsync-api.us-
west-2.amazonaws.com/graphql",
            "REALTIME": "wss://zyxwvutsrqponmlkjihgfedcba.appsync-realtime-
api.us-west-2.amazonaws.com/graphql"
        },
        "arn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz"
    }
}
```

3.

> **Note**
>
> Ini adalah perintah opsional yang mengambil skema yang ada dan mengunggahnya keAWS AppSynclayanan menggunakan gumpalan basis-64. Kami tidak akan menggunakan perintah ini demi contoh ini.

Jalankan perintah `start-schema-creation`.

Anda harus mengetikkan dua parameter untuk perintah khusus ini:

1. Anda`api-id`dari langkah sebelumnya.
2. Skema`definition`adalah gumpalan biner yang dikodekan basis-64.

Contoh perintah mungkin terlihat seperti ini:

```
 aws appsync start-schema-creation --api-id abcdefghijklmnopqrstuvwxyz --
definition "aa1111aa-123b-2bb2-c321-12hgg76cc33v"
```

Output akan dikembalikan:

```
{
    "status": "PROCESSING"
}
```

Perintah ini tidak akan mengembalikan output akhir setelah diproses. Anda harus menggunakan perintah terpisah,`get-schema-creation-status`untuk melihat hasilnya. Perhatikan bahwa kedua perintah ini asinkron, sehingga Anda dapat memeriksa status output bahkan saat skema masih dibuat.

CDK

> **Tip**
>
> Sebelum Anda menggunakan CDK, kami sarankan untuk meninjau CDKdokumentasi resmibersama denganAWS AppSynciniReferensi CDK.

Langkah-langkah yang tercantum di bawah ini hanya akan menampilkan contoh umum dari cuplikan yang digunakan untuk menambahkan sumber daya tertentu. Ini adalahtidakdimaksudkan untuk menjadi solusi yang berfungsi dalam kode produksi Anda. Kami juga menganggap Anda sudah memiliki aplikasi yang berfungsi.

1. Titik awal untuk CDK sedikit berbeda. Idealnya, Anda`schema.graphql`File seharusnya sudah dibuat. Anda hanya perlu membuat file baru dengan`.graphql`ekstensi file. Ini bisa menjadi file kosong.
2. Secara umum, Anda mungkin harus menambahkan direktif impor ke layanan yang Anda gunakan. Misalnya, mungkin mengikuti formulir:

```
import * as x from 'x'; # import wildcard as the 'x' keyword from 'x-service'
import {a, b, ...} from 'c'; # import {specific constructs} from 'c-service'
```

Untuk menambahkan API GraphQL, file stack Anda perlu mengimporAWS AppSynclayanan:

```
import * as appsync from 'aws-cdk-lib/aws-appsync';
```

> **Note**
>
> Ini berarti kami mengimpor seluruh layanan di bawah`appsync`kata kunci. Untuk menggunakan ini di aplikasi Anda,AWS AppSynckonstruksi akan menggunakan format`appsync.construct_name`. Misalnya, jika kita ingin membuat API GraphQL, kita akan mengatakan`new appsync.GraphqlApi(args_go_here)`. Langkah berikut menggambarkan ini.

3. API GraphQL paling dasar akan menyertakan a`name`untuk API dan`schema`jalan.

```
const add_api = new appsync.GraphqlApi(this, 'API_ID', {
  name: 'name_of_API_in_console',
  schema: appsync.SchemaFile.fromAsset(path.join(__dirname,
 'schema_name.graphql')),
});
```

> **Note**
>
> Mari kita tinjau apa yang dilakukan cuplikan ini. Di dalam lingkup`api`, kami membuat API GraphQL baru dengan meneleponappsync.GraphqlApi(scope: Construct, id: string, props: GraphqlApiProps). Ruang lingkup adalah`this`, yang mengacu pada objek saat ini. Id adalah*API_ID*, yang akan menjadi nama sumber daya API GraphQL Anda diAWS CloudFormationketika itu dibuat. The`GraphqlApiProps`berisinamedari API GraphQL Anda dan`schema`. The`schema`akan menghasilkan skema (`SchemaFile.fromAsset`) dengan mencari jalur absolut (`__dirname`) untuk`.graphql`berkas (*schema_name.graphql*). Dalam skenario nyata, file skema Anda mungkin akan berada di dalam aplikasi CDK. Untuk menggunakan perubahan yang dibuat pada API GraphQL Anda, Anda harus menerapkan ulang aplikasi.

Menambahkan tipe ke skema

Sekarang Anda telah menambahkan skema Anda, Anda dapat mulai menambahkan jenis input dan output Anda. Perhatikan bahwa tipe di sini tidak boleh digunakan dalam kode nyata; mereka hanya contoh untuk membantu Anda memahami prosesnya.

Pertama, kita akan membuat tipe objek. Dalam kode nyata, Anda tidak harus memulai dengan jenis ini. Anda dapat membuat jenis apa pun yang Anda inginkan kapan saja selama Anda mengikuti aturan dan sintaks GraphQL.

> **Note**
>
> Beberapa bagian berikutnya akan menggunakaneditor skema, jadi biarkan ini tetap terbuka.

Console

- Anda dapat membuat tipe objek menggunakan`type`kata kunci bersama dengan nama tipe:

  ```
  type Type_Name_Goes_Here {}
  ```

  Di dalam cakupan tipe, Anda dapat menambahkan bidang yang mewakili karakteristik objek:

```
type Type_Name_Goes_Here {
  # Add fields here
}
```

Inilah contohnya:

```
type Obj_Type_1 {
  id: ID!
  title: String
  date: AWSDateTime
}
```

> **Note**
>
> Pada langkah ini, kami menambahkan tipe objek generik dengan kebutuhan`id`bidang disimpan sebagai`ID`, sebuah`title`bidang disimpan sebagai`String`, dan`date`bidang disimpan sebagai`AWSDateTime`. Untuk melihat daftar jenis dan bidang dan apa yang mereka lakukan, lihatSkema. Untuk melihat daftar skalar dan apa yang mereka lakukan, lihatJenis referensi.

CLI

> **Note**
>
> Kami merekomendasikan membaca versi konsol terlebih dahulu jika Anda belum melakukannya.

- Anda dapat membuat tipe objek dengan menjalankan`create-type`perintah.

  Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

  1. The`api-id`dari API Anda.
  2. The`definition`, atau konten tipe Anda. Dalam contoh konsol, ini adalah:

     ```
     type Obj_Type_1 {
       id: ID!
     ```

```
  title: String
  date: AWSDateTime
}
```

3. The`format`dari masukan Anda. Dalam contoh ini, kita menggunakan`SDL`.

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-type --api-id abcdefghijklmnopqrstuvwxyz --definition "type
 Obj_Type_1{id: ID! title: String date: AWSDateTime}" --format SDL
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "type": {
        "definition": "type Obj_Type_1{id: ID! title: String date:
 AWSDateTime}",
        "name": "Obj_Type_1",
        "arn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/Obj_Type_1",
        "format": "SDL"
    }
}
```

> **Note**
>
> Pada langkah ini, kami menambahkan tipe objek generik dengan kebutuhan`id`bidang disimpan sebagai`ID`, sebuah`title`bidang disimpan sebagai`String`, dan`date`bidang disimpan sebagai`AWSDateTime`. Untuk melihat daftar jenis dan bidang dan apa yang mereka lakukan, lihatSkema. Untuk melihat daftar skalar dan apa yang mereka lakukan, lihatJenis referensi.
> Pada catatan lebih lanjut, Anda mungkin telah menyadari bahwa memasukkan definisi secara langsung berfungsi untuk tipe yang lebih kecil tetapi tidak layak untuk menambahkan jenis yang lebih besar atau lebih banyak. Anda dapat memilih untuk menambahkan semuanya di`.graphql`berkas dan kemudianmeneruskannya sebagai input.

CDK

> **Tip**
>
> Sebelum Anda menggunakan CDK, kami sarankan untuk meninjau CDKdokumentasi resmibersama denganAWS AppSynciniReferensi CDK.
> Langkah-langkah yang tercantum di bawah ini hanya akan menampilkan contoh umum dari cuplikan yang digunakan untuk menambahkan sumber daya tertentu. Ini adalahtidakdimaksudkan untuk menjadi solusi yang berfungsi dalam kode produksi Anda. Kami juga menganggap Anda sudah memiliki aplikasi yang berfungsi.

Untuk menambahkan tipe, Anda perlu menambahkannya ke`.graphql`berkas. Misalnya, contoh konsol adalah:

```
type Obj_Type_1 {
  id: ID!
  title: String
  date: AWSDateTime
}
```

Anda dapat menambahkan tipe Anda langsung ke skema seperti file lainnya.

> **Note**
>
> Untuk menggunakan perubahan yang dibuat pada API GraphQL Anda, Anda harus menerapkan ulang aplikasi.

Thetipe objekmemiliki bidang yangjenis skalarseperti string dan bilangan bulat.AWS AppSyncjuga memungkinkan Anda untuk menggunakan jenis skalar yang ditingkatkan sepertiAWSDateTimeselain skalar GraphQL dasar. Juga, bidang apa pun yang berakhir dengan tanda seru diperlukan.

TheIDtipe skalar khususnya adalah pengidentifikasi unik yang dapat berupaStringatauInt. Anda dapat mengontrol ini dalam kode resolver Anda untuk penugasan otomatis.

Ada kesamaan antara jenis objek khusus sepertiQuerydan tipe objek “biasa” seperti contoh di atas karena keduanya menggunakantypekata kunci dan dianggap objek. Namun, untuk jenis objek khusus (Query,Mutation, danSubscription), perilaku mereka sangat berbeda karena

mereka diekspos sebagai titik masuk untuk API Anda. Mereka juga lebih tentang membentuk operasi daripada data. Untuk informasi lebih lanjut, lihatJenis kueri dan mutasi.

Pada topik jenis objek khusus, langkah selanjutnya bisa menambahkan satu atau lebih dari mereka untuk melakukan operasi pada data berbentuk. Dalam skenario nyata, setiap skema GraphQL setidaknya harus memiliki tipe kueri root untuk meminta data. Anda dapat menganggap kueri sebagai salah satu titik masuk (atau titik akhir) untuk server GraphQL Anda. Mari tambahkan query sebagai contoh.

Console

- Untuk membuat kueri, Anda cukup menambahkannya ke file skema seperti jenis lainnya. Sebuah kueri akan membutuhkan`Query`ketik dan entri di root seperti ini:

```
schema {
  query: Name_of_Query
}

type Name_of_Query {
  # Add field operation here
}
```

Perhatikan bahwa*Nama_of_Query*dalam lingkungan produksi hanya akan disebut`Query`dalam banyak kasus. Kami merekomendasikan untuk menyimpannya pada nilai ini. Di dalam jenis kueri, Anda dapat menambahkan bidang. Setiap bidang akan melakukan operasi dalam permintaan. Akibatnya, sebagian besar, jika tidak semua, bidang ini akan dilampirkan ke resolver. Namun, kami tidak peduli dengan itu di bagian ini. Mengenai format operasi lapangan, mungkin terlihat seperti ini:

```
Name_of_Query(params): Return_Type # version with params
Name_of_Query: Return_Type # version without params
```

Inilah contohnya:

```
schema {
  query: Query
}

type Query {
  getObj: [Obj_Type_1]
```

```
}

type Obj_Type_1 {
  id: ID!
  title: String
  date: AWSDateTime
}
```

> **Note**
>
> Pada langkah ini, kami menambahkan`Query`mengetik dan mendefinisikannya di`schema`akar. Kami`Query`tipe didefinisikan a`getObj`bidang yang mengembalikan daftar`Obj_Type_1`benda-benda. Perhatikan bahwa`Obj_Type_1`adalah objek dari langkah sebelumnya. Dalam kode produksi, operasi lapangan Anda biasanya akan bekerja dengan data yang dibentuk oleh objek seperti`Obj_Type_1`. Selain itu, bidang seperti`getObj`biasanya akan memiliki resolver untuk melakukan logika bisnis. Itu akan dibahas di bagian yang berbeda.
>
> Sebagai catatan tambahan,AWS AppSyncsecara otomatis menambahkan root skema selama ekspor, jadi secara teknis Anda tidak perlu menambahkannya langsung ke skema. Layanan kami akan secara otomatis memproses skema duplikat. Kami menambahkannya di sini sebagai praktik terbaik.

CLI

> **Note**
>
> Kami merekomendasikan membaca versi konsol terlebih dahulu jika Anda belum melakukannya.

1. Buat`schema`root dengan`query`Definisi dengan menjalankan`create-type`perintah.

   Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

   1. The`api-id`dari API Anda.
   2. The`definition`, atau konten tipe Anda. Dalam contoh konsol, ini adalah:

      ```
      schema {
      ```

```
  query: Query
}
```

3. The`format`dari masukan Anda. Dalam contoh ini, kita menggunakan`SDL`.

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-type --api-id abcdefghijklmnopqrstuvwxyz --definition "schema
 {query: Query}" --format SDL
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "type": {
        "definition": "schema {query: Query}",
        "name": "schema",
        "arn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/schema",
        "format": "SDL"
    }
}
```

Note

Perhatikan bahwa jika Anda tidak memasukkan sesuatu dengan benar di`create-type`perintah, Anda dapat memperbarui root skema Anda (atau jenis apa pun dalam skema) dengan menjalankan`update-type`perintah. Dalam contoh ini, kita akan sementara mengubah root skema untuk berisi`subscription`definisi.
Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

1. The`api-id`dari API Anda.
2. The`type-name`dari tipe Anda. Dalam contoh konsol, ini adalah`schema`.
3. The`definition`, atau konten tipe Anda. Dalam contoh konsol, ini adalah:

```
schema {
  query: Query
}
```

Skema setelah menambahkan`subscription`akan terlihat seperti ini:

```
schema {
  query: Query
  subscription: Subscription
}
```

4. The`format`dari masukan Anda. Dalam contoh ini, kita menggunakan`SDL`.

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync update-type --api-id abcdefghijklmnopqrstuvwxyz --type-name
 schema --definition "schema {query: Query subscription: Subscription}"
 --format SDL
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "type": {
        "definition": "schema {query: Query subscription: Subscription}",
        "arn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/schema",
        "format": "SDL"
    }
}
```

Menambahkan file yang telah diformat akan tetap berfungsi dalam contoh ini.

2. Buat`Query`ketik dengan menjalankan`create-type`perintah.

Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

1. The`api-id`dari API Anda.
2. The`definition`, atau konten tipe Anda. Dalam contoh konsol, ini adalah:

```
type Query {
  getObj: [Obj_Type_1]
}
```

3. The`format`dari masukan Anda. Dalam contoh ini, kita menggunakan`SDL`.

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-type --api-id abcdefghijklmnopqrstuvwxyz --definition "type
 Query {getObj: [Obj_Type_1]}" --format SDL
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "type": {
        "definition": "Query {getObj: [Obj_Type_1]}",
        "name": "Query",
        "arn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/Query",
        "format": "SDL"
    }
}
```

> **Note**
>
> Pada langkah ini, kami menambahkan`Query`mengetik dan mendefinisikannya `dischema`akar. Kami`Query`tipe didefinisikan `agetObj`bidang yang mengembalikan daftar`Obj_Type_1`benda-benda.
> `Dischema`kode root`query: Query`,`query:`bagian menunjukkan bahwa kueri didefinisikan dalam skema Anda, sedangkan`Query`bagian menunjukkan nama objek khusus yang sebenarnya.

CDK

> **Tip**
>
> Sebelum Anda menggunakan CDK, kami sarankan untuk meninjau CDKdokumentasi resmibersama denganAWS AppSyncini[Referensi CDK](#).
> Langkah-langkah yang tercantum di bawah ini hanya akan menampilkan contoh umum dari cuplikan yang digunakan untuk menambahkan sumber daya tertentu. Ini

adalahtidakdimaksudkan untuk menjadi solusi yang berfungsi dalam kode produksi Anda. Kami juga menganggap Anda sudah memiliki aplikasi yang berfungsi.

Anda harus menambahkan kueri dan root skema ke`.graphql`berkas. Contoh kami terlihat seperti contoh di bawah ini, tetapi Anda ingin menggantinya dengan kode skema Anda yang sebenarnya:

```
schema {
  query: Query
}

type Query {
  getObj: [Obj_Type_1]
}

type Obj_Type_1 {
  id: ID!
  title: String
  date: AWSDateTime
}
```

Anda dapat menambahkan tipe Anda langsung ke skema seperti file lainnya.

> **Note**
>
> Memperbarui root skema adalah opsional. Kami menambahkannya ke contoh ini sebagai praktik terbaik.
> Untuk menggunakan perubahan yang dibuat pada API GraphQL Anda, Anda harus menerapkan ulang aplikasi.

Anda sekarang telah melihat contoh membuat objek dan objek khusus (query). Anda juga telah melihat bagaimana ini dapat saling berhubungan untuk menggambarkan data dan operasi. Anda dapat memiliki skema hanya dengan deskripsi data dan satu atau lebih kueri. Namun, kami ingin menambahkan operasi lain untuk menambahkan data ke sumber data. Kami akan menambahkan jenis objek khusus lain yang disebut`Mutation`yang memodifikasi data.

Console

- Mutasi akan disebut`Mutation`. Suka`Query`, operasi lapangan di dalam`Mutation`akan menjelaskan operasi dan akan dilampirkan ke resolver. Juga, perhatikan bahwa kita perlu mendefinisikannya di`schema`root karena ini adalah jenis objek khusus. Berikut adalah contoh mutasi:

```
schema {
  mutation: Name_of_Mutation
}

type Name_of_Mutation {
  # Add field operation here
}
```

Mutasi tipikal akan terdaftar di root seperti kueri. Mutasi didefinisikan menggunakan`type`kata kunci bersama dengan nama. *`Nama_of_mutation`*Biasanya akan disebut`Mutation`, jadi kami sarankan untuk tetap seperti itu. Setiap bidang juga akan melakukan operasi. Mengenai format operasi lapangan, mungkin terlihat seperti ini:

```
Name_of_Mutation(params): Return_Type # version with params
Name_of_Mutation: Return_Type # version without params
```

Inilah contohnya:

```
schema {
  query: Query
  mutation: Mutation
}

type Obj_Type_1 {
  id: ID!
  title: String
  date: AWSDateTime
}

type Query {
  getObj: [Obj_Type_1]
}

type Mutation {
```

```
  addObj(id: ID!, title: String, date: AWSDateTime): Obj_Type_1
}
```

> **Note**
>
> Pada langkah ini, kami menambahkan`Mutation`ketik dengan`addObj`lapangan. Mari kita rangkum apa yang dilakukan bidang ini:
>
> ```
> addObj(id: ID!, title: String, date: AWSDateTime): Obj_Type_1
> ```
>
> `addObj`menggunakan`Obj_Type_1`objek untuk melakukan operasi. Ini jelas karena bidangnya, tetapi sintaks membuktikan ini di`: Obj_Type_1`tipe pengembalian.
> Di dalam`addObj`, ia menerima`id`,`title`, dan`date`Bidang dari`Obj_Type_1`objek sebagai parameter. Seperti yang Anda lihat, ini sangat mirip dengan deklarasi metode. Namun, kami belum menjelaskan perilaku metode kami. Seperti yang dinyatakan sebelumnya, skema hanya ada untuk menentukan apa data dan operasi nantinya dan bukan bagaimana mereka beroperasi. Menerapkan logika bisnis yang sebenarnya akan datang kemudian ketika kita membuat resolver pertama kita.
> Setelah Anda selesai dengan skema Anda, ada opsi untuk mengekspornya sebagai`schema.graphql`berkas. DiEditor skema, Anda dapat memilihSkema eksporuntuk mengunduh file dalam format yang didukung.
> Sebagai catatan tambahan,AWS AppSyncsecara otomatis menambahkan root skema selama ekspor, jadi secara teknis Anda tidak perlu menambahkannya langsung ke skema. Layanan kami akan secara otomatis memproses skema duplikat. Kami menambahkannya di sini sebagai praktik terbaik.

CLI

> **Note**
>
> Kami merekomendasikan membaca versi konsol terlebih dahulu jika Anda belum melakukannya.

1. Perbarui skema root Anda dengan menjalankan`update-type`perintah.

   Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

1. The`api-id`dari API Anda.
2. The`type-name`dari tipe Anda. Dalam contoh konsol, ini adalah`schema`.
3. The`definition`, atau konten tipe Anda. Dalam contoh konsol, ini adalah:

```
schema {
  query: Query
  mutation: Mutation
}
```

4. The`format`dari masukan Anda. Dalam contoh ini, kita menggunakan`SDL`.

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync update-type --api-id abcdefghijklmnopqrstuvwxyz --type-name schema
 --definition "schema {query: Query mutation: Mutation}" --format SDL
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "type": {
        "definition": "schema {query: Query mutation: Mutation}",
        "arn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/schema",
        "format": "SDL"
    }
}
```

2. Buat`Mutation`ketik dengan menjalankan`create-type`perintah.

Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

1. The`api-id`dari API Anda.
2. The`definition`, atau konten tipe Anda. Dalam contoh konsol, ini adalah

```
type Mutation {
  addObj(id: ID!, title: String, date: AWSDateTime): Obj_Type_1
}
```

3. The`format`dari masukan Anda. Dalam contoh ini, kita menggunakan`SDL`.

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-type --api-id abcdefghijklmnopqrstuvwxyz --definition "type
 Mutation {addObj(id: ID! title: String date: AWSDateTime): Obj_Type_1}" --
format SDL
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "type": {
        "definition": "type Mutation {addObj(id: ID! title: String date:
 AWSDateTime): Obj_Type_1}",
        "name": "Mutation",
        "arn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/Mutation",
        "format": "SDL"
    }
}
```

CDK

> **Tip**
>
> Sebelum Anda menggunakan CDK, kami sarankan untuk meninjau CDKdokumentasi resmibersama denganAWS AppSynciniReferensi CDK.
> Langkah-langkah yang tercantum di bawah ini hanya akan menampilkan contoh umum dari cuplikan yang digunakan untuk menambahkan sumber daya tertentu. Ini adalahtidakdimaksudkan untuk menjadi solusi yang berfungsi dalam kode produksi Anda. Kami juga menganggap Anda sudah memiliki aplikasi yang berfungsi.

Anda harus menambahkan kueri dan root skema ke`.graphql`berkas. Contoh kami terlihat seperti contoh di bawah ini, tetapi Anda ingin menggantinya dengan kode skema Anda yang sebenarnya:

```
schema {
  query: Query
  mutation: Mutation
}
```

```
type Obj_Type_1 {
  id: ID!
  title: String
  date: AWSDateTime
}

type Query {
  getObj: [Obj_Type_1]
}

type Mutation {
  addObj(id: ID!, title: String, date: AWSDateTime): Obj_Type_1
}
```

> **Note**
>
> Memperbarui root skema adalah opsional. Kami menambahkannya ke contoh ini sebagai praktik terbaik.
> Untuk menggunakan perubahan yang dibuat pada API GraphQL Anda, Anda harus menerapkan ulang aplikasi.

Pertimbangan opsional - Menggunakan enum sebagai status

Pada titik ini, Anda tahu cara membuat skema dasar. Namun, ada banyak hal yang dapat Anda tambahkan untuk meningkatkan fungsionalitas skema. Satu hal umum yang ditemukan dalam aplikasi adalah penggunaan enum sebagai status. Anda dapat menggunakan enum untuk memaksa nilai tertentu dari sekumpulan nilai yang akan dipilih saat dipanggil. Ini bagus untuk hal-hal yang Anda tahu tidak akan berubah secara drastis dalam jangka waktu yang lama. Secara hipotetis, kita dapat menambahkan enum yang mengembalikan kode status atau String dalam respons.

Sebagai contoh, mari kita asumsikan kita membuat aplikasi media sosial yang menyimpan data posting pengguna di backend. Skema kami berisi`Post`jenis yang mewakili data posting individu:

```
type Post {
  id: ID!
  title: String
  date: AWSDateTime
  poststatus: PostStatus
```

```
}
```

KamiPostakan berisi yang unikid, postingtitle,dateposting, dan enum yang disebutPostStatusyang mewakili status posting saat diproses oleh aplikasi. Untuk operasi kami, kami akan memiliki kueri yang mengembalikan semua data posting:

```
type Query {
  getPosts: [Post]
}
```

Kami juga akan memiliki mutasi yang menambahkan posting ke sumber data:

```
type Mutation {
  addPost(id: ID!, title: String, date: AWSDateTime, poststatus: PostStatus): Post
}
```

Melihat skema kami,PostStatusenum bisa memiliki beberapa status. Kita mungkin ingin tiga keadaan dasar disebutsuccess(posting berhasil diproses),pending(posting sedang diproses), danerror(posting tidak dapat diproses). Untuk menambahkan enum, kita bisa melakukan ini:

```
enum PostStatus {
  success
  pending
  error
}
```

Skema lengkap mungkin terlihat seperti ini:

```
schema {
  query: Query
  mutation: Mutation
}

type Post {
  id: ID!
  title: String
  date: AWSDateTime
  poststatus: PostStatus
}
```

```
type Mutation {
  addPost(id: ID!, title: String, date: AWSDateTime, poststatus: PostStatus): Post
}

type Query {
  getPosts: [Post]
}

enum PostStatus {
  success
  pending
  error
}
```

Jika pengguna menambahkan`Post`dalam aplikasi,`addPost`Operasi akan dipanggil untuk memproses data tersebut. Sebagai resolver yang melekat pada`addPost`memproses data, itu akan terus memperbarui`poststatus`dengan status operasi. Ketika ditanya,`Post`akan berisi status akhir data. Perlu diingat, kami hanya menjelaskan bagaimana kami ingin data bekerja dalam skema. Kami mengasumsikan banyak tentang implementasi resolver kami, yang akan menerapkan logika bisnis aktual untuk menangani data guna memenuhi permintaan.

Pertimbangan opsional - Langganan

Langganan di AWS AppSyncdipanggil sebagai respons terhadap mutasi. Anda mengonfigurasi ini dengan`Subscription`jenis dan`@aws_subscribe()`direktif dalam skema untuk menunjukkan mutasi mana yang memanggil satu atau lebih langganan. Untuk informasi selengkapnya tentang mengonfigurasi langganan, lihatData waktu nyata.

Pertimbangan opsional - Hubungan dan pagination

Misalkan Anda memiliki satu juta`Posts`disimpan dalam tabel DynamoDB, dan Anda ingin mengembalikan beberapa data itu. Namun, contoh query yang diberikan di atas hanya mengembalikan semua posting. Anda tidak ingin mengambil semua ini setiap kali Anda membuat permintaan. Sebaliknya, Anda inginpaginasimelalui mereka. Buat perubahan berikut pada skema Anda:

- Di`getPosts`bidang, tambahkan dua argumen masukan:`nextToken`(iterator) dan`limit`(batas iterasi).
- Tambahkan yang baru`PostIterator`jenis yang mengandung`Posts`(mengambil daftar`Post`objek) dan`nextToken`bidang (iterator).

- Ubahget`Posts`sehingga ia kembali`PostIterator`dan bukan daftar`Post`benda-benda.

```
schema {
  query: Query
  mutation: Mutation
}

type Post {
  id: ID!
  title: String
  date: AWSDateTime
  poststatus: PostStatus
}

type Mutation {
  addPost(id: ID!, title: String, date: AWSDateTime, poststatus: PostStatus): Post
}

type Query {
  getPosts(limit: Int, nextToken: String): PostIterator
}

enum PostStatus {
  success
  pending
  error
}

type PostIterator {
  posts: [Post]
  nextToken: String
}
```

The`PostIterator`jenis memungkinkan Anda untuk mengembalikan sebagian dari daftar`Post`benda dan a`nextToken`untuk mendapatkan bagian selanjutnya. Di dalam`PostIterator`, ada daftar`Post`barang (`[Post]`) yang dikembalikan dengan token pagination (`nextToken`). DiAWS AppSync, ini akan terhubung ke Amazon DynamoDB melalui resolver dan secara otomatis dihasilkan sebagai token terenkripsi. Ini mengubah nilai`limit`argumen untuk`maxResults`parameter dan`nextToken`argumen untuk`exclusiveStartKey`parameter. Untuk contoh dan sampel template bawaan diAWS AppSynckonsol, lihatReferensi penyelesai (JavaScript).

Langkah 2: Melampirkan sumber data

Sumber data adalah sumber daya dalamAWSakun yang dapat berinteraksi dengan API GraphQL.AWS AppSyncmendukung banyak sumber data sepertiAWS Lambda, Amazon DynamoDB, basis data relasional (Amazon Aurora Tanpa Server), AmazonOpenSearchLayanan, dan titik akhir HTTP. SebuahAWS AppSyncAPI dapat dikonfigurasi untuk berinteraksi dengan beberapa sumber data, memungkinkan Anda untuk mengumpulkan data di satu lokasi.AWS AppSyncdapat menggunakan yang sudah adaAWSsumber daya dari akun Anda atau menyediakan tabel DynamoDB atas nama Anda dari definisi skema.

Bagian berikut akan menunjukkan cara melampirkan sumber data ke API GraphQL Anda.

Jenis sumber data

Sekarang Anda telah membuat skema diAWS AppSynckonsol, Anda dapat melampirkan sumber data untuk itu. Saat Anda pertama kali membuat API, ada opsi untuk menyediakan tabel Amazon DynamoDB selama pembuatan skema yang telah ditentukan. Namun, kami tidak akan membahas opsi itu di bagian ini. Anda dapat melihat contoh ini diMeluncurkan skemabagian.

Sebagai gantinya, kita akan melihat semua sumber dataAWS AppSyncmendukung. Ada banyak faktor yang masuk ke dalam memilih solusi yang tepat untuk aplikasi Anda. Bagian di bawah ini akan memberikan beberapa konteks tambahan untuk setiap sumber data. Untuk informasi umum tentang sumber data, lihatSumber data.

Amazon DynamoDB

Amazon DynamoDB adalah salah satuAWS'solusi penyimpanan utama untuk aplikasi yang dapat diskalakan. Komponen inti dari DynamoDB adalahmeja, yang hanya merupakan kumpulan data. Anda biasanya akan membuat tabel berdasarkan entitas sepertiBookatauAuthor. Informasi entri tabel disimpan sebagaiitem, yang merupakan kelompok bidang yang unik untuk setiap entri. Item lengkap mewakili baris/catatan dalam database. Misalnya, item untukBookentri mungkin termasuktitledanauthorbersama dengan nilai-nilai mereka. Bidang individu sepertititledanauthordisebutatribut, yang mirip dengan nilai kolom dalam database relasional.

Seperti yang bisa Anda tebak, tabel akan digunakan untuk menyimpan data dari aplikasi Anda.AWS AppSyncmemungkinkan Anda untuk menghubungkan tabel DynamoDB Anda ke GraphQL API Anda untuk memanipulasi data. Ambil inikasus penggunaandariWeb front-end dan blog seluler. Aplikasi ini memungkinkan pengguna mendaftar untuk aplikasi media sosial. Pengguna dapat bergabung dengan grup dan mengunggah posting yang disiarkan ke pengguna lain yang berlangganan grup. Aplikasi mereka menyimpan informasi pengguna, posting, dan grup pengguna di DynamoDB. API

GraphQL (dikelola olehAWS AppSync) antarmuka dengan tabel DynamoDB. Ketika pengguna membuat perubahan dalam sistem yang akan tercermin di front-end, API GraphQL mengambil perubahan ini dan menyiarkannya ke pengguna lain secara real time.

AWS Lambda

Lambda adalah layanan berbasis peristiwa yang secara otomatis membangun sumber daya yang diperlukan untuk menjalankan kode sebagai respons terhadap suatu peristiwa. Lambda menggunakanfungsi, yang merupakan pernyataan grup yang berisi kode, dependensi, dan konfigurasi untuk mengeksekusi sumber daya. Fungsi secara otomatis dijalankan ketika mereka mendeteksipelatuk, sekelompok aktivitas yang memanggil fungsi Anda. Pemicu bisa berupa apa saja seperti aplikasi yang membuat panggilan API, sebuahAWSlayanan di akun Anda memutar sumber daya, dll. Saat dipicu, fungsi akan diprosesacara, yang merupakan dokumen JSON yang berisi data untuk dimodifikasi.

Lambda bagus untuk menjalankan kode tanpa harus menyediakan sumber daya untuk menjalankannya. Ambil inikasus penggunaandariWeb front-end dan blog seluler. Kasus penggunaan ini sedikit mirip dengan yang dipamerkan di bagian DynamoDB. Dalam aplikasi ini, GraphQL API bertanggung jawab untuk mendefinisikan operasi untuk hal-hal seperti menambahkan posting (mutasi) dan mengambil data itu (kueri). Untuk mengimplementasikan fungsionalitas operasi mereka (misalnya,`getPost ( id: String ! ) : Post`,`getPostsByAuthor ( author: String ! ) : [ Post ]`), mereka menggunakan fungsi Lambda untuk memproses permintaan masuk. Di bawahOpsi 2:AWS AppSyncdengan Lambda resolver, mereka menggunakanAWS AppSynclayanan untuk mempertahankan skema mereka dan menghubungkan sumber data Lambda ke salah satu operasi. Ketika operasi dipanggil, Lambda berinteraksi dengan proxy Amazon RDS untuk melakukan logika bisnis pada database.

Amazon RDS

Amazon RDS memungkinkan Anda membangun dan mengonfigurasi database relasional dengan cepat. Di Amazon RDS, Anda akan membuat generikcontoh basis datayang akan berfungsi sebagai lingkungan database yang terisolasi di cloud. Dalam hal ini, Anda akan menggunakanMesin DB, yang merupakan perangkat lunak RDBMS yang sebenarnya (PostgreSQL, MySQL, dll.). Layanan membongkar sebagian besar pekerjaan backend dengan menyediakan skalabilitas menggunakanAWSInfrastruktur, layanan keamanan seperti patching dan enkripsi, dan menurunkan biaya administrasi untuk penyebaran.

Ambil yang samakasus penggunaandari bagian Lambda. Di bawahOpsi 3:AWS AppSyncdengan resolver Amazon RDS, opsi lain yang disajikan adalah menautkan API GraphQL diAWS AppSyncke

Amazon RDS secara langsung. MenggunakanAPI data, mereka mengaitkan database dengan API GraphQL. Resolver dilampirkan ke bidang (biasanya kueri, mutasi, atau langganan) dan mengimplementasikan pernyataan SQL yang diperlukan untuk mengakses database. Ketika permintaan memanggil bidang dibuat oleh klien, resolver mengeksekusi pernyataan dan mengembalikan respon.

Amazon EventBridge

DiEventBridge, Anda akan membuatbus acara, yang merupakan saluran pipa yang menerima peristiwa dari layanan atau aplikasi yang Anda lampirkan (sumber acara) dan memprosesnya berdasarkan seperangkat aturan. Sebuahperistiwaadalah beberapa perubahan status dalam lingkungan eksekusi, sementaraaturanadalah satu set filter untuk acara. Sebuah aturan mengikutipola acara, atau metadata perubahan status suatu peristiwa (id, Wilayah, nomor akun, ARN, dll.). Ketika sebuah acara cocok dengan pola acara,EventBridgeakan mengirim acara melintasi pipa ke layanan tujuan (target) dan memicu tindakan yang ditentukan dalam aturan.

EventBridgebaik untuk merutekan operasi yang mengubah status ke beberapa layanan lain. Ambil inikasus penggunaandariWeb front-end dan blog seluler. Contoh ini menggambarkan solusi e-commerce yang memiliki beberapa tim yang mempertahankan layanan yang berbeda. Salah satu layanan ini menyediakan pembaruan pesanan kepada pelanggan pada setiap langkah pengiriman (pesanan ditempatkan, sedang berlangsung, dikirim, dikirim, dll.) Di front-end. Namun, tim front-end yang mengelola layanan ini tidak memiliki akses langsung ke data sistem pemesanan seperti yang dikelola oleh tim backend terpisah. Sistem pemesanan tim backend juga digambarkan sebagai kotak hitam, sehingga sulit untuk mengumpulkan informasi tentang cara mereka menyusun data mereka. Namun, tim backend memang menyiapkan sistem yang menerbitkan data pesanan melalui bus acara yang dikelola olehEventBridge. Untuk mengakses data yang berasal dari bus acara dan merutekkannya ke front-end, tim front-end membuat target baru yang menunjuk ke API GraphQL mereka yang berada diAWS AppSync. Mereka juga membuat aturan untuk hanya mengirim data yang relevan dengan pembaruan pesanan. Saat pembaruan dibuat, data dari bus acara dikirim ke API GraphQL. Skema dalam API memproses data, lalu meneruskannya ke front-end.

Tidak ada sumber data

Jika Anda tidak berencana menggunakan sumber data, Anda dapat mengaturnyanone. SEBUAHnoneSumber data, meskipun masih secara eksplisit dikategorikan sebagai sumber data, bukanlah media penyimpanan. Biasanya, resolver akan memanggil satu atau lebih sumber data di beberapa titik untuk memproses permintaan. Namun, ada situasi di mana Anda mungkin tidak perlu memanipulasi sumber data. Mengatur sumber data kenoneakan menjalankan permintaan, melewati langkah pemanggilan data, lalu jalankan responsnya.

Ambil yang samakasus penggunaandariEventBridgebagian. Dalam skema, mutasi memproses pembaruan status, lalu mengirimkannya ke pelanggan. Mengingat cara kerja resolver, biasanya ada setidaknya satu pemanggilan sumber data. Namun, data dalam skenario ini sudah dikirim secara otomatis oleh bus acara. Ini berarti mutasi tidak perlu melakukan pemanggilan sumber data; status pesanan dapat ditangani secara lokal. Mutasi diatur kenone, yang bertindak sebagai nilai pass-through tanpa pemanggilan sumber data. Skema tersebut kemudian diisi dengan data, yang dikirim ke pelanggan.

OpenSearch

AmazonOpenSearchLayanan adalah seperangkat alat untuk mengimplementasikan pencarian teks lengkap, visualisasi data, dan logging. Anda dapat menggunakan layanan ini untuk menanyakan data terstruktur yang telah Anda unggah.

Dalam layanan ini, Anda akan membuat contohOpenSearch. Ini disebutnode. Dalam sebuah node, Anda akan menambahkan setidaknya satuindeks. Indeks secara konseptual sedikit mirip tabel dalam database relasional. (Namun,OpenSearchtidak sesuai dengan ACID, jadi tidak boleh digunakan seperti itu). Anda akan mengisi indeks Anda dengan data yang Anda unggah keOpenSearchlayanan. Ketika data Anda diunggah, itu akan diindeks dalam satu atau lebih pecahan yang ada di indeks. SEBUAHpecahanseperti partisi indeks Anda yang berisi beberapa data Anda dan dapat ditanyakan secara terpisah dari pecahan lainnya. Setelah diunggah, data Anda akan terstruktur sebagai file JSON yang disebutdokumen. Anda kemudian dapat menanyakan node untuk data dalam dokumen.

Titik akhir HTTP

Anda dapat menggunakan titik akhir HTTP sebagai sumber data.AWS AppSyncdapat mengirim permintaan ke titik akhir dengan informasi yang relevan seperti params dan payload. Respons HTTP akan diekspos ke resolver, yang akan mengembalikan respons akhir setelah selesai operasinya.

Menambahkan sumber data

Jika Anda membuat sumber data, Anda dapat menautkannya keAWS AppSynclayanan dan, lebih khusus lagi, API.

Console

1. Masuk keAWS Management Consoledan bukaAppSynckonsol.
    a. Pilih API Anda diDasbor.
    b. DiSidebar, pilihSumber Data.

2. Pilih Buat sumber data.

   a. Beri nama sumber data Anda. Anda juga dapat memberikan deskripsi, tetapi itu opsional.

   b. Pilih AndaJenis sumber data.

   c. Untuk DynamoDB, Anda harus memilih Region Anda, lalu tabel di Region. Anda dapat mendikte aturan interaksi dengan tabel Anda dengan memilih untuk membuat peran tabel generik baru atau mengimpor peran yang ada untuk tabel. Anda dapat mengaktifkanpembuatan versi, yang dapat secara otomatis membuat versi data untuk setiap permintaan ketika beberapa klien mencoba memperbarui data secara bersamaan. Versioning digunakan untuk menyimpan dan memelihara beberapa varian data untuk tujuan deteksi dan resolusi konflik. Anda juga dapat mengaktifkan pembuatan skema otomatis, yang mengambil sumber data Anda dan menghasilkan beberapa CRUD,`List`, dan`Query`operasi yang diperlukan untuk mengaksesnya dalam skema Anda.

      UntukOpenSearch, Anda harus memilih Region Anda, lalu domain (cluster) di Region. Anda dapat mendikte aturan interaksi dengan domain Anda dengan memilih untuk membuat peran tabel generik baru atau mengimpor peran yang ada untuk tabel.

      Untuk Lambda, Anda harus memilih Wilayah Anda, lalu ARN dari fungsi Lambda di Wilayah. Anda dapat mendikte aturan interaksi dengan fungsi Lambda Anda dengan memilih untuk membuat peran tabel generik baru atau mengimpor peran yang ada untuk tabel.

      Untuk HTTP, Anda harus memasukkan titik akhir HTTP Anda.

      UntukEventBridge, Anda harus memilih Wilayah Anda, lalu bus acara di Wilayah. Anda dapat mendikte aturan interaksi dengan bus acara Anda dengan memilih untuk membuat peran tabel generik baru atau mengimpor peran yang ada untuk tabel.

      Untuk RDS, Anda harus memilih Wilayah Anda, lalu toko rahasia (nama pengguna dan kata sandi), nama database, dan skema.

      Untuk tidak ada, Anda akan menambahkan sumber data tanpa sumber data aktual. Ini untuk menangani resolver secara lokal daripada melalui sumber data aktual.

> **Note**
>
> Jika Anda mengimpor peran yang ada, mereka memerlukan kebijakan kepercayaan. Untuk informasi lebih lanjut, lihatKebijakan kepercayaan IAM.

3. Pilih Create (Buat).

> **Note**
>
> Atau, jika Anda membuat sumber data DynamoDB, Anda dapat pergi keSkemahalaman di konsol, pilihBuat Sumber Dayadi bagian atas halaman, lalu isi model yang telah ditentukan untuk dikonversi menjadi tabel. Dalam opsi ini, Anda akan mengisi atau mengimpor tipe dasar, mengkonfigurasi data tabel dasar termasuk kunci partisi, dan meninjau perubahan skema.

CLI

- Buat sumber data Anda dengan menjalankan`create-data-source`perintah.

  Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

  1. The`api-id`dari API Anda.
  2. The`name`dari meja Anda.
  3. The`type`dari sumber data. Bergantung pada jenis sumber data yang Anda pilih, Anda mungkin perlu memasukkan`service-role-arn`dan sebuah`-config`tag.

  Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-data-source --api-id abcdefghijklmnopqrstuvwxyz
--name data_source_name --type data_source_type --service-role-arn
arn:aws:iam::107289374856:role/role_name --[data_source_type]-config {params}
```

CDK

> **Tip**
>
> Sebelum Anda menggunakan CDK, kami sarankan untuk meninjau CDKdokumentasi resmibersama denganAWS AppSynciniReferensi CDK.
> Langkah-langkah yang tercantum di bawah ini hanya akan menampilkan contoh umum dari cuplikan yang digunakan untuk menambahkan sumber daya tertentu. Ini adalahtidakdimaksudkan untuk menjadi solusi yang berfungsi dalam kode produksi Anda. Kami juga menganggap Anda sudah memiliki aplikasi yang berfungsi.

Untuk menambahkan sumber data khusus Anda, Anda harus menambahkan konstruksi ke file tumpukan Anda. Daftar tipe sumber data dapat ditemukan di sini:

- DynamoDbDataSource
- EventBridgeDataSource
- HttpDataSource
- LambdaDataSource
- NoneDataSource
- OpenSearchDataSource
- RdsDataSource

1. Secara umum, Anda mungkin harus menambahkan direktif impor ke layanan yang Anda gunakan. Misalnya, mungkin mengikuti formulir:

   ```
   import * as x from 'x'; # import wildcard as the 'x' keyword from 'x-service'
   import {a, b, ...} from 'c'; # import {specific constructs} from 'c-service'
   ```

   Misalnya, inilah cara Anda dapat mengimporAWS AppSyncdan layanan DynamoDB:

   ```
   import * as appsync from 'aws-cdk-lib/aws-appsync';
   import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
   ```

2. Beberapa layanan seperti RDS memerlukan beberapa pengaturan tambahan dalam file tumpukan sebelum membuat sumber data (misalnya, pembuatan VPC, peran, dan kredenal akses). Lihat contoh di halaman CDK yang relevan untuk informasi lebih lanjut.

3. Untuk sebagian besar sumber data, terutamaAWSlayanan, Anda akan membuat instance baru dari sumber data di file stack Anda. Biasanya, ini akan terlihat seperti berikut:

```
const add_data_source_func = new service_scope.resource_name(scope: Construct,
 id: string, props: data_source_props);
```

Misalnya, berikut adalah contoh tabel Amazon DynamoDB:

```
const add_ddb_table = new dynamodb.Table(this, 'Table_ID', {
  partitionKey: {
    name: 'id',
    type: dynamodb.AttributeType.STRING,
  },
  sortKey: {
    name: 'id',
    type: dynamodb.AttributeType.STRING,
  },
  tableClass: dynamodb.TableClass.STANDARD,
});
```

> **Note**
>
> Sebagian besar sumber data akan memiliki setidaknya satu prop yang diperlukan (akan dilambangkantanpasebuah?simbol). Konsultasikan dokumentasi CDK untuk melihat alat peraga mana yang diperlukan.

4. Selanjutnya, Anda perlu menautkan sumber data ke GraphQL API. Metode yang disarankan adalah menambahkannya saat Anda membuat fungsi untuk penyelesai pipa Anda. Misalnya, cuplikan di bawah ini adalah fungsi yang memindai semua elemen dalam tabel DynamoDB:

```
const add_func = new appsync.AppsyncFunction(this, 'func_ID', {
  name: 'func_name_in_console',
  add_api,
  dataSource: add_api.addDynamoDbDataSource('data_source_name_in_console',
 add_ddb_table),
  code: appsync.Code.fromInline(`
      export function request(ctx) {
        return { operation: 'Scan' };
      }
```

```
      export function response(ctx) {
        return ctx.result.items;
      }
  `),
  runtime: appsync.FunctionRuntime.JS_1_0_0,
});
```

Didi`dataSource`alat peraga, Anda dapat memanggil API GraphQL (`add_api`) dan menggunakan salah satu metode bawaannya (`addDynamoDbDataSource`) untuk membuat hubungan antara tabel dan API GraphQL. Argumen adalah nama tautan ini yang akan ada diAWS AppSynckonsol (`data_source_name_in_console`dalam contoh ini) dan metode tabel (`add_ddb_table`). Lebih lanjut tentang topik ini akan terungkap di bagian selanjutnya ketika Anda mulai membuat resolver.

Ada metode alternatif untuk menghubungkan sumber data. Anda secara teknis dapat menambahkan`api`ke daftar alat peraga dalam fungsi tabel. Misalnya, inilah cuplikan dari langkah 3 tetapi dengan`api`alat peraga yang berisi API GraphQL:

```
const add_api = new appsync.GraphqlApi(this, 'API_ID', {
  ...
});

const add_ddb_table = new dynamodb.Table(this, 'Table_ID', {

 ...

  api: add_api
});
```

Atau, Anda dapat menghubungi`GraphqlApi`membangun secara terpisah:

```
const add_api = new appsync.GraphqlApi(this, 'API_ID', {
  ...
});

const add_ddb_table = new dynamodb.Table(this, 'Table_ID', {
  ...
});
```

```
const link_data_source =
 add_api.addDynamoDbDataSource('data_source_name_in_console', add_ddb_table);
```

Kami merekomendasikan hanya membuat asosiasi di alat peraga fungsi. Jika tidak, Anda harus menautkan fungsi resolver Anda ke sumber data secara manual diAWS AppSynckonsol (jika Anda ingin tetap menggunakan nilai konsol`data_source_name_in_console`) atau buat asosiasi terpisah dalam fungsi dengan nama lain seperti`data_source_name_in_console_2`. Ini karena keterbatasan dalam cara alat peraga memproses informasi.

> **Note**
>
> Anda harus menerapkan ulang aplikasi untuk melihat perubahan Anda.

Kebijakan kepercayaan IAM

Jika Anda menggunakan peran IAM yang ada untuk sumber data Anda, Anda harus memberikan peran tersebut izin yang sesuai untuk melakukan operasi padaAWSsumber daya, seperti`PutItem`pada tabel Amazon DynamoDB. Anda juga perlu memodifikasi kebijakan kepercayaan pada peran itu untuk memungkinkanAWS AppSyncuntuk menggunakannya untuk akses sumber daya seperti yang ditunjukkan dalam contoh kebijakan berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "appsync.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}
```

Anda juga dapat menambahkan ketentuan ke kebijakan kepercayaan Anda untuk membatasi akses ke sumber data sesuai keinginan. Saat ini,`SourceArn`dan`SourceAccount`kunci dapat digunakan dalam kondisi ini. Misalnya, kebijakan berikut membatasi akses ke sumber data Anda ke akun`123456789012`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appsync.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Atau, Anda dapat membatasi akses ke sumber data ke API tertentu, seperti`abcdefghijklmnopq`, menggunakan kebijakan berikut:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appsync.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:appsync:us-west-2:123456789012:apis/
abcdefghijklmnopq"
        }
      }
    }
  ]
}
```

Anda dapat membatasi akses ke semuaAWS AppSyncAPI dari wilayah tertentu, seperti`us-east-1`, menggunakan kebijakan berikut:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appsync.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:appsync:us-east-1:123456789012:apis/*"
        }
      }
    }
  ]
}
```

Pada bagian selanjutnya (Mengkonfigurasi Resolver), kami akan menambahkan logika bisnis resolver kami dan melampirkannya ke bidang dalam skema kami untuk memproses data di sumber data kami.

Untuk informasi selengkapnya mengenai konfigurasi kebijakan peran, lihatMemodifikasi perandiPanduan Pengguna IAM.

Untuk informasi lebih lanjut mengenai akses lintas akunAWS Lambdaresolver untukAWS AppSync, lihatMembangun cross-accountAWS Lambdaresolver untukAWS AppSync.

Langkah 3: Mengkonfigurasi resolver

Di bagian sebelumnya, Anda mempelajari cara membuat skema GraphQL dan sumber data, lalu menautkannya bersama-sama diAWS AppSynclayanan. Dalam skema Anda, Anda mungkin telah menetapkan satu atau beberapa bidang (operasi) dalam kueri dan mutasi Anda. Sementara skema menggambarkan jenis data yang akan diminta operasi dari sumber data, skema tersebut tidak pernah menerapkan bagaimana operasi tersebut akan berperilaku di sekitar data.

Perilaku operasi selalu diimplementasikan dalam resolver, yang akan dikaitkan dengan bidang yang melakukan operasi. Untuk informasi selengkapnya tentang cara kerja resolver secara umum, lihatPenyelesaihalaman.

DiAWS AppSync, resolver Anda terkait dengan runtime, yang merupakan lingkungan di mana resolver Anda mengeksekusi. Runtime mendikte bahasa tempat resolver Anda akan ditulis. Saat ini ada dua runtime yang didukung: APPSYNC_JS (JavaScript) dan Bahasa Template Kecepatan Apache (VTL).

Saat menerapkan resolver, ada struktur umum yang mereka ikuti:

- Sebelum langkah: Ketika permintaan dibuat oleh klien, resolver untuk bidang skema yang digunakan (biasanya kueri, mutasi, langganan Anda) diteruskan data permintaan. Penyelesai akan mulai memproses data permintaan dengan penangan langkah sebelum, yang memungkinkan beberapa operasi pra-pemrosesan dilakukan sebelum data bergerak melalui resolver.
- Fungsi (s): Setelah langkah sebelum berjalan, permintaan diteruskan ke daftar fungsi. Fungsi pertama dalam daftar akan dijalankan terhadap sumber data. Fungsi adalah subset dari kode resolver Anda yang berisi permintaan dan penangan responsnya sendiri. Handler permintaan akan mengambil data permintaan dan melakukan operasi terhadap sumber data. Response handler akan memproses respon sumber data sebelum meneruskannya kembali ke daftar. Jika ada lebih dari satu fungsi, data permintaan akan dikirim ke fungsi berikutnya dalam daftar yang akan dieksekusi. Fungsi dalam daftar akan dijalankan secara serial dalam urutan yang ditentukan oleh pengembang. Setelah semua fungsi dieksekusi, hasil akhir diteruskan ke step.res setelah
- Setelah langkah: Langkah setelah adalah fungsi handler yang memungkinkan Anda melakukan beberapa operasi akhir pada respons fungsi akhir sebelum meneruskannya ke respons GraphQL.

Aliran ini adalah contoh dari resolver pipa. Resolver pipeline didukung di kedua runtime. Namun, ini adalah penjelasan yang disederhanakan tentang apa yang dapat dilakukan oleh penyelesai pipa. Juga, kami hanya menjelaskan satu konfigurasi resolver yang mungkin. Untuk informasi selengkapnya tentang konfigurasi resolver yang didukung, lihatJavaScriptikhtisar penyelesaiuntuk APPSYNC_JS ataulkhtisar template pemetaan resolveruntuk VTL.

Seperti yang Anda lihat, resolver bersifat modular. Agar komponen resolver berfungsi dengan baik, mereka harus dapat mengintip ke dalam keadaan eksekusi dari komponen lain. DariPenyelesaibagian, Anda tahu bahwa setiap komponen dalam resolver dapat diteruskan informasi penting tentang keadaan eksekusi sebagai satu set argumen (`args`,`context`, dll.). DiAWS AppSync, ini ditangani secara ketat oleh`context`. Ini adalah wadah untuk informasi tentang bidang yang sedang diselesaikan. Ini dapat mencakup semuanya mulai dari argumen yang diteruskan, hasil, data otorisasi, data header, dll. Untuk informasi lebih lanjut tentang konteksnya, lihatReferensi objek konteks penyelesaiuntuk APPSYNC_JS ataulReferensi konteks template pemetaan penyelesaiuntuk VTL.

Konteksnya bukan satu-satunya alat yang dapat Anda gunakan untuk mengimplementasikan resolver Anda.AWS AppSyncmendukung berbagai utilitas untuk menghasilkan nilai, penanganan kesalahan, penguraian, konversi, dll. Anda dapat melihat daftar utilitaskemariuntuk APPSYNC_JS ataukemariuntuk VTL.

Di bagian berikut, Anda akan belajar cara mengonfigurasi resolver di API GraphQL Anda.

Topik

- Mengkonfigurasi resolver (JavaScript)
- Mengkonfigurasi resolver (VTL)

Mengkonfigurasi resolver (JavaScript)

Resolver GraphQL menghubungkan bidang dalam skema tipe ke sumber data. Resolver adalah mekanisme dimana permintaan dipenuhi.

Resolver diAWS AppSyncmenggunakanJavaScriptuntuk mengubah ekspresi GraphQL menjadi format yang dapat digunakan sumber data. Atau, template pemetaan dapat ditulis dalamBahasa Template Kecepatan Apache (VTL)untuk mengubah ekspresi GraphQL menjadi format yang dapat digunakan sumber data.

Bagian ini menjelaskan cara mengkonfigurasi resolver menggunakanJavaScript. TheTutorial penyelesai (JavaScript)bagian menyediakan tutorial mendalam tentang cara menerapkan resolver menggunakanJavaScript. TheReferensi penyelesai (JavaScript)bagian memberikan penjelasan tentang operasi utilitas yang dapat digunakan denganJavaScriptpenyelesai.

Sebaiknya ikuti panduan ini sebelum mencoba menggunakan salah satu tutorial yang disebutkan di atas.

Di bagian ini, kita akan membahas cara membuat dan mengkonfigurasi resolver untuk kueri dan mutasi.

> **Note**
>
> Panduan ini mengasumsikan Anda telah membuat skema Anda dan memiliki setidaknya satu kueri atau mutasi. Jika Anda mencari langganan (data waktu nyata), lihatinipanduan.

Di bagian ini, kami akan memberikan beberapa langkah umum untuk mengonfigurasi resolver bersama dengan contoh yang menggunakan skema di bawah ini:

```
// schema.graphql file

input CreatePostInput {
  title: String
  date: AWSDateTime
}

type Post {
  id: ID!
  title: String
  date: AWSDateTime
}

type Mutation {
  createPost(input: CreatePostInput!): Post
}

type Query {
  getPost: [Post]
}
```

Membuat resolver kueri dasar

Bagian ini akan menunjukkan cara membuat resolver kueri dasar.

Console

1. Masuk keAWS Management Consoledan bukaAppSynckonsol.
   a. DiDasbor API, pilih API GraphQL Anda.
   b. DiSidebar, pilihSkema.
2. Masukkan detail skema dan sumber data Anda. LihatMerancang skema AndadanMelampirkan sumber databagian untuk informasi lebih lanjut.
3. Di sebelahSkemaeditor, Ada jendela bernamaPenyelesai. Kotak ini berisi daftar jenis dan bidang seperti yang didefinisikan dalamSkemajendela. Anda dapat melampirkan resolver ke bidang. Anda kemungkinan besar akan melampirkan resolver ke operasi lapangan Anda. Pada bagian ini, kita akan melihat konfigurasi query sederhana. Di bawahPermintaanketik, pilihLampirkandi sebelah bidang kueri Anda.

4. PadaLampirkan resolverhalaman, di bawahJenis penyelesai, Anda dapat memilih antara pipa atau unit resolver. Untuk informasi selengkapnya tentang jenis ini, lihatPenyelesai. Panduan ini akan menggunakan`pipeline resolvers`.

   > **Tip**
   >
   > Saat membuat resolver pipeline, sumber data Anda akan dilampirkan ke fungsi pipeline. Fungsi dibuat setelah Anda membuat resolver pipeline itu sendiri, itulah sebabnya tidak ada opsi untuk mengaturnya di halaman ini. Jika Anda menggunakan resolver unit, sumber data terikat langsung ke resolver, jadi Anda akan mengaturnya di halaman ini.

   UntukRuntime penyelesai, pilih`APPSYNC_JS`untuk mengaktifkanJavaScriptruntime.

5. Anda dapat mengaktifkancachinguntuk API ini. Sebaiknya matikan fitur ini untuk saat ini. Pilih Create (Buat).

6. PadaEdit penyelesaihalaman, ada editor kode bernamaKode penyelesaiyang memungkinkan Anda untuk menerapkan logika untuk penangan dan respons resolver (sebelum dan sesudah langkah). Untuk informasi lebih lanjut, lihatJavaScriptikhtisar penyelesai.

   > **Note**
   >
   > Dalam contoh kita, kita hanya akan membiarkan permintaan kosong dan respons diatur untuk mengembalikan hasil sumber data terakhir darikonteks:

   ```
   import {util} from '@aws-appsync/utils';

   export function request(ctx) {
       return {};
   }

   export function response(ctx) {
       return ctx.prev.result;
   }
   ```

   Di bawah bagian ini, ada tabel yang disebutFungsi. Fungsi memungkinkan Anda untuk menerapkan kode yang dapat digunakan kembali di beberapa resolver. Alih-alih terus-

menerus menulis ulang atau menyalin kode, Anda dapat menyimpan kode sumber sebagai fungsi untuk ditambahkan ke resolver kapan pun Anda membutuhkannya.

Fungsi membentuk sebagian besar daftar operasi pipa. Saat menggunakan beberapa fungsi dalam resolver, Anda mengatur urutan fungsi, dan mereka akan dijalankan dalam urutan itu secara berurutan. Mereka dieksekusi setelah fungsi permintaan berjalan dan sebelum fungsi respons dimulai.

Untuk menambahkan fungsi baru, di bawahFungsi, pilihTambahkan fungsi, makaBuat fungsi baru. Atau, Anda mungkin melihatBuat fungsitombol untuk memilih sebagai gantinya.

a. Pilih sumber data. Ini akan menjadi sumber data tempat resolver bertindak.

> **Note**
>
> Dalam contoh kami, kami melampirkan resolver untuk`getPost`, yang mengambil`Post`objek oleh`id`. Mari kita asumsikan kita sudah menyiapkan tabel DynamoDB untuk skema ini. Kunci partisi diatur ke`id`dan kosong.

b. Masukkan`Function name`.

c. Di bawahKode fungsi, Anda harus mengimplementasikan perilaku fungsi. Ini mungkin membingungkan, tetapi setiap fungsi akan memiliki permintaan lokal dan penangan respons sendiri. Permintaan berjalan, kemudian pemanggilan sumber data dibuat untuk menangani permintaan, kemudian respons sumber data diproses oleh penangan respons. Hasilnya disimpan dikonteksobjek. Setelah itu, fungsi berikutnya dalam daftar akan berjalan atau akan diteruskan ke penangan respons langkah setelah jika itu yang terakhir.

> **Note**
>
> Dalam contoh kami, kami melampirkan resolver`getPost`, yang mendapat daftar`Post`objek dari sumber data. Fungsi permintaan kami akan meminta data dari tabel kami, tabel akan meneruskan responsnya ke konteks (ctx), maka respons akan mengembalikan hasilnya dalam konteks.AWS AppSyncKekuatan terletak pada keterkaitannya dengan yang lainAWSlayanan. Karena kami menggunakan DynamoDB, kami memilikisuite operasiuntuk menyederhanakan hal-hal seperti ini. Kami memiliki beberapa contoh boilerplate untuk tipe sumber data lainnya juga.

Kode kita akan terlihat seperti ini:

```
import { util } from '@aws-appsync/utils';

/**
 * Performs a scan on the dynamodb data source
 */
export function request(ctx) {
  return { operation: 'Scan' };
}

/**
 * return a list of scanned post items
 */
export function response(ctx) {
  return ctx.result.items;
}
```

Pada langkah ini, kami menambahkan dua fungsi:

- `request`: Handler permintaan melakukan operasi pengambilan terhadap sumber data. Argumen berisi objek konteks (`ctx`), atau beberapa data yang tersedia untuk semua resolver yang melakukan operasi tertentu. Misalnya, mungkin berisi data otorisasi, nama bidang yang diselesaikan, dll. Pernyataan pengembalian melakukan`Scan`operasi (lihat`kemari`sebagai contoh). Karena kami bekerja dengan DynamoDB, kami diizinkan untuk menggunakan beberapa operasi dari layanan itu. Pemindaian melakukan pengambilan dasar semua item di tabel kami. Hasil operasi ini disimpan dalam objek konteks sebagai`result`kontainer sebelum diteruskan ke handler respons. The`request`dijalankan sebelum respons dalam pipa.
- `response`: Response handler yang mengembalikan output dari`request`. Argumennya adalah objek konteks yang diperbarui, dan pernyataan pengembaliannya adalah`ctx.prev.result`. Pada titik ini dalam panduan ini, Anda mungkin tidak terbiasa dengan nilai ini.`ctx`mengacu pada objek konteks.`prev`mengacu pada operasi sebelumnya dalam pipa, yang merupakan kami`request`. The`result`berisi hasil resolver saat bergerak melalui pipa. Jika Anda menggabungkan

semuanya,`ctx.prev.result`mengembalikan hasil operasi terakhir yang dilakukan, yang merupakan penangan permintaan.

d. PilihBuatsetelah kau selesai.

7. Kembali ke layar resolver, di bawahFungsi, pilihTambahkan fungsidrop-down dan tambahkan fungsi Anda ke daftar fungsi Anda.
8. PilihMenyimpanuntuk memperbarui resolver.

CLI

Untuk menambahkan fungsi Anda

- Buat fungsi untuk penyelesai pipeline Anda menggunakan`create-function`perintah.

  Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

  1. The`api-id`dari API Anda.
  2. The`name`dari fungsi diAWS AppSynckonsol.
  3. The`data-source-name`, atau nama sumber data yang akan digunakan fungsi. Itu harus sudah dibuat dan ditautkan ke API GraphQL Anda diAWS AppSynclayanan.
  4. The`runtime`, atau lingkungan dan bahasa fungsi. UntukJavaScript, nama itu harus`APPSYNC_JS`, dan runtime,`1.0.0`.
  5. The`code`, atau penangan permintaan dan respons fungsi Anda. Meskipun Anda dapat mengetiknya secara manual, jauh lebih mudah untuk menambahkannya ke file.txt (atau format serupa) dan kemudian meneruskannya sebagai argumen.

  > **Note**
  >
  > Kode kueri kami akan berada dalam file yang diteruskan sebagai argumen:

```
import { util } from '@aws-appsync/utils';

/**
 * Performs a scan on the dynamodb data source
 */
export function request(ctx) {
  return { operation: 'Scan' };
}
```

```
/**
 * return a list of scanned post items
 */
export function response(ctx) {
  return ctx.result.items;
}
```

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-function \
--api-id abcdefghijklmnopqrstuvwxyz \
--name get_posts_func_1 \
--data-source-name table-for-posts \
--runtime name=APPSYNC_JS,runtimeVersion=1.0.0 \
--code file://~/path/to/file/{filename}.{fileType}
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "functionConfiguration": {
        "functionId": "ejglgvmcabdn7lx75ref4qeig4",
        "functionArn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/functions/ejglgvmcabdn7lx75ref4qeig4",
        "name": "get_posts_func_1",
        "dataSourceName": "table-for-posts",
        "maxBatchSize": 0,
        "runtime": {
            "name": "APPSYNC_JS",
            "runtimeVersion": "1.0.0"
        },
        "code": "Code output goes here"
    }
}
```

> **Note**
>
> Pastikan Anda merekam`functionId`di suatu tempat karena ini akan digunakan untuk melampirkan fungsi ke resolver.

Untuk membuat resolver Anda

- Buat fungsi pipeline untuk`Query`dengan menjalankan`create-resolver`perintah.

  Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

  1. The`api-id`dari API Anda.
  2. The`type-name`, atau jenis objek khusus dalam skema Anda (Kueri, Mutasi, Langganan).
  3. The`field-name`, atau operasi bidang di dalam jenis objek khusus yang ingin Anda lampirkan resolver.
  4. The`kind`, yang menentukan unit atau resolver pipa. Setel ini ke`PIPELINE`untuk mengaktifkan fungsi pipa.
  5. The`pipeline-config`, atau fungsi untuk melampirkan ke resolver. Pastikan Anda tahu`functionId`nilai-nilai fungsi Anda. Urutan daftar penting.
  6. The`runtime`, yang`APPSYNC_JS`(JavaScript). The`runtimeVersion`saat ini adalah`1.0.0`.
  7. The`code`, yang berisi penangan langkah sebelum dan sesudah.

  > **Note**
  >
  > Kode kueri kami akan berada dalam file yang diteruskan sebagai argumen:

```
import { util } from '@aws-appsync/utils';

/**
 * Sends a request to `put` an item in the DynamoDB data source
 */
export function request(ctx) {
  const { id, ...values } = ctx.args;
  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({ id }),
    attributeValues: util.dynamodb.toMapValues(values),
  };
}

/**
 * returns the result of the `put` operation
 */
export function response(ctx) {
```

```
  return ctx.result;
}
```

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-resolver \
--api-id abcdefghijklmnopqrstuvwxyz \
--type-name Query \
--field-name getPost \
--kind PIPELINE \
--pipeline-config functions=ejglgvmcabdn7lx75ref4qeig4 \
--runtime name=APPSYNC_JS,runtimeVersion=1.0.0 \
--code file:///path/to/file/{filename}.{fileType}
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "resolver": {
        "typeName": "Mutation",
        "fieldName": "getPost",
        "resolverArn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/Mutation/resolvers/getPost",
        "kind": "PIPELINE",
        "pipelineConfig": {
            "functions": [
                "ejglgvmcabdn7lx75ref4qeig4"
            ]
        },
        "maxBatchSize": 0,
        "runtime": {
            "name": "APPSYNC_JS",
            "runtimeVersion": "1.0.0"
        },
        "code": "Code output goes here"
    }
}
```

CDK

> **Tip**
>
> Sebelum Anda menggunakan CDK, kami sarankan untuk meninjau CDKdokumentasi resmibersama denganAWS AppSynciniReferensi CDK.
> Langkah-langkah yang tercantum di bawah ini hanya akan menampilkan contoh umum dari cuplikan yang digunakan untuk menambahkan sumber daya tertentu. Ini adalahtidakdimaksudkan untuk menjadi solusi yang berfungsi dalam kode produksi Anda. Kami juga menganggap Anda sudah memiliki aplikasi yang berfungsi.

Aplikasi dasar akan membutuhkan hal-hal berikut:

1. Arahan impor layanan
2. Kode skema
3. Generator sumber data
4. Kode fungsi
5. Kode penyelesai

DariMerancang skema AndadanMelampirkan sumber databagian, kita tahu bahwa file stack akan menyertakan arahan impor formulir:

```
import * as x from 'x'; # import wildcard as the 'x' keyword from 'x-service'
import {a, b, ...} from 'c'; # import {specific constructs} from 'c-service'
```

> **Note**
>
> Di bagian sebelumnya, kami hanya menyatakan cara mengimporAWS AppSynckonstruksi. Dalam kode nyata, Anda harus mengimpor lebih banyak layanan hanya untuk menjalankan aplikasi. Dalam contoh kami, jika kami membuat aplikasi CDK yang sangat sederhana, kami setidaknya akan mengimporAWS AppSynclayanan bersama dengan sumber data kami, yang merupakan tabel DynamoDB. Kami juga perlu mengimpor beberapa konstruksi tambahan untuk menerapkan aplikasi:
>
> ```
> import * as cdk from 'aws-cdk-lib';
> import * as appsync from 'aws-cdk-lib/aws-appsync';
> ```

```
import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
import { Construct } from 'constructs';
```

Untuk meringkas masing-masing:

- `import * as cdk from 'aws-cdk-lib';`: Ini memungkinkan Anda untuk menentukan aplikasi CDK dan konstruksi seperti tumpukan. Ini juga berisi beberapa fungsi utilitas yang berguna untuk aplikasi kita seperti memanipulasi metadata. Jika Anda terbiasa dengan arahan impor ini, tetapi bertanya-tanya mengapa pustaka inti cdk tidak digunakan di sini, lihatMigrasihalaman.
- `import * as appsync from 'aws-cdk-lib/aws-appsync';`: Ini mengimporAWS AppSynclayanan.
- `import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';`: Ini mengimporLayanan DynamoDB.
- `import { Construct } from 'constructs';`: Kita membutuhkan ini untuk mendefinisikan rootmembangun.

Jenis impor tergantung pada layanan yang Anda panggil. Kami merekomendasikan untuk melihat dokumentasi CDK sebagai contoh. Skema di bagian atas halaman akan menjadi file terpisah di aplikasi CDK Anda sebagai`.graphql`berkas. Dalam file stack, kita dapat mengaitkannya dengan GraphQL baru menggunakan formulir:

```
const add_api = new appsync.GraphqlApi(this, 'graphQL-example', {
  name: 'my-first-api',
  schema: appsync.SchemaFile.fromAsset(path.join(__dirname, 'schema.graphql')),
});
```

> **Note**
>
> Dalam ruang lingkup`add_api`, kami menambahkan API GraphQL baru menggunakan`new`kata kunci diikuti oleh`appsync.GraphqlApi(scope: Construct, id: string , props: GraphqlApiProps)`. Ruang lingkup kami adalah`this`Id CFN adalah`graphQL-example`, dan alat peraga kami`my-first-api`(nama API di konsol) dan`schema.graphql`(jalur absolut ke file skema).

Untuk menambahkan sumber data, pertama-tama Anda harus menambahkan sumber data ke tumpukan. Kemudian, Anda perlu mengaitkannya dengan API GraphQL menggunakan metode khusus sumber. Asosiasi akan terjadi ketika Anda membuat fungsi resolver Anda. Sementara itu, mari kita gunakan contoh dengan membuat tabel DynamoDB menggunakan`dynamodb.Table`:

```
const add_ddb_table = new dynamodb.Table(this, 'posts-table', {
  partitionKey: {
    name: 'id',
    type: dynamodb.AttributeType.STRING,
  },
});
```

> **Note**
>
> Jika kita menggunakan ini dalam contoh kita, kita akan menambahkan tabel DynamoDB baru dengan id CFN dari`posts-table`dan kunci partisi`id (S)`.

Selanjutnya, kita perlu mengimplementasikan resolver kita di file stack. Berikut adalah contoh kueri sederhana yang memindai semua item dalam tabel DynamoDB:

```
const add_func = new appsync.AppsyncFunction(this, 'func-get-posts', {
  name: 'get_posts_func_1',
  add_api,
  dataSource: add_api.addDynamoDbDataSource('table-for-posts', add_ddb_table),
  code: appsync.Code.fromInline(`
      export function request(ctx) {
        return { operation: 'Scan' };
      }

      export function response(ctx) {
        return ctx.result.items;
      }
  `),
  runtime: appsync.FunctionRuntime.JS_1_0_0,
});

new appsync.Resolver(this, 'pipeline-resolver-get-posts', {
  add_api,
  typeName: 'Query',
  fieldName: 'getPost',
```

```
  code: appsync.Code.fromInline(`
      export function request(ctx) {
        return {};
      }

      export function response(ctx) {
        return ctx.prev.result;
      }
  `),
  runtime: appsync.FunctionRuntime.JS_1_0_0,
  pipelineConfig: [add_func],
});
```

Note

Pertama, kita membuat sebuah fungsi yang disebut`add_func`. Urutan pembuatan ini mungkin tampak agak berlawanan dengan intuisi, tetapi Anda harus membuat fungsi di resolver pipeline Anda sebelum Anda membuat resolver itu sendiri. Sebuah fungsi mengikuti formulir:

```
AppsyncFunction(scope: Construct, id: string, props: AppsyncFunctionProps)
```

Ruang lingkup kami adalah`this`, id CFN kami adalah`func-get-posts`, dan alat peraga kami berisi detail fungsi yang sebenarnya. Di dalam alat peraga, kami menyertakan:

- The`name`dari fungsi yang akan hadir diAWS AppSynckonsol (`get_posts_func_1`).
- API GraphQL yang kami buat sebelumnya (`add_api`).
- Sumber data; ini adalah titik di mana kita menghubungkan sumber data ke nilai API GraphQL, lalu melampirkannya ke fungsi. Kami mengambil tabel yang kami buat (`add_ddb_table`) dan lampirkan ke API GraphQL (`add_api`) menggunakan salah satu`GraphqlApi`metode (`addDynamoDbDataSource`). Nilai id (`table-for-posts`) adalah nama sumber data diAWS AppSynckonsol. Untuk daftar metode khusus sumber, lihat halaman berikut:
  - DynamoDbDataSource
  - EventBridgeDataSource
  - HttpDataSource
  - LambdaDataSource
  - NoneDataSource

    - OpenSearchDataSource
    - RdsDataSource
  - Kode berisi permintaan fungsi dan penangan respons, yang merupakan pemindaian dan pengembalian sederhana.
  - Runtime menentukan bahwa kita ingin menggunakan APPSYNC_JS runtime versi 1.0.0. Perhatikan bahwa saat ini satu-satunya versi yang tersedia untuk APPSYNC_JS.

  Selanjutnya, kita perlu melampirkan fungsi ke resolver pipa. Kami membuat resolver kami menggunakan formulir:

```
Resolver(scope: Construct, id: string, props: ResolverProps)
```

  Ruang lingkup kami adalah`this`, id CFN kami adalah`pipeline-resolver-get-posts`, dan alat peraga kami berisi detail fungsi yang sebenarnya. Di dalam alat peraga, kami menyertakan:

  - API GraphQL yang kami buat sebelumnya (`add_api`).
  - Nama tipe objek khusus; ini adalah operasi query, jadi kami hanya menambahkan nilai`Query`.
  - Nama bidang (`getPost`) adalah nama bidang dalam skema di bawah`Query`jenis.
  - Kode berisi penangan sebelum dan sesudah Anda. Contoh kami hanya mengembalikan hasil apa pun yang ada dalam konteks setelah fungsi melakukan operasinya.
  - Runtime menentukan bahwa kita ingin menggunakan APPSYNC_JS runtime versi 1.0.0. Perhatikan bahwa saat ini satu-satunya versi yang tersedia untuk APPSYNC_JS.
  - Konfigurasi pipeline berisi referensi ke fungsi yang kita buat (`add_func`).

Untuk meringkas apa yang terjadi dalam contoh ini, Anda melihat sebuahAWS AppSyncfungsi yang menerapkan permintaan dan penangan respons. Fungsi ini bertanggung jawab untuk berinteraksi dengan sumber data Anda. Handler permintaan mengirim`Scan`operasi keAWS AppSync, menginstruksikannya tentang operasi apa yang harus dilakukan terhadap sumber data DynamoDB Anda. Penangan respons mengembalikan daftar item (`ctx.result.items`). Daftar item kemudian dipetakan ke`Post`Jenis GraphQL secara otomatis.

Membuat resolver mutasi dasar

Bagian ini akan menunjukkan cara membuat resolver mutasi dasar.

Console

1. Masuk keAWS Management Consoledan bukaAppSynckonsol.
   a. DiDasbor API, pilih API GraphQL Anda.
   b. DiSidebar, pilihSkema.
2. Di bawahPenyelesaibagian danMutasiketik, pilihLampirkandi sebelah bidang Anda.

   > **Note**
   >
   > Dalam contoh kami, kami melampirkan resolver untuk`createPost`, yang menambahkan`Post`keberatan dengan meja kami. Mari kita asumsikan kita menggunakan tabel DynamoDB yang sama dari bagian terakhir. Kunci partisi diatur `id`dan kosong.

3. PadaLampirkan resolverhalaman, di bawahJenis penyelesai, pilih`pipeline resolvers`. Sebagai pengingat, Anda dapat menemukan informasi lebih lanjut tentang resolverkemari. UntukRuntime penyelesai, pilih`APPSYNC_JS`untuk mengaktifkanJavaScriptruntime.
4. Anda dapat mengaktifkancachinguntuk API ini. Sebaiknya matikan fitur ini untuk saat ini. Pilih Create (Buat).
5. PilihTambahkan fungsi, lalu pilihBuat fungsi baru. Atau, Anda mungkin melihatBuat fungsitombol untuk memilih sebagai gantinya.
   a. Pilih sumber data Anda. Ini harus menjadi sumber yang datanya akan Anda manipulasi dengan mutasi.
   b. Masukkan`Function name`.
   c. Di bawahKode fungsi, Anda harus mengimplementasikan perilaku fungsi. Ini adalah mutasi, jadi permintaan idealnya akan melakukan beberapa operasi perubahan status pada sumber data yang dipanggil. Hasilnya akan diproses oleh fungsi respons.

Note

`createPost`menambahkan, atau “menempatkan”, yang baru`Post`dalam tabel dengan parameter kami sebagai data. Kita bisa menambahkan sesuatu seperti ini:

```
import { util } from '@aws-appsync/utils';

/**
 * Sends a request to `put` an item in the DynamoDB data source
 */
export function request(ctx) {
  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({id: util.autoId()}),
    attributeValues: util.dynamodb.toMapValues(ctx.args.input),
  };
}

/**
 * returns the result of the `put` operation
 */
export function response(ctx) {
  return ctx.result;
}
```

Pada langkah ini, kami juga menambahkan`request`dan`response`fungsi:

- `request`: Handler permintaan menerima konteks sebagai argumen. Pernyataan pengembalian penangan permintaan melakukan`PutItem`perintah, yang merupakan operasi DynamoDB bawaan (lihatkemariataukemarisebagai contoh). The`PutItem`perintah menambahkan`Post`objek ke tabel DynamoDB kami dengan mengambil partisi`key`nilai (secara otomatis dihasilkan oleh`util.autoid()`) dan`attributes`dari input argumen konteks (ini adalah nilai yang akan kami berikan dalam permintaan kami). The`key`adalah`id`dan`attributes`adalah`date`dan`title`argumen lapangan. Keduanya telah diformat sebelumnya melalui`util.dynamodb.toMapValues`pembantu untuk bekerja dengan tabel DynamoDB.

- `response`: Respons menerima konteks yang diperbarui dan mengembalikan hasil penangan permintaan.

d. PilihBuatsetelah kau selesai.

6. Kembali ke layar resolver, di bawahFungsi, pilihTambahkan fungsidrop-down dan tambahkan fungsi Anda ke daftar fungsi Anda.
7. PilihMenyimpanuntuk memperbarui resolver.

CLI

Untuk menambahkan fungsi Anda

- Buat fungsi untuk penyelesai pipeline Anda menggunakan`create-function`perintah.

  Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

  1. The`api-id`dari API Anda.
  2. The`name`dari fungsi diAWS AppSynckonsol.
  3. The`data-source-name`, atau nama sumber data yang akan digunakan fungsi. Itu harus sudah dibuat dan ditautkan ke API GraphQL Anda diAWS AppSynclayanan.
  4. The`runtime`, atau lingkungan dan bahasa fungsi. UntukJavaScript, nama itu harus`APPSYNC_JS`, dan runtime,`1.0.0`.
  5. The`code`, atau penangan permintaan dan respons fungsi Anda. Meskipun Anda dapat mengetiknya secara manual, jauh lebih mudah untuk menambahkannya ke file.txt (atau format serupa) kemudian meneruskannya sebagai argumen.

  > **Note**
  >
  > Kode kueri kami akan berada dalam file yang diteruskan sebagai argumen:

```
import { util } from '@aws-appsync/utils';

/**
 * Sends a request to `put` an item in the DynamoDB data source
 */
export function request(ctx) {
  return {
    operation: 'PutItem',
```

```
    key: util.dynamodb.toMapValues({id: util.autoId()}),
    attributeValues: util.dynamodb.toMapValues(ctx.args.input),
  };
}

/**
 * returns the result of the `put` operation
 */
export function response(ctx) {
  return ctx.result;
}
```

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-function \
--api-id abcdefghijklmnopqrstuvwxyz \
--name add_posts_func_1 \
--data-source-name table-for-posts \
--runtime name=APPSYNC_JS,runtimeVersion=1.0.0 \
--code file:///path/to/file/{filename}.{fileType}
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "functionConfiguration": {
        "functionId": "vulcmbfcxffiram63psb4dduoa",
        "functionArn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/functions/vulcmbfcxffiram63psb4dduoa",
        "name": "add_posts_func_1",
        "dataSourceName": "table-for-posts",
        "maxBatchSize": 0,
        "runtime": {
            "name": "APPSYNC_JS",
            "runtimeVersion": "1.0.0"
        },
        "code": "Code output foes here"
    }
}
```

> **Note**
>
> Pastikan Anda merekam`functionId`di suatu tempat karena ini akan digunakan untuk melampirkan fungsi ke resolver.

Untuk membuat resolver Anda

- Buat fungsi pipeline untuk`Mutation`dengan menjalankan`create-resolver`perintah.

  Anda harus memasukkan beberapa parameter untuk perintah khusus ini:

  1. The`api-id`dari API Anda.
  2. The`type-name`, atau jenis objek khusus dalam skema Anda (Kueri, Mutasi, Langganan).
  3. The`field-name`, atau operasi bidang di dalam jenis objek khusus yang ingin Anda lampirkan resolver.
  4. The`kind`, yang menentukan unit atau resolver pipa. Setel ini ke`PIPELINE`untuk mengaktifkan fungsi pipa.
  5. The`pipeline-config`, atau fungsi untuk melampirkan ke resolver. Pastikan Anda tahu`functionId`nilai-nilai fungsi Anda. Urutan daftar penting.
  6. The`runtime`, yang`APPSYNC_JS`(JavaScript). The`runtimeVersion`saat ini adalah`1.0.0`.
  7. The`code`, yang berisi langkah sebelum dan sesudah.

  > **Note**
  >
  > Kode kueri kami akan berada dalam file yang diteruskan sebagai argumen:

```
import { util } from '@aws-appsync/utils';

/**
 * Sends a request to `put` an item in the DynamoDB data source
 */
export function request(ctx) {
  const { id, ...values } = ctx.args;
  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({ id }),
```

```
    attributeValues: util.dynamodb.toMapValues(values),
  };
}

/**
 * returns the result of the `put` operation
 */
export function response(ctx) {
  return ctx.result;
}
```

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-resolver \
--api-id abcdefghijklmnopqrstuvwxyz \
--type-name Mutation \
--field-name createPost \
--kind PIPELINE \
--pipeline-config functions=vulcmbfcxffiram63psb4dduoa \
--runtime name=APPSYNC_JS,runtimeVersion=1.0.0 \
--code file:///path/to/file/{filename}.{fileType}
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "resolver": {
        "typeName": "Mutation",
        "fieldName": "createPost",
        "resolverArn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/Mutation/resolvers/createPost",
        "kind": "PIPELINE",
        "pipelineConfig": {
            "functions": [
                "vulcmbfcxffiram63psb4dduoa"
            ]
        },
        "maxBatchSize": 0,
        "runtime": {
            "name": "APPSYNC_JS",
            "runtimeVersion": "1.0.0"
        },
```

```
        "code": "Code output goes here"
    }
}
```

CDK

> **Tip**
>
> Sebelum Anda menggunakan CDK, kami sarankan untuk meninjau CDKdokumentasi resmibersama denganAWS AppSynciniReferensi CDK.
> Langkah-langkah yang tercantum di bawah ini hanya akan menampilkan contoh umum dari cuplikan yang digunakan untuk menambahkan sumber daya tertentu. Ini adalahtidakdimaksudkan untuk menjadi solusi yang berfungsi dalam kode produksi Anda. Kami juga menganggap Anda sudah memiliki aplikasi yang berfungsi.

- Untuk membuat mutasi, dengan asumsi Anda berada di proyek yang sama, Anda dapat menambahkannya ke file tumpukan seperti kueri. Berikut adalah fungsi dan penyelesai yang dimodifikasi untuk mutasi yang menambahkan yang baru`Post`ke meja:

```
const add_func_2 = new appsync.AppsyncFunction(this, 'func-add-post', {
  name: 'add_posts_func_1',
  add_api,
  dataSource: add_api.addDynamoDbDataSource('table-for-posts-2', add_ddb_table),
      code: appsync.Code.fromInline(`
          export function request(ctx) {
            return {
              operation: 'PutItem',
              key: util.dynamodb.toMapValues({id: util.autoId()}),
              attributeValues: util.dynamodb.toMapValues(ctx.args.input),
            };
          }

          export function response(ctx) {
            return ctx.result;
          }
      `),
  runtime: appsync.FunctionRuntime.JS_1_0_0,
});
```

```
new appsync.Resolver(this, 'pipeline-resolver-create-posts', {
  add_api,
  typeName: 'Mutation',
  fieldName: 'createPost',
      code: appsync.Code.fromInline(`
          export function request(ctx) {
            return {};
          }

          export function response(ctx) {
            return ctx.prev.result;
          }
      `),
  runtime: appsync.FunctionRuntime.JS_1_0_0,
  pipelineConfig: [add_func_2],
});
```

> **Note**
>
> Karena mutasi dan kueri ini terstruktur dengan cara yang sama, kami hanya akan menjelaskan perubahan yang kami buat untuk membuat mutasi.
> Dalam fungsinya, kami mengubah id CFN menjadi`func-add-post`dan nama untuk`add_posts_func_1`untuk mencerminkan fakta bahwa kami menambahkan`Posts`ke meja. Di sumber data, kami membuat asosiasi baru ke tabel kami (`add_ddb_table`) di AWS AppSync konsol sebagai`table-for-posts-2`karena`addDynamoDbDataSource`metode membutuhkannya. Perlu diingat, asosiasi baru ini masih menggunakan tabel yang sama yang kita buat sebelumnya, tetapi kita sekarang memiliki dua koneksi ke sana di AWS AppSync konsol: satu untuk kueri sebagai`table-for-posts`dan satu untuk mutasi sebagai`table-for-posts-2`. Kode diubah untuk menambahkan`Post`dengan menghasilkan`id`nilai secara otomatis dan menerima masukan klien untuk sisa bidang.
> Di resolver, kami mengubah nilai id menjadi`pipeline-resolver-create-posts`untuk mencerminkan fakta bahwa kami menambahkan`Posts`ke meja.
> Untuk mencerminkan mutasi dalam skema, nama tipe diubah menjadi`Mutation`, dan nama,`createPost`. Konfigurasi pipeline disetel ke fungsi mutasi baru kami`add_func_2`.

Untuk meringkas apa yang terjadi dalam contoh ini,AWS AppSyncsecara otomatis mengkonversi argumen yang didefinisikan dalam`createPost`bidang dari skema GraphQL Anda ke dalam operasi DynamoDB. Contoh menyimpan catatan di DynamoDB menggunakan kunci`id`, yang secara otomatis dibuat menggunakan`util.autoId()`penolong. Semua bidang lain yang Anda berikan ke argumen konteks (`ctx.args.input`) dari permintaan yang dibuat diAWS AppSynckonsol atau sebaliknya akan disimpan sebagai atribut tabel. Baik kunci dan atribut secara otomatis dipetakan ke format DynamoDB yang kompatibel menggunakan`util.dynamodb.toMapValues(values)`penolong.

AWS AppSyncjuga mendukung alur kerja pengujian dan debug untuk mengedit resolver. Anda bisa menggunakan tiruan`context`objek untuk melihat nilai yang diubah dari template sebelum memanggilnya. Secara opsional, Anda dapat melihat permintaan lengkap ke sumber data secara interaktif saat menjalankan kueri. Untuk informasi lebih lanjut, lihatUji dan debug resolver (JavaScript)danPemantauan dan pencatatan.

Resolver tingkat lanjut

Jika Anda mengikuti bagian pagination opsional diMerancang skema Anda, Anda masih perlu menambahkan resolver Anda ke permintaan Anda untuk menggunakan pagination. Contoh kami menggunakan pagination query yang disebut`getPosts`untuk mengembalikan hanya sebagian dari hal-hal yang diminta pada suatu waktu. Kode resolver kami di bidang itu mungkin terlihat seperti ini:

```
/**
 * Performs a scan on the dynamodb data source
 */
export function request(ctx) {
  const { limit = 20, nextToken } = ctx.args;
  return { operation: 'Scan', limit, nextToken };
}

/**
 * @returns the result of the `put` operation
 */
export function response(ctx) {
  const { items: posts = [], nextToken } = ctx.result;
  return { posts, nextToken };
}
```

Dalam permintaan, kami lulus dalam konteks permintaan. Kami`limit`adalah`20`, artinya kita kembali hingga 20`Posts`dalam kueri pertama. Kami`nextToken`kursor diperbaiki ke yang pertama`Post`entri di sumber data. Ini diteruskan ke argumen. Permintaan kemudian melakukan pemindaian dari yang

pertama`Post`hingga jumlah batas pemindaian. Sumber data menyimpan hasil dalam konteks, yang diteruskan ke respons. Respons mengembalikan`Posts`situ diambil, lalu mengatur`nextToken`diatur ke`Post`masuk tepat setelah batas. Permintaan berikutnya dikirim untuk melakukan hal yang sama persis tetapi dimulai dari offset tepat setelah kueri pertama. Perlu diingat bahwa permintaan semacam ini dilakukan secara berurutan dan tidak secara paralel.

Uji dan debug resolver (JavaScript)

AWS AppSyncmengeksekusi resolver pada bidang GraphQL terhadap sumber data. Saat bekerja dengan resolver pipeline, fungsi berinteraksi dengan sumber data Anda. Seperti yang dijelaskan dalamJavaScriptikhtisar penyelesai, fungsi berkomunikasi dengan sumber data dengan menggunakan penangan permintaan dan respons yang ditulis dalamJavaScriptdan berjalan di`APPSYNC_JS`runtime. Ini memungkinkan Anda untuk memberikan logika dan kondisi khusus sebelum dan sesudah berkomunikasi dengan sumber data.

Untuk membantu pengembang menulis, menguji, dan men-debug resolver ini,AWS AppSynckonsol juga menyediakan alat untuk membuat permintaan dan respons GraphQL dengan data tiruan ke penyelesai bidang individu. Selain itu, Anda dapat melakukan kueri, mutasi, dan langganan diAWS AppSynckonsol dan lihat aliran log terperinci dari seluruh permintaan dari AmazonCloudWatch. Ini termasuk hasil dari sumber data.

Pengujian dengan data tiruan

Ketika resolver GraphQL dipanggil, itu berisi`a`context`objek yang memiliki informasi yang relevan tentang permintaan tersebut. Ini termasuk argumen dari klien, informasi identitas, dan data dari bidang GraphQL induk. Ini juga menyimpan hasil dari sumber data, yang dapat digunakan dalam penangan respons. Untuk informasi lebih lanjut tentang struktur ini dan utilitas pembantu yang tersedia untuk digunakan saat pemrograman, lihatReferensi objek konteks penyelesai.

Saat menulis atau mengedit fungsi resolver, Anda dapat melewati amengejekataukonteks tesobjek ke editor konsol. Ini memungkinkan Anda untuk melihat bagaimana permintaan dan penangan respons mengevaluasi tanpa benar-benar berjalan terhadap sumber data. Misalnya, Anda dapat lulus tes`firstname: Shaggy`argumen dan lihat bagaimana evaluasi saat menggunakan`ctx.args.firstname`dalam kode template Anda. Anda juga dapat menguji evaluasi setiap pembantu utilitas seperti`util.autoId()`atau`util.time.nowISO8601()`.

Menguji penyelesai

Contoh ini akan menggunakanAWS AppSynckonsol untuk menguji resolver.

1. Masuk keAWS Management Consoledan bukaAppSynckonsol.
   a. DiDasbor API, pilih API GraphQL Anda.
   b. DiSidebar, pilihFungsi.
2. Pilih fungsi yang ada.
3. Di bagian atasPerbarui fungsihalaman, pilihPilih konteks pengujian, lalu pilihBuat konteks baru.
4. Pilih objek konteks sampel atau isi JSON secara manual diKonfigurasikan konteks pengujianjendela di bawah ini.
5. Masukkan aNama konteks teks.
6. Pilih tombol Simpan.
7. Untuk mengevaluasi resolver Anda menggunakan objek konteks tiruan ini, pilihJalankan Uji.

Untuk contoh yang lebih praktis, misalkan Anda memiliki aplikasi yang menyimpan tipe GraphQLDogyang menggunakan pembuatan ID otomatis untuk objek dan menyimpannya di Amazon DynamoDB. Anda juga ingin menulis beberapa nilai dari argumen mutasi GraphQL dan hanya mengizinkan pengguna tertentu untuk melihat respons. Cuplikan berikut menunjukkan seperti apa skema itu:

```
type Dog {
  breed: String
  color: String
}

type Mutation {
  addDog(firstname: String, age: Int): Dog
}
```

Anda dapat menulis sebuahAWS AppSyncfungsi dan menambahkannya ke AndaaddDogresolver untuk menangani mutasi. Untuk menguji AndaAWS AppSyncfungsi, Anda dapat mengisi objek konteks seperti contoh berikut. Berikut ini memiliki argumen dari klien darinamedanage, dan ausernamedihuni diidentityobjek:

```
{
    "arguments" : {
        "firstname": "Shaggy",
        "age": 4
    },
```

```
    "source" : {},
    "result" : {
        "breed" : "Miniature Schnauzer",
        "color" : "black_grey"
    },
    "identity": {
        "sub" : "uuid",
        "issuer" : " https://cognito-idp.{region}.amazonaws.com/{userPoolId}",
        "username" : "Nadia",
        "claims" : { },
        "sourceIp" :[  "x.x.x.x" ],
        "defaultAuthStrategy" : "ALLOW"
    }
}
```

Anda dapat mengujiAWS AppSyncfungsi menggunakan kode berikut:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({ id: util.autoId() }),
    attributeValues: util.dynamodb.toMapValues(ctx.args),
  };
}

export function response(ctx) {
  if (ctx.identity.username === 'Nadia') {
    console.log("This request is allowed")
    return ctx.result;
  }
  util.unauthorized();
}
```

Penangan permintaan dan respons yang dievaluasi memiliki data dari objek konteks pengujian Anda dan nilai yang dihasilkan dari`util.autoId()`. Selain itu, jika Anda mengubah`username`untuk nilai selain`Nadia`, hasilnya tidak akan dikembalikan karena pemeriksaan otorisasi akan gagal. Untuk informasi selengkapnya tentang kontrol akses berbutir halus, lihatKasus penggunaan otorisasi.

Menguji permintaan dan penangan respons denganAWS AppSyncAPI

Anda dapat menggunakan`EvaluateCode`Perintah API untuk menguji kode Anda dari jarak jauh dengan data tiruan. Untuk memulai dengan perintah, pastikan Anda telah menambahkan`appsync:evaluateMappingCode`izin untuk kebijakan Anda. Misalnya:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "appsync:evaluateCode",
            "Resource": "arn:aws:appsync:<region>:<account>:*"
        }
    ]
}
```

Anda dapat memanfaatkan perintah dengan menggunakanAWS CLIatauAWSSDK. Misalnya, ambilDogskema dan nyaAWS AppSyncpermintaan fungsi dan penangan respons dari bagian sebelumnya. Menggunakan CLI di stasiun lokal Anda, simpan kode ke file bernama`code.js`, lalu simpan`context`objek ke file bernama`context.json`. Dari shell Anda, jalankan perintah berikut:

```
$ aws appsync evaluate-code \
  --code file://code.js \
  --function response \
  --context file://context.json \
  --runtime name=APPSYNC_JS,runtimeVersion=1.0.0
```

Tanggapan tersebut berisi sebuah`evaluationResult`berisi muatan yang dikembalikan oleh handler Anda. Ini juga berisi`logs`objek, yang menyimpan daftar log yang dihasilkan oleh handler Anda selama evaluasi. Hal ini memudahkan untuk men-debug eksekusi kode Anda dan melihat informasi tentang evaluasi Anda untuk membantu memecahkan masalah. Misalnya:

```
{
    "evaluationResult": "{\"breed\":\"Miniature Schnauzer\",\"color\":\"black_grey\"}",
    "logs": [
        "INFO - code.js:13:5: \"This request is allowed\""
    ]
}
```

The`evaluationResult`dapat diurai sebagai JSON, yang memberikan:

```
{
  "breed": "Miniature Schnauzer",
  "color": "black_grey"
}
```

Dengan menggunakan SDK, Anda dapat dengan mudah menggabungkan pengujian dari rangkaian pengujian favorit untuk memvalidasi perilaku penangan Anda. Kami merekomendasikan membuat tes menggunakanKerangka Pengujian Jest, tetapi rangkaian pengujian apa pun berfungsi. Cuplikan berikut menunjukkan validasi hipotetis berjalan. Perhatikan bahwa kami mengharapkan respons evaluasi menjadi JSON yang valid, jadi kami menggunakan`JSON.parse`untuk mengambil JSON dari respons string:

```
const AWS = require('aws-sdk')
const fs = require('fs')
const client = new AWS.AppSync({ region: 'us-east-2' })
const runtime = {name:'APPSYNC_JS',runtimeVersion:'1.0.0')

test('request correctly calls DynamoDB', async () => {
  const code = fs.readFileSync('./code.js', 'utf8')
  const context = fs.readFileSync('./context.json', 'utf8')
  const contextJSON = JSON.parse(context)

  const response = await client.evaluateCode({ code, context, runtime, function:
 'request' }).promise()
  const result = JSON.parse(response.evaluationResult)

  expect(result.key.id.S).toBeDefined()
  expect(result.attributeValues.firstname.S).toEqual(contextJSON.arguments.firstname)
})
```

Ini menghasilkan hasil sebagai berikut:

```
Ran all test suites.
> jest

PASS ./index.test.js
# request correctly calls DynamoDB (543 ms)
Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 totalTime: 1.511 s, estimated 2 s
```

Mendebug kueri langsung

Tidak ada pengganti untuk pengujian end-to-end dan logging untuk men-debug aplikasi produksi.AWS AppSyncmemungkinkan Anda mencatat kesalahan dan detail permintaan lengkap menggunakan AmazonCloudWatch. Selain itu, Anda dapat menggunakanAWS AppSynckonsol untuk menguji kueri, mutasi, dan langganan GraphQL dan data log streaming langsung untuk setiap permintaan kembali ke editor kueri untuk di-debug secara real time. Untuk langganan, log menampilkan informasi waktu koneksi.

Untuk melakukan ini, Anda harus memiliki AmazonCloudWatchlog diaktifkan sebelumnya, seperti yang dijelaskan dalamPemantauan dan pencatatan. Selanjutnya, diAWS AppSynckonsol, pilihPertanyaantab dan kemudian masukkan kueri GraphQL yang valid. Di bagian kanan bawah, klik dan seretLogjendela untuk membuka tampilan log. Di bagian atas halaman, pilih ikon panah putar untuk menjalankan kueri GraphQL Anda. Dalam beberapa saat, log permintaan dan respons lengkap Anda untuk operasi dialirkan ke bagian ini dan Anda dapat melihatnya di konsol.

Penyelesai pipa (JavaScript)

AWS AppSyncmengeksekusi resolver pada bidang GraphQL. Dalam beberapa kasus, aplikasi memerlukan eksekusi beberapa operasi untuk menyelesaikan satu bidang GraphQL. Dengan resolver pipeline, pengembang sekarang dapat membuat operasi yang disebut Fungsi dan menjalankannya secara berurutan. Pipeline resolver berguna untuk aplikasi yang, misalnya, memerlukan pemeriksaan otorisasi sebelum mengambil data untuk bidang.

Untuk informasi lebih lanjut tentang arsitekturJavaScriptpenyelesai pipa, lihatJavaScriptikhtisar penyelesai.

Buat resolver pipa

DiAWS AppSynckonsol, pergi keSkemahalaman.

Simpan skema berikut:

```
schema {
    query: Query
    mutation: Mutation
}

type Mutation {
    signUp(input: Signup): User
```

```
}

type Query {
    getUser(id: ID!): User
}

input Signup {
    username: String!
    email: String!
}

type User {
    id: ID!
    username: String
    email: AWSEmail
}
```

Kami akan memasang resolver pipa keMendaftarlapangan diMutasijenis. DiMutasiketik di sisi kanan, pilihLampirkandi sebelah`signUp`bidang mutasi. Atur resolver ke`pipeline resolver`dan`APPSYNC_JS`runtime, lalu buat resolver.

Pipeline resolver kami mendaftarkan pengguna dengan terlebih dahulu memvalidasi input alamat email dan kemudian menyimpan pengguna dalam sistem. Kami akan merangkum validasi email di dalam`ValidateEmail`fungsi dan penyimpanan pengguna di dalam`SaveUser`fungsi. The`ValidateEmail`fungsi dijalankan terlebih dahulu, dan jika email valid, maka`SaveUser`fungsi mengeksekusi.

Alur eksekusi adalah sebagai berikut:

1. Mutation.signup penangan permintaan resolver
2. fungsi validateEmail
3. fungsi SaveUser
4. Penangan respons penyelesai mutasi.signup

Karena kita mungkin akan menggunakan kembali`ValidateEmail`berfungsi di resolver lain di API kami, kami ingin menghindari mengakses`ctx.args`karena ini akan berubah dari satu bidang GraphQL ke bidang lainnya. Sebagai gantinya, kita bisa menggunakan`ctx.stash`untuk menyimpan atribut email dari`signUp(input: Signup)`argumen bidang masukan.

Perbarui kode resolver Anda dengan mengganti fungsi permintaan dan respons Anda:

```
export function request(ctx) {
    ctx.stash.email = ctx.args.input.email
    return {};
}

export function response(ctx) {
    return ctx.prev.result;
}
```

PilihBuatatauMenyimpanuntuk memperbarui resolver.

Buat fungsi

Dari halaman penyelesai pipa, diFungsibagian, klikTambahkan fungsi, makaBuat fungsi baru. Dimungkinkan juga untuk membuat fungsi tanpa melalui halaman resolver; untuk melakukan ini, diAWS AppSynckonsol, pergi keFungsihalaman. PilihBuat fungsitombol. Mari buat fungsi yang memeriksa apakah email valid dan berasal dari domain tertentu. Jika email tidak valid, fungsi tersebut menimbulkan kesalahan. Jika tidak, itu meneruskan input apa pun yang diberikan.

Pastikan Anda telah membuat sumber dataTIDAK ADAjenis. Pilih sumber data ini diNama sumber datadaftar. Untuknama fungsi, masuk`validateEmail`. Dikode fungsiarea, timpa semuanya dengan cuplikan ini:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const { email } = ctx.stash;
  const valid = util.matches(
    '^[a-zA-Z0-9_.+-]+@(?:(?:[a-zA-Z0-9-]+\.)?[a-zA-Z]+\.)?(myvaliddomain)\.com',
    email
  );
  if (!valid) {
    util.error(`"${email}" is not a valid email.`);
  }

  return { payload: { email } };
}

export function response(ctx) {
  return ctx.result;
}
```

Tinjau input Anda, lalu pilihBuat. Kami baru saja menciptakanValidateEmailfungsi. Ulangi langkah-langkah ini untuk membuatSaveUserfungsi dengan kode berikut (Demi kesederhanaan, kami menggunakanTIDAK ADAsumber data dan berpura-pura pengguna telah disimpan dalam sistem setelah fungsi dijalankan.):

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  return ctx.prev.result;
}

export function response(ctx) {
  ctx.result.id = util.autoId();
  return ctx.result;
}
```

Kami baru saja menciptakanSaveUserfungsi.

Menambahkan fungsi ke resolver pipa

Fungsi kita seharusnya ditambahkan secara otomatis ke resolver pipeline yang baru saja kita buat. Jika ini bukan masalahnya, atau Anda membuat fungsi melaluiFungsihalaman, Anda dapat mengklikTambahkan fungsikembali pada`signUp`halaman resolver untuk melampirkannya. Tambahkan keduaValidateEmaildanSaveUserberfungsi untuk resolver. TheValidateEmailfungsi harus ditempatkan sebelumSaveUserfungsi. Saat Anda menambahkan lebih banyak fungsi, Anda dapat menggunakanbergerak ke atasdanbergerak ke bawahopsi untuk mengatur ulang urutan eksekusi fungsi Anda. Tinjau perubahan Anda, lalu pilihMenyimpan.

Menjalankan kueri

DiAWS AppSynckonsol, pergi kePertanyaanhalaman. Di penjelajah, pastikan Anda menggunakan mutasi Anda. Jika tidak, pilih`Mutation`di daftar drop-down, lalu pilih+. Masukkan kueri berikut:

```
mutation {
  signUp(input: {email: "nadia@myvaliddomain.com", username: "nadia"}) {
    id
    username
  }
}
```

Ini harus mengembalikan sesuatu seperti:

```
{
  "data": {
    "signUp": {
      "id": "256b6cc2-4694-46f4-a55e-8cb14cc5d7fc",
      "username": "nadia"
    }
  }
}
```

Kami telah berhasil mendaftarkan pengguna kami dan memvalidasi email input menggunakan resolver pipeline.

Mengkonfigurasi resolver (VTL)

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

GraphQL resolver menghubungkan bidang dalam skema tipe ke sumber data. Resolver adalah mekanisme dimana permintaan dipenuhi. AWS AppSync dapat secara otomatis membuat dan menghubungkan resolver dari skema atau membuat skema dan menghubungkan resolver dari tabel yang ada tanpa Anda perlu menulis kode apa pun.

Resolver AWS AppSync digunakan JavaScript untuk mengonversi ekspresi GraphQL menjadi format yang dapat digunakan sumber data. Atau, template pemetaan dapat ditulis dalam Apache Velocity Template Language (VTL) untuk mengubah ekspresi GraphQL menjadi format yang dapat digunakan sumber data.

Bagian ini akan menunjukkan cara mengkonfigurasi resolver menggunakan VTL. Panduan pemrograman gaya tutorial pengantar untuk menulis resolver dapat ditemukan di panduan pemrograman template pemetaan Resolver, dan utilitas pembantu yang tersedia untuk digunakan ketika pemrograman dapat ditemukan di referensi konteks template pemetaan Resolver. AWS AppSync juga memiliki alur pengujian dan debug bawaan yang dapat Anda gunakan saat mengedit atau menulis dari awal. Untuk informasi selengkapnya, lihat Menguji dan men-debug resolver.

Sebaiknya ikuti panduan ini sebelum mencoba menggunakan salah satu tutorial yang disebutkan di atas.

Di bagian ini, kita akan membahas cara membuat resolver, menambahkan resolver untuk mutasi, dan menggunakan konfigurasi lanjutan.

Buat resolver pertama Anda

Mengikuti contoh dari bagian sebelumnya, langkah pertama adalah membuat resolver untuk tipe Anda`Query`.

Console

1. Masuk ke AWS Management Console dan buka AppSynckonsol.

   a. Di dasbor API, pilih GraphQL API Anda.

   b. Di Sidebar, pilih Skema.

2. Di sisi kanan halaman, ada jendela yang disebut Resolvers. Kotak ini berisi daftar jenis dan bidang seperti yang didefinisikan dalam jendela Skema Anda di sisi kiri halaman. Anda dapat melampirkan resolver ke bidang. Misalnya, di bawah Jenis kueri, pilih Lampirkan di sebelah `getTodos` bidang.

3. Pada halaman Create Resolver, pilih sumber data yang Anda buat dalam panduan Melampirkan sumber data. Di jendela Konfigurasikan templat pemetaan, Anda dapat memilih templat pemetaan permintaan umum dan respons menggunakan daftar drop-down di sebelah kanan atau menulis sendiri.

   > **Note**
   >
   > Pasangan template pemetaan permintaan ke template pemetaan respons disebut resolver unit. Resolver unit biasanya dimaksudkan untuk melakukan operasi hafalan; kami sarankan menggunakannya hanya untuk operasi tunggal dengan sejumlah kecil sumber data. Untuk operasi yang lebih kompleks, sebaiknya gunakan resolver pipa, yang dapat menjalankan beberapa operasi dengan beberapa sumber data secara berurutan.
   > Untuk informasi selengkapnya tentang perbedaan antara templat pemetaan permintaan dan respons, lihat Resolver unit.
   > Untuk informasi selengkapnya tentang penggunaan resolver pipeline, lihat Pipeline resolvers.

4. Untuk kasus penggunaan umum, AWS AppSync konsol memiliki templat bawaan yang dapat Anda gunakan untuk mendapatkan item dari sumber data (misalnya, semua kueri item,

pencarian individual, dll.). Misalnya, pada versi sederhana skema dari Merancang skema Anda di mana `getTodos` tidak memiliki pagination, template pemetaan permintaan untuk item daftar adalah sebagai berikut:

```
{
    "version" : "2017-02-28",
    "operation" : "Scan"
}
```

5. Anda selalu membutuhkan template pemetaan respons untuk menyertai permintaan. Konsol menyediakan default dengan nilai passthrough berikut untuk daftar:

```
$util.toJson($ctx.result.items)
```

Dalam contoh ini, `context` objek (alias sebagai`$ctx`) untuk daftar item memiliki formulir`$context.result.items`. Jika operasi GraphQL Anda mengembalikan satu item, itu akan menjadi. `$context.result` AWS AppSyncmenyediakan fungsi pembantu untuk operasi umum, seperti `$util.toJson` fungsi yang tercantum sebelumnya, untuk memformat respons dengan benar. Untuk daftar lengkap fungsi, lihat Referensi utilitas template pemetaan Resolver.

6. Pilih Simpan Resolver.

API

1. Buat objek resolver dengan memanggil API. `CreateResolver`
2. Anda dapat memodifikasi bidang resolver dengan memanggil API. `UpdateResolver`

CLI

1. Buat resolver dengan menjalankan perintah. `create-resolver`

   Anda harus mengetikkan 6 parameter untuk perintah khusus ini:

   1. API Anda. `api-id`
   2. `type-name`Jenis yang ingin Anda modifikasi dalam skema Anda. Dalam contoh konsol, ini adalah`Query`.

3. `field-name`Bidang yang ingin Anda modifikasi dalam tipe Anda. Dalam contoh konsol, ini adalah`getTodos`.
4. Sumber data yang Anda buat dalam panduan Melampirkan sumber data. `data-source-name`
5. Iture`request-mapping-template`, yang merupakan badan permintaan. Dalam contoh konsol, ini adalah:

```
{
    "version" : "2017-02-28",
    "operation" : "Scan"
}
```

6. Iture`response-mapping-template`, yang merupakan tubuh respons. Dalam contoh konsol, ini adalah:

```
$util.toJson($ctx.result.items)
```

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-resolver --api-id abcdefghijklmnopqrstuvwxyz --type-name
 Query --field-name getTodos --data-source-name TodoTable --request-mapping-
template "{ "version" : "2017-02-28", "operation" : "Scan", }" --response-
mapping-template ""$"util.toJson("$"ctx.result.items)"
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "resolver": {
        "kind": "UNIT",
        "dataSourceName": "TodoTable",
        "requestMappingTemplate": "{ version : 2017-02-28, operation : Scan, }",
        "resolverArn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/Query/resolvers/getTodos",
        "typeName": "Query",
        "fieldName": "getTodos",
        "responseMappingTemplate": "$util.toJson($ctx.result.items)"
    }
}
```

2. Untuk memodifikasi bidang resolver dan/atau template pemetaan, jalankan perintah. `update-resolver`

   Dengan pengecualian `api-id` parameter, parameter yang digunakan dalam `create-resolver` perintah akan ditimpa oleh nilai-nilai baru dari `update-resolver` perintah.

Menambahkan resolver untuk mutasi

Langkah selanjutnya adalah membuat resolver untuk tipe Anda`Mutation`.

Console

1. Masuk ke AWS Management Console dan buka AppSynckonsol.

   a. Di dasbor API, pilih GraphQL API Anda.

   b. Di Sidebar, pilih Skema.

2. Di bawah jenis Mutasi, pilih Lampirkan di sebelah `addTodo` bidang.
3. Pada halaman Create Resolver, pilih sumber data yang Anda buat dalam panduan Melampirkan sumber data.
4. Di jendela Configure mapping templates, Anda harus memodifikasi template permintaan karena ini adalah mutasi di mana Anda menambahkan item baru ke DynamoDB. Gunakan template pemetaan permintaan berikut:

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },
    "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
}
```

5. AWS AppSync secara otomatis mengonversi argumen yang ditentukan di `addTodo` bidang dari skema GraphQL Anda menjadi operasi DynamoDB. Contoh sebelumnya menyimpan catatan di DynamoDB menggunakan kunci `id` dari, yang dilewatkan dari argumen mutasi sebagai. `$ctx.args.id` Semua bidang lain yang Anda lewati secara otomatis dipetakan ke atribut DynamoDB. `$util.dynamodb.toMapValuesJson($ctx.args)`

   Untuk resolver ini, gunakan template pemetaan respons berikut:

```
$util.toJson($ctx.result)
```

AWS AppSync juga mendukung alur kerja pengujian dan debug untuk mengedit resolver. Anda dapat menggunakan `context` objek tiruan untuk melihat nilai template yang diubah sebelum memanggil. Secara opsional, Anda dapat melihat eksekusi permintaan penuh ke sumber data secara interaktif saat menjalankan kueri. Untuk informasi selengkapnya, lihat Menguji dan men-debug resolver serta Monitoring dan logging.

6. Pilih Simpan Resolver.

API

Anda juga dapat melakukan ini dengan API dengan memanfaatkan perintah di bagian Create your first resolver dan detail parameter dari bagian ini.

CLI

Anda juga dapat melakukan ini di CLI dengan memanfaatkan perintah di bagian Create your first resolver dan rincian parameter dari bagian ini.

Pada titik ini, jika Anda tidak menggunakan resolver lanjutan, Anda dapat mulai menggunakan GraphQL API seperti yang diuraikan dalam Menggunakan API Anda.

Resolver tingkat lanjut

Jika Anda mengikuti bagian Advanced dan Anda sedang membuat skema sampel dalam Merancang skema Anda untuk melakukan pemindaian paginasi, gunakan template permintaan berikut untuk bidang sebagai gantinya: `getTodos`

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "limit": $util.defaultIfNull(${ctx.args.limit}, 20),
    "nextToken": $util.toJson($util.defaultIfNullOrBlank($ctx.args.nextToken, null))
}
```

Untuk kasus penggunaan pagination ini, pemetaan respons lebih dari sekadar passthrough karena harus berisi kursor (sehingga klien tahu halaman apa yang akan dimulai selanjutnya) dan set hasil. Template pemetaan adalah sebagai berikut:

```
{
    "todos": $util.toJson($context.result.items),
    "nextToken": $util.toJson($context.result.nextToken)
}
```

Bidang dalam template pemetaan respons sebelumnya harus sesuai dengan bidang yang ditentukan dalam tipe Anda. `TodoConnection`

Untuk kasus relasi di mana Anda memiliki `Comments` tabel dan Anda menyelesaikan kolom komentar pada `Todo` tipe (yang mengembalikan tipe`[Comment]`), Anda dapat menggunakan template pemetaan yang menjalankan kueri terhadap tabel kedua. Untuk melakukan ini, Anda harus sudah membuat sumber data untuk `Comments` tabel seperti yang diuraikan dalam Melampirkan sumber data.

> **Note**
>
> Kami menggunakan operasi kueri terhadap tabel kedua hanya untuk tujuan ilustrasi. Anda dapat menggunakan operasi lain terhadap DynamoDB sebagai gantinya. Selain itu, Anda dapat menarik data dari sumber data lain, seperti AWS Lambda atau Amazon OpenSearch Service, karena relasinya dikendalikan oleh skema GraphQL Anda.

Console

1. Masuk ke AWS Management Console dan buka AppSynckonsol.
   a. Di dasbor API, pilih GraphQL API Anda.
   b. Di Sidebar, pilih Skema.
2. Di bawah tipe Todo, pilih Lampirkan di sebelah `comments` bidang.
3. Pada halaman Create Resolver, pilih sumber data tabel Komentar Anda. Nama default untuk tabel Komentar dari panduan mulai cepat adalah`AppSyncCommentTable`, tetapi dapat bervariasi tergantung pada nama yang Anda berikan.
4. Tambahkan cuplikan berikut ke template pemetaan permintaan Anda:

```
{
    "version": "2017-02-28",
    "operation": "Query",
    "index": "todoid-index",
```

```
    "query": {
        "expression": "todoid = :todoid",
        "expressionValues": {
            ":todoid": {
                "S": $util.toJson($context.source.id)
            }
        }
    }
}
```

5. `context.source`Referensi objek induk dari bidang saat ini yang sedang diselesaikan. Dalam contoh ini, `source.id` mengacu pada `Todo` objek individu, yang kemudian digunakan untuk ekspresi query.

   Anda dapat menggunakan template pemetaan respons passthrough sebagai berikut:

   ```
   $util.toJson($ctx.result.items)
   ```

6. Pilih Simpan Resolver.
7. Terakhir, kembali ke halaman Skema di konsol, lampirkan resolver ke `addComment` bidang, dan tentukan sumber data untuk tabel. `Comments` Templat pemetaan permintaan dalam hal ini sederhana `PutItem` dengan spesifik `todoid` yang dikomentari sebagai argumen, tetapi Anda menggunakan `$utils.autoId()` utilitas untuk membuat kunci pengurutan unik untuk komentar sebagai berikut:

   ```
   {
       "version": "2017-02-28",
       "operation": "PutItem",
       "key": {
           "todoid": { "S": $util.toJson($context.arguments.todoid) },
           "commentid": { "S": "$util.autoId()" }
       },
       "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
   }
   ```

   Gunakan template respon passthrough sebagai berikut:

   ```
   $util.toJson($ctx.result)
   ```

API

Anda juga dapat melakukan ini dengan API dengan memanfaatkan perintah di bagian Create your first resolver dan detail parameter dari bagian ini.

CLI

Anda juga dapat melakukan ini di CLI dengan memanfaatkan perintah di bagian Create your first resolver dan rincian parameter dari bagian ini.

Penyelesai Lambda Langsung (VTL)

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Dengan penyelesai Lambda langsung, Anda dapat menghindari penggunaan templat pemetaan VTL saat menggunakan sumber data. AWS Lambda AWS AppSync dapat memberikan payload default ke fungsi Lambda Anda serta terjemahan default dari respons fungsi Lambda ke tipe GraphQL. Anda dapat memilih untuk memberikan template permintaan, template respons, atau tidak keduanya dan AWS AppSync akan menanganinya sesuai dengan itu.

Untuk mempelajari selengkapnya tentang payload permintaan default dan terjemahan respons yang AWS AppSync disediakan, lihat referensi penyelesai Lambda Langsung. Untuk informasi selengkapnya tentang menyiapkan sumber AWS Lambda data dan menyiapkan Kebijakan Kepercayaan IAM, lihat Melampirkan sumber data.

Konfigurasikan penyelesai Lambda langsung

Bagian berikut akan menunjukkan cara melampirkan sumber data Lambda dan menambahkan resolver Lambda ke bidang Anda.

Tambahkan sumber data Lambda

Sebelum Anda dapat mengaktifkan resolver Lambda langsung, Anda harus menambahkan sumber data Lambda.

Console

1. Masuk ke AWS Management Console dan buka AppSynckonsol.

   a. Di dasbor API, pilih GraphQL API Anda.

   b. Di Sidebar, pilih Sumber data.

2. Pilih Buat sumber data.

   a. Untuk nama sumber data, masukkan nama untuk sumber data Anda, seperti**`myFunction`**.

   b. Untuk tipe sumber data, pilih AWS Lambdafungsi.

   c. Untuk Wilayah, pilih wilayah yang sesuai.

   d. Untuk Fungsi ARN, pilih fungsi Lambda dari daftar dropdown. Anda dapat mencari nama fungsi atau secara manual memasukkan ARN dari fungsi yang ingin Anda gunakan.

   e. Buat peran IAM baru (disarankan) atau pilih peran yang sudah ada yang memiliki izin `lambda:invokeFunction` IAM. Peran yang ada memerlukan kebijakan kepercayaan, seperti yang dijelaskan di bagian Melampirkan sumber data.

      Berikut ini adalah contoh kebijakan IAM yang memiliki izin yang diperlukan untuk melakukan operasi pada sumber daya:

```
{
     "Version": "2012-10-17",
     "Statement": [
         {
             "Effect": "Allow",
             "Action": [ "lambda:invokeFunction" ],
             "Resource": [
                 "arn:aws:lambda:us-
west-2:123456789012:function:myFunction",
                 "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:*"
             ]
         }
     ]
 }
```

3. Pilih tombol Buat.

CLI

1. Buat objek sumber data dengan menjalankan `create-data-source`perintah.

   Anda harus mengetikkan 4 parameter untuk perintah khusus ini:

   1. API Anda. `api-id`
   2. `name`Dari sumber data Anda. Dalam contoh konsol, ini adalah nama sumber data.
   3. `type`Sumber data. Dalam contoh konsol, ini adalah AWS Lambdafungsi.
   4. The`lambda-config`, yang merupakan Fungsi ARN dalam contoh konsol.

   > **Note**
   >
   > Ada parameter lain seperti `Region` yang harus dikonfigurasi tetapi biasanya akan default ke nilai konfigurasi CLI Anda.

   Contoh perintah mungkin terlihat seperti ini:

   ```
   aws appsync create-data-source --api-id abcdefghijklmnopqrstuvwxyz
    --name myFunction --type AWS_LAMBDA --lambda-config
    lambdaFunctionArn=arn:aws:lambda:us-west-2:102847592837:function:appsync-
   lambda-example
   ```

   Output akan dikembalikan dalam CLI. Inilah contohnya:

   ```
   {
       "dataSource": {
           "dataSourceArn": "arn:aws:appsync:us-west-2:102847592837:apis/
   abcdefghijklmnopqrstuvwxyz/datasources/myFunction",
           "type": "AWS_LAMBDA",
           "name": "myFunction",
           "lambdaConfig": {
               "lambdaFunctionArn": "arn:aws:lambda:us-
   west-2:102847592837:function:appsync-lambda-example"
           }
       }
   }
   ```

2. Untuk memodifikasi atribut sumber data, jalankan `update-data-source`perintah.

   Dengan pengecualian `api-id` parameter, parameter yang digunakan dalam `create-data-source` perintah akan ditimpa oleh nilai-nilai baru dari `update-data-source` perintah.

Aktifkan penyelesai Lambda langsung

Setelah membuat sumber data Lambda dan menyiapkan peran IAM yang sesuai untuk memungkinkan menjalankan fungsi, Anda dapat AWS AppSync menautkannya ke fungsi resolver atau pipeline.

Console

1. Masuk ke AWS Management Console dan buka AppSynckonsol.
   a. Di dasbor API, pilih GraphQL API Anda.
   b. Di Sidebar, pilih Skema.
2. Di jendela Resolvers, pilih bidang atau operasi dan kemudian pilih tombol Lampirkan.
3. Di halaman Create new resolver, pilih fungsi Lambda dari daftar dropdown.
4. Untuk memanfaatkan penyelesai Lambda langsung, konfirmasikan bahwa templat pemetaan permintaan dan respons dinonaktifkan di bagian Konfigurasi templat pemetaan.
5. Pilih tombol Simpan Resolver.

CLI

- Buat resolver dengan menjalankan perintah. `create-resolver`

  Anda harus mengetikkan 6 parameter untuk perintah khusus ini:

  1. API Anda. `api-id`
  2. `type-name`Jenis dalam skema Anda.
  3. `field-name`Bidang dalam skema Anda.
  4. Nama`data-source-name`, atau fungsi Lambda Anda.
  5. Itu`request-mapping-template`, yang merupakan badan permintaan. Dalam contoh konsol, ini dinonaktifkan:

```
" "
```

6. Ituresponse-mapping-template, yang merupakan tubuh respons. Dalam contoh konsol, ini juga dinonaktifkan:

```
" "
```

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync create-resolver --api-id abcdefghijklmnopqrstuvwxyz --type-name
 Subscription --field-name onCreateTodo --data-source-name LambdaTest --request-
mapping-template " " --response-mapping-template " "
```

Output akan dikembalikan dalam CLI. Inilah contohnya:

```
{
    "resolver": {
        "resolverArn": "arn:aws:appsync:us-west-2:102847592837:apis/
abcdefghijklmnopqrstuvwxyz/types/Subscription/resolvers/onCreateTodo",
        "typeName": "Subscription",
        "kind": "UNIT",
        "fieldName": "onCreateTodo",
        "dataSourceName": "LambdaTest"
    }
}
```

Ketika Anda menonaktifkan template pemetaan Anda, ada beberapa perilaku tambahan yang akan terjadi diAWS AppSync:

- Dengan menonaktifkan template pemetaan, Anda memberi sinyal AWS AppSync bahwa Anda menerima terjemahan data default yang ditentukan dalam referensi penyelesai Lambda Langsung.
- Dengan menonaktifkan template pemetaan permintaan, sumber data Lambda Anda akan menerima payload yang terdiri dari seluruh objek Context.
- Dengan menonaktifkan template pemetaan respons, hasil pemanggilan Lambda Anda akan diterjemahkan tergantung pada versi templat pemetaan permintaan atau jika templat pemetaan permintaan juga dinonaktifkan.

Uji dan debug resolver (VTL)

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync mengeksekusi resolver pada bidang GraphQL terhadap sumber data. Seperti yang dijelaskan dalam ikhtisar template pemetaan Resolver, resolver berkomunikasi dengan sumber data dengan menggunakan bahasa templating. Ini memungkinkan Anda untuk menyesuaikan perilaku dan menerapkan logika dan kondisi sebelum dan sesudah berkomunikasi dengan sumber data. Untuk panduan pemrograman gaya tutorial pengantar untuk menulis resolver, lihat panduan pemrograman template pemetaan Resolver.

Untuk membantu pengembang menulis, menguji, dan men-debug resolver ini, AWS AppSync konsol juga menyediakan alat untuk membuat permintaan dan respons GraphQL dengan data tiruan ke penyelesai bidang individual. Selain itu, Anda dapat melakukan kueri, mutasi, dan langganan di AWS AppSync konsol dan melihat aliran log terperinci dari Amazon CloudWatch dari seluruh permintaan. Ini termasuk hasil dari sumber data.

Pengujian dengan data tiruan

Ketika resolver GraphQL dipanggil, itu berisi objek yang berisi informasi tentang `context` permintaan. Ini termasuk argumen dari klien, informasi identitas, dan data dari bidang GraphQL induk. Ini juga berisi hasil dari sumber data, yang dapat digunakan dalam template respons. Untuk informasi selengkapnya tentang struktur ini dan utilitas pembantu yang tersedia untuk digunakan saat pemrograman, lihat Referensi Konteks Template Pemetaan Resolver.

Saat menulis atau mengedit resolver, Anda dapat meneruskan objek konteks tiruan atau pengujian ke editor konsol. Ini memungkinkan Anda untuk melihat bagaimana permintaan dan template respons mengevaluasi tanpa benar-benar berjalan terhadap sumber data. Misalnya, Anda dapat lulus `firstname: Shaggy` argumen pengujian dan melihat bagaimana itu mengevaluasi saat menggunakan `$ctx.args.firstname` dalam kode template Anda. Anda juga dapat menguji evaluasi setiap pembantu utilitas seperti `$util.autoId()` atau`util.time.nowISO8601()`.

Menguji penyelesai

Contoh ini akan menggunakan AWS AppSync konsol untuk menguji resolver.

1. Masuk ke AWS Management Console dan buka AppSynckonsol.

   a. Di dasbor API, pilih GraphQL API Anda.

   b. Di Sidebar, pilih Skema.

2. Jika Anda belum melakukannya, di bawah tipe dan di samping bidang, pilih Lampirkan untuk menambahkan resolver Anda.

   Untuk informasi selengkapnya tentang cara membuat resolver conplete, lihat Mengonfigurasi resolver.

   Jika tidak, pilih resolver yang sudah ada di lapangan.

3. Di bagian atas halaman Edit resolver, pilih Pilih konteks pengujian, pilih Buat konteks baru.
4. Pilih objek konteks sampel atau isi JSON secara manual di jendela konteks Eksekusi di bawah ini.
5. Masukkan nama konteks Teks.
6. Pilih tombol Simpan.
7. Di bagian atas halaman Edit Resolver, pilih Run test.

Untuk contoh yang lebih praktis, misalkan Anda memiliki aplikasi yang menyimpan tipe `Dog` GraphQL yang menggunakan pembuatan ID otomatis untuk objek dan menyimpannya di Amazon DynamoDB. Anda juga ingin menulis beberapa nilai dari argumen mutasi GraphQL, dan hanya mengizinkan pengguna tertentu untuk melihat respons. Berikut ini menunjukkan seperti apa skema itu:

```
type Dog {
  breed: String
  color: String
}

type Mutation {
  addDog(firstname: String, age: Int): Dog
}
```

Saat Anda menambahkan resolver untuk `addDog` mutasi, Anda dapat mengisi objek konteks seperti contoh berikut. Berikut ini memiliki argumen dari klien `name` dan`age`, dan `username` diisi dalam `identity` objek:

```
{
```

```
    "arguments" : {
        "firstname": "Shaggy",
        "age": 4
    },
    "source" : {},
    "result" : {
        "breed" : "Miniature Schnauzer",
        "color" : "black_grey"
    },
    "identity": {
        "sub" : "uuid",
        "issuer" : " https://cognito-idp.{region}.amazonaws.com/{userPoolId}",
        "username" : "Nadia",
        "claims" : { },
        "sourceIp" :[  "x.x.x.x" ],
        "defaultAuthStrategy" : "ALLOW"
    }
}
```

Anda dapat menguji ini menggunakan templat pemetaan permintaan dan respons berikut:

Templat Permintaan

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key" : {
        "id" : { "S" : "$util.autoId()" }
    },
    "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
}
```

Template Respon

```
#if ($context.identity.username == "Nadia")
  $util.toJson($ctx.result)
#else
  $util.unauthorized()
#end
```

Template yang dievaluasi memiliki data dari objek konteks pengujian Anda dan nilai yang dihasilkan dari`$util.autoId()`. Selain itu, jika Anda mengubah `username` ke nilai selain`Nadia`, hasilnya

tidak akan dikembalikan karena pemeriksaan otorisasi akan gagal. Untuk informasi selengkapnya tentang kontrol akses berbutir halus, lihat Kasus penggunaan otorisasi.

Menguji template pemetaan dengan AWS AppSync API

Anda dapat menggunakan perintah `EvaluateMappingTemplate` API untuk menguji template pemetaan Anda dari jarak jauh dengan data tiruan. Untuk memulai dengan perintah, pastikan Anda telah menambahkan `appsync:evaluateMappingTemplate` izin ke kebijakan Anda. Misalnya:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "appsync:evaluateMappingTemplate",
            "Resource": "arn:aws:appsync:<region>:<account>:*"
        }
    ]
}
```

Anda dapat memanfaatkan perintah dengan menggunakan AWS CLIatau AWSSDK. Misalnya, ambil Dog skema dan template pemetaan permintaan/tanggapannya dari bagian sebelumnya. Menggunakan CLI di stasiun lokal Anda, simpan template permintaan ke file bernama`request.vtl`, lalu simpan `context` objek ke file bernama. `context.json` Dari shell Anda, jalankan perintah berikut:

```
aws appsync evaluate-mapping-template --template file://request.vtl --context file://
context.json
```

Perintah mengembalikan respon berikut:

```
{
  "evaluationResult": "{\n    \"version\" : \"2017-02-28\",\n
 \"operation\" : \"PutItem\",\n    \"key\" : {\n        \"id\" : { \"S\" :
 \"afcb4c85-49f8-40de-8f2b-248949176456\" }\n    },\n    \"attributeValues\" :
 {\"firstname\":{\"S\":\"Shaggy\"},\"age\":{\"N\":4}}\n}\n"
}
```

`evaluationResult`Berisi hasil pengujian template yang Anda berikan dengan yang disediakan`context`. Anda juga dapat menguji template Anda menggunakan AWS SDK. Berikut adalah contoh menggunakan AWS SDK untuk JavaScript V2:

```
const AWS = require('aws-sdk')
const client = new AWS.AppSync({ region: 'us-east-2' })

const template = fs.readFileSync('./request.vtl', 'utf8')
const context = fs.readFileSync('./context.json', 'utf8')

client
  .evaluateMappingTemplate({ template, context })
  .promise()
  .then((data) => console.log(data))
```

Menggunakan SDK, Anda dapat dengan mudah menggabungkan pengujian dari rangkaian pengujian favorit Anda untuk memvalidasi perilaku template Anda. Sebaiknya buat pengujian menggunakan Jest Testing Framework, tetapi rangkaian pengujian apa pun berfungsi. Cuplikan berikut menunjukkan validasi hipotetis berjalan. Perhatikan bahwa kami mengharapkan respons evaluasi menjadi JSON yang valid, jadi kami gunakan `JSON.parse` untuk mengambil JSON dari respons string:

```
const AWS = require('aws-sdk')
const fs = require('fs')
const client = new AWS.AppSync({ region: 'us-east-2' })

test('request correctly calls DynamoDB', async () => {
  const template = fs.readFileSync('./request.vtl', 'utf8')
  const context = fs.readFileSync('./context.json', 'utf8')
  const contextJSON = JSON.parse(context)

  const response = await client.evaluateMappingTemplate({ template,
 context }).promise()
  const result = JSON.parse(response.evaluationResult)

  expect(result.key.id.S).toBeDefined()
  expect(result.attributeValues.firstname.S).toEqual(contextJSON.arguments.firstname)
})
```

Ini menghasilkan hasil sebagai berikut:

```
Ran all test suites.
> jest

PASS ./index.test.js
```

```
# request correctly calls DynamoDB (543 ms)

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 1.511 s, estimated 2 s
```

Mendebug kueri langsung

Tidak ada pengganti untuk end-to-end pengujian dan logging untuk men-debug aplikasi produksi. AWS AppSync memungkinkan Anda mencatat kesalahan dan detail permintaan lengkap menggunakan Amazon CloudWatch. Selain itu, Anda dapat menggunakan AWS AppSync konsol untuk menguji kueri GraphQL, mutasi, dan langganan serta data log streaming langsung untuk setiap permintaan kembali ke editor kueri untuk di-debug secara real time. Untuk langganan, log menampilkan informasi waktu koneksi.

Untuk melakukan ini, Anda harus mengaktifkan CloudWatch log Amazon terlebih dahulu, seperti yang dijelaskan dalam Pemantauan dan pencatatan. Selanjutnya, di AWS AppSync konsol, pilih tab Queries dan kemudian masukkan kueri GraphQL yang valid. Di bagian kanan bawah, klik dan seret jendela Log untuk membuka tampilan log. Di bagian atas halaman, pilih ikon panah putar untuk menjalankan kueri GraphQL Anda. Dalam beberapa saat, log permintaan dan respons lengkap Anda untuk operasi dialirkan ke bagian ini dan Anda dapat melihatnya di konsol.

Penyelesai pipa (VTL)

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync mengeksekusi resolver pada bidang GraphQL. Dalam beberapa kasus, aplikasi memerlukan eksekusi beberapa operasi untuk menyelesaikan satu bidang GraphQL. Dengan resolver pipeline, pengembang sekarang dapat membuat operasi yang disebut Fungsi dan menjalankannya secara berurutan. Pipeline resolver berguna untuk aplikasi yang, misalnya, memerlukan melakukan pemeriksaan otorisasi sebelum mengambil data untuk bidang.

Pipeline resolver terdiri dari template Sebelum pemetaan, template After mapping, dan daftar Functions. Setiap Fungsi memiliki template pemetaan permintaan dan respons yang dijalankan terhadap sumber data. Karena penyelesai pipa mendelegasikan eksekusi ke daftar fungsi, oleh

karena itu tidak ditautkan ke sumber data apa pun. Resolver dan fungsi unit adalah primitif yang menjalankan operasi terhadap sumber data. Lihat ikhtisar template pemetaan Resolver untuk informasi selengkapnya.

Buat Resolver Pipeline

Di AWS AppSync konsol, buka halaman Skema.

Simpan skema berikut:

```
schema {
    query: Query
    mutation: Mutation
}

type Mutation {
    signUp(input: Signup): User
}

type Query {
    getUser(id: ID!): User
}

input Signup {
    username: String!
    email: String!
}

type User {
    id: ID!
    username: String
    email: AWSEmail
}
```

Kita akan memasang resolver pipa ke bidang SignUp pada tipe Mutasi. Pada tipe Mutasi di sisi kanan, pilih Lampirkan di sebelah bidang `signUp` mutasi. Pada halaman create resolver, klik Actions, lalu Update runtime. Pilih`Pipeline Resolver`, lalu pilih`VTL`, lalu pilih Perbarui. Halaman sekarang harus menampilkan tiga bagian: area teks Sebelum memetakan template, bagian Fungsi, dan area teks template Setelah pemetaan.

Pipeline resolver kami mendaftarkan pengguna dengan terlebih dahulu memvalidasi input alamat email dan kemudian menyimpan pengguna dalam sistem. Kita akan merangkum validasi email

di dalam fungsi ValidateEmail, dan menyimpan pengguna di dalam fungsi SaveUser. Fungsi ValidateEmail dijalankan terlebih dahulu, dan jika email valid, maka fungsi SaveUser dijalankan.

Alur eksekusi adalah sebagai berikut:

1. Templat pemetaan permintaan resolver mutation.signup
2. fungsi ValidateEmail
3. Fungsi SaveUser
4. Templat pemetaan respons penyelesai mutasi.signup

Karena kami mungkin akan menggunakan kembali fungsi ValidateEmail di resolver lain di API kami, kami ingin menghindari mengakses `$ctx.args` karena ini akan berubah dari satu bidang GraphQL ke bidang GraphQL lainnya. Sebagai gantinya, kita dapat menggunakan `$ctx.stash` untuk menyimpan atribut email dari argumen field `signUp(input: Signup)` input.

SEBELUM template pemetaan:

```
## store email input field into a generic email key
$util.qr($ctx.stash.put("email", $ctx.args.input.email))
{}
```

Konsol menyediakan passthrough default setelah template pemetaan yang akan kita gunakan:

```
$util.toJson($ctx.result)
```

Pilih Buat atau Simpan untuk memperbarui resolver.

Buat Fungsi

Dari halaman resolver pipeline, di bagian Functions, klik Add function, lalu Create new function. Dimungkinkan juga untuk membuat fungsi tanpa melalui halaman resolver; untuk melakukan ini, di AWS AppSync konsol, buka halaman Fungsi. Pilih tombol Create Function. Mari buat fungsi yang memeriksa apakah email valid dan berasal dari domain tertentu. Jika email tidak valid, fungsi tersebut menimbulkan kesalahan. Jika tidak, itu meneruskan input apa pun yang diberikan.

Pada halaman fungsi baru, pilih Tindakan, lalu Perbarui runtime. Pilih`VTL`, lalu Perbarui. Pastikan Anda telah membuat sumber data dari tipe NONE. Pilih sumber data ini di daftar Nama sumber data. Untuk nama fungsi, masukkan`validateEmail`. Di area kode fungsi, timpa semuanya dengan cuplikan ini:

```
#set($valid = $util.matches("^[a-zA-Z0-9_.+-]+@(?:(?:[a-zA-Z0-9-]+\.)?[a-zA-Z]+\.)?
(myvaliddomain)\.com", $ctx.stash.email))
#if (!$valid)
    $util.error("$ctx.stash.email is not a valid email.")
#end
{
    "payload": { "email": $util.toJson(${ctx.stash.email}) }
}
```

Tempelkan ini ke template pemetaan respons:

```
$util.toJson($ctx.result)
```

Tinjau perubahan Anda, lalu pilih Buat. Kami baru saja membuat fungsi ValidateEmail kami. Ulangi langkah-langkah ini untuk membuat fungsi SaveUser dengan template pemetaan permintaan dan respons berikut (Demi kesederhanaan, kami menggunakan sumber data NONE dan berpura-pura pengguna telah disimpan dalam sistem setelah fungsi dijalankan.):

Permintaan template pemetaan:

```
## $ctx.prev.result contains the signup input values. We could have also
## used $ctx.args.input.
{
    "payload": $util.toJson($ctx.prev.result)
}
```

Templat pemetaan respons:

```
## an id is required so let's add a unique random identifier to the output
$util.qr($ctx.result.put("id", $util.autoId()))
$util.toJson($ctx.result)
```

Kami baru saja membuat fungsi SaveUser kami.

Menambahkan Fungsi ke Pipeline Resolver

Fungsi kita seharusnya ditambahkan secara otomatis ke resolver pipeline yang baru saja kita buat. Jika ini bukan masalahnya, atau Anda membuat fungsi melalui halaman Fungsi, Anda dapat mengklik Tambahkan fungsi pada halaman resolver untuk melampirkannya. Tambahkan fungsi ValidateEmail dan SaveUser ke resolver. Fungsi ValidateEmail harus ditempatkan sebelum fungsi SaveUser.

Saat Anda menambahkan lebih banyak fungsi, Anda dapat menggunakan opsi pindah ke atas dan bergerak ke bawah untuk mengatur ulang urutan eksekusi fungsi Anda. Tinjau perubahan Anda, lalu pilih Simpan.

Menjalankan Query

Di AWS AppSync konsol, buka halaman Kueri. Di penjelajah, pastikan Anda menggunakan mutasi Anda. Jika tidak, pilih `Mutation` di daftar drop-down, lalu pilih+. Masukkan kueri berikut:

```
mutation {
  signUp(input: {
    email: "nadia@myvaliddomain.com"
    username: "nadia"
  }) {
    id
    email
  }
}
```

Ini harus mengembalikan sesuatu seperti:

```
{
  "data": {
    "signUp": {
      "id": "256b6cc2-4694-46f4-a55e-8cb14cc5d7fc",
      "email": "nadia@myvaliddomain.com"
    }
  }
}
```

Kami telah berhasil mendaftarkan pengguna kami dan memvalidasi email input menggunakan resolver pipeline. Untuk mengikuti tutorial yang lebih lengkap yang berfokus pada resolver pipeline, Anda dapat pergi ke Tutorial: Pipeline Resolvers

Langkah 4: Menggunakan API: contoh CDK

> **Tip**
>
> Sebelum Anda menggunakan CDK, kami sarankan untuk meninjau CDKdokumentasi resmibersama denganAWS AppSynciniReferensi CDK.

Kami juga menyarankan untuk memastikan bahwa AndaAWSCLIdanNPMinstalasi bekerja pada sistem Anda.

Di bagian ini, kita akan membuat aplikasi CDK sederhana yang dapat menambahkan dan mengambil item dari tabel DynamoDB. Ini dimaksudkan untuk menjadi contoh quickstart menggunakan beberapa kode dariMerancang skema Anda,Melampirkan sumber data, danMengkonfigurasi resolver (JavaScript)bagian.

Menyiapkan proyek CDK

Warning

Langkah-langkah ini mungkin tidak sepenuhnya akurat tergantung pada lingkungan Anda. Kami berasumsi sistem Anda memiliki utilitas yang diperlukan diinstal, cara untuk berinteraksi denganAWSlayanan, dan konfigurasi yang tepat di tempat.

Langkah pertama adalah menginstalAWSCDK. Di CLI Anda, Anda dapat memasukkan perintah berikut:

```
npm install -g aws-cdk
```

Selanjutnya, Anda perlu membuat direktori proyek, lalu navigasikan ke sana. Contoh set perintah untuk membuat dan menavigasi ke direktori adalah:

```
mkdir example-cdk-app
cd example-cdk-app
```

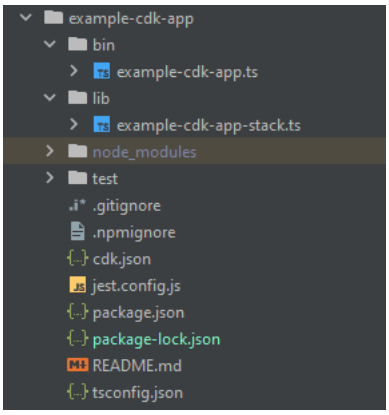
Selanjutnya, Anda perlu membuat aplikasi. Layanan kami terutama menggunakanTypeScript. Di direktori proyek Anda, masukkan perintah berikut:

```
cdk init app --language typescript
```

Ketika Anda melakukan ini, aplikasi CDK bersama dengan file inisialisasi akan diinstal:

```
Initializing a new git repository...
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Executing npm install...
✅All done!
```

Struktur proyek Anda mungkin terlihat seperti ini:



Anda akan melihat kami memiliki beberapa direktori penting:

- `bin`: File bin awal akan membuat aplikasi. Kami tidak akan menyentuh ini dalam panduan ini.
- `lib`: Direktori lib berisi file tumpukan Anda. Anda dapat menganggap file tumpukan sebagai unit eksekusi individual. Konstruksi akan berada di dalam file tumpukan kami. Pada dasarnya, ini adalah sumber daya untuk layanan yang akan diputarAWS CloudFormationsaat aplikasi diterapkan. Di sinilah sebagian besar pengkodean kami akan terjadi.
- `node_modules`: Direktori ini dibuat oleh NPM dan berisi semua dependensi paket yang Anda instal menggunakannpmperintah.

File tumpukan awal kami mungkin berisi sesuatu seperti ini:

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
// import * as sqs from 'aws-cdk-lib/aws-sqs';
```

```
export class ExampleCdkAppStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // The code that defines your stack goes here

    // example resource
    // const queue = new sqs.Queue(this, 'ExampleCdkAppQueue', {
    //   visibilityTimeout: cdk.Duration.seconds(300)
    // });
  }
}
```

Ini adalah kode boilerplate untuk membuat tumpukan di aplikasi kita. Sebagian besar kode kita dalam contoh ini akan masuk ke dalam lingkup kelas ini.

Untuk memverifikasi bahwa file stack Anda ada di aplikasi, di direktori aplikasi Anda, jalankan perintah berikut di terminal:

```
cdk ls
```

Daftar tumpukan Anda akan muncul. Jika tidak, maka Anda mungkin perlu menjalankan langkah-langkah lagi atau memeriksa dokumentasi resmi untuk bantuan.

Jika Anda ingin membuat perubahan kode sebelum menerapkan, Anda selalu dapat menjalankan perintah berikut di terminal:

```
npm run build
```

Dan, untuk melihat perubahan sebelum menerapkan:

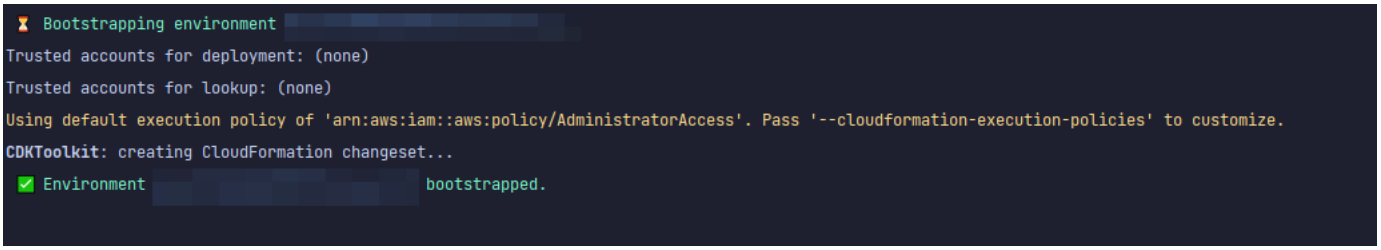
```
cdk diff
```

Sebelum kita menambahkan kode kita ke file stack, kita akan melakukan bootstrap. Bootstrapping memungkinkan kami menyediakan sumber daya untuk CDK sebelum aplikasi diterapkan. Informasi lebih lanjut tentang proses ini dapat ditemukankemari. Untuk membuat bootstrap, perintahnya adalah:

```
cdk bootstrap aws://ACCOUNT-NUMBER/REGION
```

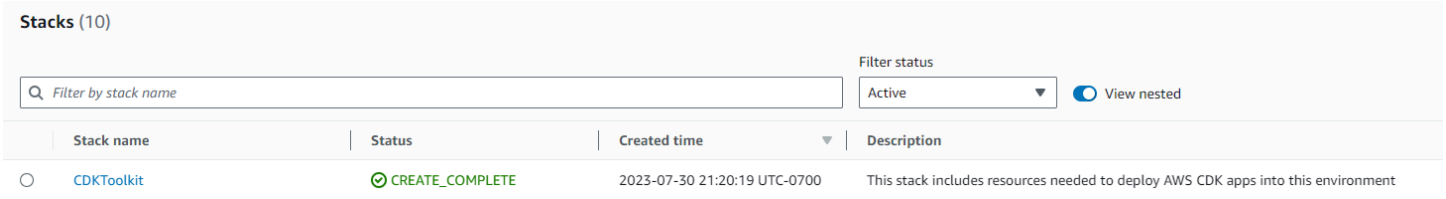
> **Tip**
>
> Langkah ini memerlukan beberapa izin IAM di akun Anda. Bootstrap Anda akan ditolak jika Anda tidak memilikinya. Jika ini terjadi, Anda mungkin harus menghapus sumber daya yang tidak lengkap yang disebabkan oleh bootstrap seperti bucket S3 yang dihasilkannya.

Bootstrap akan memutar beberapa sumber daya. Pesan terakhir akan terlihat seperti ini:

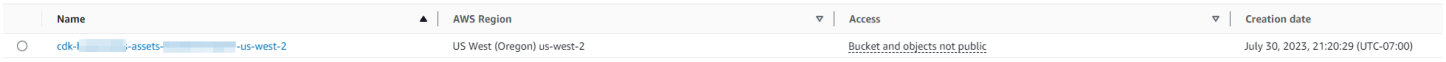


Ini dilakukan sekali per akun per Wilayah, jadi Anda tidak perlu sering melakukannya. Sumber daya utama bootstrap adalahAWS CloudFormationtumpukan dan ember Amazon S3.

Bucket Amazon S3 digunakan untuk menyimpan file dan peran IAM yang memberikan izin yang diperlukan untuk melakukan penerapan. Sumber daya yang dibutuhkan didefinisikan dalamAWS CloudFormationstack, disebut stack bootstrap, yang biasanya dinamai`CDKToolkit`. Seperti apapunAWS CloudFormationtumpukan, muncul diAWS CloudFormationkonsol setelah digunakan:



Hal yang sama dapat dikatakan untuk ember:



Untuk mengimpor layanan yang kita butuhkan di file stack kita, kita dapat menggunakan perintah berikut:

```
npm install aws-cdk-lib # V2 command
```

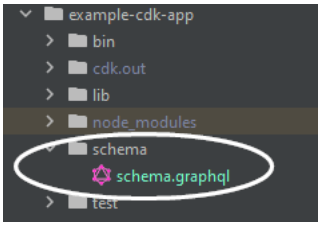
> **Tip**
>
> Jika Anda mengalami masalah dengan V2, Anda dapat menginstal pustaka individual menggunakan perintah V1:
>
> ```
> npm install @aws-cdk/aws-appsync @aws-cdk/aws-dynamodb
> ```
>
> Kami tidak merekomendasikan ini karena V1 telah usang.

Menerapkan proyek CDK - Skema

Kita sekarang dapat mulai menerapkan kode kita. Pertama, kita harus membuat skema kita. Anda cukup membuat`.graphql`berkas di aplikasi Anda:

```
mkdir schema
touch schema.graphql
```

Dalam contoh kami, kami menyertakan direktori tingkat atas yang disebut`schema`mengandung kami`schema.graphql`:



Di dalam skema kita, mari kita sertakan contoh sederhana:

```
input CreatePostInput {
    title: String
    content: String
}

type Post {
    id: ID!
    title: String
    content: String
}

type Mutation {
```

```
    createPost(input: CreatePostInput!): Post
}

type Query {
    getPost: [Post]
}
```

Kembali ke file stack kami, kami perlu memastikan arahan impor berikut didefinisikan:

```
import * as cdk from 'aws-cdk-lib';
import * as appsync from 'aws-cdk-lib/aws-appsync';
import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
import { Construct } from 'constructs';
```

Di dalam kelas, kita akan menambahkan kode untuk membuat API GraphQL kita dan menghubungkannya ke`schema.graphql`berkas:

```
export class ExampleCdkAppStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // makes a GraphQL API
    const api = new appsync.GraphqlApi(this, 'post-apis', {
      name: 'api-to-process-posts',
      schema: appsync.SchemaFile.fromAsset('schema/schema.graphql'),
    });
  }
}
```

Kami juga akan menambahkan beberapa kode untuk mencetak URL GraphQL, kunci API, dan Wilayah:

```
export class ExampleCdkAppStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Makes a GraphQL API construct
    const api = new appsync.GraphqlApi(this, 'post-apis', {
      name: 'api-to-process-posts',
      schema: appsync.SchemaFile.fromAsset('schema/schema.graphql'),
    });
```

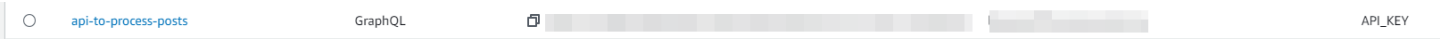
```
    // Prints out URL
    new cdk.CfnOutput(this, "GraphQLAPIURL", {
      value: api.graphqlUrl
    });

    // Prints out the AppSync GraphQL API key to the terminal
    new cdk.CfnOutput(this, "GraphQLAPIKey", {
      value: api.apiKey || ''
    });

    // Prints out the stack region to the terminal
    new cdk.CfnOutput(this, "Stack Region", {
      value: this.region
    });
  }
}
```

Pada titik ini, kita akan menggunakan deploy aplikasi kita lagi:

```
cdk deploy
```

Ini adalah hasilnya:

Tampaknya contoh kita berhasil, tapi mari kita periksaAWS AppSynckonsol hanya untuk mengonfirmasi:

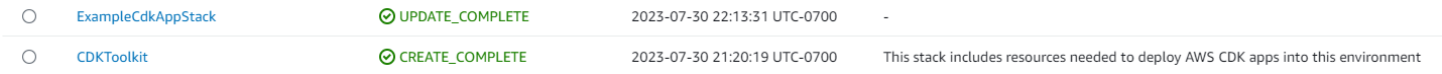


Tampaknya API kami telah dibuat. Sekarang, kita akan memeriksa skema yang dilampirkan ke API:

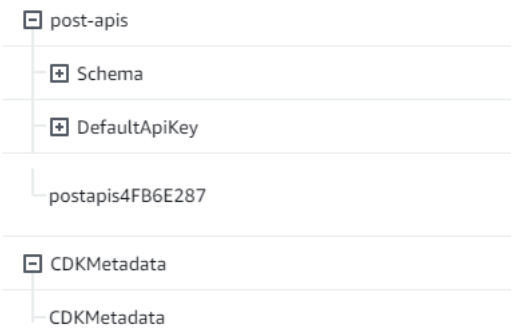
Schema

```
input CreatePostInput {
  title: String
  date: AWSDateTime
}

type Post {
  id: ID!
  title: String
  date: AWSDateTime
}

type Mutation {
  createPost(input: CreatePostInput!): Post
}

type Query {
  getPost: [Post]
}
```

Ini tampaknya cocok dengan kode skema kami, jadi berhasil. Cara lain untuk mengonfirmasi ini dari sudut pandang metadata adalah dengan melihatAWS CloudFormationtumpukan:



Saat kami menerapkan aplikasi CDK kami, itu berhasilAWS CloudFormationuntuk memutar sumber daya seperti bootstrap. Setiap tumpukan dalam aplikasi kami memetakan 1:1 denganAWS CloudFormationtumpukan. Jika Anda kembali ke kode tumpukan, nama tumpukan diambil dari nama kelas`ExampleCdkAppStack`. Anda dapat melihat sumber daya yang dibuatnya, yang juga cocok dengan konvensi penamaan kami di konstruksi API GraphQL kami:



Menerapkan proyek CDK - Sumber data

Selanjutnya, kita perlu menambahkan sumber data kita. Contoh kita akan menggunakan tabel DynamoDB. Di dalam kelas stack, kita akan menambahkan beberapa kode untuk membuat tabel baru:

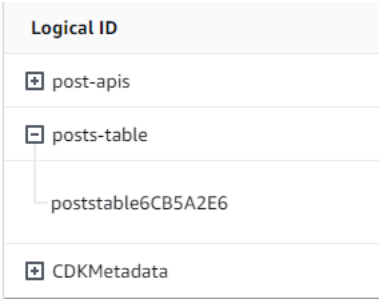
```
export class ExampleCdkAppStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Makes a GraphQL API construct
    const api = new appsync.GraphqlApi(this, 'post-apis', {
      name: 'api-to-process-posts',
      schema: appsync.SchemaFile.fromAsset('schema/schema.graphql'),
    });

    //creates a DDB table
    const add_ddb_table = new dynamodb.Table(this, 'posts-table', {
      partitionKey: {
        name: 'id',
        type: dynamodb.AttributeType.STRING,
      },
    });

    // Prints out URL
    new cdk.CfnOutput(this, "GraphQLAPIURL", {
      value: api.graphqlUrl
    });

    // Prints out the AppSync GraphQL API key to the terminal
    new cdk.CfnOutput(this, "GraphQLAPIKey", {
      value: api.apiKey || ''
    });

    // Prints out the stack region to the terminal
    new cdk.CfnOutput(this, "Stack Region", {
      value: this.region
    });
  }
}
```

Pada titik ini, mari kita gunakan lagi:

```
cdk deploy
```

Kita harus memeriksa konsol DynamoDB untuk tabel baru kita:

Nama tumpukan kami benar, dan nama tabel cocok dengan kode kami. Jika kita periksaAWS CloudFormationtumpukan lagi, kita sekarang akan melihat tabel baru:



Menerapkan proyek CDK - Resolver

Contoh ini akan menggunakan dua resolver: satu untuk menanyakan tabel dan satu untuk menambahkannya. Karena kita menggunakan resolver pipeline, kita perlu mendeklarasikan dua resolver pipeline dengan satu fungsi di masing-masing. Dalam query, kita akan menambahkan kode berikut:

```
export class ExampleCdkAppStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Makes a GraphQL API construct
    const api = new appsync.GraphqlApi(this, 'post-apis', {
      name: 'api-to-process-posts',
      schema: appsync.SchemaFile.fromAsset('schema/schema.graphql'),
    });

    //creates a DDB table
    const add_ddb_table = new dynamodb.Table(this, 'posts-table', {
      partitionKey: {
        name: 'id',
        type: dynamodb.AttributeType.STRING,
      },
    });

    // Creates a function for query
    const add_func = new appsync.AppsyncFunction(this, 'func-get-post', {
      name: 'get_posts_func_1',
      api,
      dataSource: api.addDynamoDbDataSource('table-for-posts', add_ddb_table),
      code: appsync.Code.fromInline(`
          export function request(ctx) {
```

```
        return { operation: 'Scan' };
        }

        export function response(ctx) {
        return ctx.result.items;
        }
`),
    runtime: appsync.FunctionRuntime.JS_1_0_0,
  });

  // Creates a function for mutation
  const add_func_2 = new appsync.AppsyncFunction(this, 'func-add-post', {
    name: 'add_posts_func_1',
    api,
    dataSource: api.addDynamoDbDataSource('table-for-posts-2', add_ddb_table),
    code: appsync.Code.fromInline(`
        export function request(ctx) {
          return {
          operation: 'PutItem',
          key: util.dynamodb.toMapValues({id: util.autoId()}),
          attributeValues: util.dynamodb.toMapValues(ctx.args.input),
          };
        }

        export function response(ctx) {
          return ctx.result;
        }
    `),
    runtime: appsync.FunctionRuntime.JS_1_0_0,
  });

  // Adds a pipeline resolver with the get function
  new appsync.Resolver(this, 'pipeline-resolver-get-posts', {
    api,
    typeName: 'Query',
    fieldName: 'getPost',
    code: appsync.Code.fromInline(`
        export function request(ctx) {
        return {};
        }

        export function response(ctx) {
        return ctx.prev.result;
        }
```

```
  `),
      runtime: appsync.FunctionRuntime.JS_1_0_0,
      pipelineConfig: [add_func],
    });

    // Adds a pipeline resolver with the create function
    new appsync.Resolver(this, 'pipeline-resolver-create-posts', {
      api,
      typeName: 'Mutation',
      fieldName: 'createPost',
      code: appsync.Code.fromInline(`
          export function request(ctx) {
          return {};
          }

          export function response(ctx) {
          return ctx.prev.result;
          }
  `),
      runtime: appsync.FunctionRuntime.JS_1_0_0,
      pipelineConfig: [add_func_2],
    });

    // Prints out URL
    new cdk.CfnOutput(this, "GraphQLAPIURL", {
      value: api.graphqlUrl
    });

    // Prints out the AppSync GraphQL API key to the terminal
    new cdk.CfnOutput(this, "GraphQLAPIKey", {
      value: api.apiKey || ''
    });

    // Prints out the stack region to the terminal
    new cdk.CfnOutput(this, "Stack Region", {
      value: this.region
    });
  }
}
```

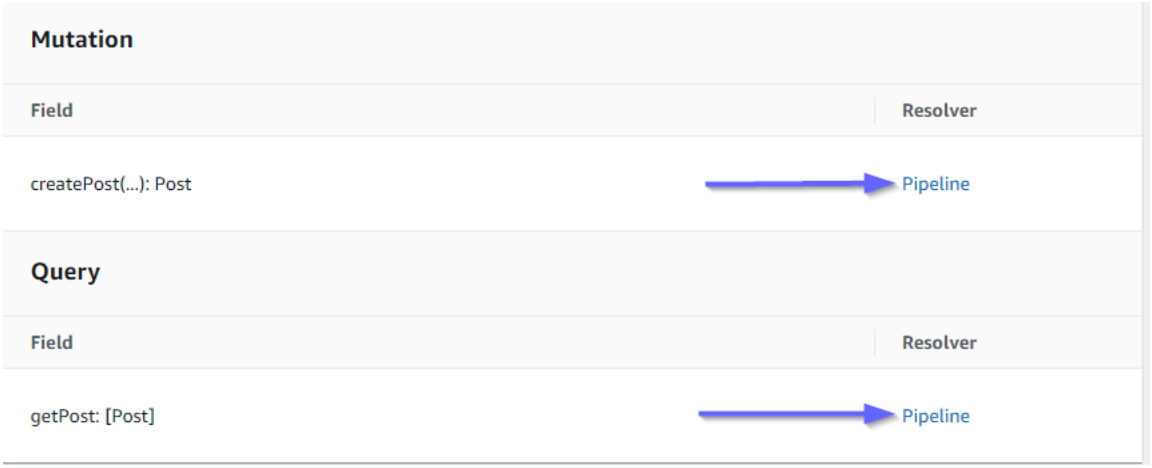
Dalam cuplikan ini, kami menambahkan resolver pipa yang disebut`pipeline-resolver-create-posts`Dengan fungsi yang disebut`func-add-post`melekat padanya. Ini adalah kode yang akan

menambahkanPostske meja. Pipeline resolver lainnya disebutpipeline-resolver-get-postsDengan fungsi yang disebutfunc-get-postyang mengambilPostsditambahkan ke meja.

Kami akan menerapkan ini untuk menambahkannya keAWS AppSynclayanan:

```
cdk deploy
```

Mari kita periksaAWS AppSynckonsol untuk melihat apakah mereka dilampirkan ke API GraphQL kami:



Tampaknya benar. Dalam kode, kedua resolver ini dilampirkan ke API GraphQL yang kami buat (dilambangkan denganapinilai props hadir dalam resolver dan fungsi). Di GraphQL API, bidang yang kami lampirkan pada resolver kami juga ditentukan dalam alat peraga (ditentukan olehtypenamedanfieldnamealat peraga di setiap resolver).

Mari kita lihat apakah konten resolver sudah benar dimulai denganpipeline-resolver-get-posts:

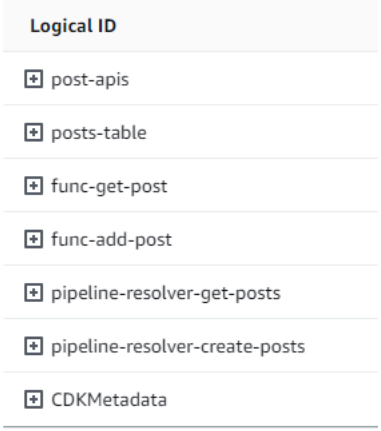
Penangan sebelum dan sesudah cocok dengan kamicodenilai alat peraga. Kita juga dapat melihat bahwa fungsi yang disebutadd_posts_func_1, yang cocok dengan nama fungsi yang kita lampirkan di resolver.

Mari kita lihat isi kode dari fungsi itu:

Ini cocok dengancodealat peraga dariadd_posts_func_1fungsi. Kueri kami berhasil diunggah, jadi mari kita periksa kueri:

Ini juga cocok dengan kode. Jika kita melihat`get_posts_func_1`:

Semuanya tampak ada di tempatnya. Untuk mengonfirmasi ini dari perspektif metadata, kita dapat memeriksa tumpukan kami diAWS CloudFormationlagi:



Sekarang, kita perlu menguji kode ini dengan melakukan beberapa permintaan.

Menerapkan proyek CDK - Permintaan

Untuk menguji aplikasi kami diAWS AppSynckonsol, kami membuat satu kueri dan satu mutasi:

```
query MyQuery {
  getPost {
    id
    date
    title
  }
}

mutation MyMutation {
  createPost(input: {date: "1970-01-01T12:30:00.000Z", title: "first post"}) {
    date
    id
    title
  }
}

```

`MyMutation`berisi `acreatePost`operasi dengan argumen`1970-01-01T12:30:00.000Z`dan`first post`. Ia mengembalikan`date`dan`title`yang kami lewati serta yang dihasilkan secara otomatis`id`nilai. Menjalankan mutasi menghasilkan hasil:

```
{
  "data": {
    "createPost": {
      "date": "1970-01-01T12:30:00.000Z",
      "id": "4dc1c2dd-0aa3-4055-9eca-7c140062ada2",
      "title": "first post"
    }
  }
}
```

Jika kita memeriksa tabel DynamoDB dengan cepat, kita dapat melihat entri kita di tabel ketika kita memindainya:

id *(String)*	date	title
9f62c4dd-49d5-48d5-b835-143284c72fe0	1970-01-01T12:30:00.000Z	first post

Kembali di`AWS AppSync`konsol, jika kita menjalankan kueri untuk mengambil ini`Post`, kami mendapatkan hasil sebagai berikut:

```
{
  "data": {
    "getPost": [
      {
        "id": "9f62c4dd-49d5-48d5-b835-143284c72fe0",
        "date": "1970-01-01T12:30:00.000Z",
```

```
        "title": "first post"
      }
    ]
  }
}
```

Data waktu nyata

AWS AppSyncmemungkinkan Anda memanfaatkan langganan untuk menerapkan pembaruan aplikasi langsung, pemberitahuan push, dll. Ketika klien menjalankan operasi berlangganan GraphQL, koneksi WebSocket aman secara otomatis dibuat dan dikelola oleh. AWS AppSync Aplikasi kemudian dapat mendistribusikan data secara real-time dari sumber data ke pelanggan sambil AWS AppSync terus mengelola koneksi aplikasi dan persyaratan penskalaan. Bagian berikut akan menunjukkan kepada Anda bagaimana langganan AWS AppSync bekerja.

Arahan langganan skema GraphQL

Langganan di AWS AppSync dipanggil sebagai respons terhadap mutasi. Ini berarti Anda dapat membuat sumber data apa pun secara AWS AppSync real time dengan menentukan direktif skema GraphQL pada mutasi.

Pustaka AWS Amplify klien secara otomatis menangani manajemen koneksi berlangganan. Pustaka menggunakan pure WebSockets sebagai protokol jaringan antara klien dan layanan.

> **Note**
>
> Untuk mengontrol otorisasi pada waktu koneksi ke langganan, Anda dapat menggunakan AWS Identity and Access Management (IAM),, kumpulan identitas Amazon CognitoAWS Lambda, atau kumpulan pengguna Amazon Cognito untuk otorisasi tingkat lapangan. Untuk kontrol akses berbutir halus pada langganan, Anda dapat melampirkan resolver ke bidang langganan Anda dan melakukan logika menggunakan identitas pemanggil dan sumber data. AWS AppSync Untuk informasi selengkapnya, lihat Otorisasi dan otentikasi.

Langganan dipicu dari mutasi dan set pemilihan mutasi dikirim ke pelanggan.

Contoh berikut menunjukkan cara bekerja dengan langganan GraphQL. Itu tidak menentukan sumber data karena sumber data bisa Lambda, Amazon DynamoDB, atau Amazon Service. OpenSearch

Untuk memulai langganan, Anda harus menambahkan titik masuk langganan ke skema Anda sebagai berikut:

```
schema {
    query: Query
    mutation: Mutation
    subscription: Subscription
}
```

Misalkan Anda memiliki situs posting blog, dan Anda ingin berlangganan blog baru dan perubahan ke blog yang ada. Untuk melakukan ini, tambahkan `Subscription` definisi berikut ke skema Anda:

```
type Subscription {
    addedPost: Post
    updatedPost: Post
    deletedPost: Post
}
```

Misalkan lebih lanjut bahwa Anda memiliki mutasi berikut:

```
type Mutation {
    addPost(id: ID! author: String! title: String content: String url: String): Post!
    updatePost(id: ID! author: String! title: String content: String url: String ups:
 Int! downs: Int! expectedVersion: Int!): Post!
    deletePost(id: ID!): Post!
}
```

Anda dapat membuat bidang ini secara real time dengan menambahkan `@aws_subscribe(mutations: ["mutation_field_1", "mutation_field_2"])` arahan untuk setiap langganan yang ingin Anda terima notifikasi, sebagai berikut:

```
type Subscription {
    addedPost: Post
    @aws_subscribe(mutations: ["addPost"])
    updatedPost: Post
    @aws_subscribe(mutations: ["updatePost"])
    deletedPost: Post
    @aws_subscribe(mutations: ["deletePost"])
}
```

Karena `@aws_subscribe(mutations: ["",..,""])` mengambil array input mutasi, Anda dapat menentukan beberapa mutasi, yang memulai langganan. Jika Anda berlangganan dari klien, kueri GraphQL Anda mungkin terlihat seperti berikut:

```
subscription NewPostSub {
    addedPost {
        __typename
        version
        title
        content
        author
        url
    }
}
```

Kueri langganan ini diperlukan untuk koneksi dan perkakas klien.

Dengan WebSockets klien murni, penyaringan set seleksi dilakukan per klien, karena setiap klien dapat menentukan set pilihannya sendiri. Dalam hal ini, set pemilihan langganan harus merupakan bagian dari set pemilihan mutasi. Misalnya, langganan yang `addedPost{author title}` ditautkan ke mutasi hanya `addPost(...){id author title url version}` menerima penulis dan judul posting. Itu tidak menerima bidang lainnya. Namun, jika mutasi tidak memiliki penulis dalam kumpulan pemilihannya, pelanggan akan mendapatkan `null` nilai untuk bidang penulis (atau kesalahan jika bidang penulis didefinisikan sebagai required/not-null dalam skema).

Set pemilihan langganan sangat penting saat menggunakan pure WebSockets. Jika bidang tidak didefinisikan secara eksplisit dalam langganan, maka bidang tersebut AWS AppSync tidak ditampilkan.

Pada contoh sebelumnya, langganan tidak memiliki argumen. Misalkan skema Anda terlihat seperti berikut:

```
type Subscription {
    updatedPost(id:ID! author:String): Post
    @aws_subscribe(mutations: ["updatePost"])
}
```

Dalam hal ini, klien Anda mendefinisikan langganan sebagai berikut:

```
subscription UpdatedPostSub {
    updatedPost(id:"XYZ", author:"ABC") {
```

```
        title
        content
    }
}
```

Jenis pengembalian `subscription` bidang dalam skema Anda harus cocok dengan tipe pengembalian bidang mutasi yang sesuai. Pada contoh sebelumnya, ini ditampilkan sebagai keduanya `addPost` dan `addedPost` dikembalikan sebagai tipe`Post`.

Untuk mengatur langganan pada klien, lihatMembangun aplikasi klien.

Menggunakan argumen berlangganan

Bagian penting dari penggunaan langganan GraphQL adalah memahami kapan dan bagaimana menggunakan argumen. Anda dapat membuat perubahan halus untuk memodifikasi bagaimana dan kapan memberi tahu klien tentang mutasi yang telah terjadi. Untuk melakukan ini, lihat skema sampel dari chapter quickstart, yang menciptakan “Todos”. Untuk skema sampel ini, mutasi berikut didefinisikan:

```
type Mutation {
    createTodo(input: CreateTodoInput!): Todo
    updateTodo(input: UpdateTodoInput!): Todo
    deleteTodo(input: DeleteTodoInput!): Todo
}
```

Dalam contoh default, klien dapat berlangganan pembaruan apa pun `Todo` `onUpdateTodo` `subscription` dengan menggunakan tanpa argumen:

```
subscription OnUpdateTodo {
  onUpdateTodo {
    description
    id
    name
    when
  }
}
```

Anda dapat memfilter Anda `subscription` dengan menggunakan argumennya. Misalnya, untuk hanya memicu `subscription` ketika a `todo` dengan spesifik `ID` diperbarui, tentukan `ID` nilainya:

```
subscription OnUpdateTodo {
```

```
  onUpdateTodo(id: "a-todo-id") {
    description
    id
    name
    when
  }
}
```

Anda juga dapat melewati beberapa argumen. Misalnya, berikut ini `subscription` menunjukkan cara mendapatkan pemberitahuan tentang `Todo` pembaruan apa pun di tempat dan waktu tertentu:

```
subscription todosAtHome {
  onUpdateTodo(when: "tomorrow", where: "at home") {
    description
    id
    name
    when
    where
  }
}
```

Perhatikan bahwa semua argumen bersifat opsional. Jika Anda tidak menentukan argumen apa pun di Anda`subscription`, Anda akan berlangganan semua `Todo` pembaruan yang terjadi di aplikasi Anda. Namun, Anda dapat memperbarui definisi bidang Anda `subscription` untuk meminta `ID` argumen. Ini akan memaksa respons spesifik, `todo` bukan semua `todo` s:

```
onUpdateTodo(
  id: ID!,
  name: String,
  when: String,
  where: String,
  description: String
): Todo
```

Argumen nilai null memiliki arti

Saat membuat kueri langganan AWS AppSync, nilai `null` argumen akan memfilter hasil secara berbeda dari menghilangkan argumen sepenuhnya.

Mari kita kembali ke contoh API todos di mana kita bisa membuat todos. Lihat skema sampel dari chapter quickstart.

Mari kita memodifikasi skema kita untuk menyertakan `owner` bidang baru, pada `Todo` tipe, yang menggambarkan siapa pemiliknya. `owner`Bidang tidak diperlukan dan hanya dapat diatur`UpdateTodoInput`. Lihat versi skema yang disederhanakan berikut ini:

```
type Todo {
  id: ID!
  name: String!
  when: String!
  where: String!
  description: String!
  owner: String
}

input CreateTodoInput {
  name: String!
  when: String!
  where: String!
  description: String!
}

input UpdateTodoInput {
  id: ID!
  name: String
  when: String
  where: String
  description: String
  owner: String
}

type Subscription {
    onUpdateTodo(
        id: ID,
        name: String,
        when: String,
        where: String,
        description: String
    ): Todo @aws_subscribe(mutations: ["updateTodo"])
}
```

Langganan berikut mengembalikan semua `Todo` pembaruan:

```
subscription MySubscription {
  onUpdateTodo {
```

```
    description
    id
    name
    when
    where
  }
}
```

Jika Anda memodifikasi langganan sebelumnya untuk menambahkan argumen bidang`owner: null`, Anda sekarang mengajukan pertanyaan yang berbeda. Langganan ini sekarang mendaftarkan klien untuk mendapatkan pemberitahuan tentang semua `Todo` pembaruan yang belum diberikan pemilik.

```
subscription MySubscription {
  onUpdateTodo(owner: null) {
    description
    id
    name
    when
    where
  }
}
```

> **Note**
>
> Mulai 1 Januari 2022, MQTT over tidak lagi tersedia sebagai WebSockets protokol untuk langganan GraphQL di API. AWS AppSync Pure WebSockets adalah satu-satunya protokol yang didukung diAWS AppSync.
> Klien berdasarkan AWS AppSync SDK atau pustaka Amplify, dirilis setelah November 2019, secara otomatis menggunakan WebSockets pure secara default. Upgrade klien ke versi terbaru memungkinkan mereka untuk menggunakan AWS AppSync WebSockets mesin murni.
> Pure WebSockets hadir dengan ukuran muatan yang lebih besar (240 KB), variasi opsi klien yang lebih luas, dan CloudWatch metrik yang ditingkatkan. Untuk informasi lebih lanjut tentang menggunakan WebSocket klien murni, lihatMembangun WebSocket klien real-time.

Membuat API pub/sub generik yang didukung oleh tanpa server WebSockets

Beberapa aplikasi hanya memerlukan WebSocket API sederhana di mana klien mendengarkan saluran atau topik tertentu. Data JSON generik tanpa bentuk spesifik atau persyaratan yang diketik dengan kuat dapat didorong ke klien yang mendengarkan salah satu saluran ini dalam pola berlangganan publikasi (pub/sub) yang murni dan sederhana.

Gunakan AWS AppSync untuk mengimplementasikan WebSocket API pub/sub sederhana dengan sedikit atau tanpa pengetahuan GraphQL dalam hitungan menit dengan secara otomatis membuat kode GraphQL di backend API dan sisi klien.

Buat dan konfigurasikan API pub-sub

Untuk memulai, lakukan hal berikut:

1. Masuk ke AWS Management Console dan buka AppSync konsol.
   - Di Dasbor, pilih Buat API.
2. Pada layar berikutnya, pilih Buat API real-time, lalu pilih Berikutnya.
3. Masukkan nama yang ramah untuk pub/sub API Anda.
4. Anda dapat mengaktifkan fitur API pribadi, tetapi kami sarankan untuk tetap menonaktifkannya untuk saat ini. Pilih Selanjutnya.
5. Anda dapat memilih untuk secara otomatis membuat pub/sub API yang berfungsi menggunakan. WebSockets Kami merekomendasikan untuk menonaktifkan fitur ini untuk saat ini juga. Pilih Selanjutnya.
6. Pilih Buat API dan kemudian tunggu beberapa menit. AWS AppSync Pub/sub API baru yang telah dikonfigurasi sebelumnya akan dibuat di akun Anda. AWS

API menggunakan AWS AppSync resolver lokal bawaan (untuk informasi selengkapnya tentang penggunaan resolver lokal, lihat Tutorial: Local Resolvers in the AWS AppSync Developer Guide) untuk mengelola beberapa pub/sub channel sementara dan WebSocket koneksi, yang secara otomatis mengirimkan dan memfilter data ke klien berlangganan hanya berdasarkan nama saluran. Panggilan API diotorisasi dengan kunci API.

Setelah API diterapkan, Anda akan disajikan dengan beberapa langkah tambahan untuk menghasilkan kode klien dan mengintegrasikannya dengan aplikasi klien Anda. Sebagai contoh

tentang cara mengintegrasikan klien dengan cepat, panduan ini akan menggunakan aplikasi web React sederhana.

1. Mulailah dengan membuat aplikasi React boilerplate menggunakan NPM di mesin lokal Anda:

```
$ npx create-react-app mypubsub-app
$ cd mypubsub-app
```

> **Note**
>
> Contoh ini menggunakan library Amplify untuk menghubungkan klien ke API backend. Namun tidak perlu membuat proyek Amplify CLI secara lokal. Meskipun React adalah klien pilihan dalam contoh ini, pustaka Amplify juga mendukung klien iOS, Android, dan Flutter, memberikan kemampuan yang sama di runtime yang berbeda ini. Klien Amplify yang didukung menyediakan abstraksi sederhana untuk berinteraksi dengan backend API AWS AppSync GraphQL dengan beberapa baris kode termasuk kemampuan bawaan yang sepenuhnya kompatibel dengan protokol real-time: WebSocket AWS AppSync WebSocket
>
> ```
> $ npm install @aws-amplify/api
> ```

2. Di AWS AppSync konsol, pilih JavaScript, lalu Unduh untuk mengunduh satu file dengan detail konfigurasi API dan kode operasi GraphQL yang dihasilkan.
3. Salin file yang diunduh ke `/src` folder dalam proyek React Anda.
4. Selanjutnya, ganti konten `src/App.js` file boilerplate yang ada dengan kode klien sampel yang tersedia di konsol.
5. Gunakan perintah berikut untuk memulai aplikasi secara lokal:

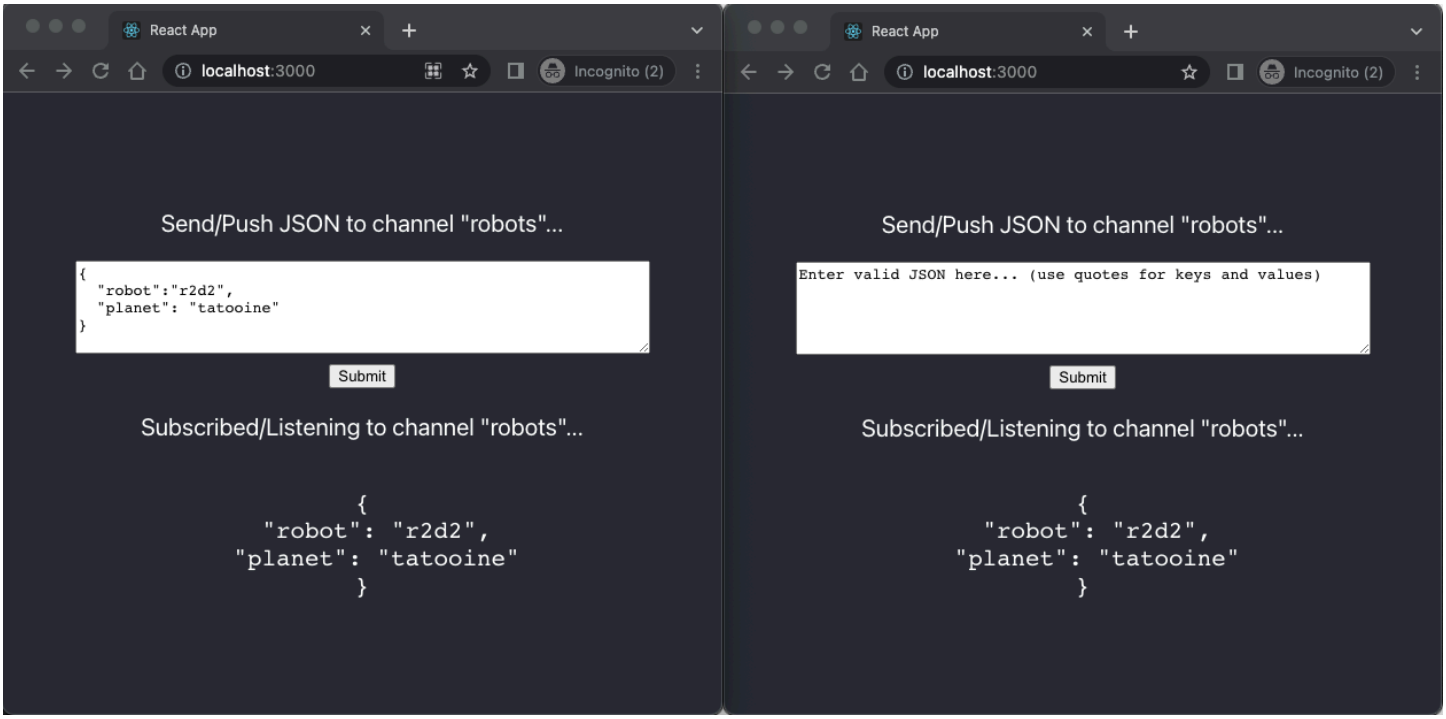
```
$ npm start
```

6. Untuk menguji pengiriman dan penerimaan data real-time, buka dua jendela browser dan akses *`localhost`*: 3000. *`Aplikasi sampel dikonfigurasi untuk mengirim data JSON generik ke saluran hard-code bernama robot.`*
7. Di salah satu jendela browser, masukkan gumpalan JSON berikut di kotak teks lalu klik Kirim:

```
{
  "robot":"r2d2",
```

```
  "planet": "tatooine"
}
```

Kedua instance browser berlangganan saluran *`robot`* dan menerima data yang dipublikasikan secara real time, ditampilkan di bagian bawah aplikasi web:



Semua kode API GraphQL yang diperlukan, termasuk skema, resolver, dan operasi dibuat secara otomatis untuk mengaktifkan kasus penggunaan pub/sub generik. Di backend, data dipublikasikan ke AWS AppSync titik akhir real-time dengan mutasi GraphQL seperti berikut ini:

```
mutation PublishData {
    publish(data: "{\"msg\": \"hello world!\"}", name: "channel") {
        data
        name
    }
}
```

Pelanggan mengakses data yang dipublikasikan yang dikirim ke saluran sementara tertentu dengan langganan GraphQL terkait:

```
subscription SubscribeToData {
    subscribe(name:"channel") {
        name
```

```
        data
    }
}
```

Menerapkan API pub-sub ke dalam aplikasi yang ada

Jika Anda hanya perlu menerapkan fitur real-time dalam aplikasi yang ada, konfigurasi pub/sub API generik ini dapat dengan mudah diintegrasikan ke dalam aplikasi atau teknologi API apa pun. Meskipun ada keuntungan dalam menggunakan titik akhir API tunggal untuk mengakses, memanipulasi, dan menggabungkan data dengan aman dari satu atau lebih sumber data dalam satu panggilan jaringan dengan GraphQL, tidak perlu mengonversi atau membangun kembali aplikasi berbasis REST yang ada dari awal untuk memanfaatkan kemampuan real-time. AWS AppSync Misalnya, Anda dapat memiliki beban kerja CRUD yang ada di titik akhir API terpisah dengan klien mengirim dan menerima pesan atau peristiwa dari aplikasi yang ada ke pub/sub API generik hanya untuk tujuan real-time dan pub/sub.

Pemfilteran langganan yang disempurnakan

DiAWS AppSync, Anda dapat menentukan dan mengaktifkan logika bisnis untuk pemfilteran data di backend secara langsung di resolver langganan GraphQL API dengan menggunakan filter yang mendukung operator logis tambahan. Anda dapat mengonfigurasi filter ini, tidak seperti argumen langganan yang ditentukan pada kueri langganan di klien. Untuk informasi selengkapnya tentang penggunaan argumen langganan, lihatMenggunakan argumen berlangganan. Untuk daftar operator, lihatReferensi utilitas template pemetaan penyelesai.

Untuk tujuan dokumen ini, kami membagi pemfilteran data real-time ke dalam kategori berikut:

- Pemfilteran dasar - Pemfilteran berdasarkan argumen yang ditentukan klien dalam kueri langganan.
- Penyaringan yang disempurnakan - Pemfilteran berdasarkan logika yang didefinisikan secara terpusat di backend layanan. AWS AppSync

Bagian berikut menjelaskan cara mengonfigurasi filter langganan yang disempurnakan dan menunjukkan penggunaan praktisnya.

Mendefinisikan langganan dalam skema GraphQL Anda

Untuk menggunakan filter langganan yang disempurnakan, Anda menentukan langganan dalam skema GraphQL lalu tentukan filter yang disempurnakan menggunakan ekstensi pemfilteran. Untuk

mengilustrasikan cara kerja penyaringan langganan yang disempurnakanAWS AppSync, gunakan skema GraphQL berikut, yang mendefinisikan API sistem manajemen tiket, sebagai contoh:

```
type Ticket {
 id: ID
 createdAt: AWSDateTime
 content: String
 severity: Int
 priority: Priority
 category: String
 group: String
 status: String

}

type Mutation {
 createTicket(input: TicketInput): Ticket
}

type Query {
 getTicket(id: ID!): Ticket
}

type Subscription {
 onSpecialTicketCreated: Ticket @aws_subscribe(mutations: ["createTicket"])
 onGroupTicketCreated(group: String!): Ticket @aws_subscribe(mutations:
 ["createTicket"])
}



enum Priority {
 none
 lowest
 low
 medium
 high
 highest
}

input TicketInput {
 content: String
 severity: Int
```

```
 priority: Priority
 category: String
 group: String
```

Misalkan Anda membuat sumber `NONE` data untuk API Anda, lalu lampirkan resolver ke `createTicket` mutasi menggunakan sumber data ini. Penangan Anda mungkin terlihat seperti ini:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
 return {
  payload: {
   id: util.autoId(),
   createdAt: util.time.nowISO8601(),
   status: 'pending',
   ...ctx.args.input,
  },
 };
}

export function response(ctx) {
 return ctx.result;
}
```

> **Note**
>
> Filter yang disempurnakan diaktifkan di handler GraphQL resolver dalam langganan tertentu. Untuk informasi selengkapnya, lihat Referensi Resolver.

Untuk mengimplementasikan perilaku filter yang disempurnakan, Anda harus menggunakan `extensions.setSubscriptionFilter()` fungsi tersebut untuk menentukan ekspresi filter yang dievaluasi terhadap data yang dipublikasikan dari mutasi GraphQL yang mungkin diminati oleh klien berlangganan. Untuk informasi selengkapnya tentang ekstensi pemfilteran, lihat Ekstensi.

Bagian berikut menjelaskan cara menggunakan ekstensi penyaringan untuk mengimplementasikan filter yang disempurnakan.

Membuat filter langganan yang disempurnakan menggunakan ekstensi penyaringan

Filter yang disempurnakan ditulis dalam JSON di handler respons dari resolver langganan. Filter dapat dikelompokkan bersama dalam daftar yang disebut `afilterGroup`. Filter didefinisikan menggunakan setidaknya satu aturan, masing-masing dengan bidang, operator, dan nilai.
Mari kita tentukan resolver baru untuk on`SpecialTicketCreated` itu menyiapkan filter yang disempurnakan. Anda dapat mengonfigurasi beberapa aturan dalam filter yang dievaluasi menggunakan logika AND, sementara beberapa filter dalam grup filter dievaluasi menggunakan logika OR:

```
import { util, extensions } from '@aws-appsync/utils';

export function request(ctx) {
 // simplfy return null for the payload
 return { payload: null };
}

export function response(ctx) {
 const filter = {
  or: [
   { severity: { ge: 7 }, priority: { in: ['high', 'medium'] } },
   { category: { eq: 'security' }, group: { in: ['admin', 'operators'] } },
  ],
 };
 extensions.setSubscriptionFilter(util.transform.toSubscriptionFilter(filter));

  // important: return null in the response
 return null;
}
```

Berdasarkan filter yang ditentukan dalam contoh sebelumnya, tiket penting secara otomatis didorong ke klien API berlangganan jika tiket dibuat dengan:

- `priority`tingkat `high` atau `medium`

  DAN

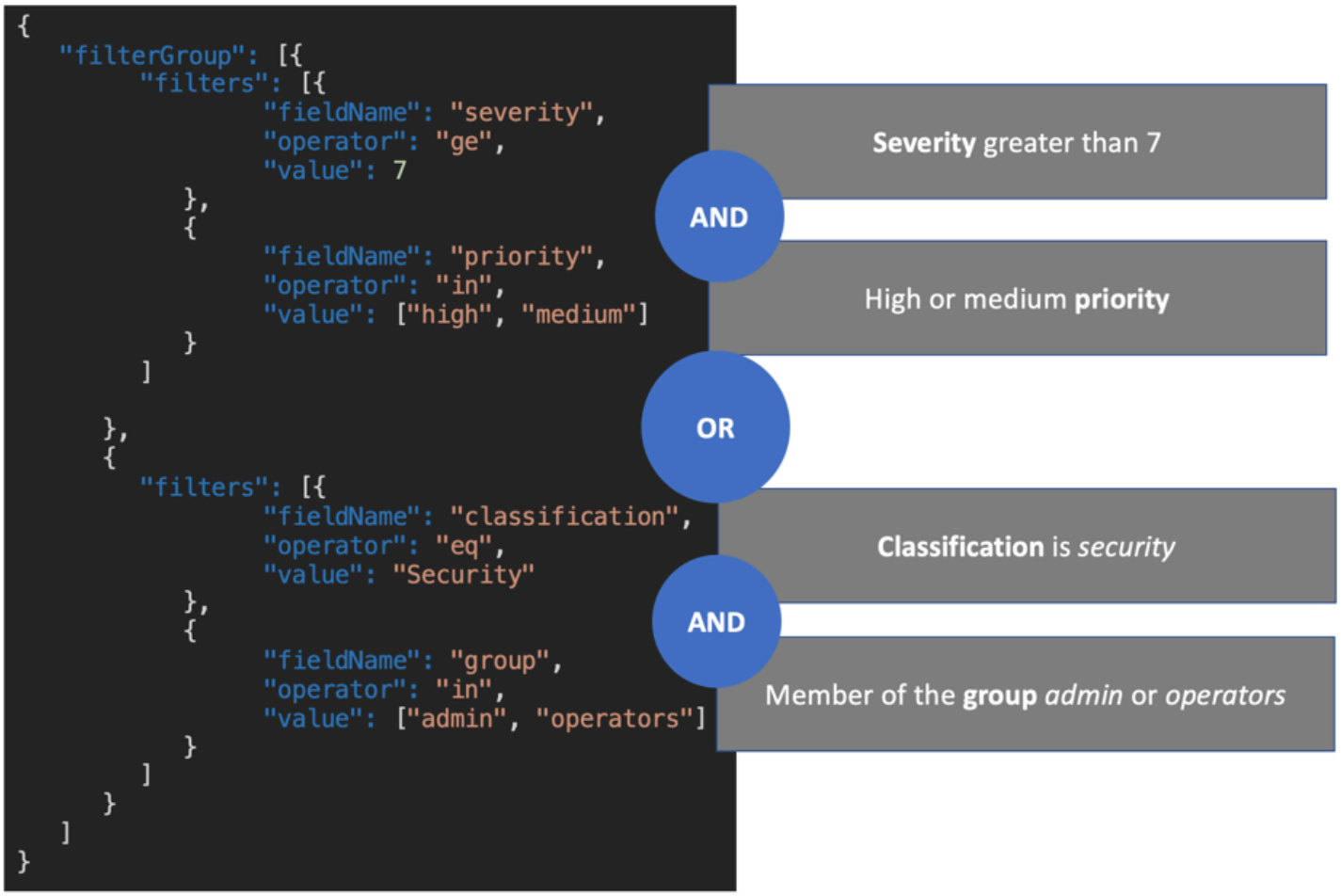
- `severity`tingkat lebih besar dari atau sama dengan 7 (`ge`)

ATAU

- `classification`tiket diatur ke `Security`

  DAN

- `group`tugas diatur ke `admin` atau `operators`



Filter yang didefinisikan dalam resolver langganan (penyaringan yang disempurnakan) lebih diutamakan daripada pemfilteran hanya berdasarkan argumen langganan (pemfilteran dasar). Untuk informasi selengkapnya tentang penggunaan argumen langganan, lihat Menggunakan argumen langganan).

Jika argumen didefinisikan dan diperlukan dalam skema GraphQL langganan, pemfilteran berdasarkan argumen yang diberikan hanya terjadi jika argumen didefinisikan sebagai aturan dalam metode resolver. `extensions.setSubscriptionFilter()` Namun, jika tidak ada metode `extensions` penyaringan dalam resolver langganan, argumen yang didefinisikan dalam klien hanya digunakan untuk pemfilteran dasar. Anda tidak dapat menggunakan pemfilteran dasar dan penyaringan yang disempurnakan secara bersamaan.

Anda dapat menggunakan `context`variabel dalam logika ekstensi filter langganan untuk mengakses informasi kontekstual tentang permintaan. Misalnya, saat menggunakan Amazon Cognito User Pools, OIDC, atau otorisasi kustom Lambda untuk otorisasi, Anda dapat mengambil informasi tentang pengguna saat langganan dibuat. `context.identity` Anda dapat menggunakan informasi tersebut untuk membuat filter berdasarkan identitas pengguna Anda.

Sekarang asumsikan bahwa Anda ingin menerapkan perilaku filter yang disempurnakan untuk`onGroupTicketCreated`. `onGroupTicketCreated`Langganan membutuhkan `group` nama wajib sebagai argumen. Saat dibuat, tiket secara otomatis diberi `pending` status. Anda dapat mengatur filter langganan untuk hanya menerima tiket yang baru dibuat milik grup yang disediakan:

```
import { util, extensions } from '@aws-appsync/utils';

export function request(ctx) {
 // simplfy return null for the payload
 return { payload: null };
}

export function response(ctx) {
 const filter = { group: { eq: ctx.args.group }, status: { eq: 'pending' } };
 extensions.setSubscriptionFilter(util.transform.toSubscriptionFilter(filter));

 return null;
}
```

Ketika data dipublikasikan menggunakan mutasi seperti pada contoh berikut:

```
mutation CreateTicket {
  createTicket(input: {priority: medium, severity: 2, group: "aws"}) {
    id
    priority
    severity
    status
    group
    createdAt
  }
}
```

Klien berlangganan mendengarkan data yang akan didorong secara otomatis WebSockets segera setelah tiket dibuat dengan `createTicket` mutasi:

```
subscription OnGroup {
  onGroupTicketCreated(group: "aws") {
    category
    status
    severity
    priority
    id
    group
    createdAt
    content
  }
}
```

Klien dapat berlangganan tanpa argumen karena logika penyaringan diimplementasikan dalam AWS AppSync layanan dengan penyaringan yang ditingkatkan, yang menyederhanakan kode klien. Klien menerima data hanya jika kriteria filter yang ditentukan terpenuhi.

Mendefinisikan filter yang disempurnakan untuk bidang skema bersarang

Anda dapat menggunakan pemfilteran langganan yang disempurnakan untuk memfilter bidang skema bersarang. Misalkan kita memodifikasi skema dari bagian sebelumnya untuk memasukkan jenis lokasi dan alamat:

```
type Ticket {
 id: ID
 createdAt: AWSDateTime
 content: String
 severity: Int
 priority: Priority
 category: String
 group: String
 status: String
 location: ProblemLocation
}

type Mutation {
 createTicket(input: TicketInput): Ticket
}

type Query {
 getTicket(id: ID!): Ticket
}
```

```
type Subscription {
 onSpecialTicketCreated: Ticket @aws_subscribe(mutations: ["createTicket"])
 onGroupTicketCreated(group: String!): Ticket @aws_subscribe(mutations:
 ["createTicket"])
}

type ProblemLocation {
 address: Address
}

type Address {
 country: String
}

enum Priority {
 none
 lowest
 low
 medium
 high
 highest
}

input TicketInput {
 content: String
 severity: Int
 priority: Priority
 category: String
 group: String
 location: AWSJSON
```

Dengan skema ini, Anda dapat menggunakan `.` pemisah untuk mewakili bersarang. Contoh berikut menambahkan aturan filter untuk bidang skema bersarang di bawah. `location.address.country` Langganan akan dipicu jika alamat tiket diatur keUSA:

```
import { util, extensions } from '@aws-appsync/utils';

export const request = (ctx) => ({ payload: null });

export function response(ctx) {
 const filter = {
  or: [
```

```
    { severity: { ge: 7 }, priority: { in: ['high', 'medium'] } },
    { category: { eq: 'security' }, group: { in: ['admin', 'operators'] } },
    { 'location.address.country': { eq: 'USA' } },
   ],
 };
 extensions.setSubscriptionFilter(util.transform.toSubscriptionFilter(filter));
 return null;
}
```

Dalam contoh di atas, `location` mewakili tingkat bersarang satu, `address` mewakili tingkat bersarang dua, dan `country` mewakili tingkat bersarang tiga, yang semuanya dipisahkan oleh pemisah. `.`

Anda dapat menguji langganan ini dengan menggunakan `createTicket` mutasi:

```
mutation CreateTicketInUSA {
  createTicket(input: {location: "{\"address\":{\"country\":\"USA\"}}"}) {
    category
    content
    createdAt
    group
    id
    location {
      address {
        country
      }
    }
    priority
    severity
    status
  }
}
```

Mendefinisikan filter yang disempurnakan dari klien

Anda dapat menggunakan pemfilteran dasar di GraphQL dengan argumen langganan. Klien yang membuat panggilan dalam kueri langganan mendefinisikan nilai argumen. Saat filter yang disempurnakan diaktifkan di resolver AWS AppSync langganan dengan `extensions` pemfilteran, filter backend yang ditentukan dalam resolver akan diutamakan dan diprioritaskan.

Konfigurasikan filter dinamis yang ditingkatkan yang ditentukan klien menggunakan `filter` argumen dalam langganan. Saat Anda mengonfigurasi filter ini, Anda harus memperbarui skema GraphQL untuk mencerminkan argumen baru:

```
...
type Subscription {
    onSpecialTicketCreated(filter: String): Ticket
        @aws_subscribe(mutations: ["createTicket"])
}
...
```

Klien kemudian dapat mengirim kueri langganan seperti pada contoh berikut:

```
subscription onSpecialTicketCreated($filter: String) {
     onSpecialTicketCreated(filter: $filter) {
        id
        group
        description
        priority
        severity
     }
 }
```

Anda dapat mengkonfigurasi variabel query seperti contoh berikut:

```
{"filter" : "{\"severity\":{\"le\":2}}"}
```

Utilitas `util.transform.toSubscriptionFilter()` resolver dapat diimplementasikan dalam template pemetaan respons langganan untuk menerapkan filter yang ditentukan dalam argumen langganan untuk setiap klien:

```
import { util, extensions } from '@aws-appsync/utils';

export function request(ctx) {
 // simplfy return null for the payload
 return { payload: null };
}

export function response(ctx) {
 const filter = ctx.args.filter;
 extensions.setSubscriptionFilter(util.transform.toSubscriptionFilter(filter));
```

```
 return null;
}
```

Dengan strategi ini, klien dapat menentukan filter mereka sendiri yang menggunakan logika penyaringan yang disempurnakan dan operator tambahan. Filter ditetapkan ketika klien tertentu memanggil kueri langganan dalam WebSocket koneksi aman. Untuk informasi selengkapnya tentang utilitas transformasi untuk penyaringan yang disempurnakan, termasuk format payload variabel `filter` kueri, lihat ikhtisar JavaScript resolver.

Pembatasan penyaringan tambahan yang ditingkatkan

Di bawah ini adalah beberapa kasus penggunaan di mana pembatasan tambahan ditempatkan pada filter yang disempurnakan:

- Filter yang disempurnakan tidak mendukung pemfilteran untuk daftar objek tingkat atas. Dalam kasus penggunaan ini, data yang dipublikasikan dari mutasi akan diabaikan untuk langganan yang ditingkatkan.
- AWS AppSync mendukung hingga lima tingkat bersarang. Filter pada bidang skema melewati tingkat lima bersarang akan diabaikan. Ambil respons GraphQL di bawah ini. `continent`Bidang di `venue.address.country.metadata.continent` diperbolehkan karena itu adalah sarang tingkat lima. Namun, `financial` in `venue.address.country.metadata.capital.financial` adalah sarang level enam, jadi filter tidak akan berfungsi:

```
{
    "data": {
        "onCreateFilterEvent": {
            "venue": {
                "address": {
                    "country": {
                        "metadata": {
                            "capital": {
                                "financial": "New York"
                            },
                            "continent" : "North America"
                        }
                    },
                    "state": "WA"
                },
                "builtYear": 2023
```

```
            },
            "private": false,
        }
    }
}
```

Berhenti berlangganan WebSocket koneksi menggunakan filter

DiAWS AppSync, Anda dapat berhenti berlangganan secara paksa dan menutup (membatalkan) WebSocket koneksi dari klien yang terhubung berdasarkan logika penyaringan tertentu. Ini berguna dalam skenario terkait otorisasi seperti ketika Anda menghapus pengguna dari grup.

Pembatalan langganan terjadi sebagai respons terhadap muatan yang ditentukan dalam mutasi. Sebaiknya Anda memperlakukan mutasi yang digunakan untuk membatalkan koneksi langganan sebagai operasi administratif di API dan izin cakupan sesuai dengan membatasi penggunaannya pada pengguna admin, grup, atau layanan backend. Misalnya, menggunakan arahan otorisasi skema seperti atau. `@aws_auth(cognito_groups: ["Administrators"])` `@aws_iam` Untuk informasi selengkapnya, lihat Menggunakan mode otorisasi tambahan.

Filter pembatalan menggunakan sintaks dan logika yang sama dengan filter langganan yang disempurnakan. Tentukan filter ini menggunakan utilitas berikut:

- `extensions.invalidateSubscriptions()`— Didefinisikan dalam penangan respons GraphQL resolver untuk mutasi.
- `extensions.setSubscriptionInvalidationFilter()`— Didefinisikan dalam penangan respons GraphQL resolver dari langganan yang ditautkan ke mutasi.

Untuk informasi selengkapnya tentang ekstensi pemfilteran pembatalan, lihat JavaScript ikhtisar penyelesai.

Menggunakan pembatalan langganan

Untuk melihat cara kerja pembatalan langgananAWS AppSync, gunakan skema GraphQL berikut:

```
type User {
  userId: ID!
  groupId: ID!
}
```

```
type Group {
  groupId: ID!
  name: String!
  members: [ID!]!
}

type GroupMessage {
  userId: ID!
  groupId: ID!
  message: String!
}

type Mutation {
    createGroupMessage(userId: ID!, groupId : ID!, message: String!): GroupMessage
    removeUserFromGroup(userId: ID!, groupId : ID!) : User @aws_iam
}

type Subscription {
    onGroupMessageCreated(userId: ID!, groupId : ID!): GroupMessage
        @aws_subscribe(mutations: ["createGroupMessage"])
}

type Query {
 none: String
}
```

Tentukan filter pembatalan dalam kode penyelesai `removeUserFromGroup` mutasi:

```
import { extensions } from '@aws-appsync/utils';

export function request(ctx) {
 return { payload: null };
}

export function response(ctx) {
 const { userId, groupId } = ctx.args;
 extensions.invalidateSubscriptions({
  subscriptionField: 'onGroupMessageCreated',
  payload: { userId, groupId },
 });
 return { userId, groupId };
}
```

Saat mutasi dipanggil, data yang ditentukan dalam `payload` objek digunakan untuk berhenti berlangganan langganan yang ditentukan dalam. `subscriptionField` Filter pembatalan juga didefinisikan dalam template pemetaan respons `onGroupMessageCreated` langganan.

Jika `extensions.invalidateSubscriptions()` payload berisi ID yang cocok dengan ID dari klien berlangganan seperti yang ditentukan dalam filter, langganan terkait akan berhenti berlangganan. Selain itu, WebSocket koneksi ditutup. Tentukan kode resolver langganan untuk langganan: `onGroupMessageCreated`

```
import { util, extensions } from '@aws-appsync/utils';

export function request(ctx) {
 // simplfy return null for the payload
 return { payload: null };
}

export function response(ctx) {
 const filter = { groupId: { eq: ctx.args.groupId } };
 extensions.setSubscriptionFilter(util.transform.toSubscriptionFilter(filter));

 const invalidation = { groupId: { eq: ctx.args.groupId }, userId: { eq:
 ctx.args.userId } };
 extensions.setSubscriptionInvalidationFilter(util.transform.toSubscriptionFilter(invalidation)

 return null;
}
```

Perhatikan bahwa handler respons langganan dapat memiliki filter langganan dan filter pembatalan yang ditentukan secara bersamaan.

Misalnya, asumsikan bahwa klien A berlangganan pengguna baru dengan ID *`user-1`* ke grup dengan ID *`group-1`* menggunakan permintaan berlangganan berikut:

```
onGroupMessageCreated(userId : "user-1", groupId: :"group-1"){...}
```

AWS AppSyncmenjalankan resolver langganan, yang menghasilkan filter langganan dan pembatalan seperti yang didefinisikan dalam template pemetaan respons sebelumnya`onGroupMessageCreated`. Untuk klien A, filter langganan memungkinkan data dikirim hanya ke*`group-1`*, dan filter pembatalan ditentukan untuk keduanya *`user-1`* dan. *`group-1`*

Sekarang asumsikan bahwa klien B berlangganan pengguna dengan ID *user-2* ke grup dengan ID *group-2* menggunakan permintaan berlangganan berikut:

```
onGroupMessageCreated(userId : "user-2", groupId: :"group-2"){...}
```

AWS AppSyncmenjalankan resolver langganan, yang menghasilkan filter langganan dan pembatalan. Untuk klien B, filter langganan memungkinkan data dikirim hanya ke*group-2*, dan filter pembatalan ditentukan untuk keduanya *user-2* dan. *group-2*

Selanjutnya, asumsikan bahwa pesan grup baru dengan ID *message-1* dibuat menggunakan permintaan mutasi seperti pada contoh berikut:

```
createGroupMessage(id: "message-1", groupId :
      "group-1", message: "test message"){...}
```

Klien berlangganan yang cocok dengan filter yang ditentukan secara otomatis menerima muatan data berikut melalui: WebSockets

```
{
  "data": {
    "onGroupMessageCreated": {
      "id": "message-1",
      "groupId": "group-1",
      "message": "test message",
    }
  }
}
```

Klien A menerima pesan karena kriteria pemfilteran cocok dengan filter langganan yang ditentukan. Namun, klien B tidak menerima pesan, karena pengguna bukan bagian dari*group-1*. Selain itu, permintaan tidak cocok dengan filter langganan yang ditentukan dalam resolver langganan.

Akhirnya, asumsikan *user-1* bahwa dihapus dari *group-1* menggunakan permintaan mutasi berikut:

```
removeUserFromGroup(userId: "user-1", groupId : "group-1"){...}
```

Mutasi memulai pembatalan langganan seperti yang didefinisikan dalam kode penangan respons `extensions.invalidateSubscriptions()` resolvernya. AWS AppSynckemudian berhenti

berlangganan klien A dan menutup koneksinya. WebSocket Klien B tidak terpengaruh, karena payload pembatalan yang ditentukan dalam mutasi tidak cocok dengan pengguna atau grupnya.

Ketika AWS AppSync membatalkan koneksi, klien menerima pesan yang mengonfirmasi bahwa mereka berhenti berlangganan:

```
{
  "message": "Subscription complete."
}
```

Menggunakan variabel konteks dalam filter pembatalan langganan

Seperti halnya filter langganan yang disempurnakan, Anda dapat menggunakan `context`variabel dalam ekstensi filter pembatalan langganan untuk mengakses data tertentu.

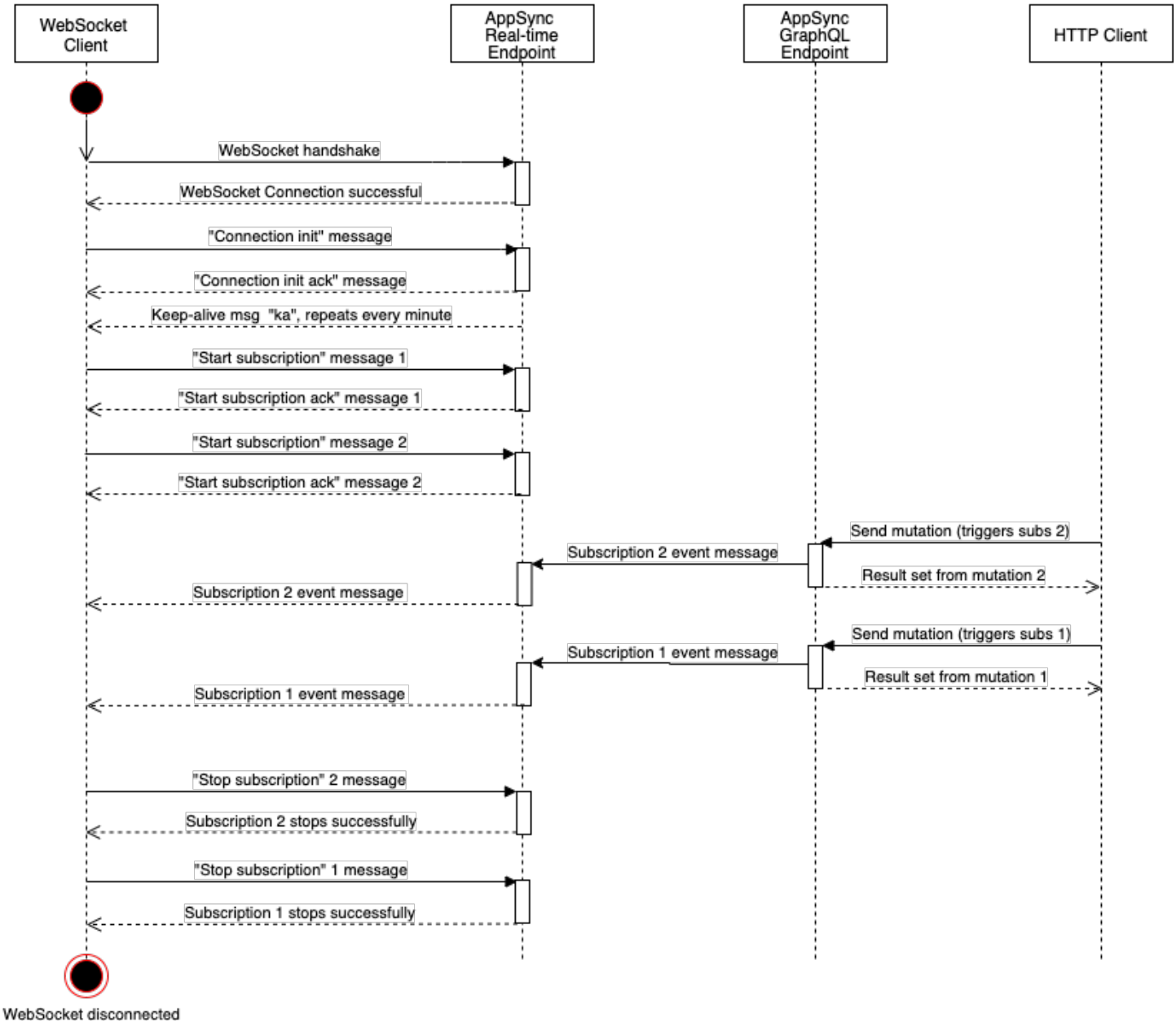
Misalnya, Anda dapat mengonfigurasi alamat email sebagai payload pembatalan dalam mutasi, lalu mencocokkannya dengan atribut email atau klaim dari pengguna berlangganan yang diotorisasi dengan kumpulan pengguna Amazon Cognito atau OpenID Connect. Filter pembatalan yang ditentukan dalam invalidator `extensions.setSubscriptionInvalidationFilter()` langganan memeriksa apakah alamat email yang ditetapkan oleh `extensions.invalidateSubscriptions()` muatan mutasi cocok dengan alamat email yang diambil dari token JWT pengguna, memulai pembatalan. `context.identity.claims.email`

Membangun WebSocket klien real-time

Bagian berikut akan menunjukkan kepada Anda arsitektur di balik AWS AppSync kemampuan real-time.

Implementasi WebSocket klien real-time untuk langganan GraphQL

Diagram urutan dan langkah-langkah berikut menunjukkan alur kerja langganan real-time antara WebSocket klien, klien HTTP, dan. AWS AppSync



1. Klien membuat WebSocket koneksi dengan titik akhir AWS AppSync real-time. Jika ada kesalahan jaringan, klien harus melakukan backoff eksponensial yang gelisah. Untuk informasi lebih lanjut, lihat Eksponensial backoff dan jitter di Blog Arsitektur. AWS
2. Setelah berhasil membangun WebSocket koneksi, klien mengirim `connection_init` pesan.
3. Klien menunggu `connection_ack` pesan dari AWS AppSync. Pesan ini menyertakan `connectionTimeoutMs` parameter, yang merupakan waktu tunggu maksimum dalam milidetik untuk pesan `"ka"` (keep-alive).

4. AWS AppSync mengirim `"ka"` pesan secara berkala. Klien melacak waktu menerima setiap `"ka"` pesan. Jika klien tidak menerima `"ka"` pesan dalam `connectionTimeoutMs` milidetik, klien harus menutup koneksi.
5. Klien mendaftarkan langganan dengan mengirim pesan `start` berlangganan. WebSocket Koneksi tunggal mendukung beberapa langganan, bahkan jika mereka berada dalam mode otorisasi yang berbeda.
6. Klien menunggu untuk mengirim `start_ack` pesan AWS AppSync untuk mengkonfirmasi langganan yang berhasil. Jika ada kesalahan, AWS AppSync mengembalikan `"type": "error"` pesan.
7. Klien mendengarkan acara berlangganan, yang dikirim setelah mutasi yang sesuai dipanggil. Kueri dan mutasi biasanya dikirim melalui `https://` titik akhir AWS AppSync GraphQL. Langganan mengalir melalui titik akhir AWS AppSync real-time menggunakan secure WebSocket ()`wss://`.
8. Klien membatalkan pendaftaran langganan dengan mengirim pesan `stop` berlangganan.
9. Setelah membatalkan pendaftaran semua langganan dan memeriksa bahwa tidak ada pesan yang ditransfer melalui WebSocket, klien dapat memutuskan sambungan dari koneksi. WebSocket

Detail jabat tangan untuk membuat koneksi WebSocket

Untuk menghubungkan dan memulai jabat tangan yang sukses dengan AWS AppSync, WebSocket klien membutuhkan yang berikut:

- Titik akhir AWS AppSync waktu nyata
- Sebuah string query yang berisi `header` dan `payload` parameter:
  - `header`: Berisi informasi yang relevan dengan AWS AppSync titik akhir dan otorisasi. Ini adalah string yang dikodekan base64 dari objek JSON yang dirangkai. Konten objek JSON bervariasi tergantung pada mode otorisasi.
  - `payload`: String yang dikodekan Base64 dari. `payload`

Dengan persyaratan ini, WebSocket klien dapat terhubung ke URL, yang berisi titik akhir real-time dengan string kueri, menggunakan `graphql-ws` sebagai WebSocket protokol.

Menemukan titik akhir real-time dari titik akhir GraphQL

Endpoint GraphQL dan endpoint real-time sedikit berbeda AWS AppSync dalam protokol dan domain. AWS AppSync Anda dapat mengambil titik akhir GraphQL menggunakan perintah (). AWS Command Line Interface AWS CLI `aws appsync get-graphql-api`

AWS AppSync Titik akhir GraphQL:

```
https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql
```

AWS AppSync titik akhir waktu nyata:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/
graphql
```

Aplikasi dapat terhubung ke titik akhir AWS AppSync GraphQL `https://` () menggunakan klien HTTP apa pun untuk kueri dan mutasi. Aplikasi dapat terhubung ke titik akhir AWS AppSync real-time (`wss://`) menggunakan WebSocket klien apa pun untuk berlangganan.

Dengan nama domain khusus, Anda dapat berinteraksi dengan kedua titik akhir menggunakan satu domain. Misalnya, jika Anda mengonfigurasi `api.example.com` sebagai domain kustom, Anda dapat berinteraksi dengan GraphQL dan titik akhir real-time menggunakan URL berikut:

AWS AppSync titik akhir GraphQL domain kustom:

```
https://api.example.com/graphql
```

AWS AppSync titik akhir real-time domain kustom:

```
wss://api.example.com/graphql/realtime
```

Format parameter header berdasarkan mode otorisasi AWS AppSync API

Format `header` objek yang digunakan dalam string kueri koneksi bervariasi tergantung pada mode otorisasi AWS AppSync API. `host`Bidang dalam objek mengacu pada titik akhir AWS AppSync GraphQL, yang digunakan untuk memvalidasi koneksi bahkan jika panggilan dilakukan `wss://` terhadap titik akhir real-time. Untuk memulai jabat tangan dan membuat koneksi resmi, `payload` harus menjadi objek JSON kosong.

Kunci API

Header kunci API

Isi header

- `"host": <string>`: Host untuk titik akhir AWS AppSync GraphQL atau nama domain kustom Anda.
- `"x-api-key": <string>`: Kunci API dikonfigurasi untuk AWS AppSync API.

Contoh

```
{
    "host":"example1234567890000.appsync-api.us-east-1.amazonaws.com",
    "x-api-key":"da2-12345678901234567890123456"
}
```

Konten muatan

```
{}
```

Permintaan URL

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJob3N0IjoiZXhhbXBsZTEyMzQ1Njc4OTAwMDAuYXBwc3luYy1hcGkudXMtZWFzdC0xLmFtYXpvbmF3cy5jb20i
```

Kumpulan pengguna Amazon Cognito dan OpenID Connect (OIDC)

Amazon Cognito dan OidCheader

Isi header:

- `"Authorization": <string>`: Token ID JWT. Header dapat menggunakan skema Bearer.
- `"host": <string>`: Host untuk titik akhir AWS AppSync GraphQL atau nama domain kustom Anda.

Contoh:

```
{

 "Authorization":"eyEXAMPLEiJjbG5xb3A5eW5MK09QYXIrMTJHWEFLSXBieU5WNHhsQjEXAMPLEnM2WldvPSIsImFsZ
zEE2DJH7sHOl2zxYi7f-SmEGoh2AD8emxQRYajByz-rE4Jh0QOymN2Ys-ZIkMpVBTPgu-
TMWDyOHhDUmUj2OP82yeZ3wlZAtr_gM4LzjXUXmI_K2yGjuXfXTaa1mvQEBG0mQfVd7SfwXB-
jcv4RYVi6j25qgow9Ew52ufurPqaK-3WAKG32KpV8J4-Wejq8t0c-
yA7sb8EnB551b7TU93uKRiVVK3E55Nk5ADPoam_WYE45i3s5qVAP_-InW75NUoOCGTsS8YWMfb6ecHYJ-1j-
bzA27zaT9VjctXn9byNFZmEXAMPLExw",
    "host":"example1234567890000.appsync-api.us-east-1.amazonaws.com"
}
```

Konten muatan:

```
{}
```

Permintaan URL:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJBdXRob3JpemF0aW9uIjoiZXlKcmFXUWlPaUpqYkc1eGIzQTVlVzVNSzA5UVlYSXJNVEpIV0VGTFNYQmllVTVX
```

IAM

Tajuk IAM

Konten header

- `"accept": "application/json, text/javascript"`: `<string>` Parameter konstan.
- `"content-encoding": "amz-1.0"`: `<string>` Parameter konstan.
- `"content-type": "application/json; charset=UTF-8"`: `<string>` Parameter konstan.
- `"host": <string>`: Ini adalah host untuk titik akhir AWS AppSync GraphQL.
  - `"x-amz-date": <string>`: Stempel waktu harus dalam UTC dan dalam format ISO 8601 berikut: YYYYMMDD'T'HHMMSS'Z'. Misalnya, 20150830T123600Z adalah stempel waktu yang valid. Jangan sertakan milidetik dalam stempel waktu. Untuk informasi selengkapnya, lihat Menangani tanggal di Tanda Tangan Versi 4 di Referensi Umum AWS.
  - `"X-Amz-Security-Token": <string>`: Token AWS sesi, yang diperlukan saat menggunakan kredensil keamanan sementara. Untuk informasi selengkapnya, lihat Menggunakan kredensyal sementara dengan AWS sumber daya di Panduan Pengguna IAM.
  - `"Authorization": <string>`: Tanda tangan Versi 4 (SiGv4) informasi penandatanganan untuk titik akhir. AWS AppSync Untuk informasi selengkapnya tentang proses penandatanganan, lihat Tugas 4: Menambahkan tanda tangan ke permintaan HTTP di Referensi Umum AWS.

Permintaan HTTP penandatanganan SigV4 menyertakan URL kanonik, yang merupakan titik akhir AWS AppSync GraphQL dengan ditambahkan. `/connect` AWSWilayah titik akhir layanan adalah Wilayah yang sama dengan tempat Anda menggunakan AWS AppSync API, dan nama layanannya adalah 'appsync'. Permintaan HTTP untuk menandatangani adalah sebagai berikut:

```
{
  url: "https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql/
connect",
```

```
  data: "{}",
  method: "POST",
  headers: {
    "accept": "application/json, text/javascript",
    "content-encoding": "amz-1.0",
    "content-type": "application/json; charset=UTF-8",
  }
}
```

Contoh

```
{
  "accept": "application/json, text/javascript",
  "content-encoding": "amz-1.0",
  "content-type": "application/json; charset=UTF-8",
  "host": "example1234567890000.appsync-api.us-east-1.amazonaws.com",
  "x-amz-date": "20200401T001010Z",
  "X-Amz-Security-Token":
 "AgEXAMPLEZ2luX2VjEAoaDmFwLXNvdXRoZWFEXAMPLEcwRQIgAh97Cljq7wOPL8KsxP3YtDuyc/9hAj8PhJ7Fvf38SgoC
+
+pEagWCveZUjKEn0zyUhBEXAMPLEjj//////////8BEXAMPLExODk2NDgyNzg1NSIMo1mWnpESWUoYw4BkKqEFSrm3DXuL8
+ZbVc4JKjDP4vUCKNR6Le9C9pZp9PsW0NoFy3vLBUdAXEXAMPLEOVG8feXfiEEA+1khgFK/
wEtwR+9zF7NaMMMse07wN2gG2tH0eKMEXAMPLEQX+sMbytQo8iepP9PZOzlZsSFb/
dP5Q8hk6YEXAMPLEYcKZsTkDAq2uKFQ8mYUVA9EtQnNRiFLEY83aKvG/tqLWNnGlSNVx7SMcfovkFDqQamm
+88y1OwwAEYK7qcoceX6Z7GGcaYuIfGpaX2MCCELeQvZ+8WxEgOnIfz7GYvsYNjLZSaRnV4G
+ILY1F0QNW64S9Nvj
+BwDg3ht2CrNvpwjVYlj9U3nmxE0UG5ne83LL5hhqMpm25kmL7enVgw2kQzmU2id4IKu0C/
WaoDRuO2F5zE63vJbxN8AYs7338+4B4HBb6BZ6OUgg96Q15RA41/
gIqxaVPxyTpDfTU5GfSLxocdYeniqqpFMtZG2n9d0u7GsQNcFkNcG3qDZm4tDo8tZbuym0a2VcF2E5hFEgXBa
+XLJCfXi/77OqAEjP0x7Qdk3B43p8KG/BaioP5RsV8zBGvH1zAgyPha2rN70/
tT13yrmPd5QYEfwzexjKrV4mWIuRg8NTHYSZJUaeyCwTom80VFUJXG
+GYTUyv5W22aBcnoRGiCiKEYTLOkgXecdKFTHmcIAejQ9Welr0a196Kq87w5KNMCkcCGFnwBNFLmfnbpNqT6rUBxxs3X5nt
aox0FtHX21eF6qIGT8j1z+l2opU+ggwUgkhUUgCH2TfqBj+MLMVVvpgqJsPKt582caFKArIFIvO
+9QupxLnEH2hz04TMTfnU6bQC6z1buVe7h
+tOLnh1YPFsLQ88anib/7TTC8k9DsBTq0ASe8R2GbSEsmO9qbbMwgEaYUhOKtGeyQsSJdhSk6XxXThrWL9EnwBCXDkICMqd
+WgtPtKOOweDlCaRs3R2qXcbNgVhleMk4IWnF8D1695AenU1LwHjOJLkCjxgNFiWAFEPH9aEXAMPLExA==",
  "Authorization": "AWS4-HMAC-SHA256 Credential=XXXXXXXXXXXXXXXXXXX/20200401/
us-east-1/appsync/aws4_request, SignedHeaders=accept;content-
encoding;content-type;host;x-amz-date;x-amz-security-token,
 Signature=83EXAMPLEbcc1fe3ee69f75cd5ebbf4cb4f150e4f99cec869f149c5EXAMPLEdc"
}
```

Konten muatan

```
{}
```

Permintaan URL

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyEXAMPLEHQiOiJhcHBsaWNhdGlvbi9qc29uLCB0ZXh0L2phdmFEXAMPLEQiLCJjb250ZW50LWVuY29kaW5nIjoE
```

Untuk menandatangani permintaan menggunakan domain kustom:

```
{
  url: "https://api.example.com/graphql/connect",
  data: "{}",
  method: "POST",
  headers: {
    "accept": "application/json, text/javascript",
    "content-encoding": "amz-1.0",
    "content-type": "application/json; charset=UTF-8",
  }
}
```

Contoh

```
{
  "accept": "application/json, text/javascript",
  "content-encoding": "amz-1.0",
  "content-type": "application/json; charset=UTF-8",
  "host": "api.example.com",
  "x-amz-date": "20200401T001010Z",
  "X-Amz-Security-Token":
 "AgEXAMPLEZ2luX2VjEAoaDmFwLXNvdXRoZWFEXAMPLEcwRQIgAh97Cljq7wOPL8KsxP3YtDuyc/9hAj8PhJ7Fvf38SgoC
+
+pEagWCveZUjKEn0zyUhBEXAMPLEjj/////////8BEXAMPLExODk2NDgyNzg1NSIMo1mWnpESWUoYw4BkKqEFSrm3DXuL8
+ZbVc4JKjDP4vUCKNR6Le9C9pZp9PsW0NoFy3vLBUdAXEXAMPLEOVG8feXfiEEA+1khgFK/
wEtwR+9zF7NaMMMse07wN2gG2tH0eKMEXAMPLEQX+sMbytQo8iepP9PZOzlZsSFb/
dP5Q8hk6YEXAMPLEYcKZsTkDAq2uKFQ8mYUVA9EtQnNRiFLEY83aKvG/tqLWNnGlSNVx7SMcfovkFDqQamm
+88y1OwwAEYK7qcoceX6Z7GGcaYuIfGpaX2MCCELeQvZ+8WxEgOnIfz7GYvsYNjLZSaRnV4G
+ILY1F0QNW64S9Nvj
+BwDg3ht2CrNvpwjVYlj9U3nmxE0UG5ne83LL5hhqMpm25kmL7enVgw2kQzmU2id4IKu0C/
WaoDRuO2F5zE63vJbxN8AYs7338+4B4HBb6BZ6OUgg96Q15RA41/
gIqxaVPxyTpDfTU5GfSLxocdYeniqqpFMtZG2n9d0u7GsQNcFkNcG3qDZm4tDo8tZbuym0a2VcF2E5hFEgXBa
+XLJCfXi/77OqAEjP0x7Qdk3B43p8KG/BaioP5RsV8zBGvH1zAgyPha2rN70/
```

```
tT13yrmPd5QYEfwzexjKrV4mWIuRg8NTHYSZJUaeyCwTom80VFUJXG
+GYTUyv5W22aBcnoRGiCiKEYTLOkgXecdKFTHmcIAejQ9Welr0a196Kq87w5KNMCkcCGFnwBNFLmfnbpNqT6rUBxxs3X5nt
aox0FtHX21eF6qIGT8j1z+l2opU+ggwUgkhUUgCH2TfqBj+MLMVVvpgqJsPKt582caFKArIFIvO
+9QupxLnEH2hz04TMTfnU6bQC6z1buVe7h
+tOLnh1YPFsLQ88anib/7TTC8k9DsBTq0ASe8R2GbSEsmO9qbbMwgEaYUhOKtGeyQsSJdhSk6XxXThrWL9EnwBCXDkICMqd
+WgtPtKOOweDlCaRs3R2qXcbNgVhleMk4IWnF8D1695AenU1LwHjOJLkCjxgNFiWAFEPH9aEXAMPLExA==",
  "Authorization": "AWS4-HMAC-SHA256 Credential=XXXXXXXXXXXXXXXXXX/20200401/
us-east-1/appsync/aws4_request, SignedHeaders=accept;content-
encoding;content-type;host;x-amz-date;x-amz-security-token,
 Signature=83EXAMPLEbcc1fe3ee69f75cd5ebbf4cb4f150e4f99cec869f149c5EXAMPLEdc"
}
```

Konten muatan

```
{}
```

Permintaan URL

```
wss://api.example.com/graphql?
header=eyEXAMPLEHQiOiJhcHBsaWNhdGlvbi9qc29uLCB0ZXh0L2phdmFEXAMPLEQiLCJjb250ZW50LWVuY29kaW5nIjoE
```

> **Note**
>
> Satu WebSocket koneksi dapat memiliki beberapa langganan (bahkan dengan mode otentikasi yang berbeda). Salah satu cara untuk menerapkan ini adalah dengan membuat WebSocket koneksi untuk langganan pertama dan kemudian menutupnya ketika langganan terakhir tidak terdaftar. Anda dapat mengoptimalkan ini dengan menunggu beberapa detik sebelum menutup WebSocket koneksi, jika aplikasi berlangganan segera setelah langganan terakhir tidak terdaftar. Untuk contoh aplikasi seluler, saat mengubah dari satu layar ke layar lainnya, saat melepas pemasangan, ia menghentikan langganan, dan pada acara pemasangan, ia memulai langganan yang berbeda.

Otorisasi Lambda

Header otorisasi Lambda

Konten header

- `"Authorization": <string>`: Nilai yang diteruskan sebagai`authorizationToken`.

- `"host": <string>`: Host untuk titik akhir AWS AppSync GraphQL atau nama domain kustom Anda.

Contoh

```
{

 "Authorization":"M0UzQzM1MkQtMkI0Ni00OTZCLUI1NkQtMUM0MTQ0QjVBRTczCkI1REEzRTIxLTk5NzItNDJENi1BQ
    "host":"example1234567890000.appsync-api.us-east-1.amazonaws.com"
}
```

Konten muatan

```
{}
```

Permintaan URL

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJBdXRob3JpemF0aW9uIjoiZXlKcmFXUWlPaUpqYkc1eGIzQTVlVzVNSzA5UVlYSXJNVEpIV0VGTFNYQmllVTVX
```

WebSocketOperasi waktu nyata

Setelah memulai WebSocket jabat tangan yang sukses dengan AWS AppSync, klien harus mengirim pesan berikutnya untuk terhubung ke AWS AppSync operasi yang berbeda. Pesan-pesan ini memerlukan data berikut:

- `type`: Jenis operasi.
- `id`: Pengidentifikasi unik untuk langganan. Kami merekomendasikan menggunakan UUID untuk tujuan ini.
- `payload`: Muatan terkait, tergantung pada jenis operasi.

`type`Bidang adalah satu-satunya bidang yang diperlukan; `id` dan `payload` bidang adalah opsional.

Urutan peristiwa

Agar berhasil memulai, membuat, mendaftar, dan memproses permintaan berlangganan, klien harus melangkah melalui urutan berikut:

1. Inisialisasi koneksi () `connection_init`

2. Pengakuan koneksi () `connection_ack`
3. Pendaftaran berlangganan (`start`)
4. Pengakuan berlangganan () `start_ack`
5. Memproses berlangganan (`data`)
6. Unregistrasi langganan () `stop`

Koneksi pesan init

Setelah jabat tangan berhasil, klien harus mengirim `connection_init` pesan untuk mulai berkomunikasi dengan titik akhir AWS AppSync real-time. Tanpa langkah ini, semua pesan lain diabaikan. Pesannya adalah string yang diperoleh dengan merangkai objek JSON berikut sebagai berikut:

```
{ "type": "connection_init" }
```

Koneksi mengakui pesan

Setelah mengirim `connection_init` pesan, klien harus menunggu `connection_ack` pesan. Semua pesan yang dikirim sebelum menerima `connection_ack` diabaikan. Pesan harus berbunyi sebagai berikut:

```
{
  "type": "connection_ack",
  "payload": {
    // Time in milliseconds waiting for ka message before the client should terminate
 the WebSocket connection
    "connectionTimeoutMs": 300000
  }
}
```

Pesan keep-alive

Selain pesan pengakuan koneksi, klien secara berkala menerima pesan keep-alive. Jika klien tidak menerima pesan keep-alive dalam periode batas waktu koneksi, klien harus menutup koneksi. AWS AppSync terus mengirim pesan-pesan ini dan melayani langganan terdaftar sampai mematikan koneksi secara otomatis (setelah 24 jam). Pesan yang tetap hidup adalah detak jantung dan tidak perlu klien untuk mengakuinya.

```
{ "type": "ka" }
```

Pesan pendaftaran langganan

Setelah klien menerima `connection_ack` pesan, klien dapat mengirim pesan pendaftaran langganan ke AWS AppSync. Jenis pesan ini adalah objek JSON stringified yang berisi bidang-bidang berikut:

- `"id": <string>`: ID langganan. ID ini harus unik untuk setiap langganan, jika tidak server mengembalikan kesalahan yang menunjukkan bahwa ID langganan diduplikasi.
- `"type": "start"`: `<string>` Parameter konstan.
- `"payload": <Object>`: Objek yang berisi informasi yang relevan dengan langganan.
  - `"data": <string>`: Sebuah objek JSON stringified yang berisi query GraphQL dan variabel.
    - `"query": <string>`: Operasi GraphQL.
    - `"variables": <Object>`: Sebuah objek yang berisi variabel untuk query.
  - `"extensions": <Object>`: Objek yang berisi objek otorisasi.
- `"authorization": <Object>`: Objek yang berisi bidang yang diperlukan untuk otorisasi.

Objek otorisasi untuk pendaftaran berlangganan

Aturan yang sama di Format parameter header berdasarkan mode otorisasi AWS AppSync API bagian ini berlaku untuk objek otorisasi. Satu-satunya pengecualian adalah untuk IAM, di mana informasi tanda tangan SiGv4 sedikit berbeda. Untuk lebih jelasnya, lihat contoh IAM.

Contoh menggunakan kumpulan pengguna Amazon Cognito:

```
{
  "id": "ee849ef0-cf23-4cb8-9fcb-152ae4fd1e69",
  "payload": {
    "data": "{\"query\":\"subscription onCreateMessage {\\n onCreateMessage {\\n
 __typename\\n message\\n }\\n }\",\"variables\":{}}",
      "extensions": {
        "authorization": {
          "Authorization":
 "eyEXAMPLEiJjbG5xb3A5eW5MK09QYXIrMTJEXAMPLEBieU5WNHhsQjhPVW9YMnM2WldvPSIsImFsZyI6IlEXAMPLEn0.e
qTCtrYeboUJ4luRSTPXaNewNeEXAMPLE14C6sfg05tO0fOMpiUwj9k19gtNCCMqoSsjtQoUweFnH4JYa5EXAMPLEVxOyQEQ
RWvW7yQU3sNQRLEXAMPLEcd0yufBiCYs3dfQxTTdvR1B6Wz6CD78lfNeKqfzzUn2beMoup2h6EXAMPLE4ow8cUPUPvG0DzR
          "host": "example1234567890000.appsync-api.us-east-1.amazonaws.com"
```

```
        }
      }
  },
  "type": "start"
}
```

Contoh menggunakan IAM:

```
{
  "id": "eEXAMPLE-cf23-1234-5678-152EXAMPLE69",
  "payload": {
    "data": "{\"query\":\"subscription onCreateMessage {\\n onCreateMessage {\\n
 __typename\\n message\\n }\\n }\",\"variables\":{}}",
    "extensions": {
      "authorization": {
        "accept": "application/json, text/javascript",
        "content-type": "application/json; charset=UTF-8",
        "X-Amz-Security-Token":
 "AgEXAMPLEZ2luX2VjEAoaDmFwLXNvdXRoZWFEXAMPLEcwRQIgAh97Cljq7wOPL8KsxP3YtDuyc/9hAj8PhJ7Fvf38SgoC
+
+pEagWCveZUjKEn0zyUhBEXAMPLEjj/////////8BEXAMPLExODk2NDgyNzg1NSIMo1mWnpESWUoYw4BkKqEFSrm3DXuL8
+ZbVc4JKjDP4vUCKNR6Le9C9pZp9PsW0NoFy3vLBUdAXEXAMPLEOVG8feXfiEEA+1khgFK/
wEtwR+9zF7NaMMMse07wN2gG2tH0eKMEXAMPLEQX+sMbytQo8iepP9PZOzlZsSFb/
dP5Q8hk6YEXAMPLEYcKZsTkDAq2uKFQ8mYUVA9EtQnNRiFLEY83aKvG/tqLWNnGlSNVx7SMcfovkFDqQamm
+88y1OwwAEYK7qcoceX6Z7GGcaYuIfGpaX2MCCELeQvZ+8WxEgOnIfz7GYvsYNjLZSaRnV4G
+ILY1F0QNW64S9Nvj
+BwDg3ht2CrNvpwjVYlj9U3nmxE0UG5ne83LL5hhqMpm25kmL7enVgw2kQzmU2id4IKu0C/
WaoDRuO2F5zE63vJbxN8AYs7338+4B4HBb6BZ6OUgg96Q15RA41/
gIqxaVPxyTpDfTU5GfSLxocdYeniqqpFMtZG2n9d0u7GsQNcFkNcG3qDZm4tDo8tZbuym0a2VcF2E5hFEgXBa
+XLJCfXi/77OqAEjP0x7Qdk3B43p8KG/BaioP5RsV8zBGvH1zAgyPha2rN70/
tT13yrmPd5QYEfwzexjKrV4mWIuRg8NTHYSZJUaeyCwTom80VFUJXG
+GYTUyv5W22aBcnoRGiCiKEYTLOkgXecdKFTHmcIAejQ9Welr0a196Kq87w5KNMCkcCGFnwBNFLmfnbpNqT6rUBxxs3X5nt
aox0FtHX21eF6qIGT8j1z+l2opU+ggwUgkhUUgCH2TfqBj+MLMVVvpgqJsPKt582caFKArIFIvO
+9QupxLnEH2hz04TMTfnU6bQC6z1buVe7h
+tOLnh1YPFsLQ88anib/7TTC8k9DsBTq0ASe8R2GbSEsmO9qbbMwgEaYUhOKtGeyQsSJdhSk6XxXThrWL9EnwBCXDkICMqd
+WgtPtKOOweDlCaRs3R2qXcbNgVhleMk4IWnF8D1695AenU1LwHjOJLkCjxgNFiWAFEPH9aEXAMPLExA==",
        "Authorization": "AWS4-HMAC-SHA256 Credential=XXXXXXXXXXXXXXXXXXXX/20200401/
us-east-1/appsync/aws4_request, SignedHeaders=accept;content-
encoding;content-type;host;x-amz-date;x-amz-security-token,
 Signature=b90131a61a7c4318e1c35ead5dbfdeb46339a7585bbdbeceeaff51f4022eb1fd",
        "content-encoding": "amz-1.0",
        "host": "example1234567890000.appsync-api.us-east-1.amazonaws.com",
        "x-amz-date": "20200401T001010Z"
```

```
      }
    }
  },
  "type": "start"
}
```

Contoh menggunakan nama domain kustom:

```
{
  "id": "key-cf23-4cb8-9fcb-152ae4fd1e69",
  "payload": {
    "data": "{\"query\":\"subscription onCreateMessage {\\n onCreateMessage {\\n
 __typename\\n message\\n }\\n }\",\"variables\":{}}",
      "extensions": {
        "authorization": {
          "x-api-key": "da2-12345678901234567890123456",
          "host": "api.example.com"
         }
      }
  },
  "type": "start"
}
```

Tanda tangan SigV4 tidak perlu ditambahkan `/connect` ke URL, dan operasi GraphQL stringified JSON menggantikan. `data` Berikut ini adalah contoh permintaan tanda tangan SigV4:

```
{
  url: "https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql",
  data: "{\"query\":\"subscription onCreateMessage {\\n onCreateMessage {\\n __typename
\\n message\\n }\\n }\",\"variables\":{}}",
  method: "POST",
  headers: {
    "accept": "application/json, text/javascript",
    "content-encoding": "amz-1.0",
    "content-type": "application/json; charset=UTF-8",
  }
}
```

Pesan pengakuan berlangganan

Setelah mengirim pesan mulai berlangganan, klien harus menunggu AWS AppSync untuk mengirim `start_ack` pesan. `start_ack`Pesan tersebut menunjukkan bahwa langganan berhasil.

Contoh pengakuan berlangganan:

```
{
  "type": "start_ack",
  "id": "eEXAMPLE-cf23-1234-5678-152EXAMPLE69"
}
```

Pesan kesalahan

Jika koneksi init atau pendaftaran langganan gagal, atau jika langganan berakhir dari server, server mengirimkan pesan kesalahan ke klien:

- `"type": "error"`: `<string>` Parameter konstan.
- `"id": <string>`: ID dari langganan terdaftar yang sesuai, jika relevan.
- `"payload" <Object>`: Objek yang berisi informasi kesalahan yang sesuai.

Contoh:

```
{
  "type": "error",
  "payload": {
    "errors": [
      {
        "errorType": "LimitExceededError",
        "message": "Rate limit exceeded"
      }
    ]
  }
}
```

Memproses pesan data

Ketika klien mengirimkan mutasi, AWS AppSync mengidentifikasi semua pelanggan yang tertarik padanya dan mengirim `"type":"data"` pesan ke masing-masing menggunakan langganan yang sesuai `id` dari operasi berlangganan. `"start"` Klien diharapkan untuk melacak langganan `id` yang dikirimkannya sehingga ketika menerima pesan data, klien dapat mencocokkannya dengan langganan yang sesuai.

- `"type": "data"`: `<string>` Parameter konstan.

- `"id": <string>`: ID dari langganan terdaftar yang sesuai.
- `"payload" <Object>`: Objek yang berisi informasi berlangganan.

Contoh:

```
{
  "type": "data",
  "id": "ee849ef0-cf23-4cb8-9fcb-152ae4fd1e69",
  "payload": {
    "data": {
      "onCreateMessage": {
        "__typename": "Message",
        "message": "test"
      }
    }
  }
}
```

Pesan pembatalan pendaftaran langganan

Saat aplikasi ingin berhenti mendengarkan acara berlangganan, klien harus mengirim pesan dengan objek JSON yang dirangkai berikut:

- `"type": "stop"`: `<string>` Parameter konstan.
- `"id": <string>`: ID langganan untuk membatalkan pendaftaran.

Contoh:

```
{
  "type":"stop",
  "id":"ee849ef0-cf23-4cb8-9fcb-152ae4fd1e69"
}
```

AWS AppSync mengirimkan kembali pesan konfirmasi dengan objek JSON stringifikasi berikut:

- `"type": "complete"`: `<string>` Parameter konstan.
- `"id": <string>`: ID langganan yang tidak terdaftar.

Setelah klien menerima pesan konfirmasi, ia tidak menerima pesan lagi untuk langganan khusus ini.

Contoh:

```
{
  "type":"complete",
  "id":"eEXAMPLE-cf23-1234-5678-152EXAMPLE69"
}
```

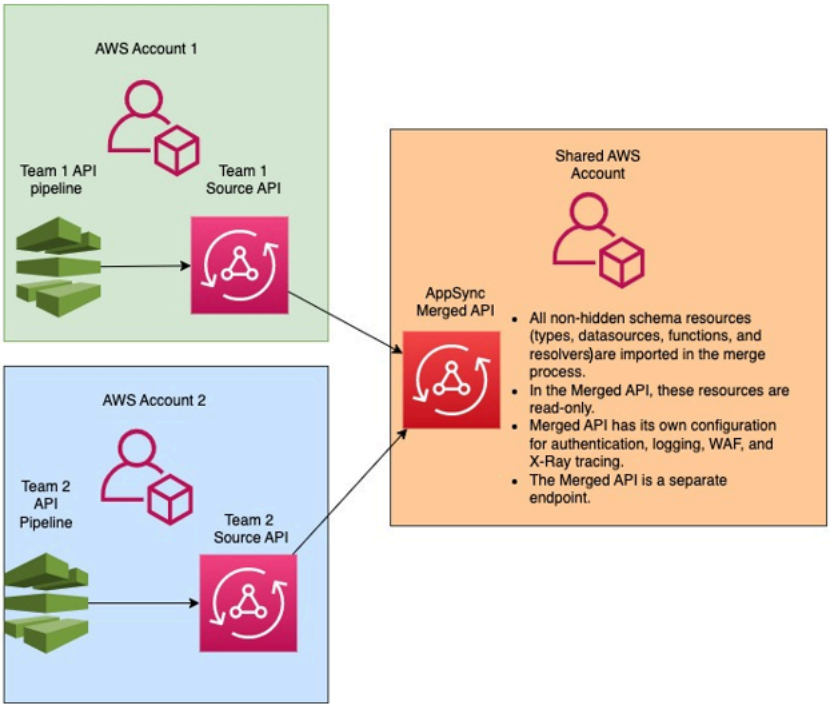
Memutuskan sambungan WebSocket

Sebelum memutuskan sambungan, untuk menghindari kehilangan data, klien harus memiliki logika yang diperlukan untuk memeriksa bahwa tidak ada operasi saat ini di tempat melalui WebSocket koneksi. Semua langganan harus tidak terdaftar sebelum memutuskan sambungan dari. WebSocket

API yang digabungkan

Karena penggunaan GraphQL meluas dalam suatu organisasi, trade-off antara ease-of-use kecepatan pengembangan API dan API dapat muncul. Di satu sisi, organisasi mengadopsi AWS AppSync dan GraphQL untuk menyederhanakan pengembangan aplikasi dengan memberi pengembang API fleksibel yang dapat mereka gunakan untuk mengakses, memanipulasi, dan menggabungkan data dengan aman dari satu atau lebih domain data dengan satu panggilan jaringan. Di sisi lain, tim dalam organisasi yang bertanggung jawab atas domain data yang berbeda digabungkan menjadi satu titik akhir API GraphQL mungkin menginginkan kemampuan untuk membuat, mengelola, dan menerapkan pembaruan API secara independen satu sama lain untuk meningkatkan kecepatan pengembangan mereka.

Untuk mengatasi ketegangan ini, fitur AWS AppSync Merged API memungkinkan tim dari domain data yang berbeda untuk secara independen membuat dan menerapkan AWS AppSync API (misalnya, skema GraphQL, resolver, sumber data, dan fungsi), yang kemudian dapat digabungkan menjadi satu API gabungan. Ini memberi organisasi kemampuan untuk mempertahankan API lintas domain yang mudah digunakan, dan cara bagi tim yang berbeda yang berkontribusi pada API tersebut kemampuan untuk secara cepat dan mandiri membuat pembaruan API.



Dengan menggunakan API Gabungan, organisasi dapat mengimpor sumber daya dari beberapa AWS AppSync API sumber independen ke dalam satu titik akhir API AWS AppSync Gabungan. Untuk melakukannya, Anda AWS AppSync dapat membuat daftar API sumber AWS AppSync sumber, lalu menggabungkan semua metadata yang terkait dengan API sumber termasuk skema, tipe, sumber data, resolver, dan fungsi, ke dalam API gabungan baru. AWS AppSync

Selama penggabungan, ada kemungkinan konflik penggabungan akan terjadi karena ketidakkonsistenan dalam konten data API sumber seperti konflik penamaan tipe saat menggabungkan beberapa skema. Untuk kasus penggunaan sederhana di mana tidak ada definisi dalam konflik API sumber, tidak perlu memodifikasi skema API sumber. API Gabungan yang dihasilkan hanya mengimpor semua jenis, resolver, sumber data, dan fungsi dari API sumber asli. AWS AppSync Untuk kasus penggunaan kompleks di mana konflik muncul, pengguna/tim harus menyelesaikan konflik melalui berbagai cara. AWS AppSyncmemberi pengguna beberapa alat dan contoh yang dapat mengurangi konflik penggabungan.

Penggabungan berikutnya yang dikonfigurasi AWS AppSync akan menyebarkan perubahan yang dibuat di API sumber ke API Gabungan terkait.

Gabungan API dan Federasi

Ada banyak solusi dan pola dalam komunitas GraphQL untuk menggabungkan skema GraphQL dan memungkinkan kolaborasi tim melalui grafik bersama. AWS AppSync API gabungan mengadopsi pendekatan waktu pembuatan untuk komposisi skema, di mana API sumber digabungkan menjadi API Gabungan yang terpisah. Pendekatan alternatif adalah dengan melapisi router run time di beberapa API sumber atau sub-grafik. Dalam pendekatan ini, router menerima permintaan, mereferensikan skema gabungan yang dipertahankannya sebagai metadata, membuat rencana permintaan, dan kemudian mendistribusikan elemen permintaan di seluruh sub-grafik/server yang mendasarinya. Tabel berikut membandingkan pendekatan build-time API Gabungan dengan pendekatan run-time berbasis router untuk komposisi skema GraphQL: AWS AppSync

Feature	AppSync Merged API	Router-based solutions
Sub-graphs managed independently	Yes	Yes
Sub-graphs addressable independently	Yes	Yes
Automated schema compositi on	Yes	Yes
Automated conflict detection	Yes	Yes
Conflict resolution via schema directives	Yes	Yes
Supported sub-graph servers	AWS AppSync*	Varies
Network complexity	Single, merged API means no extra network hops.	Multi-layer architecture requires query planning and delegation, sub-query parsing and serialization/deserializati on, and reference resolvers in sub-graphs to perform joins.
Observability support	Built-in monitoring, logging, and tracing. A single, Merged	Build-your-own observability across router and all associate

	API server means simplified debugging.	d sub-graph servers. Complex debugging across distributed system.
Authorization support	Built in support for multiple authorization modes.	Build-your-own authorization rules.
Cross account security	Built-in support for cross-AWS cloud account associations.	Build-your-own security model.
Subscriptions support	Yes	No

* API AWS AppSync gabungan hanya dapat dikaitkan dengan API AWS AppSync sumber. Jika Anda memerlukan dukungan untuk komposisi skema di seluruh AWS AppSync dan AWS AppSync non-sub-grafik, Anda dapat menghubungkan satu atau beberapa AWS AppSync GraphQL dan/atau API Gabungan ke dalam solusi berbasis router. Misalnya, lihat blog referensi untuk menambahkan AWS AppSync API sebagai sub-grafik menggunakan arsitektur berbasis router dengan Apollo Federation v2: Apollo GraphQL Federation with. AWS AppSync

Topik

Resolusi konflik API yang digabungkan

Jika terjadi konflik gabungan, AWS AppSync berikan beberapa alat dan contoh kepada pengguna untuk membantu memecahkan masalah.

Arahan skema API yang digabungkan

AWS AppSynctelah memperkenalkan beberapa arahan GraphQL yang dapat digunakan untuk-mengurangi atau menyelesaikan konflik di seluruh API sumber:

- @canonical: Arahan ini menetapkan prioritas tipe/bidang dengan nama dan data yang serupa. Jika dua atau lebih API sumber memiliki tipe atau bidang GraphQL yang sama, salah satu API dapat membuat anotasi jenis atau bidangnya sebagai kanonik, yang akan diprioritaskan selama penggabungan. Jenis/bidang yang bertentangan yang tidak dianotasi dengan arahan ini di API sumber lain diabaikan saat digabungkan.
- @hidden: Arahan ini merangkum jenis/bidang tertentu untuk menghapusnya dari proses penggabungan. Tim mungkin ingin menghapus atau menyembunyikan jenis atau operasi tertentu di API sumber sehingga hanya klien internal yang dapat mengakses data tertentu yang diketik. Dengan direktif ini dilampirkan, tipe atau bidang tidak digabungkan ke dalam API Gabungan.
- @renamed: Arahan ini mengubah nama jenis/bidang untuk mengurangi konflik penamaan. Ada situasi di mana API yang berbeda memiliki jenis atau nama bidang yang sama. Namun, semuanya harus tersedia dalam skema gabungan. Cara sederhana untuk memasukkan semuanya ke dalam API Gabungan adalah dengan mengganti nama bidang menjadi sesuatu yang serupa tetapi berbeda.

Untuk menunjukkan arahan skema utilitas yang disediakan, pertimbangkan contoh berikut:

Dalam contoh ini, mari kita asumsikan bahwa kita ingin menggabungkan dua API sumber. Kami diberi dua skema yang membuat dan mengambil posting (misalnya, bagian komentar atau posting media sosial). Dengan asumsi bahwa tipe dan bidangnya sangat mirip, ada kemungkinan besar konflik selama operasi penggabungan. Cuplikan di bawah ini menunjukkan jenis dan bidang setiap skema.

File pertama, yang disebut Source1.graphQL, adalah skema GraphQL yang memungkinkan pengguna untuk membuat menggunakan mutasi. `Posts` `putPost` Masing-masing `Post` berisi judul dan ID. ID digunakan untuk referensi`User`, atau informasi poster (email dan alamat), dan`Message`, atau muatan (konten). `User`Jenis ini dianotasi dengan tag @canonical.

```
# This snippet represents a file called Source1.graphql

type Mutation {
    putPost(id: ID!, title: String!): Post
}

type Post {
```

```
    id: ID!
    title: String!
}

type Message {
   id: ID!
   content: String
}

type User @canonical {
   id: ID!
   email: String!
   address: String!
}

type Query {
    singlePost(id: ID!): Post
    getMessage(id: ID!): Message
}
```

File kedua, yang disebut Source2.graphQL, adalah skema GraphQL yang melakukan hal yang sangat mirip dengan Source1.graphQL. Namun, perhatikan bahwa bidang masing-masing jenis berbeda. Saat menggabungkan kedua skema ini, akan ada konflik penggabungan karena perbedaan ini.

Perhatikan juga bagaimana Source2.graphQL juga berisi beberapa arahan untuk mengurangi konflik ini. `Post`Jenis ini dianotasi dengan tag @hidden untuk mengaburkan dirinya sendiri selama operasi penggabungan. `Message`Tipe dianotasi dengan tag @renamed untuk memodifikasi nama tipe jika terjadi konflik penamaan dengan tipe lain`Message`. `ChatMessage`

```
# This snippet represents a file called Source2.graphql

type Post @hidden  {
    id: ID!
    title: String!
    internalSecret: String!
}

type Message @renamed(to: "ChatMessage") {
   id: ID!
   chatId: ID!
   from: User!
```

```
   to: User!
}

# Stub user so that we can link the canonical definition from Source1
type User {
   id: ID!
}

type Query {
    getPost(id: ID!): Post
    getMessage(id: ID!): Message @renamed(to: "getChatMessage")
}
```

Ketika penggabungan terjadi, hasilnya akan menghasilkan `MergedSchema.graphql` file:

```
# This snippet represents a file called MergedSchema.graphql

type Mutation {
    putPost(id: ID!, title: String!): Post
}

# Post from Source2 was hidden so only uses the Source1 definition.
type Post {
    id: ID!
    title: String!
}

# Renamed from Message to resolve the conflict
type ChatMessage {
   id: ID!
   chatId: ID!
   from: User!
   to: User!
}

type Message {
   id: ID!
   content: String
}

# Canonical definition from Source1
type User {
   id: ID!
```

```
   email: String!
   address: String!
}

type Query {
    singlePost(id: ID!): Post
    getMessage(id: ID!): Message

    # Renamed from getMessage
    getChatMessage(id: ID!): ChatMessage
}
```

Beberapa hal terjadi dalam penggabungan:

- `User`Jenis dari source1.graphQL diprioritaskan daripada *`User`* from Source2.graphQL karena anotasi @canonical.
- `Message`Jenis dari source1.graphQL disertakan dalam penggabungan. Namun, `Message` from Source2.graphQL memiliki konflik penamaan. Karena anotasi @renamed, itu juga termasuk dalam penggabungan tetapi dengan nama alternatif. `ChatMessage`
- `Post`Jenis dari source1.graphQL disertakan, tetapi *`Post`* tipe dari Source2.graphQL tidak. Biasanya, akan ada konflik pada tipe ini, tetapi karena `Post` tipe dari Source2.graphQL memiliki anotasi @hidden, datanya dikaburkan dan tidak termasuk dalam penggabungan. Hal ini mengakibatkan tidak ada konflik.
- `Query`Jenis diperbarui untuk menyertakan konten dari kedua file. Namun, satu `GetMessage` kueri diganti namanya `GetChatMessage` karena arahan. Ini menyelesaikan konflik penamaan antara dua kueri dengan nama yang sama.

Ada juga kasus tidak ada arahan yang ditambahkan ke tipe yang bertentangan. Di sini, tipe gabungan akan mencakup penyatuan semua bidang dari semua definisi sumber dari jenis itu. Misalnya, perhatikan contoh berikut:

Skema ini, yang disebut Source1.graphQL, memungkinkan untuk membuat dan mengambil. `Posts` Konfigurasinya mirip dengan contoh sebelumnya, tetapi dengan informasi yang lebih sedikit.

```
# This snippet represents a file called Source1.graphql

type Mutation {
    putPost(id: ID!, title: String!): Post
}
```

```
type Post  {
    id: ID!
    title: String!
}

type Query {
    getPost(id: ID!): Post
}
```

Skema ini, yang disebut Source2.graphQL, memungkinkan untuk membuat dan mengambil `Reviews` (misalnya, rating film atau ulasan restoran). `Reviews`dikaitkan dengan `Post` nilai ID yang sama. Bersama-sama, mereka berisi judul, ID posting, dan pesan payload dari posting ulasan lengkap.

Saat bergabung, akan ada konflik antara kedua `Post` jenis tersebut. Karena tidak ada anotasi untuk mengatasi masalah ini, perilaku defaultnya adalah melakukan operasi gabungan pada tipe yang bertentangan.

```
# This snippet represents a file called Source2.graphql

type Mutation {
    putReview(id: ID!, postId: ID!, comment: String!): Review
}

type Post  {
    id: ID!
    reviews: [Review]
}

type Review {
   id: ID!
   postId: ID!
   comment: String!
}

type Query {
    getReview(id: ID!): Review
}
```

Ketika penggabungan terjadi, hasilnya akan menghasilkan `MergedSchema.graphql` file:

```
# This snippet represents a file called MergedSchema.graphql
```

```
type Mutation {
    putReview(id: ID!, postId: ID!, comment: String!): Review
    putPost(id: ID!, title: String!): Post
}

type Post  {
    id: ID!
    title: String!
    reviews: [Review]
}

type Review {
   id: ID!
   postId: ID!
   comment: String!
}

type Query {
    getPost(id: ID!): Post
    getReview(id: ID!): Review
}
```

Beberapa hal terjadi dalam penggabungan:

- `Mutation`Tipe tidak menghadapi konflik dan digabung.
- Bidang `Post` tipe digabungkan melalui operasi serikat pekerja. Perhatikan bagaimana penyatuan antara keduanya menghasilkan satu`id`, satu`title`, dan satu`reviews`.
- `Review`Tipe tidak menghadapi konflik dan digabung.
- `Query`Tipe tidak menghadapi konflik dan digabung.

Mengelola resolver pada tipe bersama

Dalam contoh di atas, pertimbangkan kasus di mana Source1.graphQL telah mengkonfigurasi resolver unit, `Query.getPost` yang menggunakan sumber data DynamoDB bernama. `PostDatasource` Resolver ini akan mengembalikan `id` dan dari `title` tipe. `Post` Sekarang, pertimbangkan Source2.graphQL telah mengonfigurasi resolver pipeline, yang menjalankan dua fungsi. `Post.reviews` `Function1`memiliki sumber `None` data yang dilampirkan untuk melakukan pemeriksaan otorisasi khusus. `Function2`memiliki sumber data DynamoDB yang dilampirkan untuk menanyakan tabel. `reviews`

```
query GetPostQuery {
    getPost(id: "1") {
        id,
        title,
        reviews
    }
}
```

Ketika kueri di atas dijalankan oleh klien ke titik akhir Merged API, AWS AppSync layanan pertama-tama menjalankan unit resolver for `Query.getPost` from`Source1`, yang memanggil `PostDatasource` dan mengembalikan data dari DynamoDB. Kemudian, ia menjalankan resolver `Post.reviews` pipeline di mana `Function1` melakukan logika otorisasi khusus dan `Function2` mengembalikan ulasan yang diberikan yang ditemukan di`id`. `$context.source` Layanan memproses permintaan sebagai satu GraphQL run, dan permintaan sederhana ini hanya akan memerlukan satu token permintaan.

Mengelola konflik resolver pada tipe bersama

Pertimbangkan kasus berikut di mana kami juga menerapkan resolver untuk menyediakan beberapa bidang sekaligus di luar penyelesai bidang di. `Query.getPost` `Source2` source1.graphQL mungkin terlihat seperti ini:

```
# This snippet represents a file called Source1.graphql

type Post  {
    id: ID!
    title: String!
    date: AWSDateTime!
}

type Query {
    getPost(id: ID!): Post
}
```

Source2.graphQL mungkin terlihat seperti ini:

```
# This snippet represents a file called Source2.graphql

type Post  {
  id: ID!
```

```
  content: String!
  contentHash: String!
  author: String!
}

type Query {
    getPost(id: ID!): Post
}
```

Mencoba menggabungkan kedua skema ini akan menghasilkan kesalahan AWS AppSync penggabungan karena API Gabungan tidak mengizinkan beberapa resolver sumber dilampirkan ke bidang yang sama. Untuk mengatasi konflik ini, Anda dapat menerapkan pola penyelesai bidang yang memerlukan Source2.graphQL untuk menambahkan tipe terpisah yang akan menentukan bidang yang dimilikinya dari tipe tersebut. `Post` Dalam contoh berikut, kami menambahkan tipe yang disebut`PostInfo`, yang berisi bidang konten dan penulis yang akan diselesaikan oleh Source2.graphQL. Source1.graphQL akan mengimplementasikan resolver yang dilampirkan`Query.getPost`, sementara Source2.graphQL sekarang akan melampirkan resolver untuk memastikan bahwa semua data dapat berhasil diambil: `Post.postInfo`

```
type Post  {
  id: ID!
  postInfo: PostInfo
}

type PostInfo {
   content: String!
   contentHash: String!
   author: String!
}

type Query {
    getPost(id: ID!): Post
}
```

Sementara menyelesaikan konflik semacam itu membutuhkan skema API sumber untuk ditulis ulang dan, berpotensi, klien untuk mengubah kueri mereka, keuntungan dari pendekatan ini adalah bahwa kepemilikan resolver gabungan tetap jelas di seluruh tim sumber.

Mengkonfigurasi skema

Dua pihak bertanggung jawab untuk mengonfigurasi skema untuk membuat API Gabungan:

- Pemilik API yang digabungkan - Pemilik API yang digabungkan harus mengonfigurasi logika otorisasi Gabungan API dan pengaturan lanjutan seperti pencatatan, penelusuran, caching, dan dukungan WAF.
- Pemilik API sumber terkait - Pemilik API terkait harus mengonfigurasi skema, resolver, dan sumber data yang membentuk API Gabungan.

Karena skema API Gabungan Anda dibuat dari skema API sumber terkait, skema tersebut hanya dibaca. Ini berarti perubahan skema harus dimulai di API sumber Anda. Di AWS AppSync konsol, Anda dapat beralih antara skema Gabungan dan skema individual dari API sumber yang disertakan dalam API Gabungan menggunakan daftar drop-down di atas jendela Skema.

Mengkonfigurasi mode otorisasi

Beberapa mode otorisasi tersedia untuk melindungi API Gabungan Anda. Untuk mempelajari lebih lanjut tentang mode otorisasiAWS AppSync, lihat Otorisasi dan otentikasi.

Mode otorisasi berikut tersedia untuk digunakan dengan API Gabungan:

- Kunci API: Strategi otorisasi paling sederhana. Semua permintaan harus menyertakan kunci API di bawah header `x-api-key` permintaan. Kunci API yang kedaluwarsa disimpan selama 60 hari setelah tanggal kedaluwarsa.
- AWSIdentity and Access Management (IAM): Strategi otorisasi AWS IAM mengotorisasi semua permintaan yang ditandatangani sigv4.
- Kumpulan Pengguna Amazon Cognito: Otorisasi pengguna Anda melalui Kumpulan Pengguna Amazon Cognito untuk mencapai kontrol yang lebih halus.
- AWSLambda Authorizers: Fungsi tanpa server yang memungkinkan Anda untuk mengautentikasi dan mengotorisasi akses ke API Anda menggunakan logika kustom. AWS AppSync
- OpenID Connect: Jenis otorisasi ini memberlakukan token OpenID connect (OIDC) yang disediakan oleh layanan yang sesuai dengan OIDC. Aplikasi Anda dapat memanfaatkan pengguna dan hak istimewa yang ditentukan oleh penyedia OIDC Anda untuk mengontrol akses.

Mode otorisasi API Gabungan dikonfigurasi oleh pemilik API Gabungan. Pada saat operasi gabungan, API Gabungan harus menyertakan mode otorisasi utama yang dikonfigurasi pada API sumber baik sebagai mode otorisasi utamanya sendiri atau sebagai mode otorisasi sekunder. Jika tidak, itu akan tidak kompatibel, dan operasi penggabungan akan gagal dengan konflik. Saat menggunakan arahan multi-auth di API sumber, proses penggabungan dapat secara otomatis

menggabungkan arahan ini ke titik akhir terpadu. Jika mode otorisasi utama dari API sumber tidak cocok dengan mode otorisasi utama API Gabungan, mode ini akan secara otomatis menambahkan arahan autentikasi ini untuk memastikan bahwa mode otorisasi untuk tipe di API sumber konsisten.

Mengkonfigurasi peran eksekusi

Saat membuat API Gabungan, Anda perlu menentukan peran layanan. Peran AWS layanan adalah peran AWS Identity and Access Management (IAM) and Access Management (IAM) yang digunakan oleh AWS layanan untuk melakukan tugas atas nama Anda.

Dalam konteks ini, API Gabungan Anda perlu menjalankan resolver yang mengakses data dari sumber data yang dikonfigurasi di API sumber Anda. Peran layanan yang diperlukan untuk ini adalah`mergedApiExecutionRole`, dan harus memiliki akses eksplisit untuk menjalankan permintaan pada API sumber yang disertakan dalam API gabungan Anda melalui izin `appsync:SourceGraphQL` IAM. Selama menjalankan permintaan GraphQL, AWS AppSync layanan akan mengambil peran layanan ini dan mengotorisasi peran untuk melakukan tindakan. `appsync:SourceGraphQL`

AWS AppSyncmendukung mengizinkan atau menolak izin ini pada bidang tingkat atas tertentu dalam permintaan seperti cara kerja mode otorisasi IAM untuk API IAM. Untuk non-top-level bidang, AWS AppSync mengharuskan Anda untuk menentukan izin pada sumber API ARN itu sendiri. Untuk membatasi akses ke non-top-level bidang tertentu di API Gabungan, sebaiknya terapkan logika kustom dalam Lambda Anda atau menyembunyikan bidang API sumber dari API Gabungan menggunakan direktif @hidden. Jika Anda ingin mengizinkan peran menjalankan semua operasi data dalam API sumber, Anda dapat menambahkan kebijakan di bawah ini. Perhatikan bahwa entri sumber daya pertama memungkinkan akses ke semua bidang tingkat atas dan entri kedua mencakup resolver turunan yang mengotorisasi sumber daya API sumber itu sendiri:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [ "appsync:SourceGraphQL"],
        "Resource": [
            "arn:aws:appsync:us-west-2:123456789012:apis/YourSourceGraphQLApiId/*",
            "arn:aws:appsync:us-west-2:123456789012:apis/YourSourceGraphQLApiId"]
    }]
}
```

Jika Anda ingin membatasi akses hanya ke bidang tingkat atas tertentu, Anda dapat menggunakan kebijakan seperti ini:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [ "appsync:SourceGraphQL"],
        "Resource": [
            "arn:aws:appsync:us-west-2:123456789012:apis/YourSourceGraphQLApiId/types/
Query/fields/<Field-1>",
            "arn:aws:appsync:us-west-2:123456789012:apis/YourSourceGraphQLApiId"]
    }]
}
```

Anda juga dapat menggunakan wizard pembuatan API AWS AppSync konsol untuk menghasilkan peran layanan agar API Gabungan dapat mengakses sumber daya yang dikonfigurasi dalam API sumber yang berada di akun yang sama dengan API gabungan Anda. Jika API sumber Anda tidak berada di akun yang sama dengan API gabungan, Anda harus terlebih dahulu membagikan sumber daya menggunakan AWS Resource Access Manager (AWS RAM).

Mengonfigurasi API Gabungan lintas akun menggunakan AWS RAM

Saat membuat API Gabungan, Anda dapat mengaitkan API sumber secara opsional dari akun lain yang telah dibagikan melalui AWS Resource Access Manager ()AWS RAM. AWS RAMmembantu Anda berbagi sumber daya dengan aman di seluruh AWS akun, dalam organisasi atau unit organisasi (OU) Anda, dan dengan peran dan pengguna IAM.

AWS AppSyncterintegrasi dengan AWS RAM untuk mendukung konfigurasi dan mengakses API sumber di beberapa akun dari satu API Gabungan. AWS RAMmemungkinkan Anda membuat pembagian sumber daya, atau wadah sumber daya dan set izin yang akan dibagikan untuk masing-masing sumber daya. Anda dapat menambahkan AWS AppSync API ke pembagian sumber daya diAWS RAM. Dalam pembagian sumber daya, AWS AppSync berikan tiga set izin berbeda yang dapat dikaitkan dengan AWS AppSync API di RAM:

1. `AWSRAMPermissionAppSyncSourceApiOperationAccess`: Set izin default yang ditambahkan saat berbagi AWS AppSync API AWS RAM jika tidak ada izin lain yang ditentukan. Set izin ini digunakan untuk berbagi AWS AppSync API sumber dengan pemilik API Gabungan. Set izin ini mencakup izin untuk `appsync:AssociateMergedGraphqlApi` pada API sumber serta

`appsync:SourceGraphQL` izin yang diperlukan untuk mengakses sumber daya API sumber saat runtime.

2. `AWSRAMPermissionAppSyncMergedApiOperationAccess`: Set izin ini harus dikonfigurasi saat berbagi API Gabungan dengan pemilik API sumber. Set izin ini akan memberi API sumber kemampuan untuk mengonfigurasi API Gabungan termasuk kemampuan untuk mengaitkan API sumber apa pun yang dimiliki oleh prinsipal target ke API Gabungan dan untuk membaca dan memperbarui asosiasi API sumber dari API Gabungan.
3. `AWSRAMPermissionAppSyncAllowSourceGraphQLAccess`: Set izin ini memungkinkan `appsync:SourceGraphQL` izin untuk digunakan dengan AWS AppSync API. Ini dimaksudkan untuk digunakan untuk berbagi API sumber dengan pemilik API Gabungan. Berbeda dengan izin default yang ditetapkan untuk akses operasi API sumber, set izin ini hanya menyertakan izin `appsync:SourceGraphQL` runtime. Jika pengguna memilih untuk membagikan akses operasi Gabungan API ke pemilik API sumber, mereka juga perlu membagikan izin ini dari API sumber kepada pemilik API Gabungan agar memiliki akses runtime melalui titik akhir API Gabungan.

AWS AppSyncjuga mendukung izin yang dikelola pelanggan. Jika salah satu izin AWS -managed yang disediakan tidak berfungsi, Anda dapat membuat izin yang dikelola pelanggan sendiri. Izin yang dikelola pelanggan adalah izin terkelola yang Anda buat dan pertahankan dengan menentukan secara tepat tindakan mana yang dapat dilakukan dalam kondisi mana dengan sumber daya yang digunakan bersama. AWS RAM AWS AppSyncmemungkinkan Anda memilih dari tindakan berikut saat membuat izin Anda sendiri:

1. `appsync:AssociateSourceGraphqlApi`
2. `appsync:AssociateMergedGraphqlApi`
3. `appsync:GetSourceApiAssociation`
4. `appsync:UpdateSourceApiAssociation`
5. `appsync:StartSchemaMerge`
6. `appsync:ListTypesByAssociation`
7. `appsync:SourceGraphQL`

Setelah Anda membagikan API sumber atau API Gabungan dengan benar AWS RAM dan, jika perlu, undangan berbagi sumber daya telah diterima, undangan tersebut akan terlihat di AWS AppSync konsol saat Anda membuat atau memperbarui asosiasi API sumber pada API Gabungan Anda. Anda juga dapat mencantumkan semua AWS AppSync API yang telah dibagikan menggunakan AWS RAM

akun Anda terlepas dari izin yang ditetapkan dengan memanggil `ListGraphqlApis` operasi yang disediakan oleh AWS AppSync dan menggunakan filter `OTHER_ACCOUNTS` pemilik.

> **Note**
>
> Berbagi melalui AWS RAM memerlukan pemanggil AWS RAM untuk memiliki izin untuk melakukan `appsync:PutResourcePolicy` tindakan pada API apa pun yang sedang dibagikan.

Penggabungan

Mengelola penggabungan

API gabungan dimaksudkan untuk mendukung kolaborasi tim pada titik akhir terpaduAWS AppSync. Tim dapat secara independen mengembangkan API GraphQL sumber terisolasi mereka sendiri di backend sementara AWS AppSync layanan mengelola integrasi sumber daya ke dalam titik akhir API Gabungan tunggal untuk mengurangi gesekan dalam kolaborasi dan mengurangi waktu tunggu pengembangan.

Penggabungan otomatis

API sumber yang terkait dengan API AWS AppSync Gabungan Anda dapat dikonfigurasi untuk menggabungkan (auto-merge) secara otomatis ke dalam API Gabungan setelah perubahan apa pun dilakukan pada API sumber. Ini memastikan bahwa perubahan dari API sumber selalu disebarkan ke titik akhir API Gabungan di latar belakang. Setiap perubahan dalam skema API sumber akan diperbarui di API Gabungan selama tidak menimbulkan konflik gabungan dengan definisi yang ada di API Gabungan. Jika pembaruan di API sumber memperbarui resolver, sumber data, atau fungsi, sumber daya yang diimpor juga akan diperbarui.Ketika konflik baru diperkenalkan yang tidak dapat diselesaikan secara otomatis (diselesaikan secara otomatis), pembaruan skema API Gabungan ditolak karena konflik yang tidak didukung selama operasi penggabungan. Pesan kesalahan tersedia di konsol untuk setiap asosiasi API sumber yang memiliki status`MERGE_FAILED`. Anda juga dapat memeriksa pesan kesalahan dengan memanggil `GetSourceApiAssociation` operasi untuk asosiasi API sumber tertentu menggunakan AWS SDK atau menggunakan AWS CLI seperti ini:

```
aws appsync get-source-api-association --merged-api-identifier <Merged API ARN> --
association-id <SourceApiAssociation id>
```

Ini akan menghasilkan hasil dalam format berikut:

```
{
    "sourceApiAssociation": {
        "associationId": "<association id>",
        "associationArn": "<association arn>",
        "sourceApiId": "<source api id>",
        "sourceApiArn": "<source api arn>",
        "mergedApiArn": "<merged api arn>",
        "mergedApiId": "<merged api id>",
        "sourceApiAssociationConfig": {
            "mergeType": "MANUAL_MERGE"
        },
        "sourceApiAssociationStatus": "MERGE_FAILED",
        "sourceApiAssociationStatusDetail": "Unable to resolve conflict on object with
 name title: Merging is not supported for fields with different types."
    }
}
```

Penggabungan manual

Pengaturan default untuk API sumber adalah penggabungan manual. Untuk menggabungkan perubahan apa pun yang terjadi di API sumber sejak API Gabungan terakhir diperbarui, pemilik API sumber dapat memanggil penggabungan manual dari AWS AppSync konsol atau melalui `StartSchemaMerge` operasi yang tersedia di SDK AWS dan CLI. AWS

Dukungan tambahan untuk API Gabungan

Mengkonfigurasi langganan

Tidak seperti pendekatan berbasis router untuk komposisi skema GraphQL, API Gabungan menyediakan dukungan bawaan untuk langganan AWS AppSync GraphQL. Semua operasi langganan yang ditentukan dalam API sumber terkait Anda akan secara otomatis bergabung dan berfungsi di API Gabungan Anda tanpa modifikasi. Untuk mempelajari selengkapnya tentang cara AWS AppSync mendukung langganan melalui WebSockets koneksi tanpa server, lihat Data waktu nyata.

Mengkonfigurasi observabilitas

AWS AppSyncAPI gabungan menyediakan pencatatan, pemantauan, dan metrik bawaan melalui Amazon. CloudWatch AWS AppSyncjuga menyediakan dukungan bawaan untuk melacak via AWS X-Ray.

Mengkonfigurasi domain kustom

AWS AppSyncAPI gabungan menyediakan dukungan bawaan untuk menggunakan domain khusus dengan GraphQL dan titik akhir Real-time API Gabungan Anda.

Mengkonfigurasi caching

AWS AppSyncAPI yang digabungkan memberikan dukungan bawaan untuk melakukan cache respons tingkat permintaan dan/atau tingkat resolver secara opsional serta kompresi respons. Untuk mempelajari lebih lanjut, lihat Caching dan kompresi.

Mengkonfigurasi API pribadi

AWS AppSyncAPI gabungan menyediakan dukungan bawaan untuk API Pribadi yang membatasi akses ke GraphQL API Gabungan dan titik akhir Real-time untuk lalu lintas yang berasal dari titik akhir VPC yang dapat Anda konfigurasi.

Mengkonfigurasi aturan firewall

AWS AppSyncAPI gabungan menyediakan dukungan bawaan untukAWS WAF, yang memungkinkan Anda untuk melindungi API Anda dengan mendefinisikan aturan firewall aplikasi web.

Mengkonfigurasi log audit

AWS AppSyncAPI gabungan menyediakan dukungan bawaan untuk AWS CloudTrail, yang memungkinkan Anda mengonfigurasi dan mengelola log audit.

Batasan API yang digabungkan

Saat mengembangkan API Gabungan, perhatikan aturan berikut:

1. API Gabungan tidak dapat menjadi API sumber untuk API Gabungan lainnya.
2. API sumber tidak dapat dikaitkan dengan lebih dari satu API Gabungan.
3. Batas ukuran default untuk dokumen skema API Gabungan adalah 10 MB.
4. Jumlah default API sumber yang dapat dikaitkan dengan API Gabungan adalah 10. Namun, Anda dapat meminta peningkatan batas jika membutuhkan lebih dari 10 API sumber di API Gabungan.

Membuat API Gabungan

Untuk membuat API Gabungan di konsol

1. Masuk ke AWS Management Console dan buka AWS AppSynckonsol.

   - Di Dashboard, pilih Create API.

2. Pilih API Gabungan, lalu pilih Berikutnya.
3. Di halaman Tentukan detail API, masukkan informasi berikut:

   a. Di bawah Detail API, masukkan informasi berikut:

      i. Tentukan nama API gabungan Anda. Bidang ini adalah cara untuk memberi label GraphQL API Anda agar mudah membedakannya dari API GraphQL lainnya.

      ii. Tentukan detail Kontak. Bidang ini bersifat opsional dan melampirkan nama atau grup ke GraphQL API. Ini tidak ditautkan atau dihasilkan oleh sumber daya lain dan berfungsi seperti bidang nama API.

   b. Di bawah peran Layanan, Anda harus melampirkan peran eksekusi IAM ke API gabungan Anda sehingga AWS AppSync dapat mengimpor dan menggunakan sumber daya Anda dengan aman saat runtime. Anda dapat memilih untuk membuat dan menggunakan peran layanan baru, yang akan memungkinkan Anda untuk menentukan kebijakan dan sumber daya yang AWS AppSync akan digunakan. Anda juga dapat mengimpor peran IAM yang ada dengan memilih Gunakan peran layanan yang ada, lalu memilih peran dari daftar drop-down.

   c. Di bawah konfigurasi API Pribadi, Anda dapat memilih untuk mengaktifkan fitur API pribadi. Perhatikan bahwa pilihan ini tidak dapat diubah setelah membuat API gabungan. Untuk informasi selengkapnya tentang API pribadi, lihat Menggunakan API AWS AppSync Pribadi.

   Pilih Berikutnya setelah Anda selesai.

4. Selanjutnya, Anda harus menambahkan GraphQL API yang akan digunakan sebagai dasar untuk API gabungan Anda. Di halaman Pilih sumber API, masukkan informasi berikut:

   a. Di API dari tabel AWS akun Anda, pilih Tambahkan API Sumber. Dalam daftar GraphQL API, setiap entri akan berisi data berikut:

      i. Nama: Bidang nama API GraphQL API.

      ii. API ID: Nilai ID unik GraphQL API.

      iii. Mode autentikasi primer: Mode otorisasi default untuk GraphQL API. Untuk informasi selengkapnya tentang mode otorisasiAWS AppSync, lihat Otorisasi dan otentikasi.

      iv. Mode autentikasi tambahan: Mode otorisasi sekunder yang dikonfigurasi di GraphQL API.

v. Pilih API yang akan Anda gunakan di API gabungan dengan memilih kotak centang di sebelah bidang Nama API. Setelah itu, pilih Add Source API. GraphQL API yang dipilih akan muncul di API dari AWS tabel akun Anda.

b. Di tabel API dari AWS akun lain, pilih Tambahkan API Sumber. API GraphQL dalam daftar ini berasal dari akun lain yang membagikan sumber dayanya ke akun Anda AWS Resource Access Manager melalui (). AWS RAM Proses untuk memilih GraphQL API dalam tabel ini sama dengan proses di bagian sebelumnya. Untuk informasi selengkapnya tentang berbagi sumber dayaAWS RAM, lihat Apa ituAWS Resource Access Manager? .

   Pilih Berikutnya setelah Anda selesai.

c. Tambahkan mode autentikasi utama Anda. Lihat Otorisasi dan otentikasi untuk informasi selengkapnya. Pilih Berikutnya.

d. Tinjau input Anda, lalu pilih Buat API.

Introspeksi RDS

AWS AppSyncmembuat membangun API dari database relasional yang ada menjadi mudah. Utilitas introspeksi dapat menemukan model dari tabel database dan mengusulkan tipe GraphQL. Wizard Create API AWS AppSync konsol dapat langsung menghasilkan API dari database Aurora MySQL atau PostgreSQL. Secara otomatis membuat jenis dan JavaScript resolver untuk membaca dan menulis data.

AWS AppSyncmenyediakan integrasi langsung dengan database Amazon Aurora melalui Amazon RDS Data API. Alih-alih memerlukan koneksi database persisten, Amazon RDS Data API menawarkan titik akhir HTTP aman yang AWS AppSync terhubung ke untuk menjalankan SQL pernyataan. Anda dapat menggunakan ini untuk membuat API database relasional untuk beban kerja MySQL dan PostgreSQL Anda di Aurora.

Membangun API untuk database relasional Anda dengan AWS AppSync memiliki beberapa keuntungan:

- Database Anda tidak langsung terpapar ke klien, memisahkan titik akses dari database itu sendiri.
- Anda dapat membangun API yang dibuat khusus yang disesuaikan dengan kebutuhan aplikasi yang berbeda, menghilangkan kebutuhan akan logika bisnis khusus di frontend. Ini sejalan dengan pola Backend-For-Frontend (BFF).

- Otorisasi dan kontrol akses dapat diimplementasikan pada AWS AppSync lapisan menggunakan berbagai mode otorisasi untuk mengontrol akses. Tidak ada sumber daya komputasi tambahan yang diperlukan untuk terhubung ke database, seperti hosting server web atau koneksi proxy.
- Kemampuan real-time dapat ditambahkan melalui langganan, dengan mutasi data yang dilakukan AppSync secara otomatis didorong ke klien yang terhubung.
- Klien dapat terhubung ke API melalui HTTPS menggunakan port umum seperti 443.

AWS AppSyncmembuat membangun API dari database relasional yang ada menjadi mudah. Utilitas introspeksi dapat menemukan model dari tabel database dan mengusulkan tipe GraphQL. Wizard Create API AWS AppSync konsol dapat langsung menghasilkan API dari database Aurora MySQL atau PostgreSQL. Secara otomatis membuat jenis dan JavaScript resolver untuk membaca dan menulis data.

AWS AppSyncmenyediakan JavaScript utilitas terintegrasi untuk menyederhanakan penulisan pernyataan SQL dalam resolver. Anda dapat menggunakan AWS AppSync templat `sql` tag untuk pernyataan statis dengan nilai dinamis, atau utilitas `rds` modul untuk membangun pernyataan secara terprogram. Lihat referensi fungsi resolver untuk sumber data RDS dan modul bawaan untuk informasi selengkapnya.

Menggunakan fitur introspeksi (konsol)

Untuk tutorial terperinci dan panduan memulai, lihat Tutorial: Aurora PostgreSQL Tanpa Server dengan API Data.

AWS AppSyncKonsol memungkinkan Anda membuat AWS AppSync GraphQL API dari database Aurora yang ada yang dikonfigurasi dengan Data API hanya dalam beberapa menit. Ini dengan cepat menghasilkan skema operasional berdasarkan konfigurasi database Anda. Anda dapat menggunakan API apa adanya atau membangunnya untuk menambahkan fitur.

1. Masuk ke AWS Management Console dan buka AppSynckonsol.
   - Di Dasbor, pilih Buat API.
2. Di bawah opsi API, pilih GraphQL API, Mulai dengan klaster Amazon Aurora, lalu Berikutnya.
   a. Masukkan nama API. Ini akan digunakan sebagai pengidentifikasi untuk API di konsol.
   b. Untuk detail kontak, Anda dapat memasukkan titik kontak untuk mengidentifikasi manajer API. Ini adalah bidang opsional.

c. Di bawah konfigurasi API Pribadi, Anda dapat mengaktifkan fitur API pribadi. API pribadi hanya dapat diakses dari titik akhir VPC (VPCE) yang dikonfigurasi. Untuk informasi selengkapnya, lihat API Pribadi.

   Kami tidak menyarankan mengaktifkan fitur ini untuk contoh ini. Pilih Berikutnya setelah meninjau input Anda.

3. Di halaman Database, pilih Pilih database.

   a. Anda harus memilih database Anda dari cluster Anda. Langkah pertama adalah memilih Wilayah di mana cluster Anda berada.

   b. Pilih cluster Aurora dari daftar drop-down. Perhatikan bahwa Anda harus telah membuat dan mengaktifkan API data yang sesuai sebelum menggunakan sumber daya.

   c. Selanjutnya, Anda harus menambahkan kredensil untuk database Anda ke layanan. Ini terutama dilakukan dengan menggunakanAWS Secrets Manager. Pilih Wilayah di mana rahasia Anda ada. Untuk informasi selengkapnya tentang cara mengambil informasi rahasia, lihat Menemukan rahasia atau Mengambil rahasia.

   d. Tambahkan rahasia Anda dari daftar drop-down. Perhatikan bahwa pengguna harus memiliki izin membaca untuk database Anda.

4. Pilih Import (Impor).

   AWS AppSyncakan mulai introspeksi database Anda, menemukan tabel, kolom, kunci utama, dan indeks. Ini memeriksa apakah tabel yang ditemukan dapat didukung dalam GraphQL API. Perhatikan bahwa untuk mendukung pembuatan baris baru, tabel memerlukan kunci utama, yang dapat menggunakan beberapa kolom. AWS AppSyncmemetakan kolom tabel untuk mengetik bidang sebagai berikut:

Tipe data	Jenis bidang
VARCHAR	String
CHAR	String
BINARY	String
VARBINARY	String
TINYBLOB	String

TINYTEXT	String
TEXT	String
BLOB	String
MEDIUMTEXT	String
MEDIUMBLOB	String
LONGTEXT	String
LONGBLOB	String
BOOL	Boolean
BOOLEAN	Boolean
BIT	Int
TINYINT	Int
SMALLINT	Int
MEDIUMINT	Int
INT	Int
INTEGER	Int
BIGINT	Int
YEAR	Int
FLOAT	Float
DOUBLE	Float
DECIMAL	Float
DEC	Float
NUMERIC	Float

DATE	AWSDate
TIMESTAMP	String
DATETIME	String
TIME	AWSTime
JSON	AWSJson
ENUM	ENUM

5. Setelah penemuan tabel selesai, bagian Database akan diisi dengan informasi Anda. Di bagian tabel Database baru, data dari tabel mungkin sudah diisi dan dikonversi menjadi tipe untuk skema Anda. Jika Anda tidak melihat beberapa data yang diperlukan, Anda dapat memeriksanya dengan memilih Tambahkan tabel, mengklik kotak centang untuk jenis tersebut di modal yang muncul, lalu memilih Tambah.

   Untuk menghapus tipe dari bagian tabel Database, klik kotak centang di sebelah jenis yang ingin Anda hapus, lalu pilih Hapus. Jenis yang dihapus akan ditempatkan di modal Tambahkan tabel jika Anda ingin menambahkannya lagi nanti.

   Perhatikan bahwa AWS AppSync menggunakan nama tabel sebagai nama tipe, tetapi Anda dapat mengganti namanya - misalnya, mengubah nama tabel jamak seperti film menjadi nama tipe Movie. Untuk mengganti nama tipe di bagian tabel Database, klik kotak centang dari jenis yang ingin Anda ganti nama, lalu klik ikon pensil di kolom Type name.

   Untuk melihat pratinjau konten skema berdasarkan pilihan Anda, pilih skema pratinjau. Perhatikan bahwa skema ini tidak dapat kosong, jadi Anda harus memiliki setidaknya satu tabel yang dikonversi menjadi tipe. Juga, skema ini tidak boleh melebihi 1 MB dalam ukuran.

   - Di bawah Peran layanan, pilih apakah akan membuat peran layanan baru khusus untuk impor ini atau menggunakan peran yang ada.

6. Pilih Berikutnya.
7. Selanjutnya, pilih apakah akan membuat API hanya-baca (hanya kueri) atau API untuk membaca dan menulis data (dengan kueri dan mutasi). Yang terakhir ini juga mendukung langganan real-time yang dipicu oleh mutasi.
8. Pilih Berikutnya.

9. Tinjau pilihan Anda dan kemudian pilih Buat API. AWS AppSyncakan membuat API dan melampirkan resolver ke kueri dan mutasi. API yang dihasilkan beroperasi penuh dan dapat diperpanjang sesuai kebutuhan.

Menggunakan fitur introspeksi (API)

Anda dapat menggunakan API `StartDataSourceIntrospection` introspeksi untuk menemukan model dalam database Anda secara terprogram. Untuk detail selengkapnya tentang perintah, lihat menggunakan `StartDataSourceIntrospection`API.

Untuk menggunakan`StartDataSourceIntrospection`, berikan Aurora cluster Amazon Resource Name (ARN), nama database, dan ARN rahasia Anda. AWS Secrets Manager Perintah memulai proses introspeksi. Anda dapat mengambil hasilnya dengan `GetDataSourceIntrospection` perintah. Anda dapat menentukan apakah perintah harus mengembalikan string Storage Definition Language (SDL) untuk model yang ditemukan. Ini berguna untuk menghasilkan definisi skema SDL langsung dari model yang ditemukan.

Misalnya, jika Anda memiliki pernyataan bahasa definisi Data (DDL) berikut untuk `Todos` tabel sederhana:

```
create table if not exists public.todos
(
id serial constraint todos_pk primary key,
description text,
due timestamp,
"createdAt" timestamp default now()
);
```

Anda memulai introspeksi dengan yang berikut ini.

```
aws appsync start-data-source-introspection \
  --rds-data-api-config resourceArn=<cluster-arn>,secretArn=<secret-
arn>,databaseName=database
```

Selanjutnya, gunakan `GetDataSourceIntrospection` perintah untuk mengambil hasilnya.

```
aws appsync get-data-source-introspection \
  --introspection-id a1234567-8910-abcd-efgh-identifier \
  --include-models-sdl
```

Ini mengembalikan hasil sebagai berikut.


```
{
    "introspectionId": "a1234567-8910-abcd-efgh-identifier",
    "introspectionStatus": "SUCCESS",
    "introspectionStatusDetail": null,
    "introspectionResult": {
        "models": [
            {
                "name": "todos",
                "fields": [
                    {
                        "name": "description",
                        "type": {
                            "kind": "Scalar",
                            "name": "String",
                            "type": null,
                            "values": null
                        },
                        "length": 0
                    },
                    {
                        "name": "due",
                        "type": {
                            "kind": "Scalar",
                            "name": "AWSDateTime",
                            "type": null,
                            "values": null
                        },
                        "length": 0
                    },
                    {
                        "name": "id",
                        "type": {
                            "kind": "NonNull",
                            "name": null,
                            "type": {
                                "kind": "Scalar",
                                "name": "Int",
                                "type": null,
                                "values": null
                            },
                            "values": null
                        },
```

```
                        "length": 0
                    },
                    {
                        "name": "createdAt",
                        "type": {
                            "kind": "Scalar",
                            "name": "AWSDateTime",
                            "type": null,
                            "values": null
                        },
                        "length": 0
                    }
                ],
                "primaryKey": {
                    "name": "PRIMARY_KEY",
                    "fields": [
                        "id"
                    ]
                },
                "indexes": [],
                "sdl": "type todos\n{\ndescription: String\n\ndue: AWSDateTime\n\nid:
 Int!\n\ncreatedAt: AW
SDateTime\n}\n"
            }
        ],
        "nextToken": null
    }
}
```

Membangun aplikasi klien

Anda dapat terhubung ke AWS AppSync GraphQL API Anda menggunakan klien GraphQL apa pun, tetapi kami sangat merekomendasikan klien Amplify. Amplify tidak hanya membuat SDK klien yang diketik secara otomatis untuk GraphQL API Anda, tetapi juga menawarkan dukungan untuk data waktu nyata dan kemampuan kueri GraphQL yang ditingkatkan dalam aplikasi klien. Untuk aplikasi web, Amplify dapat menghasilkan klien. JavaScript Bagi mereka yang menargetkan lingkungan lintas platform atau seluler, Amplify melayani Android, iOS, dan React Native. Untuk mempelajari lebih dalam pembuatan kode klien untuk platform ini, lihat dokumentasi Amplify. Berikut panduan untuk memulai perjalanan Anda dengan aplikasi JavaScript React:

> **Note**
>
> Anda perlu menginstal dan mengonfigurasi npm dan Amazon CLI sebelum memulai. Jika Anda menggunakan klien Amplify v6, ikuti panduan ini.

Untuk memulai:

1. Di mesin lokal Anda, navigasikan ke direktori proyek Anda. Instal perpustakaan Amplify menggunakan perintah di bawah ini:

   ```
   npm install aws-amplify
   ```

2. Unduh file konfigurasi Anda dan letakkan di folder proyek Anda. File konfigurasi Anda biasanya akan berisi `config` variabel dengan beberapa pengaturan (titik akhir, Wilayah, mode otorisasi, dll.) yang ditentukan. Misalnya, mungkin terlihat seperti ini:

   ```
   const config =  {
       API: {
           GraphQL: {
             endpoint: 'https://abcdefghijklmnopqrstuvwxyz.appsync-api.us-
   west-2.amazonaws.com/graphql',
             region: 'us-west-2',
             defaultAuthMode: 'apiKey',
             apiKey: ''
           }
       }
   };
   ```

```
export default config;
```

3. Dalam kode Anda, impor Amplify Library dan konfigurasi Anda untuk menyiapkan Amplify:

```
import { Amplify } from 'aws-amplify';
import config from './aws-exports.js';

Amplify.configure(config);
```

   Atau, gunakan cuplikan dalam konfigurasi API Anda untuk menyiapkan Amplify secara langsung:

```
import { Amplify } from 'aws-amplify';

Amplify.configure({
  API: {
    GraphQL: {
      endpoint: 'https://abcdefghijklmnopqrstuvwxyz.appsync-api.us-
west-2.amazonaws.com/graphql',
      region: 'us-west-2',
      defaultAuthMode: 'apiKey',
      apiKey: ''
    }
  }
});
```

4. Dengan menggunakan rantai alat Amplify, Anda memiliki opsi untuk membuat operasi otomatis berdasarkan skema Anda, yang menghemat upaya skrip manual. Di direktori root aplikasi Anda, gunakan perintah CLI berikut:

```
npx @aws-amplify/cli codegen add --apiId <id goes here> --region <region goes here>
```

   Ini akan mengunduh skema API Anda dan, secara default, menghasilkan kode pembantu klien ke dalam folder`src/graphql`. Setelah setiap penerapan API, Anda dapat menjalankan kembali perintah berikut untuk menghasilkan pernyataan dan tipe GraphQL yang diperbarui:

```
npx @aws-amplify/cli codegen
```

5. Anda sekarang dapat menghasilkan model untuk Android, Swift, Flutter, dan. JavaScript DataStore Gunakan perintah berikut untuk mengunduh skema Anda:

```
aws appsync get-introspection-schema --api-id <id goes here> --region <region goes
 here> --format SDL schema.graphql
```

Kemudian, jalankan perintah berikut dari direktori root aplikasi Anda:

```
npx @aws-amplify/cli codegen models \
--model-schema schema.graphql \
--target [android|ios|flutter|javascript|typescript] \
--output-dir ./
```

Tutorial penyelesai () JavaScript

Sumber data dan resolver adalah cara AWS AppSync menerjemahkan permintaan GraphQL dan mengambil informasi dari sumber daya Anda. AWS AWS AppSync memiliki dukungan untuk penyediaan otomatis dan koneksi dengan tipe sumber data tertentu. AWS AppSync mendukungAWS Lambda, Amazon DynamoDB, database relasional (Amazon Aurora Tanpa Server), Layanan OpenSearch Amazon, dan titik akhir HTTP sebagai sumber data. Anda dapat menggunakan GraphQL API dengan sumber daya yang AWS ada atau membangun sumber data dan resolver. Bagian ini membawa Anda melalui proses ini dalam serangkaian tutorial untuk lebih memahami cara kerja detail dan opsi penyetelan.

Topik

Tutorial: penyelesai DynamoDB JavaScript

Dalam tutorial ini, Anda akan mengimpor tabel Amazon DynamoDB Anda dan menghubungkannya AWS AppSync untuk membangun GraphQL API yang berfungsi penuh JavaScript menggunakan resolver pipeline yang dapat Anda manfaatkan dalam aplikasi Anda sendiri.

Anda akan menggunakan AWS AppSync konsol untuk menyediakan sumber daya Amazon DynamoDB, membuat resolver, dan menghubungkannya ke sumber data Anda. Anda juga akan dapat membaca dan menulis ke database Amazon DynamoDB Anda melalui pernyataan GraphQL dan berlangganan data waktu nyata.

Ada langkah-langkah khusus yang harus diselesaikan agar pernyataan GraphQL diterjemahkan ke operasi Amazon DynamoDB dan agar tanggapan diterjemahkan kembali ke GraphQL. Tutorial ini menguraikan proses konfigurasi melalui beberapa skenario dunia nyata dan pola akses data.

Membuat GraphQL API

Untuk membuat GraphQL API di AWS AppSync

1. Buka AppSync konsol dan pilih Buat API.
2. Pilih Desain dari awal dan pilih Berikutnya.
3. Beri nama API Anda`PostTutorialAPI`, lalu pilih Berikutnya. Lewati ke halaman ulasan sambil menjaga opsi lainnya disetel ke nilai defaultnya dan pilih`Create`.

AWS AppSyncKonsol membuat API GraphQL baru untuk Anda. Dengan detault, ini menggunakan mode otentikasi kunci API. Anda dapat menggunakan konsol untuk mengatur sisa GraphQL API dan menjalankan kueri terhadapnya selama sisa tutorial ini.

Mendefinisikan API posting dasar

Sekarang setelah Anda memiliki GraphQL API, Anda dapat menyiapkan skema dasar yang memungkinkan pembuatan dasar, pengambilan, dan penghapusan data posting.

Untuk menambahkan data ke skema Anda

1. Di API Anda, pilih tab Skema.
2. Kami akan membuat skema yang mendefinisikan `Post` tipe dan operasi `addPost` untuk menambah dan mendapatkan `Post` objek. Di panel Skema, ganti konten dengan kode berikut:

```
schema {
    query: Query
    mutation: Mutation
}

type Query {
    getPost(id: ID): Post
}

type Mutation {
    addPost(
```

```
        id: ID!
        author: String!
        title: String!
        content: String!
        url: String!
    ): Post!
}

type Post {
    id: ID!
    author: String
    title: String
    content: String
    url: String
    ups: Int!
    downs: Int!
    version: Int!
}
```

3. Pilih Simpan Skema.

Menyiapkan tabel Amazon DynamoDB Anda

AWS AppSyncKonsol dapat membantu menyediakan AWS sumber daya yang diperlukan untuk menyimpan sumber daya Anda sendiri di tabel Amazon DynamoDB. Pada langkah ini, Anda akan membuat tabel Amazon DynamoDB untuk menyimpan posting Anda. Anda juga akan menyiapkan indeks sekunder yang akan kita gunakan nanti.

Untuk membuat tabel Amazon DynamoDB

1. Pada halaman Skema, pilih Buat Sumber Daya.
2. Pilih Gunakan tipe yang ada, lalu pilih `Post` jenisnya.
3. Di bagian Indeks Tambahan, pilih Tambah Indeks.
4. Beri nama indeksnya`author-index`.
5. Atur `Primary key` ke `author` dan `Sort` kunci ke`None`.
6. Nonaktifkan Secara otomatis menghasilkan GraphQL. Dalam contoh ini, kita akan membuat resolver sendiri.
7. Pilih Create (Buat).

Anda sekarang memiliki sumber data baru yang disebut`PostTable`, yang dapat Anda lihat dengan mengunjungi Sumber data di tab samping. Anda akan menggunakan sumber data ini untuk menautkan kueri dan mutasi Anda ke tabel Amazon DynamoDB Anda.

Menyiapkan AddPost resolver (Amazon DynamoDB) PutItem

Sekarang AWS AppSync mengetahui tabel Amazon DynamoDB, Anda dapat menautkannya ke kueri dan mutasi individual dengan mendefinisikan resolver. Resolver pertama yang Anda buat adalah resolver `addPost` pipeline yang digunakan JavaScript, yang memungkinkan Anda membuat postingan di tabel Amazon DynamoDB Anda. Pipeline resolver memiliki komponen-komponen berikut:

- Lokasi dalam skema GraphQL untuk melampirkan resolver. Dalam hal ini, Anda menyiapkan resolver di `createPost` bidang pada tipe. `Mutation` Penyelesai ini akan dipanggil saat pemanggil memanggil mutasi. `{ addPost(...){...} }`
- Sumber data yang akan digunakan untuk resolver ini. Dalam hal ini, Anda ingin menggunakan sumber data DynamoDB yang Anda tentukan sebelumnya, sehingga Anda dapat menambahkan entri ke dalam `post-table-for-tutorial` tabel DynamoDB.
- Penangan permintaan. Handler permintaan adalah fungsi yang menangani permintaan masuk dari pemanggil dan menerjemahkannya ke dalam instruksi untuk AWS AppSync melakukan terhadap DynamoDB.
- Penangan respons. Tugas penangan respons adalah menangani respons dari DynamoDB dan menerjemahkannya kembali menjadi sesuatu yang diharapkan GraphQL. Ini berguna jika bentuk data di DynamoDB berbeda dengan `Post` tipe di GraphQL, tetapi dalam hal ini mereka memiliki bentuk yang sama, jadi Anda cukup melewatkan data.

Untuk mengatur resolver Anda

1. Di API Anda, pilih tab Skema.
2. Di panel Resolvers, temukan **`addPost`** bidang di bawah **`Mutation`** tipe, lalu pilih Lampirkan.
3. Pilih sumber data Anda, lalu pilih Buat.
4. Di editor kode Anda, ganti kode dengan cuplikan ini:

```
import { util } from '@aws-appsync/utils'
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
```

```
 const item = { ...ctx.arguments, ups: 1, downs: 0, version: 1 }
 const key = { id: ctx.args.id ?? util.autoId() }
 return ddb.put({ key, item })
}

export function response(ctx) {
 return ctx.result
}
```

5. Pilih Save (Simpan).

> **Note**
>
> Dalam kode ini, Anda menggunakan utilitas modul DynamoDB yang memungkinkan Anda untuk dengan mudah membuat permintaan DynamoDB.

AWS AppSyncdilengkapi dengan utilitas untuk pembuatan ID otomatis yang disebut`util.autoId()`, yang digunakan untuk menghasilkan ID untuk posting baru Anda. Jika Anda tidak menentukan ID, utilitas akan secara otomatis menghasilkannya untuk Anda.

```
const key = { id: ctx.args.id ?? util.autoId() }
```

Untuk informasi selengkapnya tentang utilitas yang tersedia JavaScript, lihat fitur JavaScript runtime untuk resolver dan fungsi.

Panggil API untuk menambahkan postingan

Sekarang resolver telah dikonfigurasi, AWS AppSync dapat menerjemahkan `addPost` mutasi yang masuk ke operasi Amazon DynamoDB. `PutItem` Anda sekarang dapat menjalankan mutasi untuk meletakkan sesuatu di tabel.

Untuk menjalankan operasi

1. Di API Anda, pilih tab Kueri.
2. Di panel Kueri, tambahkan mutasi berikut:

```
mutation addPost {
  addPost(
    id: 123,
```

```
    author: "AUTHORNAME"
    title: "Our first post!"
    content: "This is our first post."
    url: "https://aws.amazon.com/appsync/"
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Pilih Run (tombol putar oranye), lalu pilih`addPost`. Hasil posting yang baru dibuat akan muncul di panel Hasil di sebelah kanan panel Kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "addPost": {
      "id": "123",
      "author": "AUTHORNAME",
      "title": "Our first post!",
      "content": "This is our first post.",
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
      "version": 1
    }
  }
}
```

Penjelasan berikut menunjukkan apa yang terjadi:

1. AWS AppSyncmenerima permintaan `addPost` mutasi.
2. AWS AppSyncmengeksekusi penangan permintaan dari resolver. `ddb.putFungsi` membuat `PutItem` permintaan yang terlihat seperti ini:

```
{
```

```
  operation: 'PutItem',
  key: { id: { S: '123' } },
  attributeValues: {
    downs: { N: 0 },
    author: { S: 'AUTHORNAME' },
    ups: { N: 1 },
    title: { S: 'Our first post!' },
    version: { N: 1 },
    content: { S: 'This is our first post.' },
    url: { S: 'https://aws.amazon.com/appsync/' }
  }
}
```

3. AWS AppSyncmenggunakan nilai ini untuk menghasilkan dan menjalankan permintaan Amazon `PutItem` DynamoDB.
4. AWS AppSyncmengambil hasil `PutItem` permintaan dan mengubahnya kembali ke tipe GraphQL.

```
{
    "id" : "123",
    "author": "AUTHORNAME",
    "title": "Our first post!",
    "content": "This is our first post.",
    "url": "https://aws.amazon.com/appsync/",
    "ups" : 1,
    "downs" : 0,
    "version" : 1
}
```

5. Response handler mengembalikan hasilnya segera (`return ctx.result`).
6. Hasil akhir terlihat dalam respons GraphQL.

Menyiapkan resolver GetPost (Amazon DynamoDB) GetItem

Sekarang Anda dapat menambahkan data ke tabel Amazon DynamoDB, Anda perlu mengatur kueri sehingga `getPost` dapat mengambil data itu dari tabel. Untuk melakukan ini, Anda mengatur resolver lain.

Untuk menambahkan resolver Anda

1. Di API Anda, pilih tab Skema.

2. Di panel Resolvers di sebelah kanan, temukan **`getPost`** bidang pada **`Query`** jenis dan kemudian pilih Lampirkan.
3. Pilih sumber data Anda, lalu pilih Buat.
4. Di editor kode, ganti kode dengan cuplikan ini:

```
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
 return ddb.get({ key: { id: ctx.args.id } })
}

export const response = (ctx) => ctx.result
```

5. Simpan resolver Anda.

> **Note**
>
> Dalam resolver ini, kita menggunakan ekspresi fungsi panah untuk handler respon.

Panggil API untuk mendapatkan postingan

Sekarang resolver telah diatur, AWS AppSync tahu cara menerjemahkan `getPost` kueri masuk ke operasi Amazon DynamoDB. `GetItem` Anda sekarang dapat menjalankan kueri untuk mengambil posting yang Anda buat sebelumnya.

Untuk menjalankan kueri

1. Di API Anda, pilih tab Kueri.
2. Di panel Kueri, tambahkan kode berikut, dan gunakan id yang Anda salin setelah membuat posting Anda:

```
query getPost {
  getPost(id: "123") {
    id
    author
    title
    content
    url
    ups
```

```
    downs
    version
  }
}
```

3. Pilih Run (tombol putar oranye), lalu pilih`getPost`. Hasil posting yang baru dibuat akan muncul di panel Hasil di sebelah kanan panel Kueri.
4. Postingan yang diambil dari Amazon DynamoDB akan muncul di panel Hasil di sebelah kanan panel Kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "getPost": {
      "id": "123",
      "author": "AUTHORNAME",
      "title": "Our first post!",
      "content": "This is our first post.",
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
      "version": 1
    }
  }
}
```

Atau, ambil contoh berikut:

```
query getPost {
  getPost(id: "123") {
    id
    author
    title
  }
}
```

Jika `getPost` kueri Anda hanya membutuhkan`id`,, dan `author``title`, Anda dapat mengubah fungsi permintaan Anda untuk menggunakan ekspresi proyeksi untuk menentukan hanya atribut yang Anda inginkan dari tabel DynamoDB Anda untuk menghindari transfer data yang tidak perlu dari DynamoDB ke. AWS AppSync Misalnya, fungsi permintaan mungkin terlihat seperti cuplikan di bawah ini:

```
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
 return ddb.get({
  key: { id: ctx.args.id },
  projection: ['author', 'id', 'title'],
 })
}

export const response = (ctx) => ctx.result
```

Anda juga dapat menggunakan selectionSetListwith `getPost` untuk mewakili`expression`:

```
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
 const projection = ctx.info.selectionSetList.map((field) => field.replace('/', '.'))
 return ddb.get({ key: { id: ctx.args.id }, projection })
}

export const response = (ctx) => ctx.result
```

Buat mutasi UpdatePost (Amazon DynamoDB) UpdateItem

Sejauh ini, Anda dapat membuat dan mengambil `Post` objek di Amazon DynamoDB. Selanjutnya, Anda akan menyiapkan mutasi baru untuk memperbarui objek. Dibandingkan dengan `addPost` mutasi yang mengharuskan semua bidang ditentukan, mutasi ini memungkinkan Anda untuk hanya menentukan bidang yang ingin Anda ubah. Ini juga memperkenalkan `expectedVersion` argumen baru yang memungkinkan Anda menentukan versi yang ingin Anda modifikasi. Anda akan mengatur kondisi yang memastikan bahwa Anda memodifikasi versi terbaru dari objek. Anda akan melakukan ini menggunakan operasi `UpdateItem` Amazon DynamoD.sc

Untuk memperbarui resolver Anda

1. Di API Anda, pilih tab Skema.
2. Di panel Skema, ubah `Mutation` tipe untuk menambahkan `updatePost` mutasi baru sebagai berikut:

```
type Mutation {
    updatePost(
```

```
        id: ID!,
        author: String,
        title: String,
        content: String,
        url: String,
        expectedVersion: Int!
    ): Post

    addPost(
        id: ID
        author: String!
        title: String!
        content: String!
        url: String!
    ): Post!
}
```

3. Pilih Simpan Skema.
4. Di panel Resolvers di sebelah kanan, temukan **`updatePost`** bidang yang baru dibuat pada **`Mutation`** jenisnya, lalu pilih Lampirkan. Buat resolver baru Anda menggunakan cuplikan di bawah ini:

```
import { util } from '@aws-appsync/utils';
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const { id, expectedVersion, ...rest } = ctx.args;
  const values = Object.entries(rest).reduce((obj, [key, value]) => {
    obj[key] = value ?? ddb.operations.remove();
    return obj;
  }, {});

  return ddb.update({
    key: { id },
    condition: { version: { eq: expectedVersion } },
    update: { ...values, version: ddb.operations.increment(1) },
  });
}

export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type);
```

```
  }
  return result;
```

5. Simpan semua perubahan yang Anda buat.

Resolver ini digunakan `ddb.update` untuk membuat permintaan Amazon DynamoDB. `UpdateItem` Alih-alih menulis seluruh item, Anda hanya meminta Amazon DynamoDB untuk memperbarui atribut tertentu. Ini dilakukan dengan menggunakan ekspresi pembaruan Amazon DynamoDB.

`ddb.update`Fungsi mengambil kunci dan objek update sebagai argumen. Kemudian, Anda memeriksa nilai-nilai argumen yang masuk. Ketika nilai diatur ke`null`, gunakan operasi `remove` DynamoDB untuk memberi sinyal bahwa nilai harus dihapus dari item DynamoDB.

Ada juga `condition` bagian baru. Ekspresi kondisi memungkinkan Anda memberi tahu AWS AppSync dan Amazon DynamoDB apakah permintaan harus berhasil atau tidak berdasarkan status objek yang sudah ada di Amazon DynamoDB sebelum operasi dilakukan. Dalam hal ini, Anda hanya ingin `UpdateItem` permintaan berhasil jika `version` bidang item yang saat ini ada di Amazon DynamoDB sama persis dengan `expectedVersion` argumen. Ketika item diperbarui, kami ingin menambah nilai. `version` Ini mudah dilakukan dengan fungsi operasi`increment`.

Untuk informasi selengkapnya tentang ekspresi kondisi, lihat dokumentasi Ekspresi kondisi.

Untuk info selengkapnya tentang `UpdateItem` permintaan, lihat UpdateItemdokumentasi dan dokumentasi modul DynamoDB.

Untuk informasi selengkapnya tentang cara menulis ekspresi pembaruan, lihat dokumentasi DynamoDB UpdateExpressions.

Panggil API untuk memperbarui posting

Mari kita coba memperbarui `Post` objek dengan resolver baru.

Untuk memperbarui objek Anda

1. Di API Anda, pilih tab Kueri.
2. Di panel Kueri, tambahkan mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya:

```
mutation updatePost {
  updatePost(
```

```
    id:123
    title: "An empty story"
    content: null
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Pilih Run (tombol putar oranye), lalu pilih`updatePost`.
4. Posting yang diperbarui di Amazon DynamoDB akan muncul di panel Hasil di sebelah kanan panel Kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "updatePost": {
      "id": "123",
      "author": "A new author",
      "title": "An empty story",
      "content": null,
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
      "version": 2
    }
  }
}
```

Dalam permintaan ini, Anda meminta AWS AppSync dan Amazon DynamoDB untuk memperbarui dan bidang sajatitle. `content` Semua bidang lainnya dibiarkan sendiri (selain menambah `version` bidang). Anda mengatur `title` atribut ke nilai baru dan menghapus `content` atribut dari posting. Bidang `author`url,ups, dan `downs` ladang dibiarkan tak tersentuh. Coba jalankan permintaan mutasi lagi sambil meninggalkan permintaan persis apa adanya. Anda akan melihat respons yang mirip dengan berikut ini:

```
{
  "data": {
    "updatePost": null
  },
  "errors": [
    {
      "path": [
        "updatePost"
      ],
      "data": null,
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      "errorInfo": null,
      "locations": [
        {
          "line": 2,
          "column": 3,
          "sourceName": null
        }
      ],
      "message": "The conditional request failed (Service: DynamoDb, Status Code: 400,
 Request ID: 1RR3QN5F35CS8IV5VR4OQO9NNBVV4KQNSO5AEMVJF66Q9ASUAAJG)"
    }
  ]
}
```

Permintaan gagal karena ekspresi kondisi dievaluasi menjadi`false`:

1. Pertama kali Anda menjalankan permintaan, nilai `version` bidang posting di Amazon DynamoDB 1 adalah, yang cocok dengan argumen. `expectedVersion` Permintaan berhasil, yang berarti `version` bidang tersebut bertambah di Amazon DynamoDB ke. 2
2. Kedua kalinya Anda menjalankan permintaan, nilai `version` bidang posting di Amazon DynamoDB 2 adalah, yang tidak cocok dengan argumen. `expectedVersion`

Pola ini biasanya disebut penguncian optimis.

Buat mutasi suara (Amazon DynamoDB UpdateItem)

`Post`Jenis berisi `ups` dan `downs` bidang untuk mengaktifkan perekaman upvotes dan downvotes. Namun, pada saat ini, API tidak mengizinkan kami melakukan apa pun dengan mereka. Mari tambahkan mutasi untuk memungkinkan kita meningkatkan dan menurunkan suara posting.

Untuk menambahkan mutasi Anda

1. Di API Anda, pilih tab Skema.
2. Di panel Skema, ubah `Mutation` jenisnya dan tambahkan `DIRECTION` enum untuk menambahkan mutasi suara baru:

```
type Mutation {
    vote(id: ID!, direction: DIRECTION!): Post
    updatePost(
        id: ID!,
        author: String,
        title: String,
        content: String,
        url: String,
        expectedVersion: Int!
    ): Post
    addPost(
        id: ID,
        author: String!,
        title: String!,
        content: String!,
        url: String!
    ): Post!
}

enum DIRECTION {
  UP
  DOWN
}
```

3. Pilih Simpan Skema.
4. Di panel Resolvers di sebelah kanan, temukan **`vote`** bidang yang baru dibuat pada **`Mutation`** jenisnya, lalu pilih Lampirkan. Buat resolver baru dengan membuat dan mengganti kode dengan cuplikan berikut:

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const field = ctx.args.direction === 'UP' ? 'ups' : 'downs';
  return ddb.update({
    key: { id: ctx.args.id },
    update: {
```

```
      [field]: ddb.operations.increment(1),
      version: ddb.operations.increment(1),
    },
  });
}

export const response = (ctx) => ctx.result;
```

5. Simpan semua perubahan yang Anda buat.

Panggil API untuk melakukan upvote atau downvote sebuah postingan

Sekarang resolver baru telah disiapkan, AWS AppSync tahu cara menerjemahkan masuk `upvotePost` atau `downvote` mutasi ke operasi Amazon DynamoDB. `UpdateItem` Anda sekarang dapat menjalankan mutasi untuk upvote atau downvote posting yang Anda buat sebelumnya.

Untuk menjalankan mutasi Anda

1. Di API Anda, pilih tab Kueri.
2. Di panel Kueri, tambahkan mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya:

```
mutation votePost {
  vote(id:123, direction: UP) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Pilih Run (tombol putar oranye), lalu pilih`votePost`.
4. Posting yang diperbarui di Amazon DynamoDB akan muncul di panel Hasil di sebelah kanan panel Kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
```

```
    "vote": {
      "id": "123",
      "author": "A new author",
      "title": "An empty story",
      "content": null,
      "url": "https://aws.amazon.com/appsync/",
      "ups": 6,
      "downs": 0,
      "version": 4
    }
  }
}
```

5. Pilih Jalankan beberapa kali lagi. Anda akan melihat `version` bidang `ups` dan bertambah 1 setiap kali Anda menjalankan kueri.
6. Ubah kueri untuk menyebutnya dengan yang berbeda`DIRECTION`.

```
mutation votePost {
  vote(id:123, direction: DOWN) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

7. Pilih Run (tombol putar oranye), lalu pilih`votePost`.

   Kali ini, Anda akan melihat `version` bidang `downs` dan bertambah 1 setiap kali Anda menjalankan kueri.

Menyiapkan resolver DeletePost (Amazon DynamoDB) DeleteItem

Selanjutnya, Anda akan ingin membuat mutasi untuk menghapus posting. Anda akan melakukan ini menggunakan operasi `DeleteItem` Amazon DynamoDB.

Untuk menambahkan mutasi Anda

1. Dalam skema Anda, pilih tab Skema.
2. Di panel Skema, ubah `Mutation` tipe untuk menambahkan mutasi baru`deletePost`:

```
type Mutation {
    deletePost(id: ID!, expectedVersion: Int): Post
    vote(id: ID!, direction: DIRECTION!): Post
    updatePost(
        id: ID!,
        author: String,
        title: String,
        content: String,
        url: String,
        expectedVersion: Int!
    ): Post
    addPost(
        id: ID
        author: String!,
        title: String!,
        content: String!,
        url: String!
    ): Post!
}
```

3. Kali ini, Anda membuat `expectedVersion` bidang opsional. Selanjutnya, pilih Simpan Skema.
4. Di panel Resolvers di sebelah kanan, temukan **`delete`** bidang yang baru dibuat dalam **`Mutation`** tipe, lalu pilih Lampirkan. Buat resolver baru menggunakan kode berikut:

```
import { util } from '@aws-appsync/utils'

import { util } from '@aws-appsync/utils';
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  let condition = null;
  if (ctx.args.expectedVersion) {
    condition = {
      or: [
        { id: { attributeExists: false } },
        { version: { eq: ctx.args.expectedVersion } },
      ],
    };
  }
```

```
  return ddb.remove({ key: { id: ctx.args.id }, condition });
}

export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type);
  }
  return result;
}
```

> **Note**
>
> `expectedVersion`Argumen adalah argumen opsional. Jika pemanggil menetapkan `expectedVersion` argumen dalam permintaan, penangan permintaan menambahkan kondisi yang hanya memungkinkan `DeleteItem` permintaan berhasil jika item sudah dihapus atau jika `version` atribut posting di Amazon DynamoDB sama persis dengan. `expectedVersion` Jika ditinggalkan, tidak ada ekspresi kondisi yang ditentukan pada `DeleteItem` permintaan. Itu berhasil terlepas dari nilai `version` atau apakah item tersebut ada di Amazon DynamoDB.
>
> Meskipun Anda menghapus item, Anda dapat mengembalikan item yang telah dihapus, jika belum dihapus.

Untuk info lebih lanjut tentang `DeleteItem` permintaan, lihat DeleteItemdokumentasi.

Panggil API untuk menghapus postingan

Sekarang resolver telah diatur, AWS AppSync tahu cara menerjemahkan `delete` mutasi yang masuk ke operasi Amazon DynamoDB. `DeleteItem` Anda sekarang dapat menjalankan mutasi untuk menghapus sesuatu di tabel.

Untuk menjalankan mutasi Anda

1. Di API Anda, pilih tab Kueri.
2. Di panel Kueri, tambahkan mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya:

```
mutation deletePost {
  deletePost(id:123) {
```

```
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Pilih Run (tombol putar oranye), lalu pilih`deletePost`.
4. Posting dihapus dari Amazon DynamoDB. Perhatikan bahwa AWS AppSync mengembalikan nilai item yang telah dihapus dari Amazon DynamoDB, yang akan muncul di panel Hasil di sebelah kanan panel Kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "deletePost": {
      "id": "123",
      "author": "A new author",
      "title": "An empty story",
      "content": null,
      "url": "https://aws.amazon.com/appsync/",
      "ups": 6,
      "downs": 4,
      "version": 12
    }
  }
}
```

5. Nilai hanya dikembalikan jika panggilan ini `deletePost` adalah salah satu yang benar-benar menghapusnya dari Amazon DynamoDB. Pilih Jalankan lagi.
6. Panggilan masih berhasil, tetapi tidak ada nilai yang dikembalikan:

```
{
  "data": {
    "deletePost": null
  }
}
```

7. Sekarang, mari kita coba menghapus posting, tapi kali ini menentukan. `expectedValue` Pertama, Anda harus membuat posting baru karena Anda baru saja menghapus yang telah Anda kerjakan sejauh ini.
8. Di panel Kueri, tambahkan mutasi berikut:

```
mutation addPost {
  addPost(
    id:123
    author: "AUTHORNAME"
    title: "Our second post!"
    content: "A new post."
    url: "https://aws.amazon.com/appsync/"
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

9. Pilih Run (tombol putar oranye), lalu pilih`addPost`.
10. Hasil posting yang baru dibuat akan muncul di panel Hasil di sebelah kanan panel Kueri. Rekam `id` objek yang baru dibuat karena Anda akan membutuhkannya hanya dalam beberapa saat. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "addPost": {
      "id": "123",
      "author": "AUTHORNAME",
      "title": "Our second post!",
      "content": "A new post.",
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
      "version": 1
    }
  }
```

```
}
```

11. Sekarang, mari kita coba menghapus posting itu dengan nilai ilegal untuk ExpectedVersion. Di panel Kueri, tambahkan mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya:

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 9999
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

12. Pilih Run (tombol putar oranye), lalu pilih`deletePost`. Hasil berikut dikembalikan:

```
{
  "data": {
    "deletePost": null
  },
  "errors": [
    {
      "path": [
        "deletePost"
      ],
      "data": null,
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      "errorInfo": null,
      "locations": [
        {
          "line": 2,
          "column": 3,
          "sourceName": null
        }
      ],
```

```
      "message": "The conditional request failed (Service: DynamoDb, Status Code:
 400, Request ID: 7083O037M1FTFRK038A4CI9H43VV4KQNSO5AEMVJF66Q9ASUAAJG)"
    }
  ]
}
```

13. Permintaan gagal karena ekspresi kondisi dievaluasi. `false` Nilai untuk `version` posting di Amazon DynamoDB tidak cocok dengan yang ditentukan `expectedValue` dalam argumen. Nilai objek saat ini dikembalikan di `data` bidang di `errors` bagian respons GraphQL. Coba lagi permintaannya, tetapi `expectedVersion` perbaiki:

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

14. Pilih Run (tombol putar oranye), lalu pilih`deletePost`.

    Kali ini permintaan berhasil, dan nilai yang dihapus dari Amazon DynamoDB dikembalikan:

```
{
  "data": {
    "deletePost": {
      "id": "123",
      "author": "AUTHORNAME",
      "title": "Our second post!",
      "content": "A new post.",
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
      "version": 1
    }
```

```
  }
}
```

15. Pilih Jalankan lagi. Panggilan masih berhasil, tetapi kali ini tidak ada nilai yang dikembalikan karena posting sudah dihapus di Amazon DynamoDB.

```
{ "data": { "deletePost": null } }
```

Menyiapkan resolver AllPost (Amazon DynamoDB Scan)

Sejauh ini, API hanya berguna jika Anda mengetahui setiap posting yang ingin Anda lihat. `id` Mari tambahkan resolver baru yang mengembalikan semua posting dalam tabel.

Untuk menambahkan mutasi Anda

1. Di API Anda, pilih tab Skema.
2. Di panel Skema, ubah `Query` jenis untuk menambahkan `allPost` kueri baru sebagai berikut:

```
type Query {
    allPost(limit: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
}
```

3. Tambahkan `PaginationPosts` tipe baru:

```
type PaginatedPosts {
    posts: [Post!]!
    nextToken: String
}
```

4. Pilih Simpan Skema.
5. Di panel Resolvers di sebelah kanan, temukan **`allPost`** bidang yang baru dibuat dalam **`Query`** tipe, lalu pilih Lampirkan. Buat resolver baru dengan kode berikut:

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const { limit = 20, nextToken } = ctx.arguments;
  return ddb.scan({ limit, nextToken });
}
```

```
export function response(ctx) {
  const { items: posts = [], nextToken } = ctx.result;
  return { posts, nextToken };
}
```

Handler permintaan resolver ini mengharapkan dua argumen opsional:

- `limit`- Menentukan jumlah maksimum item untuk kembali dalam satu panggilan.
- `nextToken`- Digunakan untuk mengambil set hasil berikutnya (kami akan menunjukkan dari mana nilai untuk `nextToken` berasal nanti).

6. Simpan perubahan apa pun yang dilakukan pada resolver Anda.

Untuk informasi selengkapnya tentang `Scan` permintaan, lihat dokumentasi referensi Pindai.

Panggil API untuk memindai semua posting

Sekarang resolver telah diatur, AWS AppSync tahu cara menerjemahkan `allPost` kueri masuk ke operasi Amazon DynamoDB. `Scan` Anda sekarang dapat memindai tabel untuk mengambil semua posting. Sebelum Anda dapat mencobanya, Anda perlu mengisi tabel dengan beberapa data karena Anda telah menghapus semua yang telah Anda kerjakan sejauh ini.

Untuk menambah dan menanyakan data

1. Di API Anda, pilih tab Kueri.
2. Di panel Kueri, tambahkan mutasi berikut:

```
mutation addPost {
  post1: addPost(id:1 author: "AUTHORNAME" title: "A series of posts, Volume 1"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post2: addPost(id:2 author: "AUTHORNAME" title: "A series of posts, Volume 2"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post3: addPost(id:3 author: "AUTHORNAME" title: "A series of posts, Volume 3"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post4: addPost(id:4 author: "AUTHORNAME" title: "A series of posts, Volume 4"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post5: addPost(id:5 author: "AUTHORNAME" title: "A series of posts, Volume 5"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post6: addPost(id:6 author: "AUTHORNAME" title: "A series of posts, Volume 6"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
```

```
  post7: addPost(id:7 author: "AUTHORNAME" title: "A series of posts, Volume 7"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post8: addPost(id:8 author: "AUTHORNAME" title: "A series of posts, Volume 8"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post9: addPost(id:9 author: "AUTHORNAME" title: "A series of posts, Volume 9"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
}
```

3. Pilih Run (tombol putar oranye).
4. Sekarang, mari kita pindai tabel, mengembalikan lima hasil sekaligus. Di panel Kueri, tambahkan kueri berikut:

```
query allPost {
  allPost(limit: 5) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

5. Pilih Run (tombol putar oranye), lalu pilih`allPost`.

   Lima posting pertama akan muncul di panel Hasil di sebelah kanan panel Kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPost": {
      "posts": [
        {
          "id": "5",
          "title": "A series of posts, Volume 5"
        },
        {
          "id": "1",
          "title": "A series of posts, Volume 1"
        },
        {
          "id": "6",
          "title": "A series of posts, Volume 6"
        },
```

```
        {
          "id": "9",
          "title": "A series of posts, Volume 9"
        },
        {
          "id": "7",
          "title": "A series of posts, Volume 7"
        }
      ],
      "nextToken": "<token>"
    }
  }
}
```

6. Anda menerima lima hasil dan `nextToken` yang dapat Anda gunakan untuk mendapatkan set hasil berikutnya. Perbarui `allPost` kueri untuk menyertakan `nextToken` dari kumpulan hasil sebelumnya:

```
query allPost {
  allPost(
    limit: 5
    nextToken: "<token>"
  ) {
    posts {
      id
      author
    }
    nextToken
  }
}
```

7. Pilih Run (tombol putar oranye), lalu pilih`allPost`.

   Empat posting yang tersisa akan muncul di panel Hasil di sebelah kanan panel Kueri. Tidak ada `nextToken` dalam rangkaian hasil ini karena Anda telah membaca semua sembilan posting dengan tidak ada yang tersisa. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPost": {
      "posts": [
        {
          "id": "2",
```

```
          "title": "A series of posts, Volume 2"
        },
        {
          "id": "3",
          "title": "A series of posts, Volume 3"
        },
        {
          "id": "4",
          "title": "A series of posts, Volume 4"
        },
        {
          "id": "8",
          "title": "A series of posts, Volume 8"
        }
      ],
      "nextToken": null
    }
  }
}
```

Menyiapkan resolver allPostsBy Penulis (Amazon DynamoDB Query)

Selain memindai Amazon DynamoDB untuk semua posting, Anda juga dapat meminta Amazon DynamoDB untuk mengambil posting yang dibuat oleh penulis tertentu. Tabel Amazon DynamoDB yang Anda buat sebelumnya sudah memiliki panggilan `GlobalSecondaryIndex` yang `author-index` dapat Anda gunakan dengan operasi Amazon DynamoDB untuk mengambil semua `Query` posting yang dibuat oleh penulis tertentu.

Untuk menambahkan kueri Anda

1. Di API Anda, pilih tab Skema.
2. Di panel Skema, ubah `Query` jenis untuk menambahkan `allPostsByAuthor` kueri baru sebagai berikut:

```
type Query {
    allPostsByAuthor(author: String!, limit: Int, nextToken: String): PaginatedPosts!
    allPost(limit: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
}
```

Perhatikan bahwa ini menggunakan `PaginatedPosts` jenis yang sama dengan yang Anda gunakan dengan `allPost` kueri.

3. Pilih Simpan Skema.
4. Di panel Resolvers di sebelah kanan, temukan **`allPostsByAuthor`** bidang yang baru dibuat pada **`Query`** jenisnya, lalu pilih Lampirkan. Buat resolver menggunakan cuplikan di bawah ini:

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const { limit = 20, nextToken, author } = ctx.arguments;
  return ddb.query({
    index: 'author-index',
    query: { author: { eq: author } },
    limit,
    nextToken,
  });
}

export function response(ctx) {
  const { items: posts = [], nextToken } = ctx.result;
  return { posts, nextToken };
}
```

Seperti `allPost` resolver, resolver ini memiliki dua argumen opsional:

- `limit`- Menentukan jumlah maksimum item untuk kembali dalam satu panggilan.
- `nextToken`- Mengambil set hasil berikutnya (nilai untuk `nextToken` dapat diperoleh dari panggilan sebelumnya).

5. Simpan perubahan apa pun yang dilakukan pada resolver Anda.

Untuk informasi selengkapnya tentang `Query` permintaan, lihat dokumentasi referensi kueri.

Panggil API untuk menanyakan semua posting oleh penulis

Sekarang resolver telah diatur, AWS AppSync tahu bagaimana menerjemahkan `allPostsByAuthor` mutasi yang masuk ke operasi DynamoDB terhadap indeks. `Query` `author-index` Anda sekarang dapat meminta tabel untuk mengambil semua posting oleh penulis tertentu.

Sebelum ini, bagaimanapun, mari kita mengisi tabel dengan beberapa posting lagi, karena setiap posting sejauh ini memiliki penulis yang sama.

Untuk menambahkan data dan kueri

1. Di API Anda, pilih tab Kueri.
2. Di panel Kueri, tambahkan mutasi berikut:

```
mutation addPost {
  post1: addPost(id:10 author: "Nadia" title: "The cutest dog in the world" content:
 "So cute. So very, very cute." url: "https://aws.amazon.com/appsync/" ) { author,
 title }
  post2: addPost(id:11 author: "Nadia" title: "Did you know...?" content: "AppSync
 works offline?" url: "https://aws.amazon.com/appsync/" ) { author, title }
  post3: addPost(id:12 author: "Steve" title: "I like GraphQL" content: "It's great"
 url: "https://aws.amazon.com/appsync/" ) { author, title }
}
```

3. Pilih Run (tombol putar oranye), lalu pilih`addPost`.
4. Sekarang, mari kita menanyakan tabel, mengembalikan semua posting yang ditulis oleh. `Nadia` Di panel Kueri, tambahkan kueri berikut:

```
query allPostsByAuthor {
  allPostsByAuthor(author: "Nadia") {
    posts {
      id
      title
    }
    nextToken
  }
}
```

5. Pilih Run (tombol putar oranye), lalu pilih`allPostsByAuthor`. Semua posting yang ditulis oleh akan **`Nadia`** muncul di panel Hasil di sebelah kanan panel Queries. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
```

```
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you know...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

6. Pagination bekerja untuk hal yang `Query` sama seperti yang dilakukannya. `Scan` Sebagai contoh, mari kita cari semua posting dengan`AUTHORNAME`, mendapatkan lima sekaligus.
7. Di panel Kueri, tambahkan kueri berikut:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    limit: 5
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

8. Pilih Run (tombol putar oranye), lalu pilih`allPostsByAuthor`. Semua posting yang ditulis oleh akan **`AUTHORNAME`** muncul di panel Hasil di sebelah kanan panel Queries. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "6",
          "title": "A series of posts, Volume 6"
        },
```

```
        {
          "id": "4",
          "title": "A series of posts, Volume 4"
        },
        {
          "id": "2",
          "title": "A series of posts, Volume 2"
        },
        {
          "id": "7",
          "title": "A series of posts, Volume 7"
        },
        {
          "id": "1",
          "title": "A series of posts, Volume 1"
        }
      ],
      "nextToken": "<token>"
    }
  }
}
```

9. Perbarui `nextToken` argumen dengan nilai yang dikembalikan dari kueri sebelumnya sebagai berikut:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    limit: 5
    nextToken: "<token>"
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

10. Pilih Run (tombol putar oranye), lalu pilih `allPostsByAuthor`. Posting yang tersisa yang ditulis oleh **`AUTHORNAME`** akan muncul di panel Hasil di sebelah kanan panel Kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "8",
          "title": "A series of posts, Volume 8"
        },
        {
          "id": "5",
          "title": "A series of posts, Volume 5"
        },
        {
          "id": "3",
          "title": "A series of posts, Volume 3"
        },
        {
          "id": "9",
          "title": "A series of posts, Volume 9"
        }
      ],
      "nextToken": null
    }
  }
}
```

Menggunakan set

Sampai saat ini, `Post` tipe telah menjadi objek kunci/nilai datar. Anda juga dapat memodelkan objek kompleks dengan resolver Anda, seperti set, daftar, dan peta. Mari kita perbarui `Post` jenis untuk menyertakan tag. Sebuah posting dapat memiliki nol atau lebih tag, yang disimpan di DynamoDB sebagai String Set. Anda juga akan menyiapkan beberapa mutasi untuk menambah dan menghapus tag, dan kueri baru untuk memindai posting dengan tag tertentu.

Untuk mengatur data Anda

1. Di API Anda, pilih tab Skema.
2. Di panel Skema, ubah `Post` jenis untuk menambahkan `tags` bidang baru sebagai berikut:

```
type Post {
```

```
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
  tags: [String!]
}
```

3. Di panel Skema, ubah `Query` jenis untuk menambahkan `allPostsByTag` kueri baru sebagai berikut:

```
type Query {
  allPostsByTag(tag: String!, limit: Int, nextToken: String): PaginatedPosts!
  allPostsByAuthor(author: String!, limit: Int, nextToken: String): PaginatedPosts!
  allPost(limit: Int, nextToken: String): PaginatedPosts!
  getPost(id: ID): Post
}
```

4. Di panel Skema, ubah `Mutation` tipe untuk menambahkan baru `addTag` dan `removeTag` mutasi sebagai berikut:

```
type Mutation {
  addTag(id: ID!, tag: String!): Post
  removeTag(id: ID!, tag: String!): Post
  deletePost(id: ID!, expectedVersion: Int): Post
  upvotePost(id: ID!): Post
  downvotePost(id: ID!): Post
  updatePost(
    id: ID!,
    author: String,
    title: String,
    content: String,
    url: String,
    expectedVersion: Int!
  ): Post
  addPost(
    author: String!,
    title: String!,
    content: String!,
    url: String!
```

```
  ): Post!
}
```

5. Pilih Simpan Skema.
6. Di panel Resolvers di sebelah kanan, temukan **`allPostsByTag`** bidang yang baru dibuat pada **`Query`** jenisnya, lalu pilih Lampirkan. Buat resolver Anda menggunakan cuplikan di bawah ini:

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const { limit = 20, nextToken, tag } = ctx.arguments;
  return ddb.scan({ limit, nextToken, filter: { tags: { contains: tag } } });
}

export function response(ctx) {
  const { items: posts = [], nextToken } = ctx.result;
  return { posts, nextToken };
}
```

7. Simpan perubahan apa pun yang telah Anda buat pada resolver Anda.
8. Sekarang, lakukan hal yang sama untuk `Mutation` bidang `addTag` menggunakan cuplikan di bawah ini:

> **Note**
>
> Meskipun utilitas DynamoDB saat ini tidak mendukung operasi set, Anda masih dapat berinteraksi dengan set dengan membuat permintaan sendiri.

```
import { util } from '@aws-appsync/utils'

export function request(ctx) {
 const { id, tag } = ctx.arguments
 const expressionValues = util.dynamodb.toMapValues({ ':plusOne': 1 })
 expressionValues[':tags'] = util.dynamodb.toStringSet([tag])

 return {
  operation: 'UpdateItem',
  key: util.dynamodb.toMapValues({ id }),
  update: {
   expression: `ADD tags :tags, version :plusOne`,
```

```
    expressionValues,
   },
 }
}

export const response = (ctx) => ctx.result
```

9. Simpan perubahan apa pun yang dilakukan pada resolver Anda.
10. Ulangi ini sekali lagi untuk `Mutation` bidang `removeTag` menggunakan cuplikan di bawah ini:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
   const { id, tag } = ctx.arguments;
   const expressionValues = util.dynamodb.toMapValues({ ':plusOne': 1 });
   expressionValues[':tags'] = util.dynamodb.toStringSet([tag]);

   return {
     operation: 'UpdateItem',
     key: util.dynamodb.toMapValues({ id }),
     update: {
       expression: `DELETE tags :tags ADD version :plusOne`,
       expressionValues,
     },
   };
 }

 export const response = (ctx) => ctx.resultexport
```

11. Simpan perubahan apa pun yang dilakukan pada resolver Anda.

Panggil API untuk bekerja dengan tag

Sekarang setelah Anda menyiapkan resolver, AWS AppSync tahu cara menerjemahkan masuk `addTagremoveTag`, dan permintaan `allPostsByTag` ke DynamoDB dan operasi. `UpdateItem` `Scan` Untuk mencobanya, mari pilih salah satu posting yang Anda buat sebelumnya. Misalnya, mari kita gunakan posting yang ditulis oleh. `Nadia`

Untuk menggunakan tag

1. Di API Anda, pilih tab Kueri.

2. Di panel Kueri, tambahkan kueri berikut:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "Nadia"
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

3. Pilih Run (tombol putar oranye), lalu pilih`allPostsByAuthor`.
4. Semua postingan Nadia akan muncul di panel Hasil di sebelah kanan panel Kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you known...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

5. Mari kita gunakan yang memiliki judul The cutest dog in the world. Rekam `id` karena Anda akan menggunakannya nanti. Sekarang, mari kita coba menambahkan dog tag.
6. Di panel Kueri, tambahkan mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya.

```
mutation addTag {
```

```
  addTag(id:10 tag: "dog") {
    id
    title
    tags
  }
}
```

7. Pilih Run (tombol putar oranye), lalu pilihaddTag. Posting diperbarui dengan tag baru:

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

8. Anda dapat menambahkan lebih banyak tag. Perbarui mutasi untuk mengubah `tag` argumen menjadipuppy:

```
mutation addTag {
  addTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

9. Pilih Run (tombol putar oranye), lalu pilihaddTag. Posting diperbarui dengan tag baru:

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog",
        "puppy"
      ]
```

```
    }
  }
}
```

10 Anda juga dapat menghapus tag. Di panel Kueri, tambahkan mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya:

```
mutation removeTag {
  removeTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

11 Pilih Run (tombol putar oranye), lalu pilih`removeTag`. Posting diperbarui dan `puppy` tag dihapus.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

12 Anda juga dapat mencari semua posting yang memiliki tag. Di panel Kueri, tambahkan kueri berikut:

```
query allPostsByTag {
  allPostsByTag(tag: "dog") {
    posts {
      id
      title
      tags
    }
    nextToken
  }
}
```

13. Pilih Run (tombol putar oranye), lalu pilih `allPostsByTag`. Semua posting yang memiliki dog tag dikembalikan sebagai berikut:

```
{
  "data": {
    "allPostsByTag": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world",
          "tags": [
            "dog",
            "puppy"
          ]
        }
      ],
      "nextToken": null
    }
  }
}
```

Kesimpulan

Dalam tutorial ini, Anda telah membangun API yang memungkinkan Anda memanipulasi `Post` objek di AWS AppSync DynamoDB menggunakan dan GraphQL.

Untuk membersihkan, Anda dapat menghapus AWS AppSync GraphQL API dari konsol.

Untuk menghapus peran yang terkait dengan tabel DynamoDB Anda, pilih sumber data Anda di tabel Sumber Data dan klik edit. Perhatikan nilai peran di bawah Buat atau gunakan peran yang ada. Buka konsol IAM untuk menghapus peran.

Untuk menghapus tabel DynamoDB Anda, klik pada nama tabel dalam daftar sumber data. Ini akan membawa Anda ke konsol DynamoDB tempat Anda dapat menghapus tabel.

Tutorial: Penyelesai Lambda

Anda dapat menggunakanAWS LambdabersamaAWS AppSyncuntuk menyelesaikan bidang GraphQL apa pun. Misalnya, kueri GraphQL mungkin mengirim panggilan ke instans Amazon Relational Database Service (Amazon RDS), dan mutasi GraphQL mungkin menulis ke aliran

Amazon Kinesis. Di bagian ini, kami akan menunjukkan cara menulis fungsi Lambda yang melakukan logika bisnis berdasarkan pemanggilan operasi bidang GraphQL.

Buat fungsi Lambda

Contoh berikut menunjukkan fungsi Lambda ditulis dalam`Node.js`(runtime: Node.js 18.x) yang melakukan operasi berbeda pada posting blog sebagai bagian dari aplikasi posting blog. Perhatikan bahwa kode harus disimpan dalam nama file dengan ekstensi.mis.

```
export const handler = async (event) => {
console.log('Received event {}', JSON.stringify(event, 3))

  const posts = {
1: { id: '1', title: 'First book', author: 'Author1', url: 'https://amazon.com/',
 content: 'SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT
 AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1', ups: '100', downs: '10', },
    2: { id: '2', title: 'Second book', author: 'Author2', url: 'https://amazon.com',
 content: 'SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT', ups: '100', downs:
 '10', },
    3: { id: '3', title: 'Third book', author: 'Author3', url: null, content: null,
 ups: null, downs: null },
    4: { id: '4', title: 'Fourth book', author: 'Author4', url: 'https://
www.amazon.com/', content: 'SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4', ups: '1000', downs: '0', },
    5: { id: '5', title: 'Fifth book', author: 'Author5', url: 'https://
www.amazon.com/', content: 'SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT
 AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT', ups: '50', downs: '0', },
  }

  const relatedPosts = {
1: [posts['4']],
    2: [posts['3'], posts['5']],
    3: [posts['2'], posts['1']],
    4: [posts['2'], posts['1']],
    5: [],
  }

  console.log('Got an Invoke Request.')
  let result
  switch (event.field) {
case 'getPost':
      return posts[event.arguments.id]
```

```
    case 'allPosts':
      return Object.values(posts)
    case 'addPost':
      // return the arguments back
return event.arguments
    case 'addPostErrorWithData':
      result = posts[event.arguments.id]
      // attached additional error information to the post
      result.errorMessage = 'Error with the mutation, data has changed'
      result.errorType = 'MUTATION_ERROR'
return result
    case 'relatedPosts':
      return relatedPosts[event.source.id]
    default:
      throw new Error('Unknown field, unable to resolve ' + event.field)
  }
}
```

Fungsi Lambda ini mengambil posting dengan ID, menambahkan posting, mengambil daftar posting, dan mengambil posting terkait untuk posting tertentu.

> **Note**
>
> Fungsi Lambda menggunakan`switch`pernyataan tentang`event.field`untuk menentukan bidang mana yang saat ini sedang diselesaikan.

Buat fungsi Lambda ini menggunakanAWSKonsol Manajemen.

Konfigurasikan sumber data untuk Lambda

Setelah Anda membuat fungsi Lambda, navigasikan ke GraphQL API Anda diAWS AppSynckonsol, dan kemudian pilihSumber Datatab.

PilihBuat sumber data, masukkan yang ramahNama sumber data(misalnya,**`Lambda`**), dan kemudian untukJenis sumber data, pilihAWS Lambdafungsi. UntukWilayah, pilih Wilayah yang sama dengan fungsi Anda. UntukFungsi ARN, pilih Amazon Resource Name (ARN) dari fungsi Lambda Anda.

Setelah memilih fungsi Lambda Anda, Anda dapat membuat yang baruAWS Identity and Access Management(IAM) peran (untuk yangAWS AppSyncmenetapkan izin yang sesuai) atau memilih peran yang ada yang memiliki kebijakan inline berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:REGION:ACCOUNTNUMBER:function/LAMBDA_FUNCTION"
        }
    ]
}
```

Anda juga harus membangun hubungan kepercayaan denganAWS AppSyncuntuk peran IAM sebagai berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "appsync.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Buat skema GraphQL

Sekarang sumber data terhubung ke fungsi Lambda Anda, buat skema GraphQL.

Dari editor skema diAWS AppSyncconsole, pastikan skema Anda cocok dengan skema berikut:

```
schema {
    query: Query
    mutation: Mutation
}
type Query {
    getPost(id:ID!): Post
```

```
    allPosts: [Post]
}
type Mutation {
    addPost(id: ID!, author: String!, title: String, content: String, url: String):
 Post!
}
type Post {
    id: ID!
    author: String!
    title: String
    content: String
    url: String
    ups: Int
    downs: Int
    relatedPosts: [Post]
}
```

Konfigurasikan resolver

Setelah mendaftarkan sumber data Lambda dan skema GraphQL yang valid, Anda dapat menghubungkan bidang GraphQL ke sumber data Lambda menggunakan resolver.

Anda akan membuat resolver yang menggunakanAWS AppSync JavaScript(APPSYNC_JS) runtime dan berinteraksi dengan fungsi Lambda Anda. Untuk mempelajari lebih lanjut tentang menulisAWS AppSyncresolver dan fungsi denganJavaScript, lihatJavaScriptfitur runtime untuk resolver dan fungsi.

Untuk informasi selengkapnya tentang template pemetaan Lambda, lihatJavaScriptreferensi fungsi resolver untuk Lambda.

Pada langkah ini, Anda melampirkan resolver ke fungsi Lambda untuk bidang berikut:`getPost(id:ID!): Post`,`allPosts: [Post]`,`addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!`, dan`Post.relatedPosts: [Post]`. DariSkemaeditor diAWS AppSynckonsol, diPenyelesaipanel, pilihLampirkandi sebelah`getPost(id:ID!): Post`lapangan. Pilih sumber data Lambda Anda. Selanjutnya, berikan kode berikut:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const {source, args} = ctx
  return {
```

```
    operation: 'Invoke',
    payload: { field: ctx.info.fieldName, arguments: args, source },
  };
}

export function response(ctx) {
  return ctx.result;
}
```

Kode resolver ini meneruskan nama bidang, daftar argumen, dan konteks tentang objek sumber ke fungsi Lambda saat memanggilnya. Pilih Save (Simpan).

Anda telah berhasil melampirkan resolver pertama Anda. Ulangi operasi ini untuk bidang yang tersisa.

Uji API GraphQL Anda

Sekarang fungsi Lambda Anda terhubung ke resolver GraphQL, Anda dapat menjalankan beberapa mutasi dan kueri menggunakan konsol atau aplikasi klien.

Di sisi kiriAWS AppSynckonsol, pilihPertanyaan, dan kemudian paste dalam kode berikut:

AddPost Mutasi

```
mutation AddPost {
    addPost(
        id: 6
        author: "Author6"
        title: "Sixth book"
        url: "https://www.amazon.com/"
        content: "This is the book is a tutorial for using GraphQL with AWS AppSync."
    ) {
        id
        author
        title
        content
        url
        ups
        downs
    }
}
```

Kueri GetPost

```
query GetPost {
    getPost(id: "2") {
        id
        author
        title
        content
        url
        ups
        downs
    }
}
```

AllPosts Query

```
query AllPosts {
    allPosts {
        id
        author
        title
        content
        url
        ups
        downs
        relatedPosts {
            id
            title
        }
    }
}
```

Mengembalikan kesalahan

Setiap resolusi bidang yang diberikan dapat mengakibatkan kesalahan. denganAWS AppSync, Anda dapat meningkatkan kesalahan dari sumber-sumber berikut:

- Penangan respons penyelesai
- Fungsi Lambda

Dari penangan respons resolver

Untuk meningkatkan kesalahan yang disengaja, Anda dapat menggunakan`util.error`metode utilitas. Dibutuhkan argumen sebua`errorMessage`, sebua`errorType`, dan opsional`data`nilai. The`data`berguna untuk mengembalikan data tambahan kembali ke klien ketika terjadi kesalahan. The`data`objek ditambahkan ke`errors`dalam respons akhir GraphQL.

Contoh berikut menunjukkan cara menggunakannya di`Post.relatedPosts: [Post]`penangan respons resolver.

```
// the Post.relatedPosts response handler
export function response(ctx) {
    util.error("Failed to fetch relatedPosts", "LambdaFailure", ctx.result)
    return ctx.result;
}
```

Ini menghasilkan respons GraphQL yang mirip dengan yang berikut:

```
{
    "data": {
        "allPosts": [
            {
                "id": "2",
                "title": "Second book",
                "relatedPosts": null
            },
            ...
        ]
    },
    "errors": [
        {
            "path": [
                "allPosts",
                0,
                "relatedPosts"
            ],
            "errorType": "LambdaFailure",
            "locations": [
                {
                    "line": 5,
                    "column": 5
                }
```

```
            ],
            "message": "Failed to fetch relatedPosts",
            "data": [
                {
                  "id": "2",
                  "title": "Second book"
                },
                {
                  "id": "1",
                  "title": "First book"
                }
            ]
        }
    ]
}
```

Dimana`allPosts[0].relatedPosts`adalahnolkarena kesalahan dan`errorMessage`,`errorType`, dan`data`hadir di`data.errors[0]`objek.

Dari fungsi Lambda

AWS AppSyncjuga memahami kesalahan yang dilemparkan fungsi Lambda. Model pemrograman Lambda memungkinkan Anda meningkatkanditanganikesalahan. Jika fungsi Lambda melempar kesalahan,AWS AppSyncgagal menyelesaikan bidang saat ini. Hanya pesan kesalahan yang dikembalikan dari Lambda yang disetel dalam respons. Saat ini, Anda tidak dapat meneruskan data asing apa pun kembali ke klien dengan memunculkan kesalahan dari fungsi Lambda.

> **Note**
>
> Jika fungsi Lambda Anda memunculkantidak tertanganikesalahanAWS AppSyncmenggunakan pesan kesalahan yang ditetapkan Lambda.

Fungsi Lambda berikut menimbulkan kesalahan:

```
export const handler = async (event) => {
  console.log('Received event {}', JSON.stringify(event, 3))
  throw new Error('I always fail.')
}
```

Kesalahan diterima di handler respons Anda. Anda dapat mengirimnya kembali dalam respons GraphQL dengan menambahkan kesalahan ke respons dengan`util.appendError`. Untuk melakukannya, ubahAWS AppSyncpenangan respons fungsi untuk ini:

```
// the lambdaInvoke response handler
export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type, result);
  }
  return result;
}
```

Ini mengembalikan respon GraphQL mirip dengan berikut ini:

```
{
  "data": {
    "allPosts": null
  },
  "errors": [
    {
      "path": [
        "allPosts"
      ],
      "data": null,
      "errorType": "Lambda:Unhandled",
      "errorInfo": null,
      "locations": [
        {
          "line": 2,
          "column": 3,
          "sourceName": null
        }
      ],
      "message": "I fail. always"
    }
  ]
}
```

Kasus penggunaan lanjutan: Batching

Fungsi Lambda dalam contoh ini memilikirelatedPostsbidang yang mengembalikan daftar posting terkait untuk posting tertentu. Dalam contoh kueri,allPostspemanggilan bidang dari fungsi Lambda mengembalikan lima posting. Karena kami menetapkan bahwa kami juga ingin menyelesaikannyarelatedPostsuntuk setiap pos yang dikembalikan,relatedPostsoperasi lapangan dipanggil lima kali.

```
query {
    allPosts {   // 1 Lambda invocation - yields 5 Posts
        id
        author
        title
        content
        url
        ups
        downs
        relatedPosts {   // 5 Lambda invocations - each yields 5 posts
            id
            title
        }
    }
}
```

Meskipun ini mungkin tidak terdengar substansif dalam contoh spesifik ini, pengambilan berlebihan yang diperparah ini dapat dengan cepat merusak aplikasi.

Jika Anda mengambilrelatedPostslagi pada yang dikembalikan terkaitPostsdalam kueri yang sama, jumlah pemanggilan akan meningkat secara dramatis.

```
query {
    allPosts {   // 1 Lambda invocation - yields 5 Posts
        id
        author
        title
        content
        url
        ups
        downs
        relatedPosts {   // 5 Lambda invocations - each yield 5 posts = 5 x 5 Posts
            id
```

```
            title
            relatedPosts {  // 5 x 5 Lambda invocations - each yield 5 posts = 25 x 5
 Posts
                id
                title
                author
            }
        }
    }
}
```

Dalam pertanyaan yang relatif sederhana ini,AWS AppSyncakan memanggil fungsi Lambda 1 + 5 + 25 = 31 kali.

Ini adalah tantangan yang cukup umum dan sering disebut masalah N+1 (dalam hal ini, N = 5), dan dapat menimbulkan peningkatan latensi dan biaya untuk aplikasi.

Salah satu pendekatan untuk memecahkan masalah ini adalah dengan mengumpulkan permintaan penyelesai bidang yang serupa bersama-sama. Dalam contoh ini, alih-alih memiliki fungsi Lambda menyelesaikan daftar posting terkait untuk satu posting tertentu, itu malah bisa menyelesaikan daftar posting terkait untuk kumpulan posting tertentu.

Untuk mendemonstrasikan ini, mari perbarui resolver untuk`relatedPosts`untuk menangani batching.

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const {source, args} = ctx
  return {
    operation: ctx.info.fieldName === 'relatedPosts' ? 'BatchInvoke' : 'Invoke',
    payload: { field: ctx.info.fieldName, arguments: args, source },
  };
}

export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type, result);
  }
  return result;
}
```

Kode sekarang mengubah operasi dari`Invoke`kepada`BatchInvoke`ketika`fieldName`diselesaikan adalah`relatedPosts`. Sekarang, aktifkan batching pada fungsi di**Konfigurasikan Batching**bagian. Atur ukuran batching maksimum yang disetel ke`5`. Pilih **Save (Simpan)**.

Dengan perubahan ini, saat menyelesaikan`relatedPosts`, fungsi Lambda menerima yang berikut sebagai input:

```
[
    {
        "field": "relatedPosts",
        "source": {
            "id": 1
        }
    },
    {
        "field": "relatedPosts",
        "source": {
            "id": 2
        }
    },
    ...
]
```

Kapan`BatchInvoke`ditentukan dalam permintaan, fungsi Lambda menerima daftar permintaan dan mengembalikan daftar hasil.

Secara khusus, daftar hasil harus sesuai dengan ukuran dan urutan entri payload permintaan sehingga**AWS AppSync**dapat mencocokkan hasil yang sesuai.

Dalam contoh batching ini, fungsi Lambda mengembalikan sekumpulan hasil sebagai berikut:

```
[
    [{"id":"2","title":"Second book"}, {"id":"3","title":"Third book"}],   //
 relatedPosts for id=1
    [{"id":"3","title":"Third book"}]                                      //
 relatedPosts for id=2
]
```

Anda dapat memperbarui kode Lambda Anda untuk menangani batching`relatedPosts`:

```
export const handler = async (event) => {
  console.log('Received event {}', JSON.stringify(event, 3))
```

```
  //throw new Error('I fail. always')

  const posts = {
    1: { id: '1', title: 'First book', author: 'Author1', url: 'https://amazon.com/',
 content: 'SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT
 AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1', ups: '100', downs: '10', },
    2: { id: '2', title: 'Second book', author: 'Author2', url: 'https://amazon.com',
 content: 'SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT', ups: '100', downs:
 '10', },
    3: { id: '3', title: 'Third book', author: 'Author3', url: null, content: null,
 ups: null, downs: null },
    4: { id: '4', title: 'Fourth book', author: 'Author4', url: 'https://
www.amazon.com/', content: 'SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4', ups: '1000', downs: '0', },
    5: { id: '5', title: 'Fifth book', author: 'Author5', url: 'https://
www.amazon.com/', content: 'SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT
 AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT', ups: '50', downs: '0', },
  }

  const relatedPosts = {
    1: [posts['4']],
    2: [posts['3'], posts['5']],
    3: [posts['2'], posts['1']],
    4: [posts['2'], posts['1']],
    5: [],
  }

  if (!event.field && event.length){
    console.log(`Got a BatchInvoke Request. The payload has ${event.length} items to
 resolve.`);
    return event.map(e => relatedPosts[e.source.id])
  }

  console.log('Got an Invoke Request.')
  let result
  switch (event.field) {
    case 'getPost':
      return posts[event.arguments.id]
    case 'allPosts':
      return Object.values(posts)
    case 'addPost':
      // return the arguments back
      return event.arguments
```

```
    case 'addPostErrorWithData':
      result = posts[event.arguments.id]
      // attached additional error information to the post
      result.errorMessage = 'Error with the mutation, data has changed'
      result.errorType = 'MUTATION_ERROR'
      return result
    case 'relatedPosts':
      return relatedPosts[event.source.id]
    default:
      throw new Error('Unknown field, unable to resolve ' + event.field)
  }
}
```

Mengembalikan kesalahan individu

Contoh sebelumnya menunjukkan bahwa Anda dapat mengembalikan satu kesalahan dari fungsi Lambda atau memunculkan kesalahan dari penangan respons Anda. Untuk pemanggilan batch, memunculkan kesalahan dari fungsi Lambda menandai seluruh batch sebagai gagal. Ini mungkin dapat diterima untuk skenario tertentu di mana terjadi kesalahan yang tidak dapat dipulihkan, seperti koneksi yang gagal ke penyimpanan data. Namun, dalam kasus di mana beberapa item dalam batch berhasil dan yang lainnya gagal, dimungkinkan untuk mengembalikan kesalahan dan data yang valid. KarenaAWS AppSyncmemerlukan respons batch untuk elemen daftar yang cocok dengan ukuran asli batch, Anda harus menentukan struktur data yang dapat membedakan data yang valid dari kesalahan.

Misalnya, jika fungsi Lambda diharapkan mengembalikan sekumpulan posting terkait, Anda dapat memilih untuk mengembalikan daftar`Response`objek di mana setiap objek memiliki opsionaldata,ErrorMessage, danErrorTypeladang. JikaErrorMessagebidang hadir, itu berarti bahwa kesalahan terjadi.

Kode berikut menunjukkan bagaimana Anda dapat memperbarui fungsi Lambda:

```
export const handler = async (event) => {
console.log('Received event {}', JSON.stringify(event, 3))
  // throw new Error('I fail. always')
const posts = {
1: { id: '1', title: 'First book', author: 'Author1', url: 'https://amazon.com/',
 content: 'SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT
 AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1', ups: '100', downs: '10', },
```

```
    2: { id: '2', title: 'Second book', author: 'Author2', url: 'https://amazon.com',
 content: 'SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT', ups: '100', downs:
 '10', },
    3: { id: '3', title: 'Third book', author: 'Author3', url: null, content: null,
 ups: null, downs: null },
    4: { id: '4', title: 'Fourth book', author: 'Author4', url: 'https://
www.amazon.com/', content: 'SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4', ups: '1000', downs: '0', },
    5: { id: '5', title: 'Fifth book', author: 'Author5', url: 'https://
www.amazon.com/', content: 'SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT
 AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT', ups: '50', downs: '0', },
  }

  const relatedPosts = {
1: [posts['4']],
    2: [posts['3'], posts['5']],
    3: [posts['2'], posts['1']],
    4: [posts['2'], posts['1']],
    5: [],
  }

  if (!event.field && event.length){
console.log(`Got a BatchInvoke Request. The payload has ${event.length} items to
 resolve.`);
    return event.map(e => {
// return an error for post 2
if (e.source.id === '2') {
return { 'data': null, 'errorMessage': 'Error Happened', 'errorType': 'ERROR' }
      }
      return {data: relatedPosts[e.source.id]}
      })
  }

  console.log('Got an Invoke Request.')
  let result
  switch (event.field) {
case 'getPost':
      return posts[event.arguments.id]
    case 'allPosts':
      return Object.values(posts)
    case 'addPost':
      // return the arguments back
return event.arguments
```

```
    case 'addPostErrorWithData':
      result = posts[event.arguments.id]
      // attached additional error information to the post
      result.errorMessage = 'Error with the mutation, data has changed'
      result.errorType = 'MUTATION_ERROR'
return result
    case 'relatedPosts':
      return relatedPosts[event.source.id]
    default:
      throw new Error('Unknown field, unable to resolve ' + event.field)
  }
}
```

Perbarui`relatedPosts`kode penyelesai:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const {source, args} = ctx
  return {
    operation: ctx.info.fieldName === 'relatedPosts' ? 'BatchInvoke' : 'Invoke',
    payload: { field: ctx.info.fieldName, arguments: args, source },
  };
}

export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type, result);
  } else if (result.errorMessage) {
    util.appendError(result.errorMessage, result.errorType, result.data)
  } else if (ctx.info.fieldName === 'relatedPosts') {
      return result.data
  } else {
      return result
  }
}
```

Penangan respons sekarang memeriksa kesalahan yang dikembalikan oleh fungsi LambdaInvokeoperasi, memeriksa kesalahan yang dikembalikan untuk item individualBatchInvokeoperasi, dan akhirnya memeriksa`fieldName`. Untuk`relatedPosts`, fungsi

kembali`result.data`. Untuk semua bidang lainnya, fungsi hanya kembali`result`. Misalnya, lihat kueri di bawah ini:

```
query AllPosts {
  allPosts {
    id
    title
    content
    url
    ups
    downs
    relatedPosts {
      id
    }
    author
  }
}
```

Query ini mengembalikan respon GraphQL mirip dengan berikut:

```
{
  "data": {
    "allPosts": [
      {
        "id": "1",
        "relatedPosts": [
          {
            "id": "4"
          }
        ]
      },
      {
        "id": "2",
        "relatedPosts": null
      },
      {
        "id": "3",
        "relatedPosts": [
          {
            "id": "2"
          },
          {
            "id": "1"
```

```
          }
        ]
      },
      {
        "id": "4",
        "relatedPosts": [
          {
            "id": "2"
          },
          {
            "id": "1"
          }
        ]
      },
      {
        "id": "5",
        "relatedPosts": []
      }
    ]
  },
  "errors": [
    {
      "path": [
        "allPosts",
        1,
        "relatedPosts"
      ],
      "data": null,
      "errorType": "ERROR",
      "errorInfo": null,
      "locations": [
        {
          "line": 4,
          "column": 5,
          "sourceName": null
        }
      ],
      "message": "Error Happened"
    }
  ]
}
```

Mengkonfigurasi ukuran batching maksimum

Untuk mengonfigurasi ukuran batching maksimum pada resolver, gunakan perintah berikut diAWS Command Line Interface(AWS CLI):

```
$ aws appsync create-resolver --api-id <api-id> --type-name Query --field-name
 relatedPosts \
 --code "<code-goes-here>" \
 --runtime name=APPSYNC_JS,runtimeVersion=1.0.0 \
 --data-source-name "<lambda-datasource>" \
 --max-batch-size X
```

> **Note**
>
> Saat menyediakan templat pemetaan permintaan, Anda harus menggunakan`BatchInvoke`operasi untuk menggunakan batching.

Tutorial: Penyelesai lokal

AWS AppSyncmemungkinkan Anda menggunakan sumber data yang didukung (AWS Lambda, Amazon DynamoDB, atau AmazonOpenSearchLayanan) untuk melakukan berbagai operasi. Namun, dalam skenario tertentu, panggilan ke sumber data yang didukung mungkin tidak diperlukan.

Di sinilah resolver lokal berguna. Alih-alih memanggil sumber data jarak jauh, resolver lokal hanya akanmajuhasil dari penangan permintaan ke handler respons. Resolusi lapangan tidak akan meninggalkanAWS AppSync.

Resolver lokal berguna dalam banyak situasi. Kasus penggunaan yang paling populer adalah mempublikasikan notifikasi tanpa memicu panggilan sumber data. Untuk mendemonstrasikan kasus penggunaan ini, mari buat aplikasi pub/sub tempat pengguna dapat mempublikasikan dan berlangganan pesan. Contoh ini memanfaatkanLangganan, jadi jika Anda tidak terbiasa denganLangganan, Anda dapat mengikutiData Waktu Nyatatutorial.

Membuat aplikasi pub/sub

Pertama, buat API GraphQL kosong dengan memilihDesain dari awalopsi dan mengonfigurasi detail opsional saat membuat API GraphQL Anda.

Dalam aplikasi pub/sub kami, klien dapat berlangganan dan mempublikasikan pesan. Setiap pesan yang diterbitkan mencakup nama dan data. Tambahkan ini ke skema:

```
type Channel {
 name: String!
 data: AWSJSON!
}

type Mutation {
 publish(name: String!, data: AWSJSON!): Channel
}

type Query {
 getChannel: Channel
}

type Subscription {
 subscribe(name: String!): Channel
  @aws_subscribe(mutations: ["publish"])
}
```

Selanjutnya, mari kita lampirkan resolver ke`Mutation.publish`lapangan. Dalam`Penyelesai`panel di sebelah`Skema`panel, temukan`Mutation`ketik, lalu`publish(...): Channel`bidang, lalu klik`Lampirkan`.

Buat`Tidak ada`sumber data dan beri nama`PageDataSource`. Pasang ke resolver Anda.

Tambahkan implementasi resolver Anda menggunakan cuplikan berikut:

```
export function request(ctx) {
  return { payload: ctx.args };
}

export function response(ctx) {
  return ctx.result;
}
```

Pastikan Anda membuat resolver dan menyimpan perubahan yang Anda buat.

Kirim dan berlangganan pesan

Agar klien dapat menerima pesan, mereka harus terlebih dahulu berlangganan kotak masuk.

DalamPertanyaanpanel, jalankan`SubscribeToData`berlangganan:

```
subscription SubscribeToData {
    subscribe(name:"channel") {
        name
        data
    }
}
```

Pelanggan akan menerima pesan kapan pun`publish`mutasi dipanggil tetapi hanya ketika pesan dikirim ke`channel`berlangganan. Mari kita coba ini diPertanyaanpanel. Saat langganan Anda masih berjalan di konsol, buka konsol lain dan jalankan permintaan berikut diPertanyaanpanel:

> **Note**
>
> Kami menggunakan string JSON yang valid dalam contoh ini.

```
mutation PublishData {
    publish(data: "{\"msg\": \"hello world!\"}", name: "channel") {
        data
        name
    }
}
```

Hasilnya akan terlihat seperti ini:

```
{
  "data": {
    "publish": {
      "data": "{\"msg\":\"hello world!\"}",
      "name": "channel"
    }
  }
}
```

Kami baru saja mendemonstrasikan penggunaan resolver lokal, dengan menerbitkan pesan dan menerimanya tanpa meninggalkanAWS AppSynclayanan.

Tutorial: Menggabungkan resolver GraphQL

Resolver dan bidang dalam skema GraphQL memiliki hubungan 1:1 dengan tingkat fleksibilitas yang besar. Karena sumber data dikonfigurasi pada resolver secara independen dari skema, Anda memiliki kemampuan untuk menyelesaikan atau memanipulasi tipe GraphQL Anda melalui sumber data yang berbeda, memungkinkan Anda untuk mencampur dan mencocokkan skema untuk memenuhi kebutuhan Anda.

Skenario berikut menunjukkan cara mencampur dan mencocokkan sumber data dalam skema Anda. Sebelum Anda mulai, Anda harus terbiasa dengan mengonfigurasi sumber data dan resolver untukAWS Lambda, Amazon DynamoDB, dan AmazonOpenSearchLayanan.

Contoh skema

Skema berikut memiliki jenis`Post`dengan tiga`Query`dan`Mutation`operasi masing-masing:

```
type Post {
    id: ID!
    author: String!
    title: String
    content: String
    url: String
    ups: Int
    downs: Int
    version: Int!
}

type Query {
    allPost: [Post]
    getPost(id: ID!): Post
    searchPosts: [Post]
}

type Mutation {
    addPost(
        id: ID!,
        author: String!,
        title: String,
        content: String,
        url: String
    ): Post
```

```
    updatePost(
        id: ID!,
        author: String!,
        title: String,
        content: String,
        url: String,
        ups: Int!,
        downs: Int!,
        expectedVersion: Int!
    ): Post
    deletePost(id: ID!): Post
}
```

Dalam contoh ini, Anda akan memiliki total enam resolver dengan masing-masing membutuhkan sumber data. Salah satu cara untuk mengatasi masalah ini adalah dengan menghubungkannya ke satu tabel Amazon DynamoDB, yang disebut`Posts`, di mana`AllPost`bidang menjalankan pemindaian dan`searchPosts`bidang menjalankan kueri (lihatJavaScriptreferensi fungsi resolver untuk DynamoDB). Namun, Anda tidak terbatas pada Amazon DynamoDB; sumber data yang berbeda seperti Lambda atauOpenSearchLayanan ada untuk memenuhi kebutuhan bisnis Anda.

Mengubah data melalui resolver

Anda mungkin perlu mengembalikan hasil dari database pihak ketiga yang tidak didukung secara langsung olehAWS AppSyncsumber data. Anda mungkin juga harus melakukan modifikasi kompleks pada data sebelum dikembalikan ke klien API. Hal ini dapat disebabkan oleh pemformatan tipe data yang tidak tepat, seperti perbedaan stempel waktu pada klien, atau penanganan masalah kompatibilitas mundur. Dalam hal ini, menghubungkanAWS Lambdaberfungsi sebagai sumber data untuk AndaAWS AppSyncAPI adalah solusi yang tepat. Untuk tujuan ilustrasi, dalam contoh berikut, sebuahAWS Lambdafungsi memanipulasi data yang diambil dari penyimpanan data pihak ketiga:

```
export const handler = (event, context, callback) => {
    // fetch data
    const result = fetcher()

    // apply complex business logic
    const data = transform(result)

    // return to AppSync
    return data
};
```

Ini adalah fungsi Lambda yang sangat valid dan dapat dilampirkan ke`AllPost`bidang dalam skema GraphQL sehingga kueri apa pun yang mengembalikan semua hasil mendapat angka acak untuk naik/turun.

DynamoDB danOpenSearchLayanan

Untuk beberapa aplikasi, Anda mungkin melakukan mutasi atau kueri pencarian sederhana terhadap DynamoDB dan memiliki proses latar belakang mentransfer dokumen keOpenSearchLayanan. Anda cukup melampirkan`searchPosts`penyelesai keOpenSearchLayanan sumber data dan mengembalikan hasil pencarian (dari data yang berasal dari DynamoDB) menggunakan query GraphQL. Ini bisa sangat kuat saat menambahkan operasi pencarian lanjutan ke aplikasi Anda seperti kata kunci, kecocokan kata kabur, atau bahkan pencarian geospasial. Mentransfer data dari DynamoDB dapat dilakukan melalui proses ETL, atau sebagai alternatif, Anda dapat melakukan streaming dari DynamoDB menggunakan Lambda.

Untuk memulai dengan sumber data khusus ini, lihatDynamoDBdanLambdatutorial.

Misalnya, menggunakan skema dari tutorial kami sebelumnya, mutasi berikut menambahkan item ke DynamoDB:

```
mutation addPost {
  addPost(
    id: 123
    author: "Nadia"
    title: "Our first post!"
    content: "This is our first post."
    url: "https://aws.amazon.com/appsync/"
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

Ini menulis data ke DynamoDB, yang kemudian mengalirkan data melalui Lambda ke AmazonOpenSearchLayanan, yang kemudian Anda gunakan untuk mencari posting berdasarkan

bidang yang berbeda. Misalnya, karena datanya ada di AmazonOpenSearchLayanan, Anda dapat mencari bidang penulis atau konten dengan teks bentuk bebas, bahkan dengan spasi, sebagai berikut:

```
query searchName{
    searchAuthor(name:"   Nadia   "){
        id
        title
        content
    }
}

---------- or ----------

query searchContent{
    searchContent(text:"test"){
        id
        title
        content
    }
}
```

Karena data ditulis langsung ke DynamoDB, Anda masih dapat melakukan operasi pencarian daftar atau item yang efisien terhadap tabel dengan`allPost{...}`dan`getPost{...}`pertanyaan. Tumpukan ini menggunakan kode contoh berikut untuk aliran DynamoDB:

> **Note**
>
> Kode Python ini adalah contoh dan tidak dimaksudkan untuk digunakan dalam kode produksi.

```
import boto3
import requests
from requests_aws4auth import AWS4Auth

region = '' # e.g. us-east-1
service = 'es'
credentials = boto3.Session().get_credentials()
awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,
 session_token=credentials.token)
```

```
host = '' # the OpenSearch Service domain, e.g. https://search-mydomain.us-
west-1.es.amazonaws.com
index = 'lambda-index'
datatype = '_doc'
url = host + '/' + index + '/' + datatype + '/'

headers = { "Content-Type": "application/json" }

def handler(event, context):
    count = 0
    for record in event['Records']:
        # Get the primary key for use as the OpenSearch ID
        id = record['dynamodb']['Keys']['id']['S']

        if record['eventName'] == 'REMOVE':
            r = requests.delete(url + id, auth=awsauth)
        else:
            document = record['dynamodb']['NewImage']
            r = requests.put(url + id, auth=awsauth, json=document, headers=headers)
        count += 1
    return str(count) + ' records processed.'
```

Anda kemudian dapat menggunakan aliran DynamoDB untuk melampirkan ini ke tabel DynamoDB dengan kunci utama`id`, dan setiap perubahan pada sumber DynamoDB akan mengalir ke AndaOpenSearchDomain layanan. Untuk informasi selengkapnya tentang mengonfigurasi ini, lihatDokumentasi DynamoDB Streams.

Tutorial: AmazonOpenSearchPenyelesai Layanan

AWS AppSyncmendukung menggunakan AmazonOpenSearchLayanan dari domain yang telah Anda sediakan sendiriAWSakun, asalkan tidak ada di dalam VPC. Setelah domain Anda disediakan, Anda dapat menghubungkannya menggunakan sumber data, di mana Anda dapat mengonfigurasi resolver dalam skema untuk melakukan operasi GraphQL seperti kueri, mutasi, dan langganan. Tutorial ini akan membawa Anda melalui beberapa contoh umum.

Untuk informasi lebih lanjut, lihatJavaScriptreferensi fungsi resolver untukOpenSearch.

Buat yang baruOpenSearchDomain layanan

Untuk memulai tutorial ini, Anda memerlukan yang sudah adaOpenSearchDomain layanan. Jika Anda tidak memilikinya, Anda dapat menggunakan sampel berikut. Perhatikan bahwa ini bisa

memakan waktu hingga 15 menit untukOpenSearchDomain layanan yang akan dibuat sebelum Anda dapat melanjutkan untuk mengintegrasikannya denganAWS AppSyncsumber data.

```
aws cloudformation create-stack --stack-name AppSyncOpenSearch \
--template-url https://s3.us-west-2.amazonaws.com/awsappsync/resources/elasticsearch/
ESResolverCFTemplate.yaml \
--parameters ParameterKey=OSDomainName,ParameterValue=ddtestdomain
 ParameterKey=Tier,ParameterValue=development \
--capabilities CAPABILITY_NAMED_IAM
```

Anda dapat meluncurkan yang berikutAWS CloudFormationtumpukan di Wilayah AS-Barat-2 (Oregon) di wilayah AndaAWSakun:

Konfigurasikan sumber data untukOpenSearchLayanan

SetelahOpenSearchDomain layanan dibuat, navigasikan keAWS AppSyncGraphQL API dan pilihSumber Datatab. PilihBuat sumber datadan masukkan nama yang ramah untuk sumber data seperti"*oss*". Kemudian, pilihAmazonOpenSearchranahuntukJenis sumber data, pilih Wilayah yang sesuai, dan Anda akan melihatOpenSearchDomain layanan terdaftar. Setelah memilihnya, Anda dapat membuat peran baru, danAWS AppSyncakan menetapkan izin yang sesuai peran, atau Anda dapat memilih peran yang ada, yang memiliki kebijakan sebaris berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1234234",
            "Effect": "Allow",
            "Action": [
                "es:ESHttpDelete",
                "es:ESHttpHead",
                "es:ESHttpGet",
                "es:ESHttpPost",
                "es:ESHttpPut"
            ],
            "Resource": [
                "arn:aws:es:REGION:ACCOUNTNUMBER:domain/democluster/*"
            ]
```

```
        }
    ]
}
```

Anda juga harus membangun hubungan kepercayaan denganAWS AppSyncuntuk peran itu:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "appsync.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Selain itu,OpenSearchDomain layanan memiliki miliknya sendiriKebijakan Aksesyang dapat Anda modifikasi melalui AmazonOpenSearchKonsol layanan. Anda harus menambahkan kebijakan yang serupa dengan yang di bawah ini dengan tindakan dan sumber daya yang sesuai untukOpenSearchDomain layanan. Perhatikan bahwaKepala Sekolahakan menjadiAWS AppSyncperan sumber data, yang dapat ditemukan di konsol IAM jika Anda membiarkan konsol tersebut membuatnya.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::ACCOUNTNUMBER:role/service-role/
APPSYNC_DATASOURCE_ROLE"
        },
        "Action": [
            "es:ESHttpDelete",
            "es:ESHttpHead",
            "es:ESHttpGet",
            "es:ESHttpPost",
            "es:ESHttpPut"
        ],
```

```
        "Resource": "arn:aws:es:REGION:ACCOUNTNUMBER:domain/DOMAIN_NAME/*"
        }
    ]
}
```

Menghubungkan resolver

Sekarang sumber data terhubung keOpenSearchDomain layanan, Anda dapat menghubungkannya ke skema GraphQL Anda dengan resolver seperti yang ditunjukkan pada contoh berikut:

```
 type Query {
   getPost(id: ID!): Post
   allPosts: [Post]
 }

 type Mutation {
   addPost(id: ID!, author: String, title: String, url: String, ups: Int, downs: Int,
 content: String): AWSJSON
 }

type Post {
  id: ID!
  author: String
  title: String
  url: String
  ups: Int
  downs: Int
  content: String
}
```

Perhatikan bahwa ada yang ditentukan penggunaPostketik dengan bidangid. Dalam contoh berikut, kami berasumsi ada proses (yang dapat diotomatisasi) untuk memasukkan tipe ini ke dalamOpenSearchDomain layanan, yang akan dipetakan ke root jalur/post/_docdi manapostadalah indeks. Dari jalur root ini, Anda dapat melakukan pencarian dokumen individual, pencarian wildcard dengan/id/post*, atau pencarian multi-dokumen dengan jalur/post/_search. Misalnya, jika Anda memiliki tipe lain yang disebutUser, Anda dapat mengindeks dokumen di bawah indeks baru yang disebutuser, kemudian melakukan pencarian denganjalandari/user/_search.

DariSkemaeditor diAWS AppSynckonsol, memodifikasi sebelumnyaPostsskema untuk memasukkansearchPostspertanyaan:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  searchPosts: [Post]
}
```

Simpan skema. DalamPenyelesaipanel, temukan`searchPosts`dan pilihLampirkan. Pilih AndaOpenSearchSumber data layanan dan simpan resolver. Perbarui kode resolver Anda menggunakan cuplikan di bawah ini:

```
import { util } from '@aws-appsync/utils'

/**
 * Searches for documents by using an input term
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the request
 */
export function request(ctx) {
 return {
  operation: 'GET',
  path: `/post/_search`,
  params: { body: { from: 0, size: 50 } },
 }
}

/**
 * Returns the fetched items
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the result
 */
export function response(ctx) {
 if (ctx.error) {
  util.error(ctx.error.message, ctx.error.type)
 }
 return ctx.result.hits.hits.map((hit) => hit._source)
}
```

Ini mengasumsikan bahwa skema sebelumnya memiliki dokumen yang telah diindeks diOpenSearchLayanan di bawah`post`lapangan. Jika Anda menyusun data Anda secara berbeda, Anda harus memperbaruinya.

Memodifikasi pencarian Anda

Handler permintaan resolver sebelumnya melakukan kueri sederhana untuk semua catatan. Misalkan Anda ingin mencari oleh penulis tertentu. Selanjutnya, misalkan Anda ingin penulis itu menjadi argumen yang didefinisikan dalam kueri GraphQL Anda. DalamSkemaeditor dariAWS AppSynckonsol, tambahkan`allPostsByAuthor`pertanyaan:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  allPostsByAuthor(author: String!): [Post]
  searchPosts: [Post]
}
```

DalamPenyelesaipanel, temukan`allPostsByAuthor`dan pilihLampirkan. PilihOpenSearchLayanan sumber data dan gunakan kode berikut:

```
import { util } from '@aws-appsync/utils'

/**
 * Searches for documents by `author`
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the request
 */
export function request(ctx) {
 return {
  operation: 'GET',
  path: '/post/_search',
  params: {
   body: {
    from: 0,
    size: 50,
    query: { match: { author: ctx.args.author } },
   },
  },
 }
}

/**
 * Returns the fetched items
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the result
```

```
 */
export function response(ctx) {
 if (ctx.error) {
  util.error(ctx.error.message, ctx.error.type)
 }
 return ctx.result.hits.hits.map((hit) => hit._source)
}
```

Perhatikan bahwabodydiisi dengan kueri istilah untuk`author`bidang, yang dilewatkan dari klien sebagai argumen. Secara opsional, Anda dapat menggunakan informasi yang telah diisi sebelumnya, seperti teks standar.

Menambahkan data keOpenSearchLayanan

Anda mungkin ingin menambahkan data keOpenSearchDomain layanan sebagai hasil dari mutasi GraphQL. Ini adalah mekanisme yang kuat untuk pencarian dan tujuan lainnya. Karena Anda dapat menggunakan langganan GraphQL untukbuat data Anda secara real-time, ini dapat berfungsi sebagai mekanisme untuk memberi tahu klien tentang pembaruan data di AndaOpenSearchDomain layanan.

Kembali keSkemahalaman diAWS AppSynckonsol dan pilihLampirkanuntuk`addPost()`mutasi. PilihOpenSearchLayanan sumber data lagi dan gunakan kode berikut:

```
import { util } from '@aws-appsync/utils'

/**
 * Searches for documents by `author`
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the request
 */
export function request(ctx) {
 return {
  operation: 'PUT',
  path: `/post/_doc/${ctx.args.id}`,
  params: { body: ctx.args },
 }
}

/**
 * Returns the inserted post
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the result
```

```
 */
export function response(ctx) {
 if (ctx.error) {
  util.error(ctx.error.message, ctx.error.type)
 }
 return ctx.result
}
```

Seperti sebelumnya, ini adalah contoh bagaimana data Anda mungkin terstruktur. Jika Anda memiliki nama bidang atau indeks yang berbeda, Anda perlu memperbarui`path`dan`body`. Contoh ini juga menunjukkan cara menggunakan`context.arguments`, yang juga dapat ditulis sebagai`ctx.args`, di handler permintaan Anda.

Mengambil satu dokumen

Akhirnya, jika Anda ingin menggunakan`getPost(id:ID)`kueri dalam skema Anda untuk mengembalikan dokumen individual, temukan kueri ini diSkemaeditor dariAWS AppSynckonsol dan pilihLampirkan. PilihOpenSearchLayanan sumber data lagi dan gunakan kode berikut:

```
import { util } from '@aws-appsync/utils'

/**
 * Searches for documents by `author`
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the request
 */
export function request(ctx) {
 return {
  operation: 'GET',
  path: `/post/_doc/${ctx.args.id}`,
 }
}

/**
 * Returns the post
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the result
 */
export function response(ctx) {
 if (ctx.error) {
  util.error(ctx.error.message, ctx.error.type)
 }
```

```
 return ctx.result._source
}
```

Lakukan kueri dan mutasi

Anda sekarang harus dapat melakukan operasi GraphQL terhadapOpenSearchDomain layanan. Arahkan kePertanyaantab dariAWS AppSynckonsol dan tambahkan catatan baru:

```
mutation AddPost {
    addPost (
        id:"12345"
        author: "Fred"
        title: "My first book"
        content: "This will be fun to write!"
        url: "publisher website",
        ups: 100,
        downs:20
       )
}
```

Anda akan melihat hasil mutasi di sebelah kanan. Demikian pula, Anda sekarang dapat menjalankan`searchPosts`kueri terhadap AndaOpenSearchDomain layanan:

```
query search {
    searchPosts {
        id
        title
        author
        content
    }
}
```

Praktik terbaik

- OpenSearchLayanan harus untuk query data, bukan sebagai database utama Anda. Anda mungkin ingin menggunakanOpenSearchLayanan dalam hubungannya dengan Amazon DynamoDB sebagaimana diuraikan dalam[Menggabungkan Resolver GraphQL](#).
- Hanya memberikan akses ke domain Anda dengan mengizinkanAWS AppSyncperan layanan untuk mengakses cluster.

- Anda dapat memulai dari yang kecil dalam pengembangan, dengan cluster berbiaya terendah, dan kemudian pindah ke cluster yang lebih besar dengan ketersediaan tinggi (HA) saat Anda pindah ke produksi.

Tutorial: Penyelesai Transaksi DynamoDB

AWS AppSyncmendukung penggunaan operasi transaksi Amazon DynamoDB di satu atau beberapa tabel dalam satu Wilayah. Operasi yang didukung adalah`TransactGetItems`dan`TransactWriteItems`. Dengan menggunakan fitur-fitur ini diAWS AppSync, Anda dapat melakukan tugas-tugas seperti:

- Melewati daftar kunci dalam satu kueri dan mengembalikan hasil dari tabel
- Membaca catatan dari satu atau beberapa tabel dalam satu kueri
- Menulis catatan dalam transaksi ke satu atau lebih tabel dalamall-or-nothingjalan
- Menjalankan transaksi ketika beberapa kondisi terpenuhi

Izin

Seperti resolver lainnya, Anda perlu membuat sumber data diAWS AppSyncdan membuat peran atau menggunakan yang sudah ada. Karena operasi transaksi memerlukan izin yang berbeda pada tabel DynamoDB, Anda perlu memberikan izin peran yang dikonfigurasi untuk tindakan baca atau tulis:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:region:accountId:table/TABLENAME",
                "arn:aws:dynamodb:region:accountId:table/TABLENAME/*"
```

```
            ]
        }
    ]
}
```

> **Note**
>
> Peran terkait dengan sumber data diAWS AppSync, dan resolver pada bidang dipanggil terhadap sumber data. Sumber data yang dikonfigurasi untuk diambil terhadap DynamoDB hanya memiliki satu tabel yang ditentukan untuk menjaga konfigurasi tetap sederhana. Oleh karena itu, saat melakukan operasi transaksi terhadap beberapa tabel dalam satu resolver, yang merupakan tugas yang lebih maju, Anda harus memberikan peran pada akses sumber data tersebut ke tabel mana pun yang akan berinteraksi dengan resolver. Ini akan dilakukan diSumber Dayabidang dalam kebijakan IAM di atas. Konfigurasi panggilan transaksi terhadap tabel dilakukan dalam kode resolver, yang kami jelaskan di bawah ini.

Sumber data

Demi kesederhanaan, kita akan menggunakan sumber data yang sama untuk semua resolver yang digunakan dalam tutorial ini.

Kami akan memiliki dua tabel yang disebutMenyimpanAkundanMemeriksa Akun, keduanya dengan`accountNumber`sebagai kunci partisi, danTransactionHistorymeja dengan`transactionId`sebagai kunci partisi. Anda dapat menggunakan perintah CLI di bawah ini untuk membuat tabel Anda. Pastikan untuk mengganti`region`dengan wilayah Anda.

Dengan CLI

```
aws dynamodb create-table --table-name savingAccounts \
  --attribute-definitions AttributeName=accountNumber,AttributeType=S \
  --key-schema AttributeName=accountNumber,KeyType=HASH \
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
  --table-class STANDARD --region region

aws dynamodb create-table --table-name checkingAccounts \
  --attribute-definitions AttributeName=accountNumber,AttributeType=S \
  --key-schema AttributeName=accountNumber,KeyType=HASH \
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
  --table-class STANDARD --region region
```

```
aws dynamodb create-table --table-name transactionHistory \
  --attribute-definitions AttributeName=transactionId,AttributeType=S \
  --key-schema AttributeName=transactionId,KeyType=HASH \
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
  --table-class STANDARD --region region
```

DiAWS AppSynckonsol, diSumber data, buat sumber data DynamoDB baru dan beri namaTransactTutorial. PilihMenyimpanAkunsebagai tabel (meskipun tabel tertentu tidak masalah saat menggunakan transaksi). Pilih untuk membuat peran baru dan sumber data. Anda dapat meninjau konfigurasi sumber data untuk melihat nama peran yang dihasilkan. Di konsol IAM, Anda dapat menambahkan kebijakan in-line yang memungkinkan sumber data berinteraksi dengan semua tabel.

Ganti`region`dan`accountID`dengan Region dan ID akun Anda:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:region:accountId:table/savingAccounts",
                "arn:aws:dynamodb:region:accountId:table/savingAccounts/*",
                "arn:aws:dynamodb:region:accountId:table/checkingAccounts",
                "arn:aws:dynamodb:region:accountId:table/checkingAccounts/*",
                "arn:aws:dynamodb:region:accountId:table/transactionHistory",
                "arn:aws:dynamodb:region:accountId:table/transactionHistory/*"
            ]
        }
    ]
}
```

Transaksi

Untuk contoh ini, konteksnya adalah transaksi perbankan klasik, di mana kita akan menggunakan`TransactWriteItems`ke:

- Transfer uang dari rekening tabungan ke rekening giro
- Menghasilkan catatan transaksi baru untuk setiap transaksi

Dan kemudian kita akan menggunakan`TransactGetItems`untuk mengambil detail dari rekening tabungan dan rekening giro.

Kami mendefinisikan skema GraphQL kami sebagai berikut:

```
type SavingAccount {
    accountNumber: String!
    username: String
    balance: Float
}

type CheckingAccount {
    accountNumber: String!
    username: String
    balance: Float
}

type TransactionHistory {
    transactionId: ID!
    from: String
    to: String
    amount: Float
}

type TransactionResult {
    savingAccounts: [SavingAccount]
    checkingAccounts: [CheckingAccount]
    transactionHistory: [TransactionHistory]
}

input SavingAccountInput {
    accountNumber: String!
    username: String
    balance: Float
```

```
}

input CheckingAccountInput {
    accountNumber: String!
    username: String
    balance: Float
}

input TransactionInput {
    savingAccountNumber: String!
    checkingAccountNumber: String!
    amount: Float!
}

type Query {
    getAccounts(savingAccountNumbers: [String], checkingAccountNumbers: [String]):
 TransactionResult
}

type Mutation {
    populateAccounts(savingAccounts: [SavingAccountInput], checkingAccounts:
 [CheckingAccountInput]): TransactionResult
    transferMoney(transactions: [TransactionInput]): TransactionResult
}
```

TransactWriteItems- Mengisi akun

Untuk mentransfer uang antar akun, kita perlu mengisi tabel dengan detailnya. Kami akan menggunakan operasi GraphQL`Mutation.populateAccounts`untuk melakukannya.

Di bagian Skema, klikLampirkandi sebelah`Mutation.populateAccounts`operasi. Pilih`TransactTutorial`sumber data dan pilihBuat.

Sekarang gunakan kode berikut:

```
import { util } from '@aws-appsync/utils'

export function request(ctx) {
 const { savingAccounts, checkingAccounts } = ctx.args

 const savings = savingAccounts.map(({ accountNumber, ...rest }) => {
  return {
   table: 'savingAccounts',
```

```
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({ accountNumber }),
    attributeValues: util.dynamodb.toMapValues(rest),
   }
 })

 const checkings = checkingAccounts.map(({ accountNumber, ...rest }) => {
  return {
   table: 'checkingAccounts',
   operation: 'PutItem',
   key: util.dynamodb.toMapValues({ accountNumber }),
   attributeValues: util.dynamodb.toMapValues(rest),
  }
 })
 return {
  version: '2018-05-29',
  operation: 'TransactWriteItems',
  transactItems: [...savings, ...checkings],
 }
}

export function response(ctx) {
 if (ctx.error) {
  util.error(ctx.error.message, ctx.error.type, null, ctx.result.cancellationReasons)
 }
 const { savingAccounts: sInput, checkingAccounts: cInput } = ctx.args
 const keys = ctx.result.keys
 const savingAccounts = sInput.map((_, i) => keys[i])
 const sLength = sInput.length
 const checkingAccounts = cInput.map((_, i) => keys[sLength + i])
 return { savingAccounts, checkingAccounts }
}
```

Simpan resolver dan navigasikan kePertanyaanbagian dariAWS AppSynckonsol untuk mengisi akun.

Jalankan mutasi berikut:

```
mutation populateAccounts {
  populateAccounts (
    savingAccounts: [
      {accountNumber: "1", username: "Tom", balance: 100},
      {accountNumber: "2", username: "Amy", balance: 90},
      {accountNumber: "3", username: "Lily", balance: 80},
    ]
```

```
    checkingAccounts: [
      {accountNumber: "1", username: "Tom", balance: 70},
      {accountNumber: "2", username: "Amy", balance: 60},
      {accountNumber: "3", username: "Lily", balance: 50},
    ]) {
    savingAccounts {
      accountNumber
    }
    checkingAccounts {
      accountNumber
    }
  }
}
```

Kami mengisi tiga rekening tabungan dan tiga rekening giro dalam satu mutasi.

Gunakan konsol DynamoDB untuk memvalidasi bahwa data muncul di keduaMenyimpanAkundanMemeriksa Akuntabel.

TransactWriteItems- Transfer uang

Pasang resolver ke`transferMoney`mutasi dengan kode berikut. Untuk setiap transfer, kami membutuhkan pengubah sukses untuk rekening giro dan tabungan, dan kami perlu melacak transfer dalam transaksi.

```
import { util } from '@aws-appsync/utils'

export function request(ctx) {
 const transactions = ctx.args.transactions

 const savings = []
 const checkings = []
 const history = []
 transactions.forEach((t) => {
  const { savingAccountNumber, checkingAccountNumber, amount } = t
  savings.push({
   table: 'savingAccounts',
   operation: 'UpdateItem',
   key: util.dynamodb.toMapValues({ accountNumber: savingAccountNumber }),
   update: {
    expression: 'SET balance = balance - :amount',
    expressionValues: util.dynamodb.toMapValues({ ':amount': amount }),
   },
```

```
  })
  checkings.push({
   table: 'checkingAccounts',
   operation: 'UpdateItem',
   key: util.dynamodb.toMapValues({ accountNumber: checkingAccountNumber }),
   update: {
    expression: 'SET balance = balance + :amount',
    expressionValues: util.dynamodb.toMapValues({ ':amount': amount }),
   },
  })
  history.push({
   table: 'transactionHistory',
   operation: 'PutItem',
   key: util.dynamodb.toMapValues({ transactionId: util.autoId() }),
   attributeValues: util.dynamodb.toMapValues({
    from: savingAccountNumber,
    to: checkingAccountNumber,
    amount,
   }),
  })
 })

 return {
  version: '2018-05-29',
  operation: 'TransactWriteItems',
  transactItems: [...savings, ...checkings, ...history],
 }
}

export function response(ctx) {
 if (ctx.error) {
  util.error(ctx.error.message, ctx.error.type, null, ctx.result.cancellationReasons)
 }
 const tInput = ctx.args.transactions
 const tLength = tInput.length
 const keys = ctx.result.keys
 const savingAccounts = tInput.map((_, i) => keys[tLength * 0 + i])
 const checkingAccounts = tInput.map((_, i) => keys[tLength * 1 + i])
 const transactionHistory = tInput.map((_, i) => keys[tLength * 2 + i])
 return { savingAccounts, checkingAccounts, transactionHistory }
}
```

Sekarang, navigasikan kePertanyaanbagian dariAWS AppSynckonsol dan jalankanTransferUangmutasi sebagai berikut:

```
mutation write {
  transferMoney(
    transactions: [
      {savingAccountNumber: "1", checkingAccountNumber: "1", amount: 7.5},
      {savingAccountNumber: "2", checkingAccountNumber: "2", amount: 6.0},
      {savingAccountNumber: "3", checkingAccountNumber: "3", amount: 3.3}
    ]) {
    savingAccounts {
      accountNumber
    }
    checkingAccounts {
      accountNumber
    }
    transactionHistory {
      transactionId
    }
  }
}
```

Kami mengirim tiga transaksi perbankan dalam satu mutasi. Gunakan konsol DynamoDB untuk memvalidasi bahwa data muncul diMenyimpanAkun,Memeriksa Akun, danTransactionHistorytabel.

TransactGetItems- Ambil akun

Untuk mengambil detail dari tabungan dan rekening giro dalam satu permintaan transaksional, kami akan melampirkan resolver ke`Query.getAccounts`Operasi GraphQL pada skema kami. PilihLampirkan, pilih yang sama`TransactTutorial`sumber data dibuat di awal tutorial. Gunakan kode berikut:

```
import { util } from '@aws-appsync/utils'

export function request(ctx) {
 const { savingAccountNumbers, checkingAccountNumbers } = ctx.args

 const savings = savingAccountNumbers.map((accountNumber) => {
  return { table: 'savingAccounts', key: util.dynamodb.toMapValues({ accountNumber }) }
 })
 const checkings = checkingAccountNumbers.map((accountNumber) => {
```

```
  return { table: 'checkingAccounts', key:
 util.dynamodb.toMapValues({ accountNumber }) }
 })
 return {
  version: '2018-05-29',
  operation: 'TransactGetItems',
  transactItems: [...savings, ...checkings],
 }
}

export function response(ctx) {
 if (ctx.error) {
  util.error(ctx.error.message, ctx.error.type, null, ctx.result.cancellationReasons)
 }

 const { savingAccountNumbers: sInput, checkingAccountNumbers: cInput } = ctx.args
 const items = ctx.result.items
 const savingAccounts = sInput.map((_, i) => items[i])
 const sLength = sInput.length
 const checkingAccounts = cInput.map((_, i) => items[sLength + i])
 return { savingAccounts, checkingAccounts }
}
```

Simpan resolver dan navigasikan kePertanyaanbagian dariAWS AppSynckonsol. Untuk mengambil rekening tabungan dan giro, jalankan kueri berikut:

```
query getAccounts {
  getAccounts(
    savingAccountNumbers: ["1", "2", "3"],
    checkingAccountNumbers: ["1", "2"]
  ) {
    savingAccounts {
      accountNumber
      username
      balance
    }
    checkingAccounts {
      accountNumber
      username
      balance
    }
  }
}
```

Kami telah berhasil menunjukkan penggunaan transaksi DynamoDB menggunakanAWS AppSync.

Tutorial: Penyelesai batch DynamoDB

AWS AppSyncmendukung penggunaan operasi batch Amazon DynamoDB di satu atau beberapa tabel dalam satu Wilayah. Operasi yang didukung adalah`BatchGetItem`,`BatchPutItem`, dan`BatchDeleteItem`. Dengan menggunakan fitur-fitur ini diAWS AppSync, Anda dapat melakukan tugas-tugas seperti:

- Melewati daftar kunci dalam satu kueri dan mengembalikan hasil dari tabel
- Membaca catatan dari satu atau beberapa tabel dalam satu kueri
- Menulis catatan secara massal ke satu atau lebih tabel
- Menulis atau menghapus catatan secara kondisional dalam beberapa tabel yang mungkin memiliki hubungan

Operasi batch diAWS AppSyncmemiliki dua perbedaan utama dari operasi non-batch:

- Peran sumber data harus memiliki izin ke semua tabel yang akan diakses oleh resolver.
- Spesifikasi tabel untuk resolver adalah bagian dari objek permintaan.

Batch tabel tunggal

Untuk memulai, mari buat API GraphQL baru. DiAWS AppSynckonsol, pilihBuat API,API GraphQL, danDesain dari awal. Beri nama API Anda`BatchTutorial API`, pilihBerikutnya, dan padaTentukan sumber daya GraphQLlangkah, pilihBuat sumber daya GraphQL nantidan klikBerikutnya. Tinjau detail Anda dan buat API. Pergi keSkemahalaman dan tempel skema berikut, mencatat bahwa untuk kueri, kami akan meneruskan daftar ID:

```
type Post {
    id: ID!
    title: String
}

input PostInput {
    id: ID!
    title: String
}
```

```
type Query {
    batchGet(ids: [ID]): [Post]
}

type Mutation {
    batchAdd(posts: [PostInput]): [Post]
    batchDelete(ids: [ID]): [Post]
}
```

Simpan skema Anda dan pilihBuat Sumber Dayadi bagian atas halaman. PilihGunakan tipe yang adadan pilih`Post`jenis. Beri nama meja Anda`Posts`. PastikanKunci Utamadiatur ke`id`, batalkan pilihanSecara otomatis menghasilkan GraphQL(Anda akan memberikan kode Anda sendiri), dan pilihBuat. Untuk memulai,AWS AppSyncmembuat tabel DynamoDB baru dan sumber data yang terhubung ke tabel dengan peran yang sesuai. Namun, masih ada beberapa izin yang perlu Anda tambahkan ke peran tersebut. Pergi keSumber datahalaman dan pilih sumber data baru. Di bawahPilih peran yang ada, Anda akan melihat bahwa peran secara otomatis dibuat untuk tabel. Perhatikan perannya (harus terlihat seperti`appsync-ds-ddb-aaabbbcccddd-Posts`) dan kemudian pergi ke konsol IAM (https://console.aws.amazon.com/iam/). Di konsol IAM, pilihPeran, lalu pilih peran Anda dari tabel. Dalam peran Anda, di bawahKebijakan izin, klik pada”+“di sebelah kebijakan (harus memiliki nama yang mirip dengan nama peran). PilihSuntingdi bagian atas yang dapat dilipat saat kebijakan muncul. Anda perlu menambahkan izin batch ke kebijakan Anda, khususnya`dynamodb:BatchGetItem`dan`dynamodb:BatchWriteItem`. Ini akan terlihat seperti ini:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:UpdateItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:BatchGetItem"
            ],
            "Resource": [
```

```
                "arn:aws:dynamodb:…",
                "arn:aws:dynamodb:…"
            ]
        }
    ]
}
```

PilihBerikutnya, makaSimpan perubahan. Kebijakan Anda harus mengizinkan pemrosesan batch sekarang.

Kembali diAWS AppSynckonsol, pergi keSkemahalaman dan pilihLampirkandi sebelah`Mutation.batchAdd`lapangan. Buat resolver Anda menggunakan`Posts`tabel sebagai sumber data. Di editor kode, ganti handler dengan cuplikan di bawah ini. Cuplikan ini secara otomatis mengambil setiap item di GraphQL`input PostInput`mengetik dan membangun peta, yang diperlukan untuk`BatchPutItem`operasi:

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  return {
    operation: "BatchPutItem",
    tables: {
      Posts: ctx.args.posts.map((post) => util.dynamodb.toMapValues(post)),
    },
  };
}

export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type);
  }
  return ctx.result.data.Posts;
}
```

Arahkan kePertanyaanhalaman dariAWS AppSynckonsol dan jalankan yang berikut`batchAdd`mutasi:

```
mutation add {
    batchAdd(posts:[{
            id: 1 title: "Running in the Park"},{
            id: 2 title: "Playing fetch"
        }]){
```

```
            id
            title
    }
}
```

Anda akan melihat hasil yang dicetak di layar; ini dapat divalidasi dengan meninjau konsol DynamoDB untuk memindai nilai yang ditulis ke`Posts`meja.

Selanjutnya, ulangi proses melampirkan resolver tetapi untuk`Query.batchGet`bidang menggunakan`Posts`tabel sebagai sumber data. Ganti handler dengan kode di bawah ini. Ini secara otomatis mengambil setiap item di GraphQL`ids:[]`ketik dan buat peta yang diperlukan untuk`BatchGetItem`operasi:

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  return {
    operation: "BatchGetItem",
    tables: {
      Posts: {
        keys: ctx.args.ids.map((id) => util.dynamodb.toMapValues({ id })),
        consistentRead: true,
      },
    },
  };
}

export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type);
  }
  return ctx.result.data.Posts;
}
```

Sekarang, kembali kePertanyaanhalaman dariAWS AppSynckonsol dan jalankan yang berikut`batchGet`pertanyaan:

```
query get {
    batchGet(ids:[1,2,3]){
        id
        title
    }
```

```
}
```

Ini akan mengembalikan hasil untuk keduanya`id`nilai yang Anda tambahkan sebelumnya. Perhatikan bahwa an`null`nilai dikembalikan untuk`id`dengan nilai`3`. Ini karena tidak ada catatan di`Posts`tabel dengan nilai itu belum. Perhatikan juga bahwaAWS AppSyncmengembalikan hasil dalam urutan yang sama dengan kunci yang diteruskan ke kueri, yang merupakan fitur tambahan yangAWS AppSyncmelakukan atas nama Anda. Jadi, jika Anda beralih ke`batchGet(ids:[1,3,2])`, Anda akan melihat bahwa pesanan berubah. Anda juga akan tahu yang mana`id`mengembalikan`null`nilai.

Akhirnya, pasang satu resolver lagi ke`Mutation.batchDelete`bidang menggunakan`Posts`tabel sebagai sumber data. Ganti handler dengan kode di bawah ini. Ini secara otomatis mengambil setiap item di GraphQL`ids:[]`ketik dan buat peta yang diperlukan untuk`BatchGetItem`operasi:

```
import { util } from "@aws-appsync/utils";

export function request(ctx) {
  return {
    operation: "BatchDeleteItem",
    tables: {
      Posts: ctx.args.ids.map((id) => util.dynamodb.toMapValues({ id })),
    },
  };
}

export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type);
  }
  return ctx.result.data.Posts;
}
```

Sekarang, kembali kePertanyaanhalaman dariAWS AppSynckonsol dan jalankan yang berikut`batchDelete`mutasi:

```
mutation delete {
    batchDelete(ids:[1,2]){ id }
}
```

Catatan dengan`id` `1`dan`2`sekarang harus dihapus. Jika Anda menjalankan kembali`batchGet()`kueri dari sebelumnya, ini harus kembali`null`.

Batch multi-tabel

AWS AppSyncjuga memungkinkan Anda untuk melakukan operasi batch di seluruh tabel. Mari kita membangun aplikasi yang lebih kompleks. Bayangkan kita sedang membangun aplikasi kesehatan hewan peliharaan di mana sensor melaporkan lokasi dan suhu tubuh hewan peliharaan. Sensor bertenaga baterai dan mencoba terhubung ke jaringan setiap beberapa menit. Ketika sensor membuat koneksi, ia mengirimkan bacaannya ke kamiAWS AppSyncAPI. Pemicu kemudian menganalisis data sehingga dasbor dapat disajikan kepada pemilik hewan peliharaan. Mari kita fokus pada mewakili interaksi antara sensor dan penyimpanan data backend.

DiAWS AppSynckonsol, pilihBuat API,API GraphQL, danDesain dari awal. Beri nama API Anda`MultiBatchTutorial API`, pilihBerikutnya, dan padaTentukan sumber daya GraphQLlangkah, pilihBuat sumber daya GraphQL nantidan klikBerikutnya. Tinjau detail Anda dan buat API. Pergi keSkemahalaman dan tempel dan simpan skema berikut:

```
type Mutation {
    # Register a batch of readings
    recordReadings(tempReadings: [TemperatureReadingInput], locReadings:
 [LocationReadingInput]): RecordResult
    # Delete a batch of readings
    deleteReadings(tempReadings: [TemperatureReadingInput], locReadings:
 [LocationReadingInput]): RecordResult
}

type Query {
    # Retrieve all possible readings recorded by a sensor at a specific time
    getReadings(sensorId: ID!, timestamp: String!): [SensorReading]
}

type RecordResult {
    temperatureReadings: [TemperatureReading]
    locationReadings: [LocationReading]
}

interface SensorReading {
    sensorId: ID!
    timestamp: String!
}

# Sensor reading representing the sensor temperature (in Fahrenheit)
type TemperatureReading implements SensorReading {
    sensorId: ID!
```

```
    timestamp: String!
    value: Float
}

# Sensor reading representing the sensor location (lat,long)
type LocationReading implements SensorReading {
    sensorId: ID!
    timestamp: String!
    lat: Float
    long: Float
}

input TemperatureReadingInput {
    sensorId: ID!
    timestamp: String
    value: Float
}

input LocationReadingInput {
    sensorId: ID!
    timestamp: String
    lat: Float
    long: Float
}
```

Kita perlu membuat dua tabel DynamoDB:

- `locationReadings`akan menyimpan pembacaan lokasi sensor.
- `temperatureReadings`akan menyimpan pembacaan suhu sensor.

Kedua tabel akan berbagi struktur kunci utama yang sama:`sensorId (String)`sebagai kunci partisi dan`timestamp (String)`sebagai kunci sortir.

PilihBuat Sumber Dayadi bagian atas halaman. PilihGunakan tipe yang adadan pilih`locationReadings`jenis. Beri nama meja Anda`locationReadings`. PastikanKunci Utamadiatur ke`sensorId`dan kunci semacam untuk`timestamp`. Batalkan pilihanSecara otomatis menghasilkan GraphQL(Anda akan memberikan kode Anda sendiri), dan pilihBuat. Ulangi proses ini untuk`temperatureReadings`menggunakan`temperatureReadings`sebagai jenis dan nama tabel. Gunakan tombol yang sama seperti di atas.

Tabel baru Anda akan berisi peran yang dihasilkan secara otomatis. Masih ada beberapa izin yang perlu Anda tambahkan ke peran tersebut. Pergi keSumber datahalaman dan pilih`locationReadings`. Di bawahPilih peran yang adaAnda bisa melihat perannya. Perhatikan perannya (harus terlihat seperti`appsync-ds-ddb-aaabbbcccddd-locationReadings`) dan kemudian pergi ke konsol IAM (https://console.aws.amazon.com/iam/). Di konsol IAM, pilihPeran, lalu pilih peran Anda dari tabel. Dalam peran Anda, di bawahKebijakan izin, klik pada"+"di sebelah kebijakan (harus memiliki nama yang mirip dengan nama peran). PilihSuntingdi bagian atas yang dapat dilipat saat kebijakan muncul. Anda perlu menambahkan izin ke kebijakan ini. Ini akan terlihat seperti ini:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:UpdateItem",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:region:account:table/locationReadings",
                "arn:aws:dynamodb:region:account:table/locationReadings/*",
                "arn:aws:dynamodb:region:account:table/temperatureReadings",
                "arn:aws:dynamodb:region:account:table/temperatureReadings/*"
            ]
        }
    ]
}
```

PilihBerikutnya, makaSimpan perubahan. Ulangi proses ini untuk`temperatureReadings`sumber data menggunakan cuplikan kebijakan yang sama di atas.

BatchPutItem- Merekam pembacaan sensor

Sensor kami harus dapat mengirim bacaan mereka setelah mereka terhubung ke internet. Bidang GraphQL`Mutation.recordReadings`adalah API yang akan mereka gunakan untuk melakukannya. Kita perlu menambahkan resolver ke bidang ini.

DiAWS AppSynckonsolSkemahalaman, pilihLampirkandi sebelah`Mutation.recordReadings`lapangan. Pada layar berikutnya, buat resolver Anda menggunakan`locationReadings`tabel sebagai sumber data.

Setelah membuat resolver Anda, ganti handler dengan kode berikut di editor. Ini`BatchPutItem`operasi memungkinkan kita untuk menentukan beberapa tabel:

```
import { util } from '@aws-appsync/utils'

export function request(ctx) {
 const { locReadings, tempReadings } = ctx.args
 const locationReadings = locReadings.map((loc) => util.dynamodb.toMapValues(loc))
 const temperatureReadings = tempReadings.map((tmp) => util.dynamodb.toMapValues(tmp))

 return {
  operation: 'BatchPutItem',
  tables: {
   locationReadings,
   temperatureReadings,
  },
 }
}

export function response(ctx) {
 if (ctx.error) {
  util.appendError(ctx.error.message, ctx.error.type)
 }
 return ctx.result.data
}
```

Dengan operasi batch, mungkin ada kesalahan dan hasil yang dikembalikan dari pemanggilan. Dalam hal ini, kami bebas melakukan beberapa penanganan kesalahan tambahan.

> **Note**
>
> Penggunaan`utils.appendError()`Mirip dengan`util.error()`, dengan perbedaan utama bahwa itu tidak mengganggu evaluasi permintaan atau penangan respons. Sebaliknya, ini menandakan ada kesalahan dengan bidang tetapi memungkinkan penangan untuk dievaluasi dan akibatnya mengembalikan data kembali ke pemanggil. Kami menyarankan Anda menggunakan`utils.appendError()`ketika aplikasi Anda perlu mengembalikan sebagian hasil.

Simpan resolver dan arahkan kePertanyaanhalaman diAWS AppSynckonsol. Kami sekarang dapat mengirim beberapa pembacaan sensor.

Jalankan mutasi berikut:

```
mutation sendReadings {
  recordReadings(
    tempReadings: [
      {sensorId: 1, value: 85.5, timestamp: "2018-02-01T17:21:05.000+08:00"},
      {sensorId: 1, value: 85.7, timestamp: "2018-02-01T17:21:06.000+08:00"},
      {sensorId: 1, value: 85.8, timestamp: "2018-02-01T17:21:07.000+08:00"},
      {sensorId: 1, value: 84.2, timestamp: "2018-02-01T17:21:08.000+08:00"},
      {sensorId: 1, value: 81.5, timestamp: "2018-02-01T17:21:09.000+08:00"}
    ]
    locReadings: [
      {sensorId: 1, lat: 47.615063, long: -122.333551, timestamp:
 "2018-02-01T17:21:05.000+08:00"},
      {sensorId: 1, lat: 47.615163, long: -122.333552, timestamp:
 "2018-02-01T17:21:06.000+08:00"},
      {sensorId: 1, lat: 47.615263, long: -122.333553, timestamp:
 "2018-02-01T17:21:07.000+08:00"},
      {sensorId: 1, lat: 47.615363, long: -122.333554, timestamp:
 "2018-02-01T17:21:08.000+08:00"},
      {sensorId: 1, lat: 47.615463, long: -122.333555, timestamp:
 "2018-02-01T17:21:09.000+08:00"}
    ]) {
    locationReadings {
      sensorId
      timestamp
      lat
      long
    }
```

```
    temperatureReadings {
      sensorId
      timestamp
      value
    }
  }
}
```

Kami mengirim sepuluh pembacaan sensor dalam satu mutasi dengan pembacaan dibagi menjadi dua tabel. Gunakan konsol DynamoDB untuk memvalidasi bahwa data muncul di kedua`locationReadings`dan`temperatureReadings`tabel.

BatchDeleteItem- Menghapus pembacaan sensor

Demikian pula, kita juga harus dapat menghapus batch pembacaan sensor. Mari kita gunakan`Mutation.deleteReadings`Bidang GraphQL untuk tujuan ini. DiAWS AppSynckonsolSkemahalaman, pilihLampirkandi sebelah`Mutation.deleteReadings`lapangan. Pada layar berikutnya, buat resolver Anda menggunakan`locationReadings`tabel sebagai sumber data.

Setelah membuat resolver Anda, ganti handler di editor kode dengan cuplikan di bawah ini. Dalam resolver ini, kami menggunakan pemeta fungsi pembantu yang mengekstrak`sensorId`dan`timestamp`dari input yang disediakan.

```
import { util } from '@aws-appsync/utils'

export function request(ctx) {
 const { locReadings, tempReadings } = ctx.args
 const mapper = ({ sensorId, timestamp }) => util.dynamodb.toMapValues({ sensorId,
 timestamp })

 return {
  operation: 'BatchDeleteItem',
  tables: {
   locationReadings: locReadings.map(mapper),
   temperatureReadings: tempReadings.map(mapper),
  },
 }
}

export function response(ctx) {
```

```
 if (ctx.error) {
  util.appendError(ctx.error.message, ctx.error.type)
 }
 return ctx.result.data
}
```

Simpan resolver dan arahkan kePertanyaanhalaman diAWS AppSynckonsol. Sekarang, mari kita hapus beberapa pembacaan sensor.

Jalankan mutasi berikut:

```
mutation deleteReadings {
  # Let's delete the first two readings we recorded
  deleteReadings(
    tempReadings: [{sensorId: 1, timestamp: "2018-02-01T17:21:05.000+08:00"}]
    locReadings: [{sensorId: 1, timestamp: "2018-02-01T17:21:05.000+08:00"}]) {
    locationReadings {
      sensorId
      timestamp
      lat
      long
    }
    temperatureReadings {
      sensorId
      timestamp
      value
    }
  }
}
```

> **Note**
>
> Bertentangan dengan`DeleteItem`operasi, item yang dihapus sepenuhnya tidak dikembalikan dalam respons. Hanya kunci yang dilewati yang dikembalikan. Untuk mempelajari lebih lanjut, lihatBatchDeleteItemdiJavaScriptreferensi fungsi resolver untuk DynamoDB.

Validasi melalui konsol DynamoDB bahwa kedua bacaan ini telah dihapus dari`locationReadings`dan`temperatureReadings`tabel.

BatchGetItem- Ambil bacaan

Operasi umum lainnya untuk aplikasi kami adalah mengambil pembacaan untuk sensor pada titik waktu tertentu. Mari kita lampirkan resolver ke`Query.getReadings`Bidang GraphQL pada skema kami. DiAWS AppSynckonsolSkemahalaman, pilihLampirkandi sebelah`Query.getReadings`lapangan. Pada layar berikutnya, buat resolver Anda menggunakan`locationReadings`tabel sebagai sumber data.

Mari kita gunakan kode berikut:

```
import { util } from '@aws-appsync/utils'

export function request(ctx) {
 const keys = [util.dynamodb.toMapValues(ctx.args)]
 const consistentRead = true
 return {
  operation: 'BatchGetItem',
  tables: {
   locationReadings: { keys, consistentRead },
   temperatureReadings: { keys, consistentRead },
  },
 }
}

export function response(ctx) {
 if (ctx.error) {
  util.appendError(ctx.error.message, ctx.error.type)
 }
 const { locationReadings: locs, temperatureReadings: temps } = ctx.result.data

 return [
  ...locs.map((l) => ({ ...l, __typename: 'LocationReading' })),
  ...temps.map((t) => ({ ...t, __typename: 'TemperatureReading' })),
 ]
}
```

Simpan resolver dan arahkan kePertanyaanhalaman diAWS AppSynckonsol. Sekarang, mari kita ambil pembacaan sensor kita.

Jalankan kueri berikut:

```
query getReadingsForSensorAndTime {
```

```
  # Let's retrieve the very first two readings
  getReadings(sensorId: 1, timestamp: "2018-02-01T17:21:06.000+08:00") {
    sensorId
    timestamp
    ...on TemperatureReading {
      value
    }
    ...on LocationReading {
      lat
      long
    }
  }
}
```

Kami telah berhasil mendemonstrasikan penggunaan operasi batch DynamoDB menggunakanAWS AppSync.

Penanganan kesalahan

DiAWS AppSync, operasi sumber data terkadang dapat mengembalikan sebagian hasil. Hasil sebagian adalah istilah yang akan kita gunakan untuk menunjukkan ketika output dari suatu operasi terdiri dari beberapa data dan kesalahan. Karena penanganan kesalahan secara inheren spesifik aplikasi,AWS AppSyncmemberi Anda kesempatan untuk menangani kesalahan dalam penangan respons. Kesalahan pemanggilan resolver, jika ada, tersedia dari konteks sebagai`ctx.error`. Kesalahan pemanggilan selalu menyertakan pesan dan tipe, dapat diakses sebagai properti`ctx.error.message`dan`ctx.error.type`. Dalam handler respon, Anda dapat menangani sebagian hasil dalam tiga cara:

1. Menelan kesalahan pemanggilan hanya dengan mengembalikan data.
2. Naikkan kesalahan (menggunakan`util.error(...)`) dengan menghentikan evaluasi handler, yang tidak akan mengembalikan data apa pun.
3. Tambahkan kesalahan (menggunakan`util.appendError(...)`) dan juga mengembalikan data.

Mari kita tunjukkan masing-masing dari tiga poin di atas dengan operasi batch DynamoDB.

Operasi Batch DynamoDB

Dengan operasi batch DynamoDB, ada kemungkinan bahwa batch sebagian selesai. Artinya, ada kemungkinan bahwa beberapa item atau kunci yang diminta dibiarkan tidak diproses. JikaAWS

AppSynctidak dapat menyelesaikan batch, item yang belum diproses dan kesalahan pemanggilan akan disetel pada konteksnya.

Kami akan menerapkan penanganan kesalahan menggunakan`Query.getReadings`konfigurasi bidang dari`BatchGetItem`operasi dari bagian sebelumnya dari tutorial ini. Kali ini, mari kita berpura-pura bahwa saat mengeksekusi`Query.getReadings`lapangan,`temperatureReadings`Tabel DynamoDB kehabisan throughput yang disediakan. DynamoDB mengangkat`ProvisionedThroughputExceededException`Dalam upaya kedua oleh AWS AppSyncuntuk memproses elemen yang tersisa dalam batch.

JSON berikut mewakili konteks serial setelah pemanggilan batch DynamoDB tetapi sebelum penangan respons dipanggil:

```
{
  "arguments": {
    "sensorId": "1",
    "timestamp": "2018-02-01T17:21:05.000+08:00"
  },
  "source": null,
  "result": {
    "data": {
      "temperatureReadings": [
        null
      ],
      "locationReadings": [
        {
          "lat": 47.615063,
          "long": -122.333551,
          "sensorId": "1",
          "timestamp": "2018-02-01T17:21:05.000+08:00"
        }
      ]
    },
    "unprocessedKeys": {
      "temperatureReadings": [
        {
          "sensorId": "1",
          "timestamp": "2018-02-01T17:21:05.000+08:00"
        }
      ],
      "locationReadings": []
    }
```

```
  },
  "error": {
    "type": "DynamoDB:ProvisionedThroughputExceededException",
    "message": "You exceeded your maximum allowed provisioned throughput for a table or
 for one or more global secondary indexes. (...)"
  },
  "outErrors": []
}
```

Beberapa hal yang perlu diperhatikan pada konteksnya:

- Kesalahan pemanggilan telah disetel pada konteks dic`tx.error`olehAWS AppSync, dan jenis kesalahan telah disetel ke`DynamoDB:ProvisionedThroughputExceededException`.
- Hasil dipetakan per tabel di bawah`ctx.result.data`meskipun ada kesalahan.
- Kunci yang dibiarkan tidak diproses tersedia dic`tx.result.data.unprocessedKeys`. Di sini,AWS AppSynctidak dapat mengambil item dengan kunci (sensorid:1, stempel waktu: 2018-02-01T 17:21:05.000 + 08:00) karena throughput tabel yang tidak mencukupi.

> **Note**
>
> Untuk`BatchPutItem`, itu`ctx.result.data.unprocessedItems`.
> Untuk`BatchDeleteItem`, itu`ctx.result.data.unprocessedKeys`.

Mari kita tangani kesalahan ini dengan tiga cara berbeda.

1. Menelan kesalahan pemanggilan

Mengembalikan data tanpa menangani kesalahan pemanggilan secara efektif menelan kesalahan, membuat hasil untuk bidang GraphQL yang diberikan selalu berhasil.

Kode yang kami tulis sudah tidak asing lagi dan hanya berfokus pada data hasil.

Penangan respons

```
export function response(ctx) {
  return ctx.result.data
}
```

Tanggapan GraphQL

```
{
  "data": {
    "getReadings": [
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00",
        "lat": 47.615063,
        "long": -122.333551
      },
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00",
        "value": 85.5
      }
    ]
  }
}
```

Tidak ada kesalahan yang akan ditambahkan ke respons kesalahan karena hanya data yang ditindaklanjuti.

2. Memunculkan kesalahan untuk membatalkan eksekusi handler respons

Ketika kegagalan sebagian harus diperlakukan sebagai kegagalan total dari perspektif klien, Anda dapat membatalkan eksekusi handler respons untuk mencegah pengembalian data. The`util.error(...)`metode utilitas mencapai persis perilaku ini.

Kode penangan respons

```
export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type, null,
 ctx.result.data.unprocessedKeys);
  }
  return ctx.result.data;
}
```

Tanggapan GraphQL

```
{
  "data": {
```

```
    "getReadings": null
  },
  "errors": [
    {
      "path": [
        "getReadings"
      ],
      "data": null,
      "errorType": "DynamoDB:ProvisionedThroughputExceededException",
      "errorInfo": {
        "temperatureReadings": [
          {
            "sensorId": "1",
            "timestamp": "2018-02-01T17:21:05.000+08:00"
          }
        ],
        "locationReadings": []
      },
      "locations": [
        {
          "line": 58,
          "column": 3
        }
      ],
      "message": "You exceeded your maximum allowed provisioned throughput for a table
 or for one or more global secondary indexes. (...)"
    }
  ]
}
```

Meskipun beberapa hasil mungkin telah dikembalikan dari operasi batch DynamoDB, kami memilih untuk memunculkan kesalahan sehinggagetReadingsBidang GraphQL adalah nol dan kesalahan telah ditambahkan ke respons GraphQLkesalahanblok.

3. Menambahkan kesalahan untuk mengembalikan data dan kesalahan

Dalam kasus tertentu, untuk memberikan pengalaman pengguna yang lebih baik, aplikasi dapat mengembalikan sebagian hasil dan memberi tahu klien mereka tentang item yang belum diproses. Klien dapat memutuskan untuk menerapkan coba lagi atau menerjemahkan kesalahan kembali ke pengguna akhir. Theutil.appendError(...)adalah metode utilitas yang memungkinkan perilaku ini dengan membiarkan perancang aplikasi menambahkan kesalahan pada konteks tanpa mengganggu evaluasi penangan respons. Setelah mengevaluasi handler respons,AWS

AppSyncakan memproses kesalahan konteks apa pun dengan menambahkannya ke blok kesalahan respons GraphQL.

Kode penangan respons

```
export function response(ctx) {
  if (ctx.error) {
    util.appendError(ctx.error.message, ctx.error.type, null,
 ctx.result.data.unprocessedKeys);
  }
  return ctx.result.data;
}
```

Kami meneruskan kesalahan pemanggilan dan`unprocessedKeys`elemen di dalam blok kesalahan respons GraphQL. The`getReadings`field juga mengembalikan sebagian data dari`locationReadings`tabel seperti yang Anda lihat pada tanggapan di bawah ini.

Tanggapan GraphQL

```
{
  "data": {
    "getReadings": [
      null,
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00",
        "value": 85.5
      }
    ]
  },
  "errors": [
    {
      "path": [
        "getReadings"
      ],
      "data": null,
      "errorType": "DynamoDB:ProvisionedThroughputExceededException",
      "errorInfo": {
        "temperatureReadings": [
          {
            "sensorId": "1",
            "timestamp": "2018-02-01T17:21:05.000+08:00"
```

```
          }
        ],
        "locationReadings": []
      },
      "locations": [
        {
          "line": 58,
          "column": 3
        }
      ],
      "message": "You exceeded your maximum allowed provisioned throughput for a table
 or for one or more global secondary indexes. (...)"
    }
  ]
}
```

Tutorial: HTTP resolver

AWS AppSyncmemungkinkan Anda untuk menggunakan sumber data yang didukung (yaitu,AWS Lambda, Amazon DynamoDB, AmazonOpenSearchLayanan, atau Amazon Aurora) untuk melakukan berbagai operasi, selain titik akhir HTTP arbitrer untuk menyelesaikan bidang GraphQL. Setelah titik akhir HTTP Anda tersedia, Anda dapat menghubungkannya menggunakan sumber data. Kemudian, Anda dapat mengonfigurasi resolver dalam skema untuk melakukan operasi GraphQL seperti kueri, mutasi, dan langganan. Tutorial ini memandu Anda melalui beberapa contoh umum.

Dalam tutorial ini Anda menggunakan REST API (dibuat menggunakan Amazon API Gateway dan Lambda) denganAWS AppSyncTitik akhir GraphQL.

Membuat REST API

Anda dapat menggunakan yang berikutAWS CloudFormationtemplate untuk mengatur titik akhir REST yang berfungsi untuk tutorial ini:

TheAWS CloudFormationstack melakukan langkah-langkah berikut:

1. Menyiapkan fungsi Lambda yang berisi logika bisnis Anda untuk layanan mikro Anda.
2. Menyiapkan API REST API Gateway dengan kombinasi titik akhir/metode/tipe konten berikut:

Jalur Sumber Daya API	Metode HTTP	Jenis Konten yang Didukung
/v1/pengguna	POST	aplikasi/json
/v1/pengguna	DAPATKAN	aplikasi/json
/v1/pengguna/1	DAPATKAN	aplikasi/json
/v1/pengguna/1	PUT	aplikasi/json
/v1/pengguna/1	HAPUS	aplikasi/json

Membuat API GraphQL

Untuk membuat API GraphQL diAWS AppSync:

1. BukaAWS AppSynckonsol dan pilihBuat API.
2. PilihAPI GraphQLdan kemudian memilihDesain dari awal. Pilih Selanjutnya.
3. Untuk nama API, ketik`UserData`. Pilih Selanjutnya.
4. Pilih `Create GraphQL resources later`. Pilih Selanjutnya.
5. Tinjau input Anda dan pilihBuat API.

TheAWS AppSynckonsol membuat API GraphQL baru untuk Anda menggunakan mode autentikasi kunci API. Anda dapat menggunakan konsol untuk mengonfigurasi API GraphQL Anda lebih lanjut dan menjalankan permintaan.

Membuat skema GraphQL

Sekarang setelah Anda memiliki API GraphQL, mari buat skema GraphQL. DalamSkemaeditor diAWS AppSynckonsol, gunakan cuplikan di bawah ini:

```
type Mutation {
    addUser(userInput: UserInput!): User
    deleteUser(id: ID!): User
}

type Query {
```

```
    getUser(id: ID): User
    listUser: [User!]!
}

type User {
    id: ID!
    username: String!
    firstname: String
    lastname: String
    phone: String
    email: String
}

input UserInput {
    id: ID!
    username: String!
    firstname: String
    lastname: String
    phone: String
    email: String
}
```

Konfigurasikan sumber data HTTP Anda

Untuk mengonfigurasi sumber data HTTP Anda, lakukan hal berikut:

1. DalamSumber datahalaman di AndaAWS AppSyncGraphQL API, pilihBuat sumber data.
2. Masukkan nama untuk sumber data sepertiHTTP_Example.
3. DiJenis sumber data, pilihTitik akhir HTTP.
4. Setel titik akhir ke titik akhir API Gateway yang dibuat di awal tutorial. Anda dapat menemukan titik akhir yang dihasilkan tumpukan jika Anda menavigasi ke konsol Lambda dan menemukan aplikasi Anda di bawahAplikasi. Di dalam pengaturan aplikasi Anda, Anda akan melihatTitik akhir APIyang akan menjadi titik akhir Anda diAWS AppSync. Pastikan Anda tidak menyertakan nama panggung sebagai bagian dari titik akhir. Misalnya, jika titik akhir Andahttps://aaabbbcccd.execute-api.us-east-1.amazonaws.com/v1, Anda akan mengetikhttps://aaabbbcccd.execute-api.us-east-1.amazonaws.com.

> **Note**
>
> Saat ini, hanya titik akhir publik yang didukung olehAWS AppSync.

Untuk informasi lebih lanjut tentang otoritas sertifikasi yang diakui olehAWS AppSynclayanan, lihatOtoritas Sertifikat (CA) Diakui olehAWS AppSyncuntuk Titik Akhir HTTPS.

Mengkonfigurasi resolver

Pada langkah ini, Anda akan menghubungkan sumber data HTTP ke`getUser`dan`addUser`pertanyaan.

Untuk mengatur`getUser`penyelesai:

1. Dalam AndaAWS AppSyncGraphQL API, pilihSkematab.
2. Di sebelah kananSkemaeditor, diPenyelesaipanel dan di bawahPermintaanketik, temukan`getUser`bidang dan pilihLampirkan.
3. Pertahankan jenis resolver ke`Unit`dan runtime untuk`APPSYNC_JS`.
4. DiNama sumber data, pilih titik akhir HTTP yang Anda buat sebelumnya.
5. Pilih Create (Buat).
6. DalamPenyelesaieditor kode, tambahkan cuplikan berikut sebagai penangan permintaan Anda:

```
import { util } from '@aws-appsync/utils'

export function request(ctx) {
 return {
  version: '2018-05-29',
  method: 'GET',
  params: {
   headers: {
    'Content-Type': 'application/json',
   },
  },
  resourcePath: `/v1/users/${ctx.args.id}`,
 }
}
```

7. Tambahkan cuplikan berikut sebagai handler respons Anda:

```
export function response(ctx) {
 const { statusCode, body } = ctx.result
 // if response is 200, return the response
 if (statusCode === 200) {
```

```
  return JSON.parse(body)
 }
 // if response is not 200, append the response to error block.
 util.appendError(body, statusCode)
}
```

8. PilihPermintaantab, dan kemudian jalankan query berikut:

```
query GetUser{
    getUser(id:1){
        id
        username
    }
}
```

Ini akan mengembalikan respons berikut:

```
{
    "data": {
        "getUser": {
            "id": "1",
            "username": "nadia"
        }
    }
}
```

Untuk mengaturaddUserpenyelesai:

1. PilihSkematab.
2. Di sebelah kananSkemaeditor, diPenyelesaipanel dan di bawahPermintaanketik, temukanaddUserbidang dan pilihLampirkan.
3. Pertahankan jenis resolver keUnitdan runtime untukAPPSYNC_JS.
4. DiNama sumber data, pilih titik akhir HTTP yang Anda buat sebelumnya.
5. Pilih Create (Buat).
6. DalamPenyelesaieditor kode, tambahkan cuplikan berikut sebagai penangan permintaan Anda:

```
export function request(ctx) {
    return {
        "version": "2018-05-29",
```

```
        "method": "POST",
        "resourcePath": "/v1/users",
        "params":{
            "headers":{
                "Content-Type": "application/json"
            },
        "body": ctx.args.userInput
        }
    }
}
```

7. Tambahkan cuplikan berikut sebagai handler respons Anda:

```
export function response(ctx) {
    if(ctx.error) {
        return util.error(ctx.error.message, ctx.error.type)
    }
    if (ctx.result.statusCode == 200) {
        return ctx.result.body
    } else {
        return util.appendError(ctx.result.body, "ctx.result.statusCode")
    }
}
```

8. PilihPermintaantab, dan kemudian jalankan query berikut:

```
mutation addUser{
    addUser(userInput:{
        id:"2",
        username:"shaggy"
    }){
        id
        username
    }
}
```

Jika Anda menjalankan`getUser`query lagi, itu harus mengembalikan respons berikut:

```
{
    "data": {
        "getUser": {
        "id": "2",
        "username": "shaggy"
```

```
        }
    }
}
```

MemohonAWSLayanan

Anda dapat menggunakan resolver HTTP untuk menyiapkan antarmuka API GraphQLAWSlayanan. Permintaan HTTP keAWSharus ditandatangani denganProses Signature Version 4sehinggaAWSdapat mengidentifikasi siapa yang mengirimnya.AWS AppSyncmenghitung tanda tangan atas nama Anda saat Anda mengaitkan peran IAM dengan sumber data HTTP.

Anda menyediakan dua komponen tambahan untuk dipanggilAWSlayanan dengan resolver HTTP:

- Peran IAM dengan izin untuk memanggilAWSAPI layanan
- Konfigurasi penandatanganan di sumber data

Misalnya, jika Anda ingin meneleponListGraphqlApisoperasidengan resolver HTTP, Anda terlebih dahulubuat peran IAMituAWS AppSyncmengasumsikan dengan kebijakan berikut terlampir:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "appsync:ListGraphqlApis"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Selanjutnya, buat sumber data HTTP untukAWS AppSync. Dalam contoh ini, Anda meneleponAWS AppSyncdi Wilayah Barat AS (Oregon). Siapkan konfigurasi HTTP berikut dalam file bernamahttp.json, yang mencakup wilayah penandatanganan dan nama layanan:

```
{
    "endpoint": "https://appsync.us-west-2.amazonaws.com/",
    "authorizationConfig": {
```

```
        "authorizationType": "AWS_IAM",
        "awsIamConfig": {
            "signingRegion": "us-west-2",
            "signingServiceName": "appsync"
        }
    }
}
```

Kemudian, gunakanAWS CLIuntuk membuat sumber data dengan peran terkait sebagai berikut:

```
aws appsync create-data-source --api-id <API-ID> \
                               --name AWSAppSync \
                               --type HTTP \
                               --http-config file:///http.json \
                               --service-role-arn <ROLE-ARN>
```

Saat Anda melampirkan resolver ke bidang dalam skema, gunakan template pemetaan permintaan berikut untuk memanggilAWS AppSync:

```
{
    "version": "2018-05-29",
    "method": "GET",
    "resourcePath": "/v1/apis"
}
```

Saat Anda menjalankan kueri GraphQL untuk sumber data ini,AWS AppSyncmenandatangani permintaan menggunakan peran yang Anda berikan dan menyertakan tanda tangan dalam permintaan. Query mengembalikan daftarAWS AppSyncAPI GraphQL di akun Anda di dalamnyaAWSWilayah.

Tutorial: Aurora PostgreSQL dengan Data API

AWS AppSyncmenyediakan sumber data untuk mengeksekusi pernyataan SQL terhadap klaster Amazon Aurora yang diaktifkan dengan API Data. Anda dapat menggunakan AWS AppSync resolver untuk menjalankan pernyataan SQL terhadap API data dengan kueri, mutasi, dan langganan GraphQL.

> **Note**
>
> Tutorial ini menggunakan Wilayah `US-EAST-1`.

Membuat cluster

Sebelum menambahkan sumber data Amazon RDSAWS AppSync, aktifkan API Data terlebih dahulu pada klaster Aurora Tanpa Server. Anda juga harus mengkonfigurasi rahasia menggunakanAWS Secrets Manager. Untuk membuat cluster Aurora Tanpa Server, Anda dapat menggunakan: AWS CLI

```
aws rds create-db-cluster \
    --db-cluster-identifier appsync-tutorial \
    --engine aurora-postgresql --engine-version 13.11 \
    --engine-mode serverless \
    --master-username USERNAME \
    --master-user-password COMPLEX_PASSWORD
```

Ini akan mengembalikan ARN untuk cluster. Anda dapat memeriksa status cluster Anda dengan perintah:

```
aws rds describe-db-clusters \
    --db-cluster-identifier appsync-tutorial \
    --query "DBClusters[0].Status"
```

Buat Rahasia melalui AWS Secrets Manager Konsol atau AWS CLI dengan file input seperti berikut menggunakan `USERNAME` dan `COMPLEX_PASSWORD` dari langkah sebelumnya:

```
{
    "username": "USERNAME",
    "password": "COMPLEX_PASSWORD"
}
```

Berikan ini sebagai parameter ke CLI:

```
aws secretsmanager create-secret \
    --name appsync-tutorial-rds-secret \
    --secret-string file://creds.json
```

Ini akan mengembalikan ARN untuk rahasianya. Catat ARN cluster Aurora Serverless Anda dan Rahasia untuk nanti saat membuat sumber data di konsol. AWS AppSync

Mengaktifkan API data

Setelah status klaster Anda berubah`available`, aktifkan Data API dengan mengikuti dokumentasi Amazon RDS. Data API harus diaktifkan sebelum menambahkannya sebagai sumber AWS AppSync data. Anda juga dapat mengaktifkan API Data menggunakanAWS CLI:

```
aws rds modify-db-cluster \
    --db-cluster-identifier appsync-tutorial \
    --enable-http-endpoint \
    --apply-immediately
```

Membuat database dan tabel

Setelah mengaktifkan API Data Anda, validasi itu berfungsi menggunakan `aws rds-data execute-statement` perintah di. AWS CLI Ini memastikan bahwa klaster Aurora Tanpa Server Anda dikonfigurasi dengan benar sebelum menambahkannya ke API. AWS AppSync Pertama, buat database TESTDB dengan `--sql` parameter:

```
aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:appsync-tutorial" \
    --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:appsync-
tutorial-rds-secret" \
    --sql "create DATABASE \"testdb\""
```

Jika ini berjalan tanpa kesalahan, tambahkan dua tabel dengan `create table` perintah:

```
 aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:appsync-tutorial" \
    --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:appsync-
tutorial-rds-secret" \
    --database "testdb" \
    --sql 'create table public.todos (id serial constraint todos_pk primary key,
 description text not null, due date not null, "createdAt" timestamp default now());'

aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:appsync-tutorial" \
    --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:appsync-
tutorial-rds-secret" \
```

```
    --database "testdb" \
    --sql 'create table public.tasks (id serial constraint tasks_pk primary key,
description varchar, "todoId" integer not null constraint tasks_todos_id_fk references
public.todos);'
```

Jika semuanya berjalan tanpa masalah, Anda sekarang dapat menambahkan cluster sebagai sumber data di API Anda.

Membuat skema GraphQL

Sekarang setelah Aurora Serverless Data API Anda berjalan dengan tabel yang dikonfigurasi, kami akan membuat skema GraphQL. Anda dapat melakukannya secara manual, tetapi AWS AppSync memungkinkan Anda memulai dengan cepat dengan mengimpor konfigurasi tabel dari database yang ada menggunakan wizard pembuatan API.

Untuk memulai:

1. Di AWS AppSync konsol, pilih Buat API, lalu Mulai dengan cluster Amazon Aurora.
2. Tentukan detail API seperti nama API, lalu pilih database Anda untuk menghasilkan API.
3. Pilih database Anda. Jika perlu, perbarui Wilayah, lalu pilih cluster Aurora dan database TESTDB Anda.
4. Pilih Rahasia Anda, lalu pilih Impor.
5. Setelah tabel ditemukan, perbarui nama jenisnya. Ubah `Todos` ke `Todo` dan `Tasks` ke`Task`.
6. Pratinjau skema yang dihasilkan dengan memilih Skema Pratinjau. Skema Anda akan terlihat seperti ini:

```
type Todo {
  id: Int!
  description: String!
  due: AWSDate!
  createdAt: String
}

type Task {
  id: Int!
  todoId: Int!
  description: String
}
```

7. Untuk peran tersebut, Anda dapat AWS AppSync membuat peran baru atau membuat peran dengan kebijakan yang mirip dengan yang di bawah ini:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-data:ExecuteStatement",
            ],
            "Resource": [
                "arn:aws:rds:us-east-1:123456789012:cluster:appsync-tutorial",
                "arn:aws:rds:us-east-1:123456789012:cluster:appsync-tutorial:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:your:secret:arn:appsync-tutorial-rds-secret",
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:your:secret:arn:appsync-tutorial-rds-secret:*"
            ]
        }
    ]
}
```

Perhatikan bahwa ada dua pernyataan dalam kebijakan ini yang Anda berikan akses peran. Sumber daya pertama adalah cluster Aurora Anda dan yang kedua adalah AWS Secrets Manager ARN Anda.

Pilih Berikutnya, tinjau detail konfigurasi, lalu pilih Buat API. Anda sekarang memiliki API yang beroperasi penuh. Anda dapat meninjau detail lengkap API Anda di halaman Skema.

Resolver untuk RDS

Alur pembuatan API secara otomatis membuat resolver untuk berinteraksi dengan tipe kami. Jika Anda melihat halaman Skema, Anda akan menemukan resolver yang diperlukan untuk:

- Buat `todo` melalui `Mutation.createTodo` bidang.
- Perbarui `todo` melalui `Mutation.updateTodo` bidang.
- Hapus `todo` melalui `Mutation.deleteTodo` bidang.
- Dapatkan satu `todo` melalui `Query.getTodo` lapangan.
- Daftar semua `todos` melalui `Query.listTodos` bidang.

Anda akan menemukan bidang dan resolver serupa yang dilampirkan untuk jenisnya. `Task` Mari kita lihat lebih dekat beberapa resolver.

mutasi.createTodo

Dari editor skema di AWS AppSync konsol, di sisi kanan, pilih di `testdb` sebelah`createTodo(...): Todo`. Kode resolver menggunakan `insert` fungsi dari `rds` modul untuk secara dinamis membuat pernyataan insert yang menambahkan data ke tabel. `todos` Karena kami bekerja dengan Postgres, kami dapat memanfaatkan `returning` pernyataan untuk mendapatkan data yang dimasukkan kembali.

Mari kita perbarui resolver untuk menentukan `DATE` jenis bidang dengan benar: `due`

```
import { util } from '@aws-appsync/utils';
import { insert, createPgStatement, toJsonObject, typeHint } from '@aws-appsync/utils/
rds';

export function request(ctx) {
    const { input } = ctx.args;
    // if a due date is provided, cast is as `DATE`
    if (input.due) {
        input.due = typeHint.DATE(input.due)
    }
    const insertStatement = insert({
        table: 'todos',
        values: input,
        returning: '*',
    });
    return createPgStatement(insertStatement)
```

```
}

export function response(ctx) {
    const { error, result } = ctx;
    if (error) {
        return util.appendError(
            error.message,
            error.type,
            result
        )
    }
    return toJsonObject(result)[0][0]
}
```

Simpan resolver. Petunjuk tipe menandai `due` dengan benar di objek input kami sebagai `DATE` tipe. Hal ini memungkinkan mesin Postgres untuk menafsirkan nilai dengan benar. Selanjutnya, perbarui skema Anda untuk menghapus `id` dari `CreateTodo` input. Karena database Postgres kami dapat mengembalikan ID yang dihasilkan, kami dapat mengandalkannya untuk pembuatan dan mengembalikan hasilnya sebagai satu permintaan:

```
input CreateTodoInput {
    due: AWSDate!
    createdAt: String
    description: String!
}
```

Buat perubahan dan perbarui skema Anda. Buka editor Kueri untuk menambahkan item ke database:

```
mutation CreateTodo {
  createTodo(input: {description: "Hello World!", due: "2023-12-31"}) {
    id
    due
    description
    createdAt
  }
}
```

Anda mendapatkan hasilnya:

```
{
  "data": {
    "createTodo": {
```

```
      "id": 1,
      "due": "2023-12-31",
      "description": "Hello World!",
      "createdAt": "2023-11-14 20:47:11.875428"
    }
  }
}
```

Query.listTodos

Dari editor skema di konsol, di sisi kanan, pilih di `testdb` sebelah`listTodos(id: ID!): Todo`. Handler permintaan menggunakan fungsi utilitas pilih untuk membangun permintaan secara dinamis pada waktu berjalan.

```
export function request(ctx) {
    const { filter = {}, limit = 100, nextToken } = ctx.args;
    const offset = nextToken ? +util.base64Decode(nextToken) : 0;
    const statement = select({
        table: 'todos',
        columns: '*',
        limit,
        offset,
        where: filter,
    });
    return createPgStatement(statement)
}
```

Kami ingin memfilter `todos` berdasarkan due tanggal. Mari kita perbarui resolver untuk mentransmisikan due nilai ke. `DATE` Perbarui daftar impor dan penangan permintaan:

```
import { util } from '@aws-appsync/utils';
import * as rds from '@aws-appsync/utils/rds';

export function request(ctx) {
  const { filter: where = {}, limit = 100, nextToken } = ctx.args;
  const offset = nextToken ? +util.base64Decode(nextToken) : 0;

  // if `due` is used in a filter, CAST the values to DATE.
  if (where.due) {
    Object.entries(where.due).forEach(([k, v]) => {
      if (k === 'between') {
        where.due[k] = v.map((d) => rds.typeHint.DATE(d));
```

```
      } else {
        where.due[k] = rds.typeHint.DATE(v);
      }
    });
  }

  const statement = rds.select({
    table: 'todos',
    columns: '*',
    limit,
    offset,
    where,
  });
  return rds.createPgStatement(statement);
}

export function response(ctx) {
  const {
    args: { limit = 100, nextToken },
    error,
    result,
  } = ctx;
  if (error) {
    return util.appendError(error.message, error.type, result);
  }
  const offset = nextToken ? +util.base64Decode(nextToken) : 0;
  const items = rds.toJsonObject(result)[0];
  const endOfResults = items?.length < limit;
  const token = endOfResults ? null : util.base64Encode(`${offset + limit}`);
  return { items, nextToken: token };
}
```

Mari kita coba query. Di editor Queries:

```
query LIST {
  listTodos(limit: 10, filter: {due: {between: ["2021-01-01", "2025-01-02"]}}) {
    items {
      id
      due
      description
    }
  }
}
```

mutasi.updateTodo

Anda juga bisa `update` aTodo. Dari editor Queries, mari kita perbarui `Todo` item pertama kami. `id` 1

```
mutation UPDATE {
  updateTodo(input: {id: 1, description: "edits"}) {
    description
    due
    id
  }
}
```

Perhatikan bahwa Anda harus menentukan item `id` yang Anda perbarui. Anda juga dapat menentukan kondisi untuk hanya memperbarui item yang memenuhi kondisi tertentu. Misalnya, kami mungkin hanya ingin mengedit item jika deskripsi dimulai dengan`edits`:

```
mutation UPDATE {
  updateTodo(input: {id: 1, description: "edits: make a change"}, condition:
 {description: {beginsWith: "edits"}}) {
    description
    due
    id
  }
}
```

Sama seperti cara kami menangani `create` dan `list` operasi kami, kami dapat memperbarui resolver kami untuk mentransmisikan `due` bidang ke a. `DATE` Simpan perubahan ini keu`pdateTodo`:

```
import { util } from '@aws-appsync/utils';
import * as rds from '@aws-appsync/utils/rds';

export function request(ctx) {
  const { input: { id, ...values }, condition = {}, } = ctx.args;
  const where = { ...condition, id: { eq: id } };

  // if `due` is used in a condition, CAST the values to DATE.
  if (condition.due) {
    Object.entries(condition.due).forEach(([k, v]) => {
      if (k === 'between') {
        condition.due[k] = v.map((d) => rds.typeHint.DATE(d));
      } else {
```

```
        condition.due[k] = rds.typeHint.DATE(v);
      }
    });
  }

  // if a due date is provided, cast is as `DATE`
  if (values.due) {
    values.due = rds.typeHint.DATE(values.due);
  }

  const updateStatement = rds.update({
    table: 'todos',
    values,
    where,
    returning: '*',
  });
  return rds.createPgStatement(updateStatement);
}

export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    return util.appendError(error.message, error.type, result);
  }
  return rds.toJsonObject(result)[0][0];
}
```

Sekarang coba pembaruan dengan kondisi:

```
mutation UPDATE {
  updateTodo(
    input: {
        id: 1, description: "edits: make a change", due: "2023-12-12"},
    condition: {
        description: {beginsWith: "edits"}, due: {ge: "2023-11-08"}})
    {
          description
          due
          id
        }
}
```

mutasi.deleteTodo

Anda bisa `delete` a `Todo` dengan `deleteTodo` mutasi. Ini berfungsi seperti `updateTodo` mutasi, dan Anda harus menentukan item `id` yang ingin Anda hapus:

```
mutation DELETE {
  deleteTodo(input: {id: 1}) {
    description
    due
    id
  }
}
```

Menulis kueri kustom

Kami telah menggunakan utilitas `rds` modul untuk membuat pernyataan SQL kami. Kami juga dapat menulis pernyataan statis kustom kami sendiri untuk berinteraksi dengan database kami. Pertama, perbarui skema untuk menghapus `id` bidang dari `CreateTask` input.

```
input CreateTaskInput {
    todoId: Int!
    description: String
}
```

Selanjutnya, buat beberapa tugas. Suatu tugas memiliki hubungan kunci asing dengan`Todo`:

```
mutation TASKS {
  a: createTask(input: {todoId: 2, description: "my first sub task"}) { id }
  b:createTask(input: {todoId: 2, description: "another sub task"}) { id }
  c: createTask(input: {todoId: 2, description: "a final sub task"}) { id }
}
```

Buat bidang baru dalam `Query` tipe Anda yang disebut`getTodoAndTasks`:

```
getTodoAndTasks(id: Int!): Todo
```

Tambahkan `tasks` bidang ke `Todo` jenis:

```
type Todo {
    due: AWSDate!
```

```
    id: Int!
    createdAt: String
    description: String!
    tasks:TaskConnection
}
```

Simpan skema. Dari editor skema di konsol, di sisi kanan, pilih Lampirkan Resolver untuk. `getTodosAndTasks(id: Int!): Todo` Pilih sumber data Amazon RDS Anda. Perbarui resolver Anda dengan kode berikut:

```
import { sql, createPgStatement,toJsonObject } from '@aws-appsync/utils/rds';

export function request(ctx) {
    return createPgStatement(
        sql`SELECT * from todos where id = ${ctx.args.id}`,
        sql`SELECT * from tasks where "todoId" = ${ctx.args.id}`);
}

export function response(ctx) {
    const result = toJsonObject(ctx.result);
    const todo = result[0][0];
    if (!todo) {
        return null;
    }
    todo.tasks = { items: result[1] };
    return todo;
}
```

Dalam kode ini, kita menggunakan template `sql` tag untuk menulis pernyataan SQL bahwa kita dapat dengan aman meneruskan nilai dinamis pada waktu berjalan. `createPgStatement`dapat mengambil hingga dua permintaan SQL pada satu waktu. Kami menggunakannya untuk mengirim satu kueri untuk kami `todo` dan yang lain untuk kami`tasks`. Anda bisa melakukan ini dengan `JOIN` pernyataan atau metode lain dalam hal ini. Idenya adalah mampu menulis pernyataan SQL Anda sendiri untuk mengimplementasikan logika bisnis Anda. Untuk menggunakan kueri di editor Queries, kita dapat mencoba ini:

```
query TodoAndTasks {
  getTodosAndTasks(id: 2) {
    id
    due
    description
```

```
    tasks {
      items {
        id
        description
      }
    }
  }
}
```

Menghapus klaster Anda

> **Important**
>
> Menghapus cluster bersifat permanen. Tinjau proyek Anda secara menyeluruh sebelum melakukan tindakan ini.

Untuk menghapus klaster Anda:

```
$ aws rds delete-db-cluster \
    --db-cluster-identifier appsync-tutorial \
    --skip-final-snapshot
```

Tutorial penyelesai (VTL)

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Sumber data dan resolver adalah cara AWS AppSync menerjemahkan permintaan GraphQL dan mengambil informasi dari sumber daya Anda. AWS AWS AppSync memiliki dukungan untuk penyediaan otomatis dan koneksi dengan tipe sumber data tertentu. AWS AppSync mendukungAWS Lambda, Amazon DynamoDB, database relasional (Amazon Aurora Tanpa Server), Layanan Amazon OpenSearch , dan titik akhir HTTP sebagai sumber data. Anda dapat menggunakan GraphQL API dengan sumber daya yang AWS ada atau membangun sumber data dan resolver. Bagian ini membawa Anda melalui proses ini dalam serangkaian tutorial untuk lebih memahami cara kerja detail dan opsi penyetelan.

AWS AppSync menggunakan template pemetaan yang ditulis dalam Apache Velocity Template Language (VTL) untuk resolver. Untuk informasi selengkapnya tentang penggunaan templat pemetaan, lihat referensi template pemetaan Resolver. Informasi lebih lanjut tentang bekerja dengan VTL tersedia di panduan pemrograman template pemetaan Resolver.

AWS AppSync mendukung penyediaan otomatis tabel DynamoDB dari skema GraphQL seperti yang dijelaskan dalam Penyediaan dari skema (opsional) dan Luncurkan skema sampel. Anda juga dapat mengimpor dari tabel DynamoDB yang ada yang akan membuat skema dan menghubungkan resolver. Ini diuraikan dalam Impor dari Amazon DynamoDB (opsional).

Topik

- Tutorial: Resolver HTTP
- Tutorial: Aurora Tanpa Server
- Tutorial: Penyelesai Pipa
- Tutorial: Sinkronisasi Delta

Tutorial: penyelesai DynamoDB

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Tutorial ini menunjukkan bagaimana Anda dapat membawa tabel Amazon DynamoDB Anda sendiri dan menghubungkannya AWS AppSync ke GraphQL API.

Anda dapat mengizinkan AWS AppSync penyediaan sumber daya DynamoDB atas nama Anda. Atau, jika Anda mau, Anda dapat menghubungkan tabel yang ada ke skema GraphQL dengan membuat sumber data dan resolver. Dalam kedua kasus, Anda akan dapat membaca dan menulis ke database DynamoDB Anda melalui pernyataan GraphQL dan berlangganan data real-time.

Ada langkah-langkah konfigurasi khusus yang perlu diselesaikan agar pernyataan GraphQL diterjemahkan ke operasi DynamoDB, dan agar tanggapan diterjemahkan kembali ke GraphQL. Tutorial ini menguraikan proses konfigurasi melalui beberapa skenario dunia nyata dan pola akses data.

Menyiapkan tabel DynamoDB Anda

Untuk memulai tutorial ini, pertama-tama Anda harus mengikuti langkah-langkah di bawah ini untuk menyediakan AWS sumber daya.

1. Menyediakan AWS sumber daya menggunakan AWS CloudFormation template berikut di CLI:

```
aws cloudformation create-stack \
    --stack-name AWSAppSyncTutorialForAmazonDynamoDB \
    --template-url https://s3.us-west-2.amazonaws.com/awsappsync/resources/
dynamodb/AmazonDynamoDBCFTemplate.yaml \
```

```
    --capabilities CAPABILITY_NAMED_IAM
```

Atau, Anda dapat meluncurkan AWS CloudFormation tumpukan berikut di wilayah AS-Barat 2 (Oregon) di akun AndaAWS.

Ini menciptakan yang berikut:

- Sebuah tabel DynamoDB `AppSyncTutorial-Post` disebut yang akan menyimpan data. `Post`
- Peran IAM dan kebijakan terkelola IAM terkait AWS AppSync untuk memungkinkan berinteraksi dengan tabel. `Post`

2. Untuk melihat detail selengkapnya tentang tumpukan dan sumber daya yang dibuat, jalankan perintah CLI berikut:

```
aws cloudformation describe-stacks --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

3. Untuk menghapus sumber daya nanti, Anda dapat menjalankan yang berikut:

```
aws cloudformation delete-stack --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

Membuat GraphQL API

Untuk membuat GraphQL API di: AWS AppSync

1. Masuk ke AWS Management Console dan buka AppSync konsol.
   - Di dasbor API, pilih Buat API.
2. Di bawah jendela Sesuaikan API atau impor dari Amazon DynamoDB, pilih Bangun dari awal.
   - Pilih Mulai di sebelah kanan jendela yang sama.
3. Di bidang nama API, atur nama API ke`AWSAppSyncTutorial`.
4. Pilih Create (Buat).

AWS AppSync Konsol membuat API GraphQL baru untuk Anda menggunakan mode autentikasi kunci API. Anda dapat menggunakan konsol untuk mengatur sisa GraphQL API dan menjalankan kueri terhadapnya selama sisa tutorial ini.

Mendefinisikan API posting dasar

Sekarang setelah Anda membuat AWS AppSync GraphQL API, Anda dapat menyiapkan skema dasar yang memungkinkan pembuatan dasar, pengambilan, dan penghapusan data posting.

1. Masuk ke AWS Management Console dan buka AppSync konsol.
   - Di dasbor API, pilih API yang baru saja Anda buat.
2. Di Sidebar, pilih Skema.
   - Di panel Skema, ganti konten dengan kode berikut:

```
schema {
    query: Query
    mutation: Mutation
}

type Query {
    getPost(id: ID): Post
}

type Mutation {
    addPost(
        id: ID!
        author: String!
        title: String!
        content: String!
        url: String!
    ): Post!
}

type Post {
    id: ID!
    author: String
    title: String
    content: String
    url: String
    ups: Int!
```

```
    downs: Int!
    version: Int!
}
```

3. Pilih Save (Simpan).

Skema ini mendefinisikan `Post` jenis dan operasi untuk menambah dan mendapatkan `Post` objek.

Mengkonfigurasi Sumber Data untuk Tabel DynamoDB

Selanjutnya, tautkan kueri dan mutasi yang ditentukan dalam skema ke tabel `AppSyncTutorial-Post` DynamoDB.

Pertama, AWS AppSync perlu menyadari tabel Anda. Anda melakukan ini dengan menyiapkan sumber data di AWS AppSync:

1. Masuk ke AWS Management Console dan buka AppSync konsol.
   a. Di dasbor API, pilih GraphQL API Anda.
   b. Di Sidebar, pilih Sumber Data.
2. Pilih Buat sumber data.
   a. Untuk nama sumber data, masukkan`PostDynamoDBTable`.
   b. Untuk tipe sumber data, pilih tabel Amazon DynamoDB.
   c. Untuk Wilayah, pilih US-WEST-2.
   d. Untuk nama Tabel, pilih tabel AppSyncTutorial-Post DynamoDB.
   e. Buat peran IAM baru (disarankan) atau pilih peran yang sudah ada yang memiliki izin `lambda:invokeFunction` IAM. Peran yang ada memerlukan kebijakan kepercayaan, seperti yang dijelaskan di bagian Melampirkan sumber data.

      Berikut ini adalah contoh kebijakan IAM yang memiliki izin yang diperlukan untuk melakukan operasi pada sumber daya:

```
{
     "Version": "2012-10-17",
     "Statement": [
         {
             "Effect": "Allow",
             "Action": [ "lambda:invokeFunction" ],
```

```
            "Resource": [
                "arn:aws:lambda:us-west-2:123456789012:function:myFunction",
                "arn:aws:lambda:us-west-2:123456789012:function:myFunction:*"
            ]
        }
    ]
}
```

3. Pilih Create (Buat).

Menyiapkan AddPost resolver (DynamoDB) PutItem

Setelah AWS AppSync mengetahui tabel DynamoDB, Anda dapat menautkannya ke kueri dan mutasi individual dengan mendefinisikan Resolvers. Resolver pertama yang Anda buat adalah `addPost` resolver, yang memungkinkan Anda untuk membuat posting di tabel DynamoDB. `AppSyncTutorial-Post`

Sebuah resolver memiliki komponen-komponen berikut:

- Lokasi dalam skema GraphQL untuk melampirkan resolver. Dalam hal ini, Anda menyiapkan resolver di `addPost` bidang pada tipe. `Mutation` Penyelesai ini akan dipanggil saat pemanggil memanggil. `mutation { addPost(...){...} }`
- Sumber data yang akan digunakan untuk resolver ini. Dalam hal ini, Anda ingin menggunakan sumber `PostDynamoDBTable` data yang Anda tentukan sebelumnya, sehingga Anda dapat menambahkan entri ke dalam tabel `AppSyncTutorial-Post` DynamoDB.
- Templat pemetaan permintaan . Tujuan dari template pemetaan permintaan adalah untuk mengambil permintaan masuk dari pemanggil dan menerjemahkannya ke dalam instruksi untuk melakukan AWS AppSync terhadap DynamoDB.
- Templat pemetaan respons. Tugas template pemetaan respons adalah mengambil respons dari DynamoDB dan menerjemahkannya kembali menjadi sesuatu yang diharapkan GraphQL. Ini berguna jika bentuk data di DynamoDB berbeda dengan `Post` tipe di GraphQL, tetapi dalam hal ini mereka memiliki bentuk yang sama, jadi Anda cukup melewatkan data.

Untuk mengatur resolver:

1. Masuk ke AWS Management Console dan buka AppSync konsol.

    a. Di dasbor API, pilih GraphQL API Anda.

b. Di Sidebar, pilih Sumber Data.

2. Pilih Buat sumber data.

   a. Untuk nama sumber data, masukkan`PostDynamoDBTable`.

   b. Untuk tipe sumber data, pilih tabel Amazon DynamoDB.

   c. Untuk Wilayah, pilih US-WEST-2.

   d. Untuk nama Tabel, pilih tabel AppSyncTutorial-Post DynamoDB.

   e. Buat peran IAM baru (disarankan) atau pilih peran yang sudah ada yang memiliki izin `lambda:invokeFunction` IAM. Peran yang ada memerlukan kebijakan kepercayaan, seperti yang dijelaskan di bagian Melampirkan sumber data.

      Berikut ini adalah contoh kebijakan IAM yang memiliki izin yang diperlukan untuk melakukan operasi pada sumber daya:

```
{
     "Version": "2012-10-17",
     "Statement": [
         {
             "Effect": "Allow",
             "Action": [ "lambda:invokeFunction" ],
             "Resource": [
                 "arn:aws:lambda:us-west-2:123456789012:function:myFunction",
                 "arn:aws:lambda:us-west-2:123456789012:function:myFunction:*"
             ]
         }
     ]
 }
```

3. Pilih Create (Buat).

4. Pilih tab Skema.

5. Di panel tipe Data di sebelah kanan, temukan bidang AddPost pada tipe Mutasi, lalu pilih Lampirkan.

6. Di menu Action, pilih Update runtime, lalu pilih Unit Resolver (hanya VTL).

7. Dalam nama sumber data, pilih PostDynamoDbTable.

8. Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
    "version" : "2017-02-28",
```

```
    "operation" : "PutItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($context.arguments.id)
    },
    "attributeValues" : {
        "author" : $util.dynamodb.toDynamoDBJson($context.arguments.author),
        "title" : $util.dynamodb.toDynamoDBJson($context.arguments.title),
        "content" : $util.dynamodb.toDynamoDBJson($context.arguments.content),
        "url" : $util.dynamodb.toDynamoDBJson($context.arguments.url),
        "ups" : { "N" : 1 },
        "downs" : { "N" : 0 },
        "version" : { "N" : 1 }
    }
}
```

Catatan: Tipe ditentukan pada semua kunci dan nilai atribut. Misalnya, Anda mengatur `author` bidang ke`{ "S" : "${context.arguments.author}" }`. SBagian menunjukkan kepada AWS AppSync dan DynamoDB bahwa nilai akan menjadi nilai string. Nilai sebenarnya akan diisi dari `author` argumen. Demikian pula, `version` bidang adalah bidang angka karena digunakan `N` untuk tipe. Akhirnya, Anda juga menginisialisasi`ups`, `downs` dan `version` bidang.

Untuk tutorial ini Anda telah menentukan bahwa tipe `ID!` GraphQL, yang mengindeks item baru yang dimasukkan ke DynamoDB, datang sebagai bagian dari argumen klien. AWS AppSyncdilengkapi dengan utilitas untuk pembuatan ID otomatis `$utils.autoId()` yang disebut yang bisa Anda gunakan juga dalam bentuk`"id" : { "S" : "${$utils.autoId()}" }`. Kemudian Anda bisa meninggalkan `id: ID!` keluar dari definisi skema `addPost()` dan itu akan dimasukkan secara otomatis. Anda tidak akan menggunakan teknik ini untuk tutorial ini, tetapi Anda harus menganggapnya sebagai praktik yang baik saat menulis ke tabel DynamoDB.

Untuk informasi selengkapnya tentang templat pemetaan, lihat dokumentasi referensi Ikhtisar Templat Pemetaan Resolver. Untuk informasi selengkapnya tentang pemetaan GetItem permintaan, lihat dokumentasi GetItemreferensi. Untuk informasi selengkapnya tentang jenis, lihat dokumentasi referensi Sistem Jenis (Permintaan Pemetaan).

9. Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
$utils.toJson($context.result)
```

Catatan: Karena bentuk data dalam `AppSyncTutorial-Post` tabel sama persis dengan bentuk `Post` tipe di GraphQL, template pemetaan respons hanya meneruskan hasilnya secara langsung. Perhatikan juga bahwa semua contoh dalam tutorial ini menggunakan template pemetaan respons yang sama, jadi Anda hanya membuat satu file.

10. Pilih Save (Simpan).

Panggil API untuk Menambahkan Posting

Sekarang resolver sudah diatur, AWS AppSync dapat menerjemahkan `addPost` mutasi yang masuk ke operasi DynamoDB. PutItem Anda sekarang dapat menjalankan mutasi untuk meletakkan sesuatu di tabel.

- Pilih tab Kueri.
- Di panel Kueri, tempel mutasi berikut:

```
mutation addPost {
  addPost(
    id: 123
    author: "AUTHORNAME"
    title: "Our first post!"
    content: "This is our first post."
    url: "https://aws.amazon.com/appsync/"
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Hasil posting yang baru dibuat akan muncul di panel hasil di sebelah kanan panel kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
```

```
  "data": {
    "addPost": {
      "id": "123",
      "author": "AUTHORNAME",
      "title": "Our first post!",
      "content": "This is our first post.",
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
      "version": 1
    }
  }
}
```

Inilah yang terjadi:

- AWS AppSync menerima permintaan `addPost` mutasi.
- AWS AppSync mengambil permintaan, dan template pemetaan permintaan, dan menghasilkan dokumen pemetaan permintaan. Ini akan terlihat seperti:

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key" : {
        "id" : { "S" : "123" }
    },
    "attributeValues" : {
        "author": { "S" : "AUTHORNAME" },
        "title": { "S" : "Our first post!" },
        "content": { "S" : "This is our first post." },
        "url": { "S" : "https://aws.amazon.com/appsync/" },
        "ups" : { "N" : 1 },
        "downs" : { "N" : 0 },
        "version" : { "N" : 1 }
    }
}
```

- AWS AppSync menggunakan dokumen pemetaan permintaan untuk menghasilkan dan mengeksekusi permintaan DynamoDB`PutItem`.
- AWS AppSync mengambil hasil `PutItem` permintaan dan mengubahnya kembali ke tipe GraphQL.

```
{
    "id" : "123",
    "author": "AUTHORNAME",
    "title": "Our first post!",
    "content": "This is our first post.",
    "url": "https://aws.amazon.com/appsync/",
    "ups" : 1,
    "downs" : 0,
    "version" : 1
}
```

- Melewatkannya melalui dokumen pemetaan respons, yang baru saja melewatinya tanpa perubahan.
- Mengembalikan objek yang baru dibuat dalam respons GraphQL.

Menyiapkan GetPost Resolver (DynamoDB) GetItem

Sekarang bahwa Anda dapat menambahkan data ke tabel `AppSyncTutorial-Post` DynamoDB, Anda perlu mengatur query sehingga dapat mengambil data dari tabel. `getPost` `AppSyncTutorial-Post` Untuk melakukan ini, Anda mengatur resolver lain.

- Pilih tab Skema.
- Di panel tipe Data di sebelah kanan, temukan bidang GetPost pada tipe Query, lalu pilih Lampirkan.
- Di menu Action, pilih Update runtime, lalu pilih Unit Resolver (hanya VTL).
- Dalam nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    }
}
```

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
$utils.toJson($context.result)
```

- Pilih Save (Simpan).

Panggil API untuk Mendapatkan Posting

Sekarang resolver telah diatur, AWS AppSync tahu bagaimana menerjemahkan `getPost` query masuk ke operasi DynamoDB. `GetItem` Anda sekarang dapat menjalankan kueri untuk mengambil posting yang Anda buat sebelumnya.

- Pilih tab Kueri.
- Di panel Kueri, tempel berikut ini:

```
query getPost {
  getPost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Posting yang diambil dari DynamoDB akan muncul di panel hasil di sebelah kanan panel kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "getPost": {
      "id": "123",
      "author": "AUTHORNAME",
      "title": "Our first post!",
      "content": "This is our first post.",
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
```

```
      "version": 1
    }
  }
}
```

Inilah yang terjadi:

- AWS AppSync menerima permintaan `getPost` kueri.
- AWS AppSync mengambil permintaan, dan template pemetaan permintaan, dan menghasilkan dokumen pemetaan permintaan. Ini akan terlihat seperti:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "id" : { "S" : "123" }
    }
}
```

- AWS AppSync menggunakan dokumen pemetaan permintaan untuk menghasilkan dan mengeksekusi permintaan DynamoDB GetItem .
- AWS AppSync mengambil hasil `GetItem` permintaan dan mengubahnya kembali ke tipe GraphQL.

```
{
    "id" : "123",
    "author": "AUTHORNAME",
    "title": "Our first post!",
    "content": "This is our first post.",
    "url": "https://aws.amazon.com/appsync/",
    "ups" : 1,
    "downs" : 0,
    "version" : 1
}
```

- Melewatkannya melalui dokumen pemetaan respons, yang baru saja melewatinya tanpa perubahan.
- Mengembalikan objek yang diambil dalam respons.

Atau, ambil contoh berikut:

```
query getPost {
  getPost(id:123) {
    id
    author
    title
  }
}
```

Jika `getPost` kueri Anda hanya membutuhkan`id`,, dan `authortitle`, Anda dapat mengubah template pemetaan permintaan Anda untuk menggunakan ekspresi proyeksi untuk menentukan hanya atribut yang Anda inginkan dari tabel DynamoDB Anda untuk menghindari transfer data yang tidak perlu dari DynamoDB ke. AWS AppSync Misalnya, template pemetaan permintaan mungkin terlihat seperti cuplikan di bawah ini:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },
    "projection" : {
     "expression" : "#author, id, title",
     "expressionNames" : { "#author" : "author"}
    }
}
```

Membuat Mutasi UpdatePost (DynamoDB) UpdateItem

Sejauh ini Anda dapat membuat dan mengambil `Post` objek di DynamoDB. Selanjutnya, Anda akan mengatur mutasi baru untuk memungkinkan kita untuk memperbarui objek. Anda akan melakukan ini menggunakan operasi UpdateItem DynamoDB.

- Pilih tab Skema.
- Di panel Skema, ubah `Mutation` tipe untuk menambahkan `updatePost` mutasi baru sebagai berikut:

```
type Mutation {
    updatePost(
        id: ID!,
        author: String!,
```

```
        title: String!,
        content: String!,
        url: String!
    ): Post
    addPost(
        author: String!
        title: String!
        content: String!
        url: String!
    ): Post!
}
```

- Pilih Save (Simpan).
- Di panel tipe Data di sebelah kanan, temukan bidang UpdatePost yang baru dibuat pada tipe Mutasi dan kemudian pilih Lampirkan.
- Di menu Action, pilih Update runtime, lalu pilih Unit Resolver (hanya VTL).
- Dalam nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
    "version" : "2017-02-28",
    "operation" : "UpdateItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($context.arguments.id)
    },
    "update" : {
        "expression" : "SET author = :author, title = :title, content = :content,
 #url = :url ADD version :one",
        "expressionNames": {
            "#url" : "url"
        },
        "expressionValues": {
            ":author" : $util.dynamodb.toDynamoDBJson($context.arguments.author),
            ":title" : $util.dynamodb.toDynamoDBJson($context.arguments.title),
            ":content" : $util.dynamodb.toDynamoDBJson($context.arguments.content),
            ":url" : $util.dynamodb.toDynamoDBJson($context.arguments.url),
            ":one" : { "N": 1 }
        }
    }
}
```

Catatan: Resolver ini menggunakan DynamoDB UpdateItem, yang secara signifikan berbeda dari operasi. PutItem Alih-alih menulis seluruh item, Anda hanya meminta DynamoDB untuk memperbarui atribut tertentu. Ini dilakukan dengan menggunakan DynamoDB Update Expressions. Ekspresi itu sendiri ditentukan dalam `expression` bidang di `update` bagian. Dikatakan untuk mengatur atribut `authortitle`,, `content` dan url, dan kemudian menambah `version` bidang. Nilai yang digunakan tidak muncul dalam ekspresi itu sendiri; ekspresi memiliki placeholder yang memiliki nama dimulai dengan titik dua, yang kemudian didefinisikan di `expressionValues` bidang. Akhirnya, DynamoDB memiliki kata-kata cadangan yang tidak dapat muncul di. `expression` Misalnya, `url` adalah kata yang dicadangkan, jadi untuk memperbarui `url` bidang Anda dapat menggunakan placeholder nama dan mendefinisikannya di `expressionNames` bidang.

Untuk info selengkapnya tentang pemetaan `UpdateItem` permintaan, lihat dokumentasi UpdateItemreferensi. Untuk informasi selengkapnya tentang cara menulis ekspresi pembaruan, lihat dokumentasi DynamoDB UpdateExpressions .

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
$utils.toJson($context.result)
```

Panggil API untuk Memperbarui Posting

Sekarang resolver telah diatur, AWS AppSync tahu bagaimana menerjemahkan `update` mutasi yang masuk ke operasi DynamoDB. `Update` Anda sekarang dapat menjalankan mutasi untuk memperbarui item yang Anda tulis sebelumnya.

- Pilih tab Kueri.
- Di panel Kueri, tempel mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya.

```
mutation updatePost {
  updatePost(
    id:"123"
    author: "A new author"
    title: "An updated author!"
    content: "Now with updated content!"
    url: "https://aws.amazon.com/appsync/"
  ) {
```

```
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Posting yang diperbarui di DynamoDB akan muncul di panel hasil di sebelah kanan panel kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "updatePost": {
      "id": "123",
      "author": "A new author",
      "title": "An updated author!",
      "content": "Now with updated content!",
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
      "version": 2
    }
  }
}
```

Dalam contoh ini, `downs` bidang `ups` and tidak dimodifikasi karena template pemetaan permintaan tidak meminta AWS AppSync dan DynamoDB melakukan apa pun dengan bidang tersebut. Juga, `version` bidang bertambah 1 karena Anda meminta AWS AppSync dan DynamoDB untuk menambahkan 1 ke bidang. `version`

Memodifikasi UpdatePost Resolver (DynamoDB) UpdateItem

Ini adalah awal yang baik untuk `updatePost` mutasi, tetapi memiliki dua masalah utama:

- Jika Anda ingin memperbarui hanya satu bidang, Anda harus memperbarui semua bidang.
- Jika dua orang memodifikasi objek, Anda berpotensi kehilangan informasi.

Untuk mengatasi masalah ini, Anda akan memodifikasi `updatePost` mutasi untuk hanya memodifikasi argumen yang ditentukan dalam permintaan, dan kemudian menambahkan kondisi ke `UpdateItem` operasi.

1. Pilih tab Skema.
2. Di panel Skema, ubah `updatePost` bidang dalam `Mutation` tipe untuk menghapus tanda seru dari,,, dan `url` argumen `author` `titlecontent`, pastikan untuk membiarkan bidang apa adanya. `id` Ini akan membuat mereka argumen opsional. Juga, tambahkan `expectedVersion` argumen baru yang diperlukan.

```
type Mutation {
    updatePost(
        id: ID!,
        author: String,
        title: String,
        content: String,
        url: String,
        expectedVersion: Int!
    ): Post
    addPost(
        author: String!
        title: String!
        content: String!
        url: String!
    ): Post!
}
```

3. Pilih Save (Simpan).
4. Di panel tipe Data di sebelah kanan, temukan bidang UpdatePost pada tipe Mutasi.
5. Pilih PostDynamoDbTable untuk membuka resolver yang ada.
6. Di Konfigurasikan templat pemetaan permintaan, ubah templat pemetaan permintaan sebagai berikut:

```
{
    "version" : "2017-02-28",
    "operation" : "UpdateItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($context.arguments.id)
    },
```

```
    ## Set up some space to keep track of things you're updating **
    #set( $expNames  = {} )
    #set( $expValues = {} )
    #set( $expSet = {} )
    #set( $expAdd = {} )
    #set( $expRemove = [] )

    ## Increment "version" by 1 **
    $!{expAdd.put("version", ":one")}
    $!{expValues.put(":one", { "N" : 1 })}

    ## Iterate through each argument, skipping "id" and "expectedVersion" **
    #foreach( $entry in $context.arguments.entrySet() )
        #if( $entry.key != "id" && $entry.key != "expectedVersion" )
            #if( (!$entry.value) && ("$!{entry.value}" == "") )
                ## If the argument is set to "null", then remove that attribute from
the item in DynamoDB **

                #set( $discard = ${expRemove.add("#${entry.key}")} )
                $!{expNames.put("#${entry.key}", "$entry.key")}
            #else
                ## Otherwise set (or update) the attribute on the item in DynamoDB **

                $!{expSet.put("#${entry.key}", ":${entry.key}")}
                $!{expNames.put("#${entry.key}", "$entry.key")}
                $!{expValues.put(":${entry.key}", { "S" : "${entry.value}" })}
            #end
        #end
    #end

    ## Start building the update expression, starting with attributes you're going to
SET **
    #set( $expression = "" )
    #if( !${expSet.isEmpty()} )
        #set( $expression = "SET" )
        #foreach( $entry in $expSet.entrySet() )
            #set( $expression = "${expression} ${entry.key} = ${entry.value}" )
            #if ( $foreach.hasNext )
                #set( $expression = "${expression}," )
            #end
        #end
    #end
```

```
   ## Continue building the update expression, adding attributes you're going to ADD
**
   #if( !${expAdd.isEmpty()} )
       #set( $expression = "${expression} ADD" )
       #foreach( $entry in $expAdd.entrySet() )
           #set( $expression = "${expression} ${entry.key} ${entry.value}" )
           #if ( $foreach.hasNext )
               #set( $expression = "${expression}," )
           #end
       #end
   #end

   ## Continue building the update expression, adding attributes you're going to
REMOVE **
   #if( !${expRemove.isEmpty()} )
       #set( $expression = "${expression} REMOVE" )

       #foreach( $entry in $expRemove )
           #set( $expression = "${expression} ${entry}" )
           #if ( $foreach.hasNext )
               #set( $expression = "${expression}," )
           #end
       #end
   #end

   ## Finally, write the update expression into the document, along with any
expressionNames and expressionValues **
   "update" : {
       "expression" : "${expression}"
       #if( !${expNames.isEmpty()} )
           ,"expressionNames" : $utils.toJson($expNames)
       #end
       #if( !${expValues.isEmpty()} )
           ,"expressionValues" : $utils.toJson($expValues)
       #end
   },

   "condition" : {
       "expression"       : "version = :expectedVersion",
       "expressionValues" : {
           ":expectedVersion" :
$util.dynamodb.toDynamoDBJson($context.arguments.expectedVersion)
       }
   }
```

```
}
```

7. Pilih Save (Simpan).

Template ini adalah salah satu contoh yang lebih kompleks. Ini menunjukkan kekuatan dan fleksibilitas template pemetaan. Ini mengulang semua argumen, melompati `id` dan`expectedVersion`. Jika argumen diatur ke sesuatu, ia meminta AWS AppSync dan DynamoDB untuk memperbarui atribut itu pada objek di DynamoDB. Jika atribut diatur ke null, ia meminta AWS AppSync dan DynamoDB untuk menghapus atribut itu dari objek post. Jika argumen tidak ditentukan, itu meninggalkan atribut saja. Ini juga menambah `version` bidang.

Juga, ada `condition` bagian baru. Ekspresi kondisi memungkinkan Anda memberi tahu AWS AppSync dan DynamoDB apakah permintaan harus berhasil atau tidak berdasarkan status objek yang sudah ada di DynamoDB sebelum operasi dilakukan. Dalam hal ini, Anda hanya ingin `UpdateItem` permintaan berhasil jika `version` bidang item saat ini di DynamoDB sama persis dengan argumen. `expectedVersion`

Untuk informasi selengkapnya tentang ekspresi kondisi, lihat dokumentasi referensi Ekspresi Kondisi.

Panggil API untuk Memperbarui Posting

Mari kita coba memperbarui `Post` objek dengan resolver baru:

- Pilih tab Kueri.
- Di panel Kueri, tempel mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya.

```
mutation updatePost {
  updatePost(
    id:123
    title: "An empty story"
    content: null
    expectedVersion: 2
  ) {
    id
    author
    title
    content
    url
    ups
    downs
```

```
    version
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Posting yang diperbarui di DynamoDB akan muncul di panel hasil di sebelah kanan panel kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "updatePost": {
      "id": "123",
      "author": "A new author",
      "title": "An empty story",
      "content": null,
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
      "version": 3
    }
  }
}
```

Dalam permintaan ini, Anda meminta AWS AppSync dan DynamoDB untuk memperbarui bidang dan sajatitle. content Itu meninggalkan semua bidang lainnya sendirian (selain menambah version bidang). Anda mengatur title atribut ke nilai baru, dan menghapus content atribut dari posting. Bidang authorurl,ups, dan downs ladang dibiarkan tak tersentuh.

Coba jalankan permintaan mutasi lagi, biarkan permintaan persis apa adanya. Anda akan melihat respons yang mirip dengan berikut ini:

```
{
  "data": {
    "updatePost": null
  },
  "errors": [
    {
      "path": [
        "updatePost"
      ],
      "data": {
```

```
        "id": "123",
        "author": "A new author",
        "title": "An empty story",
        "content": null,
        "url": "https://aws.amazon.com/appsync/",
        "ups": 1,
        "downs": 0,
        "version": 3
      },
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "message": "The conditional request failed (Service: AmazonDynamoDBv2;
  Status Code: 400; Error Code: ConditionalCheckFailedException; Request ID:
  ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ)"
    }
  ]
}
```

Permintaan gagal karena ekspresi kondisi dievaluasi menjadi false:

- Pertama kali Anda menjalankan permintaan, nilai `version` bidang posting di DynamoDB 2 adalah, yang cocok dengan argumen. `expectedVersion` Permintaan berhasil, yang berarti `version` bidang tersebut bertambah di DynamoDB ke. 3
- Kedua kalinya Anda menjalankan permintaan, nilai `version` bidang posting di DynamoDB 3 adalah, yang tidak cocok dengan argumen. `expectedVersion`

Pola ini biasanya disebut penguncian optimis.

Sebuah fitur dari AWS AppSync DynamoDB resolver adalah bahwa ia mengembalikan nilai saat ini dari objek post di DynamoDB. Anda dapat menemukan ini di `data` bidang di `errors` bagian respons GraphQL. Aplikasi Anda dapat menggunakan informasi ini untuk memutuskan bagaimana hal itu harus dilanjutkan. Dalam hal ini, Anda dapat melihat `version` bidang objek di DynamoDB diatur 3 ke, sehingga Anda bisa memperbarui `expectedVersion` argumen 3 ke dan permintaan akan berhasil lagi.

Untuk informasi selengkapnya tentang menangani kegagalan pemeriksaan kondisi, lihat dokumentasi referensi templat pemetaan Ekspresi Kondisi.

Buat Mutasi UpVotePost dan DownVotePost (DynamoDB) UpdateItem

`Post`Jenis memiliki `ups` dan `downs` bidang untuk mengaktifkan rekam suara positif dan suara turun, tetapi sejauh ini API tidak mengizinkan kami melakukan apa pun dengannya. Mari tambahkan beberapa mutasi untuk memungkinkan kita meningkatkan dan menurunkan suara posting.

- Pilih tab Skema.
- Di panel Skema, ubah `Mutation` tipe untuk menambahkan baru `upvotePost` dan `downvotePost` mutasi sebagai berikut:

```
type Mutation {
    upvotePost(id: ID!): Post
    downvotePost(id: ID!): Post
    updatePost(
        id: ID!,
        author: String,
        title: String,
        content: String,
        url: String,
        expectedVersion: Int!
    ): Post
    addPost(
        author: String!,
        title: String!,
        content: String!,
        url: String!
    ): Post!
}
```

- Pilih Save (Simpan).
- Di panel Jenis data di sebelah kanan, temukan bidang UpVotePost yang baru dibuat pada tipe Mutasi, lalu pilih Lampirkan.
- Di menu Action, pilih Update runtime, lalu pilih Unit Resolver (hanya VTL).
- Dalam nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
```

```
    "version" : "2017-02-28",
    "operation" : "UpdateItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($context.arguments.id)
    },
    "update" : {
        "expression" : "ADD ups :plusOne, version :plusOne",
        "expressionValues" : {
            ":plusOne" : { "N" : 1 }
        }
    }
}
```

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
$utils.toJson($context.result)
```

- Pilih Save (Simpan).
- Di panel Jenis data di sebelah kanan, temukan **`downvotePost`** bidang yang baru dibuat pada tipe Mutasi, lalu pilih Lampirkan.
- Dalam nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
    "version" : "2017-02-28",
    "operation" : "UpdateItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($context.arguments.id)
    },
    "update" : {
        "expression" : "ADD downs :plusOne, version :plusOne",
        "expressionValues" : {
            ":plusOne" : { "N" : 1 }
        }
    }
}
```

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
$utils.toJson($context.result)
```

- Pilih Save (Simpan).

Panggil API untuk upvote dan downvote sebuah Post

Sekarang resolver baru telah diatur, AWS AppSync tahu bagaimana menerjemahkan masuk `upvotePost` atau `downvote` mutasi ke operasi DynamoDB. UpdateItem Anda sekarang dapat menjalankan mutasi untuk upvote atau downvote posting yang Anda buat sebelumnya.

- Pilih tab Kueri.
- Di panel Kueri, tempel mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya.

```
mutation votePost {
  upvotePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Posting diperbarui di DynamoDB dan akan muncul di panel hasil di sebelah kanan panel kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "upvotePost": {
      "id": "123",
      "author": "A new author",
      "title": "An empty story",
      "content": null,
      "url": "https://aws.amazon.com/appsync/",
      "ups": 6,
      "downs": 0,
      "version": 4
    }
  }
}
```

- Pilih Jalankan kueri beberapa kali lagi. Anda akan melihat `version` bidang `ups` dan bertambah 1 setiap kali Anda menjalankan kueri.
- Ubah kueri untuk memanggil `downvotePost` mutasi sebagai berikut:

```
mutation votePost {
  downvotePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Pilih Execute query (tombol putar oranye). Kali ini, Anda akan melihat `version` bidang `downs` dan bertambah 1 setiap kali Anda menjalankan kueri.

```
{
  "data": {
    "downvotePost": {
      "id": "123",
      "author": "A new author",
      "title": "An empty story",
      "content": null,
      "url": "https://aws.amazon.com/appsync/",
      "ups": 6,
      "downs": 4,
      "version": 12
    }
  }
}
```

Menyiapkan DeletePost Resolver (DynamoDB) DeleteItem

Mutasi berikutnya yang ingin Anda atur adalah menghapus posting. Anda akan melakukan ini menggunakan operasi `DeleteItem` DynamoDB.

- Pilih tab Skema.

- Di panel Skema, ubah `Mutation` tipe untuk menambahkan `deletePost` mutasi baru sebagai berikut:

```
type Mutation {
    deletePost(id: ID!, expectedVersion: Int): Post
    upvotePost(id: ID!): Post
    downvotePost(id: ID!): Post
    updatePost(
        id: ID!,
        author: String,
        title: String,
        content: String,
        url: String,
        expectedVersion: Int!
    ): Post
    addPost(
        author: String!,
        title: String!,
        content: String!,
        url: String!
    ): Post!
}
```

  Kali ini Anda membuat `expectedVersion` bidang opsional, yang dijelaskan nanti ketika Anda menambahkan template pemetaan permintaan.

- Pilih Save (Simpan).
- Di panel Jenis data di sebelah kanan, temukan bidang hapus yang baru dibuat pada Jenis mutasi, lalu pilih Lampirkan.
- Di menu Action, pilih Update runtime, lalu pilih Unit Resolver (hanya VTL).
- Dalam nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
    "version" : "2017-02-28",
    "operation" : "DeleteItem",
    "key": {
        "id": $util.dynamodb.toDynamoDBJson($context.arguments.id)
    }
    #if( $context.arguments.containsKey("expectedVersion") )
        ,"condition" : {
```

```
            "expression"       : "attribute_not_exists(id) OR version
 = :expectedVersion",
            "expressionValues" : {
                ":expectedVersion" :
 $util.dynamodb.toDynamoDBJson($context.arguments.expectedVersion)
            }
        }
    #end
}
```

Catatan: `expectedVersion` Argumen adalah argumen opsional. Jika pemanggil menetapkan `expectedVersion` argumen dalam permintaan, template menambahkan kondisi yang hanya memungkinkan `DeleteItem` permintaan untuk berhasil jika item sudah dihapus atau jika `version` atribut posting di DynamoDB sama persis dengan. `expectedVersion` Jika ditinggalkan, tidak ada ekspresi kondisi yang ditentukan pada `DeleteItem` permintaan. Ini berhasil terlepas dari nilai`version`, atau apakah item ada atau tidak di DynamoDB.

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
$utils.toJson($context.result)
```

Catatan: Meskipun Anda menghapus item, Anda dapat mengembalikan item yang telah dihapus, jika belum dihapus.

- Pilih Save (Simpan).

Untuk info selengkapnya tentang pemetaan `DeleteItem` permintaan, lihat dokumentasi DeleteItemreferensi.

Panggil API untuk Menghapus Posting

Sekarang resolver telah diatur, AWS AppSync tahu bagaimana menerjemahkan `delete` mutasi yang masuk ke operasi DynamoDB. `DeleteItem` Anda sekarang dapat menjalankan mutasi untuk menghapus sesuatu di tabel.

- Pilih tab Kueri.
- Di panel Kueri, tempel mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya.

```
mutation deletePost {
```

```
  deletePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Posting dihapus dari DynamoDB. Perhatikan bahwa AWS AppSync mengembalikan nilai item yang telah dihapus dari DynamoDB, yang akan muncul di panel hasil di sebelah kanan panel kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "deletePost": {
      "id": "123",
      "author": "A new author",
      "title": "An empty story",
      "content": null,
      "url": "https://aws.amazon.com/appsync/",
      "ups": 6,
      "downs": 4,
      "version": 12
    }
  }
}
```

Nilai hanya dikembalikan jika panggilan ini adalah salah satu yang `deletePost` benar-benar menghapusnya dari DynamoDB.

- Pilih Jalankan kueri lagi.
- Panggilan masih berhasil, tetapi tidak ada nilai yang dikembalikan.

```
{
  "data": {
    "deletePost": null
```

```
  }
}
```

Sekarang mari kita coba menghapus posting, tapi kali ini menentukan. `expectedValue` Pertama-tama, Anda harus membuat posting baru karena Anda baru saja menghapus yang telah Anda kerjakan sejauh ini.

- Di panel Kueri, tempel mutasi berikut:

```
mutation addPost {
  addPost(
    id:123
    author: "AUTHORNAME"
    title: "Our second post!"
    content: "A new post."
    url: "https://aws.amazon.com/appsync/"
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Hasil posting yang baru dibuat akan muncul di panel hasil di sebelah kanan panel kueri. Catat objek `id` yang baru dibuat karena Anda membutuhkannya hanya dalam beberapa saat. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "addPost": {
      "id": "123",
      "author": "AUTHORNAME",
      "title": "Our second post!",
      "content": "A new post.",
      "url": "https://aws.amazon.com/appsync/",
```

```
      "ups": 1,
      "downs": 0,
      "version": 1
    }
  }
}
```

Sekarang mari kita coba menghapus posting itu, tetapi masukkan nilai yang salah untuk`expectedVersion`:

- Di panel Kueri, tempel mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya.

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 9999
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Pilih Execute query (tombol putar oranye).

```
{
  "data": {
    "deletePost": null
  },
  "errors": [
    {
      "path": [
        "deletePost"
      ],
      "data": {
        "id": "123",
```

```
        "author": "AUTHORNAME",
        "title": "Our second post!",
        "content": "A new post.",
        "url": "https://aws.amazon.com/appsync/",
        "ups": 1,
        "downs": 0,
        "version": 1
      },
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "message": "The conditional request failed (Service: AmazonDynamoDBv2;
 Status Code: 400; Error Code: ConditionalCheckFailedException; Request ID:
 ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ)"
    }
  ]
}
```

Permintaan gagal karena ekspresi kondisi dievaluasi menjadi false: nilai untuk `version` posting di DynamoDB tidak cocok dengan `expectedValue` yang ditentukan dalam argumen. Nilai objek saat ini dikembalikan di `data` bidang di `errors` bagian respons GraphQL.

- Coba lagi permintaannya, tetapi `expectedVersion` perbaiki:

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Kali ini permintaan berhasil, dan nilai yang dihapus dari DynamoDB dikembalikan:

```
{
  "data": {
    "deletePost": {
      "id": "123",
      "author": "AUTHORNAME",
      "title": "Our second post!",
      "content": "A new post.",
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
      "version": 1
    }
  }
}
```

- Pilih Jalankan kueri lagi.
- Panggilan masih berhasil, tetapi kali ini tidak ada nilai yang dikembalikan karena posting sudah dihapus di DynamoDB.

```
{
  "data": {
    "deletePost": null
  }
}
```

Menyiapkan AllPost Resolver (DynamoDB Scan)

Sejauh ini API hanya berguna jika Anda mengetahui setiap posting yang ingin Anda lihat. `id` Mari tambahkan resolver baru yang mengembalikan semua posting dalam tabel.

- Pilih tab Skema.
- Di panel Skema, ubah `Query` jenis untuk menambahkan `allPost` kueri baru sebagai berikut:

```
type Query {
    allPost(count: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
```

```
}
```

- Tambahkan `PaginationPosts` tipe baru:

```
type PaginatedPosts {
    posts: [Post!]!
    nextToken: String
}
```

- Pilih Save (Simpan).
- Di panel tipe Data di sebelah kanan, temukan bidang AllPost yang baru dibuat pada tipe Query, lalu pilih Lampirkan.
- Di menu Action, pilih Update runtime, lalu pilih Unit Resolver (hanya VTL).
- Di nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
    "version" : "2017-02-28",
    "operation" : "Scan"
    #if( ${context.arguments.count} )
        ,"limit": $util.toJson($context.arguments.count)
    #end
    #if( ${context.arguments.nextToken} )
        ,"nextToken": $util.toJson($context.arguments.nextToken)
    #end
}
```

Penyelesai ini memiliki dua argumen opsional:`count`, yang menentukan jumlah maksimum item yang akan dikembalikan dalam satu panggilan, dan`nextToken`, yang dapat digunakan untuk mengambil set hasil berikutnya (Anda akan menunjukkan dari mana nilai untuk `nextToken` berasal nanti).

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
{
    "posts": $utils.toJson($context.result.items)
    #if( ${context.result.nextToken} )
        ,"nextToken": $util.toJson($context.result.nextToken)
    #end
}
```

Catatan: Template pemetaan respons ini berbeda dari yang lainnya sejauh ini. Hasil `allPost` kueri adalah`PaginatedPosts`, yang berisi daftar posting dan token pagination. Bentuk objek ini berbeda dengan apa yang dikembalikan dari AWS AppSync DynamoDB Resolver: daftar posting dipanggil dalam hasil AWS AppSync DynamoDB Resolver, tetapi dipanggil `items` dalam. `posts` `PaginatedPosts`

- Pilih Save (Simpan).

Untuk informasi selengkapnya tentang pemetaan `Scan` permintaan, lihat dokumentasi referensi Pindai.

Panggil API untuk Memindai Semua Posting

Sekarang resolver telah diatur, AWS AppSync tahu bagaimana menerjemahkan `allPost` query masuk ke operasi DynamoDB. `Scan` Anda sekarang dapat memindai tabel untuk mengambil semua posting.

Sebelum Anda dapat mencobanya, Anda perlu mengisi tabel dengan beberapa data karena Anda telah menghapus semua yang telah Anda kerjakan sejauh ini.

- Pilih tab Kueri.
- Di panel Kueri, tempel mutasi berikut:

```
mutation addPost {
  post1: addPost(id:1 author: "AUTHORNAME" title: "A series of posts, Volume 1"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post2: addPost(id:2 author: "AUTHORNAME" title: "A series of posts, Volume 2"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post3: addPost(id:3 author: "AUTHORNAME" title: "A series of posts, Volume 3"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post4: addPost(id:4 author: "AUTHORNAME" title: "A series of posts, Volume 4"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post5: addPost(id:5 author: "AUTHORNAME" title: "A series of posts, Volume 5"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post6: addPost(id:6 author: "AUTHORNAME" title: "A series of posts, Volume 6"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post7: addPost(id:7 author: "AUTHORNAME" title: "A series of posts, Volume 7"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
  post8: addPost(id:8 author: "AUTHORNAME" title: "A series of posts, Volume 8"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
```

```
  post9: addPost(id:9 author: "AUTHORNAME" title: "A series of posts, Volume 9"
 content: "Some content" url: "https://aws.amazon.com/appsync/" ) { title }
}
```

- Pilih Execute query (tombol putar oranye).

Sekarang, mari kita pindai tabel, mengembalikan lima hasil sekaligus.

- Di panel Kueri, tempel kueri berikut:

```
query allPost {
  allPost(count: 5) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Lima posting pertama akan muncul di panel hasil di sebelah kanan panel kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPost": {
      "posts": [
        {
          "id": "5",
          "title": "A series of posts, Volume 5"
        },
        {
          "id": "1",
          "title": "A series of posts, Volume 1"
        },
        {
          "id": "6",
          "title": "A series of posts, Volume 6"
        },
        {
          "id": "9",
```

```
          "title": "A series of posts, Volume 9"
        },
        {
          "id": "7",
          "title": "A series of posts, Volume 7"
        }
      ],
      "nextToken":
 "eyJ2ZXJzaW9uIjoxLCJ0b2tlbiI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eThOSmRvcG9ad2RHYjI
    }
  }
}
```

Anda mendapat lima hasil dan `nextToken` yang dapat Anda gunakan untuk mendapatkan hasil berikutnya.

- Perbarui `allPost` kueri untuk menyertakan `nextToken` dari kumpulan hasil sebelumnya:

```
query allPost {
  allPost(
    count: 5
    nextToken:
 "eyJ2ZXJzaW9uIjoxLCJ0b2tlbiI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eThOSmRvcG9ad2RHYjI
  ) {
    posts {
      id
      author
    }
    nextToken
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Empat posting yang tersisa akan muncul di panel hasil di sebelah kanan panel kueri. Tidak ada `nextToken` dalam rangkaian hasil ini karena Anda telah membaca semua sembilan posting, dengan tidak ada yang tersisa. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPost": {
      "posts": [
```

```
        {
          "id": "2",
          "title": "A series of posts, Volume 2"
        },
        {
          "id": "3",
          "title": "A series of posts, Volume 3"
        },
        {
          "id": "4",
          "title": "A series of posts, Volume 4"
        },
        {
          "id": "8",
          "title": "A series of posts, Volume 8"
        }
      ],
      "nextToken": null
    }
  }
}
```

Menyiapkan Resolver allPostsBy Penulis (Query DynamoDB)

Selain memindai DynamoDB untuk semua posting, Anda juga dapat meminta DynamoDB untuk mengambil posting yang dibuat oleh penulis tertentu. Tabel DynamoDB yang Anda buat sebelumnya sudah memiliki panggilan `GlobalSecondaryIndex` yang dapat Anda `author-index` gunakan dengan operasi DynamoDB untuk mengambil `Query` semua posting yang dibuat oleh penulis tertentu.

- Pilih tab Skema.
- Di panel Skema, ubah `Query` jenis untuk menambahkan `allPostsByAuthor` kueri baru sebagai berikut:

```
type Query {
    allPostsByAuthor(author: String!, count: Int, nextToken: String): PaginatedPosts!
    allPost(count: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
}
```

Catatan: Ini menggunakan `PaginatedPosts` jenis yang sama yang Anda gunakan dengan `allPost` kueri.

- Pilih Save (Simpan).
- Di panel Jenis data di sebelah kanan, temukan bidang allPostsByPenulis yang baru dibuat pada jenis kueri, lalu pilih Lampirkan.
- Di menu Action, pilih Update runtime, lalu pilih Unit Resolver (hanya VTL).
- Di nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
    "version" : "2017-02-28",
    "operation" : "Query",
    "index" : "author-index",
    "query" : {
      "expression": "author = :author",
        "expressionValues" : {
          ":author" : $util.dynamodb.toDynamoDBJson($context.arguments.author)
        }
    }
    #if( ${context.arguments.count} )
        ,"limit": $util.toJson($context.arguments.count)
    #end
    #if( ${context.arguments.nextToken} )
        ,"nextToken": "${context.arguments.nextToken}"
    #end
}
```

Seperti `allPost` resolver, resolver ini memiliki dua argumen opsional:`count`, yang menentukan jumlah maksimum item yang akan dikembalikan dalam satu panggilan, dan`nextToken`, yang dapat digunakan untuk mengambil set hasil berikutnya (nilai untuk `nextToken` dapat diperoleh dari panggilan sebelumnya).

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
{
    "posts": $utils.toJson($context.result.items)
    #if( ${context.result.nextToken} )
        ,"nextToken": $util.toJson($context.result.nextToken)
    #end
```

```
}
```

Catatan: Ini adalah template pemetaan respons yang sama yang Anda gunakan dalam `allPost` resolver.

- Pilih Save (Simpan).

Untuk informasi selengkapnya tentang pemetaan `Query` permintaan, lihat dokumentasi referensi kueri.

Panggil API untuk Menanyakan Semua Postingan oleh Penulis

Sekarang resolver telah diatur, AWS AppSync tahu bagaimana menerjemahkan `allPostsByAuthor` mutasi yang masuk ke operasi DynamoDB terhadap indeks. `Query` `author-index` Anda sekarang dapat meminta tabel untuk mengambil semua posting oleh penulis tertentu.

Sebelum Anda melakukan itu, bagaimanapun, mari kita mengisi tabel dengan beberapa posting lagi, karena setiap posting sejauh ini memiliki penulis yang sama.

- Pilih tab Kueri.
- Di panel Kueri, tempel mutasi berikut:

```
mutation addPost {
  post1: addPost(id:10 author: "Nadia" title: "The cutest dog in the world" content:
 "So cute. So very, very cute." url: "https://aws.amazon.com/appsync/" ) { author,
 title }
  post2: addPost(id:11 author: "Nadia" title: "Did you know...?" content: "AppSync
 works offline?" url: "https://aws.amazon.com/appsync/" ) { author, title }
  post3: addPost(id:12 author: "Steve" title: "I like GraphQL" content: "It's great"
 url: "https://aws.amazon.com/appsync/" ) { author, title }
}
```

- Pilih Execute query (tombol putar oranye).

Sekarang, mari kita menanyakan tabel, mengembalikan semua posting yang ditulis oleh. `Nadia`

- Di panel Kueri, tempel kueri berikut:

```
query allPostsByAuthor {
  allPostsByAuthor(author: "Nadia") {
```

```
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Semua posting yang ditulis oleh akan `Nadia` muncul di panel hasil di sebelah kanan panel query. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you know...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

Pagination bekerja untuk hal yang `Query` sama seperti yang dilakukannya. `Scan` Sebagai contoh, mari kita cari semua posting dengan`AUTHORNAME`, mendapatkan lima sekaligus.

- Di panel Kueri, tempel kueri berikut:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    count: 5
  ) {
    posts {
```

```
      id
      title
    }
    nextToken
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Semua posting yang ditulis oleh akan AUTHORNAME muncul di panel hasil di sebelah kanan panel query. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "6",
          "title": "A series of posts, Volume 6"
        },
        {
          "id": "4",
          "title": "A series of posts, Volume 4"
        },
        {
          "id": "2",
          "title": "A series of posts, Volume 2"
        },
        {
          "id": "7",
          "title": "A series of posts, Volume 7"
        },
        {
          "id": "1",
          "title": "A series of posts, Volume 1"
        }
      ],
      "nextToken":
 "eyJ2ZXJzaW9uIjoxLCJ0b2tlbiI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eThOSmRvcG9ad2RHYjI
    }
  }
}
```

- Perbarui `nextToken` argumen dengan nilai yang dikembalikan dari kueri sebelumnya sebagai berikut:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    count: 5
    nextToken:
 "eyJ2ZXJzaW9uIjoxLCJ0b2tlbiI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eThOSmRvcG9ad2RHYjI
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Posting yang tersisa yang ditulis oleh AUTHORNAME akan muncul di panel hasil di sebelah kanan panel kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "8",
          "title": "A series of posts, Volume 8"
        },
        {
          "id": "5",
          "title": "A series of posts, Volume 5"
        },
        {
          "id": "3",
          "title": "A series of posts, Volume 3"
        },
        {
          "id": "9",
          "title": "A series of posts, Volume 9"
        }
```

```
      ],
      "nextToken": null
    }
  }
}
```

Menggunakan Set

Sampai titik ini `Post` tipe telah menjadi objek kunci/nilai datar. Anda juga dapat memodelkan objek kompleks dengan resolver AWS AppSyncDynamo DB, seperti set, daftar, dan peta.

Mari kita perbarui `Post` jenis untuk menyertakan tag. Sebuah posting dapat memiliki 0 atau lebih tag, yang disimpan di DynamoDB sebagai String Set. Anda juga akan menyiapkan beberapa mutasi untuk menambah dan menghapus tag, dan kueri baru untuk memindai posting dengan tag tertentu.

- Pilih tab Skema.
- Di panel Skema, ubah `Post` jenis untuk menambahkan `tags` bidang baru sebagai berikut:

```
type Post {
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
  tags: [String!]
}
```

- Di panel Skema, ubah `Query` jenis untuk menambahkan `allPostsByTag` kueri baru sebagai berikut:

```
type Query {
  allPostsByTag(tag: String!, count: Int, nextToken: String): PaginatedPosts!
  allPostsByAuthor(author: String!, count: Int, nextToken: String): PaginatedPosts!
  allPost(count: Int, nextToken: String): PaginatedPosts!
  getPost(id: ID): Post
}
```

- Di panel Skema, ubah `Mutation` tipe untuk menambahkan baru `addTag` dan `removeTag` mutasi sebagai berikut:

```
type Mutation {
  addTag(id: ID!, tag: String!): Post
  removeTag(id: ID!, tag: String!): Post
  deletePost(id: ID!, expectedVersion: Int): Post
  upvotePost(id: ID!): Post
  downvotePost(id: ID!): Post
  updatePost(
    id: ID!,
    author: String,
    title: String,
    content: String,
    url: String,
    expectedVersion: Int!
  ): Post
  addPost(
    author: String!,
    title: String!,
    content: String!,
    url: String!
  ): Post!
}
```

- Pilih Save (Simpan).
- Di panel Tipe data di sebelah kanan, temukan bidang allPostsByTag yang baru dibuat pada Jenis kueri, lalu pilih Lampirkan.
- Di nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "filter": {
      "expression": "contains (tags, :tag)",
        "expressionValues": {
          ":tag": $util.dynamodb.toDynamoDBJson($context.arguments.tag)
        }
    }
    #if( ${context.arguments.count} )
        ,"limit": $util.toJson($context.arguments.count)
```

```
    #end
    #if( ${context.arguments.nextToken} )
        ,"nextToken": $util.toJson($context.arguments.nextToken)
    #end
}
```

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
{
    "posts": $utils.toJson($context.result.items)
    #if( ${context.result.nextToken} )
        ,"nextToken": $util.toJson($context.result.nextToken)
    #end
}
```

- Pilih Save (Simpan).
- Di panel Jenis data di sebelah kanan, temukan bidang AddTag yang baru dibuat pada tipe Mutasi, lalu pilih Lampirkan.
- Di nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
    "version" : "2017-02-28",
    "operation" : "UpdateItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($context.arguments.id)
    },
    "update" : {
        "expression" : "ADD tags :tags, version :plusOne",
        "expressionValues" : {
            ":tags" : { "SS": [ $util.toJson($context.arguments.tag) ] },
            ":plusOne" : { "N" : 1 }
        }
    }
}
```

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
$utils.toJson($context.result)
```

- Pilih Save (Simpan).

- Di panel tipe Data di sebelah kanan, temukan bidang RemoveTag yang baru dibuat pada tipe Mutasi, lalu pilih Lampirkan.
- Di nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
    "version" : "2017-02-28",
    "operation" : "UpdateItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($context.arguments.id)
    },
    "update" : {
        "expression" : "DELETE tags :tags ADD version :plusOne",
        "expressionValues" : {
            ":tags" : { "SS": [ $util.toJson($context.arguments.tag) ] },
            ":plusOne" : { "N" : 1 }
        }
    }
}
```

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
$utils.toJson($context.result)
```

- Pilih Save (Simpan).

Panggil API untuk Bekerja dengan Tag

Sekarang setelah Anda menyiapkan resolver, AWS AppSync tahu cara menerjemahkan masuk `addTagremoveTag`, dan permintaan `allPostsByTag` ke DynamoDB dan operasi. `UpdateItem` `Scan`

Untuk mencobanya, mari pilih salah satu posting yang Anda buat sebelumnya. Misalnya, mari kita gunakan posting yang ditulis oleh. `Nadia`

- Pilih tab Kueri.
- Di panel Kueri, tempel kueri berikut:

```
query allPostsByAuthor {
  allPostsByAuthor(
```

```
    author: "Nadia"
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Semua posting Nadia akan muncul di panel hasil di sebelah kanan panel kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you known...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

- Mari kita gunakan yang dengan judul`"The cutest dog in the world"`. Catat `id` karena Anda akan menggunakannya nanti.

Sekarang mari kita coba menambahkan dog tag.

- Di panel Kueri, tempel mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya.

```
mutation addTag {
  addTag(id:10 tag: "dog") {
```

```
    id
    title
    tags
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Posting diperbarui dengan tag baru.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

Anda dapat menambahkan lebih banyak tag sebagai berikut:

- Perbarui mutasi untuk mengubah `tag` argumen menjadi`puppy`.

```
mutation addTag {
  addTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Posting diperbarui dengan tag baru.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
```

```
      "tags": [
        "dog",
        "puppy"
      ]
    }
  }
}
```

Anda juga dapat menghapus tag:

- Di panel Kueri, tempel mutasi berikut. Anda juga perlu memperbarui `id` argumen ke nilai yang Anda catat sebelumnya.

```
mutation removeTag {
  removeTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Posting diperbarui dan puppy tag dihapus.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

Anda juga dapat mencari semua posting yang memiliki tag:

- Di panel Kueri, tempel kueri berikut:

```
query allPostsByTag {
  allPostsByTag(tag: "dog") {
    posts {
      id
      title
      tags
    }
    nextToken
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Semua posting yang memiliki dog tag dikembalikan sebagai berikut:

```
{
  "data": {
    "allPostsByTag": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world",
          "tags": [
            "dog",
            "puppy"
          ]
        }
      ],
      "nextToken": null
    }
  }
}
```

Menggunakan Daftar dan Peta

Selain menggunakan set DynamoDB, Anda juga dapat menggunakan daftar dan peta DynamoDB untuk memodelkan data kompleks dalam satu objek.

Mari tambahkan kemampuan untuk menambahkan komentar ke posting. Ini akan dimodelkan sebagai daftar objek peta pada objek di DynamoDB. `Post`

Catatan: dalam aplikasi nyata, Anda akan memodelkan komentar di tabel mereka sendiri. Untuk tutorial ini, Anda hanya akan menambahkannya di `Post` tabel.

- Pilih tab Skema.
- Di panel Skema, tambahkan `Comment` tipe baru sebagai berikut:

```
type Comment {
    author: String!
    comment: String!
}
```

- Di panel Skema, ubah `Post` jenis untuk menambahkan `comments` bidang baru sebagai berikut:

```
type Post {
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
  tags: [String!]
  comments: [Comment!]
}
```

- Di panel Skema, ubah `Mutation` tipe untuk menambahkan `addComment` mutasi baru sebagai berikut:

```
type Mutation {
  addComment(id: ID!, author: String!, comment: String!): Post
  addTag(id: ID!, tag: String!): Post
  removeTag(id: ID!, tag: String!): Post
  deletePost(id: ID!, expectedVersion: Int): Post
  upvotePost(id: ID!): Post
  downvotePost(id: ID!): Post
  updatePost(
    id: ID!,
    author: String,
    title: String,
    content: String,
    url: String,
```

```
    expectedVersion: Int!
  ): Post
  addPost(
    author: String!,
    title: String!,
    content: String!,
    url: String!
  ): Post!
}
```

- Pilih Save (Simpan).
- Di panel tipe Data di sebelah kanan, temukan bidang AddComment yang baru dibuat pada tipe Mutasi, lalu pilih Lampirkan.
- Di nama sumber data, pilih PostDynamoDbTable.
- Di Konfigurasikan templat pemetaan permintaan, tempel yang berikut ini:

```
{
  "version" : "2017-02-28",
  "operation" : "UpdateItem",
  "key" : {
    "id" : $util.dynamodb.toDynamoDBJson($context.arguments.id)
  },
  "update" : {
    "expression" : "SET comments =
 list_append(if_not_exists(comments, :emptyList), :newComment) ADD version :plusOne",
    "expressionValues" : {
      ":emptyList": { "L" : [] },
      ":newComment" : { "L" : [
        { "M": {
          "author": $util.dynamodb.toDynamoDBJson($context.arguments.author),
          "comment": $util.dynamodb.toDynamoDBJson($context.arguments.comment)
          }
        }
      ] },
      ":plusOne" : $util.dynamodb.toDynamoDBJson(1)
    }
  }
}
```

Ekspresi pembaruan ini akan menambahkan daftar yang berisi komentar baru kami ke `comments` daftar yang ada. Jika daftar belum ada, itu akan dibuat.

- Di Konfigurasikan template pemetaan respons, tempel yang berikut ini:

```
$utils.toJson($context.result)
```

- Pilih Save (Simpan).

Panggil API untuk Menambahkan Komentar

Sekarang setelah Anda menyiapkan resolver, AWS AppSync tahu cara menerjemahkan permintaan masuk `addComment` ke dalam operasi DynamoDB. `UpdateItem`

Mari kita coba dengan menambahkan komentar ke posting yang sama dengan yang Anda tambahkan tag.

- Pilih tab Kueri.
- Di panel Kueri, tempel kueri berikut:

```
mutation addComment {
  addComment(
    id:10
    author: "Steve"
    comment: "Such a cute dog."
  ) {
    id
    comments {
      author
      comment
    }
  }
}
```

- Pilih Execute query (tombol putar oranye).
- Semua posting Nadia akan muncul di panel hasil di sebelah kanan panel kueri. Itu terlihat serupa dengan yang berikut ini:

```
{
  "data": {
    "addComment": {
      "id": "10",
      "comments": [
        {
```

```
            "author": "Steve",
            "comment": "Such a cute dog."
          }
        ]
      }
    }
}
```

Jika Anda menjalankan permintaan beberapa kali, beberapa komentar akan ditambahkan ke daftar.

Kesimpulan

Dalam tutorial ini, Anda telah membangun sebuah API yang memungkinkan kita memanipulasi objek Post di AWS AppSync DynamoDB menggunakan dan GraphQL. Untuk informasi selengkapnya, lihat Referensi Template Pemetaan Resolver.

Untuk membersihkan, Anda dapat menghapus AppSync GraphQL API dari konsol.

Untuk menghapus tabel DynamoDB dan peran IAM yang Anda buat untuk tutorial ini, Anda dapat menjalankan yang berikut ini untuk menghapus `AWSAppSyncTutorialForAmazonDynamoDB` tumpukan, atau mengunjungi AWS CloudFormation konsol dan menghapus tumpukan:

```
aws cloudformation delete-stack \
    --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

Tutorial: Penyelesai Lambda

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Anda dapat menggunakan AWS Lambda dengan AWS AppSync untuk menyelesaikan bidang GraphQL apa pun. Misalnya, kueri GraphQL mungkin mengirim panggilan ke instance Amazon Relational Database Service (Amazon RDS), dan mutasi GraphQL mungkin menulis ke aliran Amazon Kinesis. Di bagian ini, kami akan menunjukkan cara menulis fungsi Lambda yang melakukan logika bisnis berdasarkan pemanggilan operasi lapangan GraphQL.

Buat fungsi Lambda

Contoh berikut menunjukkan fungsi Lambda ditulis dalam `Node.js` yang melakukan operasi yang berbeda pada posting blog sebagai bagian dari aplikasi posting blog.

```
exports.handler = (event, context, callback) => {
    console.log("Received event {}", JSON.stringify(event, 3));
    var posts = {
         "1": {"id": "1", "title": "First book", "author": "Author1", "url": "https://
amazon.com/", "content": "SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR
 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1", "ups": "100",
 "downs": "10"},
         "2": {"id": "2", "title": "Second book", "author": "Author2", "url": "https://
amazon.com", "content": "SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT", "ups":
 "100", "downs": "10"},
         "3": {"id": "3", "title": "Third book", "author": "Author3", "url": null,
 "content": null, "ups": null, "downs": null },
         "4": {"id": "4", "title": "Fourth book", "author": "Author4", "url": "https://
www.amazon.com/", "content": "SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4", "ups": "1000", "downs": "0"},
         "5": {"id": "5", "title": "Fifth book", "author": "Author5", "url": "https://
www.amazon.com/", "content": "SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT
 AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT", "ups": "50", "downs": "0"} };

    var relatedPosts = {
        "1": [posts['4']],
        "2": [posts['3'], posts['5']],
        "3": [posts['2'], posts['1']],
        "4": [posts['2'], posts['1']],
        "5": []
    };

    console.log("Got an Invoke Request.");
    switch(event.field) {
        case "getPost":
            var id = event.arguments.id;
            callback(null, posts[id]);
            break;
        case "allPosts":
            var values = [];
            for(var d in posts){
                values.push(posts[d]);
```

```
            }
            callback(null, values);
            break;
        case "addPost":
            // return the arguments back
            callback(null, event.arguments);
            break;
        case "addPostErrorWithData":
            var id = event.arguments.id;
            var result = posts[id];
            // attached additional error information to the post
            result.errorMessage = 'Error with the mutation, data has changed';
            result.errorType = 'MUTATION_ERROR';
            callback(null, result);
            break;
        case "relatedPosts":
            var id = event.source.id;
            callback(null, relatedPosts[id]);
            break;
        default:
            callback("Unknown field, unable to resolve" + event.field, null);
            break;
    }
};
```

Fungsi Lambda ini mengambil posting dengan ID, menambahkan posting, mengambil daftar posting, dan mengambil posting terkait untuk posting tertentu.

Catatan: Fungsi Lambda menggunakan `switch` pernyataan `event.field` untuk menentukan bidang mana yang sedang diselesaikan.

Buat fungsi Lambda ini menggunakan Konsol AWS Manajemen atau tumpukan. AWS CloudFormation Untuk membuat fungsi dari CloudFormation tumpukan, Anda dapat menggunakan perintah AWS Command Line Interface (AWS CLI) berikut:

```
aws cloudformation create-stack --stack-name AppSyncLambdaExample \
--template-url https://s3.us-west-2.amazonaws.com/awsappsync/resources/lambda/
LambdaCFTemplate.yaml \
--capabilities CAPABILITY_NAMED_IAM
```

Anda juga dapat meluncurkan AWS CloudFormation tumpukan di AWS Wilayah AS Barat (Oregon) di AWS akun Anda dari sini:

Konfigurasikan sumber data untuk Lambda

Setelah Anda membuat fungsi Lambda, navigasikan ke GraphQL API di AWS AppSync konsol, lalu pilih tab Sumber Data.

Pilih Buat sumber data, masukkan nama sumber data yang ramah (misalnya,**`Lambda`**), dan kemudian untuk tipe sumber data, pilih AWS Lambdafungsi. Untuk Region, pilih Region yang sama dengan fungsi Anda. (Jika Anda membuat fungsi dari CloudFormation tumpukan yang disediakan, fungsinya mungkin ada di US-WEST-2.) Untuk Fungsi ARN, pilih Amazon Resource Name (ARN) dari fungsi Lambda Anda.

Setelah memilih fungsi Lambda, Anda dapat membuat peran baru AWS Identity and Access Management (IAM) (yang AWS AppSync menetapkan izin yang sesuai) atau memilih peran yang ada yang memiliki kebijakan sebaris berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:REGION:ACCOUNTNUMBER:function/LAMBDA_FUNCTION"
        }
    ]
}
```

Anda juga harus mengatur hubungan kepercayaan dengan AWS AppSync peran IAM sebagai berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "appsync.amazonaws.com"
```

```
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Buat skema GraphQL

Sekarang sumber data terhubung ke fungsi Lambda Anda, buat skema GraphQL.

Dari editor skema di AWS AppSync konsol, pastikan skema Anda cocok dengan skema berikut:

```
schema {
    query: Query
    mutation: Mutation
}

type Query {
    getPost(id:ID!): Post
    allPosts: [Post]
}

type Mutation {
    addPost(id: ID!, author: String!, title: String, content: String, url: String):
 Post!
}

type Post {
    id: ID!
    author: String!
    title: String
    content: String
    url: String
    ups: Int
    downs: Int
    relatedPosts: [Post]
}
```

Konfigurasikan resolver

Setelah mendaftarkan sumber data Lambda dan skema GraphQL yang valid, Anda dapat menghubungkan bidang GraphQL ke sumber data Lambda menggunakan resolver.

Untuk membuat resolver, Anda memerlukan template pemetaan. Untuk mempelajari lebih lanjut tentang templat pemetaan, lihatResolver Mapping Template Overview.

Untuk informasi selengkapnya tentang templat pemetaan Lambda, lihat. Resolver mapping template reference for Lambda

Pada langkah ini, Anda melampirkan resolver ke fungsi Lambda untuk bidang berikut:`getPost(id:ID!): Post`,,, `allPosts: [Post]` dan. `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!` `Post.relatedPosts: [Post]`

Dari editor skema di AWS AppSync konsol, di sisi kanan, pilih Lampirkan Resolver untuk. `getPost(id:ID!): Post`

Kemudian, di menu Action, pilih Update runtime, lalu pilih Unit Resolver (hanya VTL).

Setelah itu, pilih sumber data Lambda Anda. Di bagian template pemetaan permintaan, pilih Invoke And Forward Arguments.

Ubah `payload` objek untuk menambahkan nama bidang. Template Anda akan terlihat seperti berikut:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments":  $utils.toJson($context.arguments)
    }
}
```

Di bagian template pemetaan respons, pilih Kembalikan Hasil Lambda.

Dalam hal ini, gunakan template dasar apa adanya. Seharusnya terlihat seperti berikut:

```
$utils.toJson($context.result)
```

Pilih Save (Simpan). Anda telah berhasil melampirkan resolver pertama Anda. Ulangi operasi ini untuk bidang yang tersisa sebagai berikut:

Untuk templat pemetaan `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!` permintaan:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "addPost",
        "arguments":  $utils.toJson($context.arguments)
    }
}
```

Untuk template pemetaan `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!` respons:

```
$utils.toJson($context.result)
```

Untuk templat pemetaan `allPosts: [Post]` permintaan:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "allPosts"
    }
}
```

Untuk template pemetaan `allPosts: [Post]` respons:

```
$utils.toJson($context.result)
```

Untuk templat pemetaan `Post.relatedPosts: [Post]` permintaan:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "relatedPosts",
        "source":  $utils.toJson($context.source)
    }
}
```

Untuk template pemetaan `Post.relatedPosts: [Post]` respons:

```
$utils.toJson($context.result)
```

Uji GraphQL API

Sekarang fungsi Lambda Anda terhubung ke resolver GraphQL, Anda dapat menjalankan beberapa mutasi dan kueri menggunakan konsol atau aplikasi klien.

Di sisi kiri AWS AppSync konsol, pilih Kueri, lalu tempel kode berikut:

AddPost Mutasi

```
mutation addPost {
    addPost(
        id: 6
        author: "Author6"
        title: "Sixth book"
        url: "https://www.amazon.com/"
        content: "This is the book is a tutorial for using GraphQL with AWS AppSync."
    ) {
        id
        author
        title
        content
        url
        ups
        downs
    }
}
```

Kueri GetPost

```
query getPost {
    getPost(id: "2") {
        id
        author
        title
        content
        url
        ups
        downs
    }
```

```
}
```

AllPosts Query

```
query allPosts {
    allPosts {
        id
        author
        title
        content
        url
        ups
        downs
        relatedPosts {
            id
            title
        }
    }
}
```

Mengembalikan kesalahan

Setiap resolusi bidang yang diberikan dapat mengakibatkan kesalahan. Dengan AWS AppSync, Anda dapat meningkatkan kesalahan dari sumber-sumber berikut:

- Templat pemetaan permintaan atau respons
- Fungsi Lambda

Dari template pemetaan

Untuk meningkatkan kesalahan yang disengaja, Anda dapat menggunakan metode `$utils.error` pembantu dari template Velocity Template Language (VTL). Dibutuhkan sebagai argumen`errorMessage`, an`errorType`, dan `data` nilai opsional. `data`Ini berguna untuk mengembalikan data tambahan kembali ke klien ketika terjadi kesalahan. `data`Objek ditambahkan ke `errors` dalam respon akhir GraphQL.

Contoh berikut menunjukkan cara menggunakannya dalam template pemetaan `Post.relatedPosts: [Post]` respons:

```
$utils.error("Failed to fetch relatedPosts", "LambdaFailure", $context.result)
```

Ini menghasilkan respons GraphQL yang mirip dengan yang berikut:

```
{
    "data": {
        "allPosts": [
            {
                "id": "2",
                "title": "Second book",
                "relatedPosts": null
            },
            ...
        ]
    },
    "errors": [
        {
            "path": [
                "allPosts",
                0,
                "relatedPosts"
            ],
            "errorType": "LambdaFailure",
            "locations": [
                {
                    "line": 5,
                    "column": 5
                }
            ],
            "message": "Failed to fetch relatedPosts",
            "data": [
                {
                  "id": "2",
                  "title": "Second book"
                },
                {
                  "id": "1",
                  "title": "First book"
                }
            ]
        }
    ]
}
```

`allPosts[0].relatedPosts`Dimana nol karena kesalahan dan`errorMessage`,`errorType`, dan `data` hadir dalam `data.errors[0]` objek.

Dari fungsi Lambda

AWS AppSync juga memahami kesalahan yang dilemparkan fungsi Lambda. Model pemrograman Lambda memungkinkan Anda meningkatkan kesalahan yang ditangani. Jika fungsi Lambda melempar kesalahan, AWS AppSync gagal menyelesaikan bidang saat ini. Hanya pesan kesalahan yang dikembalikan dari Lambda yang disetel dalam respons. Saat ini, Anda tidak dapat meneruskan data asing apa pun kembali ke klien dengan memunculkan kesalahan dari fungsi Lambda.

Catatan: Jika fungsi Lambda Anda memunculkan kesalahan yang tidak tertangani, AWS AppSync gunakan pesan kesalahan yang ditetapkan Lambda.

Fungsi Lambda berikut menimbulkan kesalahan:

```
exports.handler = (event, context, callback) => {
    console.log("Received event {}", JSON.stringify(event, 3));
    callback("I fail. Always.");
};
```

Ini mengembalikan respon GraphQL mirip dengan berikut ini:

```
{
    "data": {
        "allPosts": [
            {
                "id": "2",
                "title": "Second book",
                "relatedPosts": null
            },
            ...
        ]
    },
    "errors": [
        {
            "path": [
                "allPosts",
                0,
                "relatedPosts"
            ],
```

```
            "errorType": "Lambda:Handled",
            "locations": [
                {
                    "line": 5,
                    "column": 5
                }
            ],
            "message": "I fail. Always."
        }
    ]
}
```

Kasus penggunaan lanjutan: Batching

Fungsi Lambda dalam contoh ini memiliki `relatedPosts` bidang yang mengembalikan daftar posting terkait untuk posting tertentu. Dalam contoh kueri, pemanggilan `allPosts` bidang dari fungsi Lambda mengembalikan lima posting. Karena kami menetapkan bahwa kami juga ingin menyelesaikan `relatedPosts` untuk setiap posting yang dikembalikan, operasi `relatedPosts` lapangan dipanggil lima kali.

```
query allPosts {
    allPosts {   // 1 Lambda invocation - yields 5 Posts
        id
        author
        title
        content
        url
        ups
        downs
        relatedPosts {   // 5 Lambda invocations - each yields 5 posts
            id
            title
        }
    }
}
```

Meskipun ini mungkin tidak terdengar substansif dalam contoh spesifik ini, pengambilan berlebihan yang diperparah ini dapat dengan cepat merusak aplikasi.

Jika Anda mengambil `relatedPosts` lagi pada terkait yang dikembalikan `Posts` dalam kueri yang sama, jumlah pemanggilan akan meningkat secara dramatis.

```
query allPosts {
    allPosts {   // 1 Lambda invocation - yields 5 Posts
        id
        author
        title
        content
        url
        ups
        downs
        relatedPosts {   // 5 Lambda invocations - each yield 5 posts = 5 x 5 Posts
            id
            title
            relatedPosts {  // 5 x 5 Lambda invocations - each yield 5 posts = 25 x 5
 Posts
                id
                title
                author
            }
        }
    }
}
```

Dalam kueri yang relatif sederhana ini, AWS AppSync akan memanggil fungsi Lambda 1 + 5 + 25 = 31 kali.

Ini adalah tantangan yang cukup umum dan sering disebut masalah N+1 (dalam hal ini, N = 5), dan dapat menimbulkan peningkatan latensi dan biaya untuk aplikasi.

Salah satu pendekatan untuk memecahkan masalah ini adalah dengan mengumpulkan permintaan penyelesai bidang yang serupa bersama-sama. Dalam contoh ini, alih-alih memiliki fungsi Lambda menyelesaikan daftar posting terkait untuk satu posting tertentu, itu malah bisa menyelesaikan daftar posting terkait untuk kumpulan posting tertentu.

Untuk mendemonstrasikan hal ini, mari beralih `Post.relatedPosts: [Post]` resolver ke resolver batch-enabled.

Di sisi kanan AWS AppSync konsol, pilih `Post.relatedPosts: [Post]` resolver yang ada. Ubah template pemetaan permintaan menjadi berikut:

```
{
    "version": "2017-02-28",
```

```
    "operation": "BatchInvoke",
    "payload": {
        "field": "relatedPosts",
        "source":  $utils.toJson($context.source)
    }
}
```

Hanya `operation` bidang yang berubah dari `Invoke` menjadi`BatchInvoke`. Bidang payload sekarang menjadi array dari apa pun yang ditentukan dalam template. Dalam contoh ini, fungsi Lambda menerima yang berikut sebagai input:

```
[
    {
        "field": "relatedPosts",
        "source": {
            "id": 1
        }
    },
    {
        "field": "relatedPosts",
        "source": {
            "id": 2
        }
    },
    ...
]
```

Kapan `BatchInvoke` ditentukan dalam template pemetaan permintaan, fungsi Lambda menerima daftar permintaan dan mengembalikan daftar hasil.

Secara khusus, daftar hasil harus sesuai dengan ukuran dan urutan entri payload permintaan sehingga AWS AppSync dapat cocok dengan hasil yang sesuai.

Dalam contoh batching ini, fungsi Lambda mengembalikan sekumpulan hasil sebagai berikut:

```
[
    [{"id":"2","title":"Second book"}, {"id":"3","title":"Third book"}],   //
 relatedPosts for id=1
    [{"id":"3","title":"Third book"}]
           // relatedPosts for id=2
]
```

Fungsi Lambda berikut di Node.js menunjukkan fungsionalitas batching ini untuk bidang sebagai `Post.relatedPosts` berikut:

```
exports.handler = (event, context, callback) => {
    console.log("Received event {}", JSON.stringify(event, 3));
    var posts = {
         "1": {"id": "1", "title": "First book", "author": "Author1", "url": "https://
amazon.com/", "content": "SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR
 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1", "ups": "100",
 "downs": "10"},
         "2": {"id": "2", "title": "Second book", "author": "Author2", "url": "https://
amazon.com", "content": "SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT", "ups":
 "100", "downs": "10"},
         "3": {"id": "3", "title": "Third book", "author": "Author3", "url": null,
 "content": null, "ups": null, "downs": null },
         "4": {"id": "4", "title": "Fourth book", "author": "Author4", "url": "https://
www.amazon.com/", "content": "SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4", "ups": "1000", "downs": "0"},
         "5": {"id": "5", "title": "Fifth book", "author": "Author5", "url": "https://
www.amazon.com/", "content": "SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT
 AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT", "ups": "50", "downs": "0"} };

    var relatedPosts = {
        "1": [posts['4']],
        "2": [posts['3'], posts['5']],
        "3": [posts['2'], posts['1']],
        "4": [posts['2'], posts['1']],
        "5": []
    };

    console.log("Got a BatchInvoke Request. The payload has %d items to resolve.",
 event.length);
    // event is now an array
    var field = event[0].field;
    switch(field) {
        case "relatedPosts":
            var results = [];
            // the response MUST contain the same number
            // of entries as the payload array
            for (var i=0; i< event.length; i++) {
                console.log("post {}", JSON.stringify(event[i].source));
                results.push(relatedPosts[event[i].source.id]);
```

```
            }
            console.log("results {}", JSON.stringify(results));
            callback(null, results);
            break;
        default:
            callback("Unknown field, unable to resolve" + field, null);
            break;
    }
};
```

Mengembalikan kesalahan individu

Contoh sebelumnya menunjukkan bahwa dimungkinkan untuk mengembalikan satu kesalahan dari fungsi Lambda atau memunculkan kesalahan dari templat pemetaan. Untuk pemanggilan batch, memunculkan kesalahan dari fungsi Lambda menandai seluruh batch sebagai gagal. Ini mungkin dapat diterima untuk skenario tertentu di mana terjadi kesalahan yang tidak dapat dipulihkan, seperti koneksi yang gagal ke penyimpanan data. Namun, dalam kasus di mana beberapa item dalam batch berhasil dan yang lainnya gagal, dimungkinkan untuk mengembalikan kesalahan dan data yang valid. Karena AWS AppSync memerlukan respons batch untuk mencantumkan elemen yang cocok dengan ukuran asli batch, Anda harus menentukan struktur data yang dapat membedakan data yang valid dari kesalahan.

Misalnya, jika fungsi Lambda diharapkan mengembalikan kumpulan posting terkait, Anda dapat memilih untuk mengembalikan daftar *Response* objek di mana setiap objek memiliki data opsional, ErrorMessage, dan bidang ErrorType. Jika bidang ErrorMessage ada, itu berarti terjadi kesalahan.

Kode berikut menunjukkan bagaimana Anda dapat memperbarui fungsi Lambda:

```
exports.handler = (event, context, callback) => {
    console.log("Received event {}", JSON.stringify(event, 3));
    var posts = {
         "1": {"id": "1", "title": "First book", "author": "Author1", "url": "https://
amazon.com/", "content": "SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR
 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1", "ups": "100",
 "downs": "10"},
         "2": {"id": "2", "title": "Second book", "author": "Author2", "url": "https://
amazon.com", "content": "SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT", "ups":
 "100", "downs": "10"},
         "3": {"id": "3", "title": "Third book", "author": "Author3", "url": null,
 "content": null, "ups": null, "downs": null },
         "4": {"id": "4", "title": "Fourth book", "author": "Author4", "url": "https://
www.amazon.com/", "content": "SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
```

```
 AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4", "ups": "1000", "downs": "0"},
         "5": {"id": "5", "title": "Fifth book", "author": "Author5", "url": "https://
www.amazon.com/", "content": "SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT
 AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT", "ups": "50", "downs": "0"} };

    var relatedPosts = {
        "1": [posts['4']],
        "2": [posts['3'], posts['5']],
        "3": [posts['2'], posts['1']],
        "4": [posts['2'], posts['1']],
        "5": []
    };

    console.log("Got a BatchInvoke Request. The payload has %d items to resolve.",
 event.length);
    // event is now an array
    var field = event[0].field;
    switch(field) {
        case "relatedPosts":
            var results = [];
            results.push({ 'data': relatedPosts['1'] });
            results.push({ 'data': relatedPosts['2'] });
            results.push({ 'data': null, 'errorMessage': 'Error Happened', 'errorType':
 'ERROR' });
            results.push(null);
            results.push({ 'data': relatedPosts['3'], 'errorMessage': 'Error Happened
 with last result', 'errorType': 'ERROR' });
            callback(null, results);
            break;
        default:
            callback("Unknown field, unable to resolve" + field, null);
            break;
    }
};
```

Untuk contoh ini, template pemetaan respons berikut mem-parsing setiap item fungsi Lambda dan memunculkan kesalahan yang terjadi:

```
#if( $context.result && $context.result.errorMessage )
    $utils.error($context.result.errorMessage, $context.result.errorType,
 $context.result.data)
#else
```

```
    $utils.toJson($context.result.data)
#end
```

Contoh ini mengembalikan respon GraphQL mirip dengan berikut:

```
{
  "data": {
    "allPosts": [
      {
        "id": "1",
        "relatedPostsPartialErrors": [
          {
            "id": "4",
            "title": "Fourth book"
          }
        ]
      },
      {
        "id": "2",
        "relatedPostsPartialErrors": [
          {
            "id": "3",
            "title": "Third book"
          },
          {
            "id": "5",
            "title": "Fifth book"
          }
        ]
      },
      {
        "id": "3",
        "relatedPostsPartialErrors": null
      },
      {
        "id": "4",
        "relatedPostsPartialErrors": null
      },
      {
        "id": "5",
        "relatedPostsPartialErrors": null
      }
    ]
```

```
  },
  "errors": [
    {
      "path": [
        "allPosts",
        2,
        "relatedPostsPartialErrors"
      ],
      "errorType": "ERROR",
      "locations": [
        {
          "line": 4,
          "column": 9
        }
      ],
      "message": "Error Happened"
    },
    {
      "path": [
        "allPosts",
        4,
        "relatedPostsPartialErrors"
      ],
      "data": [
        {
          "id": "2",
          "title": "Second book"
        },
        {
          "id": "1",
          "title": "First book"
        }
      ],
      "errorType": "ERROR",
      "locations": [
        {
          "line": 4,
          "column": 9
        }
      ],
      "message": "Error Happened with last result"
    }
  ]
```

```
}
```

Mengkonfigurasi ukuran batching maksimum

Secara default, saat menggunakan`BatchInvoke`, AWS AppSync mengirim permintaan ke fungsi Lambda Anda dalam batch hingga lima item. Anda dapat mengonfigurasi ukuran batch maksimum resolver Lambda Anda.

Untuk mengonfigurasi ukuran batching maksimum pada resolver, gunakan perintah berikut di (): AWS Command Line Interface AWS CLI

```
$ aws appsync create-resolver --api-id <api-id> --type-name Query --field-name
 relatedPosts \
 --request-mapping-template "<template>" --response-mapping-template "<template>" --
data-source-name "<lambda-datasource>" \
 --max-batch-size X
```

> **Note**
>
> Saat menyediakan template pemetaan permintaan, Anda harus menggunakan `BatchInvoke` operasi untuk menggunakan batching.

Anda juga dapat menggunakan perintah berikut untuk mengaktifkan dan mengkonfigurasi batching pada Direct Lambda Resolvers:

```
$ aws appsync create-resolver --api-id <api-id> --type-name Query --field-name
 relatedPosts \
 --data-source-name "<lambda-datasource>" \
 --max-batch-size X
```

Konfigurasi ukuran batching maksimum dengan template VTL

Untuk Resolver Lambda yang memiliki templat dalam permintaan VTL, ukuran batch maksimum tidak akan berpengaruh kecuali mereka secara langsung menentukannya sebagai operasi di VTL. `BatchInvoke` Demikian pula, jika Anda melakukan mutasi tingkat atas, batching tidak dilakukan untuk mutasi karena spesifikasi GraphQL memerlukan mutasi paralel untuk dieksekusi secara berurutan.

Misalnya, ambil mutasi berikut:

```
type Mutation {
    putItem(input: Item): Item
    putItems(inputs: [Item]): [Item]
}
```

Dengan menggunakan mutasi pertama, kita dapat membuat 10 `Items` seperti yang ditunjukkan pada cuplikan di bawah ini:

```
mutation MyMutation {
    v1: putItem($someItem1) {
        id,
        name
    }
    v2: putItem($someItem2) {
        id,
        name
    }
    v3: putItem($someItem3) {
        id,
        name
    }
    v4: putItem($someItem4) {
        id,
        name
    }
    v5: putItem($someItem5) {
        id,
        name
    }
    v6: putItem($someItem6) {
        id,
        name
    }
    v7: putItem($someItem7) {
        id,
        name
    }
    v8: putItem($someItem8) {
        id,
        name
    }
    v9: putItem($someItem9) {
        id,
```

```
        name
    }
    v10: putItem($someItem10) {
        id,
        name
    }
}
```

Dalam contoh ini, tidak `Items` akan dikelompokkan dalam grup 10 bahkan jika ukuran batch maksimum diatur ke 10 di Lambda Resolver. Sebaliknya, mereka akan mengeksekusi secara berurutan sesuai dengan spesifikasi GraphQL.

Untuk melakukan mutasi batch yang sebenarnya, Anda dapat mengikuti contoh di bawah ini menggunakan mutasi kedua:

```
mutation MyMutation {
    putItems([$someItem1, $someItem2, $someItem3,$someItem4, $someItem5, $someItem6,
    $someItem7, $someItem8, $someItem9, $someItem10]) {
    id,
    name
    }
}
```

Untuk informasi selengkapnya tentang penggunaan batching dengan Direct Lambda Resolvers, lihat. Resolver Lambda Langsung

Tutorial: Penyelesai OpenSearch Layanan Amazon

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync mendukung penggunaan OpenSearch Layanan Amazon dari domain yang telah Anda sediakan di AWS akun Anda sendiri, asalkan tidak ada di dalam VPC. Setelah domain Anda disediakan, Anda dapat menghubungkannya menggunakan sumber data, di mana Anda dapat mengonfigurasi resolver dalam skema untuk melakukan operasi GraphQL seperti kueri, mutasi, dan langganan. Tutorial ini akan membawa Anda melalui beberapa contoh umum.

Untuk informasi selengkapnya, lihat Referensi Template Pemetaan Resolver untuk. OpenSearch

Pengaturan Satu-Klik

Untuk secara otomatis mengatur titik AWS AppSync akhir GraphQL dengan OpenSearch Amazon Service yang dikonfigurasi, Anda dapat menggunakan template ini: AWS CloudFormation

Setelah AWS CloudFormation penerapan selesai, Anda dapat langsung melompat ke menjalankan kueri dan mutasi GraphQL.

Buat Domain OpenSearch Layanan Baru

Untuk memulai tutorial ini, Anda memerlukan domain OpenSearch Layanan yang ada. Jika Anda tidak memilikinya, Anda dapat menggunakan sampel berikut. Perhatikan bahwa diperlukan waktu hingga 15 menit untuk membuat domain OpenSearch Layanan sebelum Anda dapat melanjutkan untuk mengintegrasikannya dengan sumber AWS AppSync data.

```
aws cloudformation create-stack --stack-name AppSyncOpenSearch \
--template-url https://s3.us-west-2.amazonaws.com/awsappsync/resources/elasticsearch/
ESResolverCFTemplate.yaml \
--parameters ParameterKey=OSDomainName,ParameterValue=ddtestdomain
 ParameterKey=Tier,ParameterValue=development \
--capabilities CAPABILITY_NAMED_IAM
```

Anda dapat meluncurkan AWS CloudFormation tumpukan berikut di wilayah US West 2 (Oregon) di AWS akun Anda:

Konfigurasikan Sumber Data untuk OpenSearch Layanan

Setelah domain OpenSearch Layanan dibuat, navigasikan ke AWS AppSync GraphQL API Anda dan pilih tab Sumber Data. Pilih Baru dan masukkan nama ramah untuk sumber data, seperti “oss”. Kemudian pilih OpenSearch domain Amazon untuk tipe sumber data, pilih wilayah yang sesuai, dan Anda akan melihat domain OpenSearch Layanan Anda terdaftar. Setelah memilihnya, Anda dapat membuat peran baru dan AWS AppSync akan menetapkan izin yang sesuai peran, atau Anda dapat memilih peran yang ada, yang memiliki kebijakan sebaris berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1234234",
            "Effect": "Allow",
            "Action": [
                "es:ESHttpDelete",
                "es:ESHttpHead",
                "es:ESHttpGet",
                "es:ESHttpPost",
                "es:ESHttpPut"
            ],
            "Resource": [
                "arn:aws:es:REGION:ACCOUNTNUMBER:domain/democluster/*"
            ]
        }
    ]
}
```

Anda juga perlu mengatur hubungan kepercayaan AWS AppSync untuk peran itu:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "appsync.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Selain itu, domain OpenSearch Layanan memiliki Kebijakan Akses sendiri yang dapat Anda modifikasi melalui konsol OpenSearch Layanan Amazon. Anda perlu menambahkan kebijakan yang mirip dengan yang berikut ini, dengan tindakan dan sumber daya yang sesuai untuk domain OpenSearch Layanan. Perhatikan bahwa Principal akan menjadi peran sumber AppSync data, yang jika Anda membiarkan konsol membuat ini, dapat ditemukan di konsol IAM.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::ACCOUNTNUMBER:role/service-role/
APPSYNC_DATASOURCE_ROLE"
        },
        "Action": [
            "es:ESHttpDelete",
            "es:ESHttpHead",
            "es:ESHttpGet",
            "es:ESHttpPost",
            "es:ESHttpPut"
        ],
        "Resource": "arn:aws:es:REGION:ACCOUNTNUMBER:domain/DOMAIN_NAME/*"
        }
    ]
}
```

Menghubungkan Resolver

Setelah sumber data terhubung ke domain OpenSearch Layanan Anda, Anda dapat menghubungkannya ke skema GraphQL Anda dengan resolver, seperti yang ditunjukkan pada contoh berikut:

```
 schema {
   query: Query
   mutation: Mutation
 }

 type Query {
   getPost(id: ID!): Post
   allPosts: [Post]
 }

 type Mutation {
   addPost(id: ID!, author: String, title: String, url: String, ups: Int, downs: Int,
 content: String): AWSJSON
 }

type Post {
  id: ID!
```

```
  author: String
  title: String
  url: String
  ups: Int
  downs: Int
  content: String
}
...
```

Perhatikan bahwa ada `Post` tipe yang ditentukan pengguna dengan bidang. `id` Dalam contoh berikut, kami berasumsi ada proses (yang dapat diotomatisasi) untuk memasukkan jenis ini ke domain OpenSearch Layanan Anda, yang akan memetakan ke root jalur`/post/_doc`, di mana `post` indeks. Dari jalur root ini, Anda dapat melakukan pencarian dokumen individual, pencarian wildcard dengan`/id/post*`, atau pencarian multi-dokumen dengan jalur. `/post/_search` Misalnya, jika Anda memiliki jenis lain yang dipanggil`User`, Anda dapat mengindeks dokumen di bawah indeks baru yang disebut`user`, lalu melakukan pencarian dengan jalur. `/user/_search`

Dari editor skema di AWS AppSync konsol, ubah `Posts` skema sebelumnya untuk menyertakan kueri: `searchPosts`

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  searchPosts: [Post]
}
```

Simpan skema. Di sisi kanan, untuk`searchPosts`, pilih Lampirkan resolver. Di menu Action, pilih Update runtime, lalu pilih Unit Resolver (hanya VTL). Kemudian, pilih sumber data OpenSearch Layanan Anda. Di bawah bagian template pemetaan permintaan, pilih drop-down untuk posting Query untuk mendapatkan template dasar. `path`Memodifikasi menjadi`/post/_search`. Seharusnya terlihat seperti berikut:

```
{
    "version":"2017-02-28",
    "operation":"GET",
    "path":"/post/_search",
    "params":{
        "headers":{},
        "queryString":{},
        "body":{
```

```
            "from":0,
            "size":50
        }
    }
}
```

Ini mengasumsikan bahwa skema sebelumnya memiliki dokumen yang telah diindeks di Layanan di bawah bidang. OpenSearch `post` Jika Anda menyusun data Anda secara berbeda, maka Anda harus memperbaruinya.

Di bawah bagian template pemetaan respons, Anda perlu menentukan `_source` filter yang sesuai jika Anda ingin mendapatkan kembali hasil data dari kueri OpenSearch Layanan dan menerjemahkan ke GraphQL. Gunakan template berikut:

```
[
    #foreach($entry in $context.result.hits.hits)
    #if( $velocityCount > 1 ) , #end
    $utils.toJson($entry.get("_source"))
    #end
]
```

Memodifikasi Pencarian Anda

Template pemetaan permintaan sebelumnya melakukan kueri sederhana untuk semua catatan. Misalkan Anda ingin mencari oleh penulis tertentu. Selanjutnya, misalkan Anda ingin penulis itu menjadi argumen yang didefinisikan dalam kueri GraphQL Anda. Di editor skema AWS AppSync konsol, tambahkan `allPostsByAuthor` kueri:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  allPostsByAuthor(author: String!): [Post]
  searchPosts: [Post]
}
```

Sekarang pilih Lampirkan resolver dan pilih sumber data OpenSearch Layanan, tetapi gunakan contoh berikut dalam template pemetaan respons:

```
{
```

```
    "version":"2017-02-28",
    "operation":"GET",
    "path":"/post/_search",
    "params":{
        "headers":{},
        "queryString":{},
        "body":{
            "from":0,
            "size":50,
            "query":{
                "match" :{
                    "author": $util.toJson($context.arguments.author)
                }
            }
        }
    }
}
```

Perhatikan bahwa `body` diisi dengan kueri istilah untuk `author` bidang, yang diteruskan dari klien sebagai argumen. Anda dapat secara opsional memiliki informasi yang telah diisi sebelumnya, seperti teks standar, atau bahkan menggunakan utilitas lain.

Jika Anda menggunakan resolver ini, isi template pemetaan respons dengan informasi yang sama seperti contoh sebelumnya.

Menambahkan Data ke OpenSearch Layanan

Anda mungkin ingin menambahkan data ke domain OpenSearch Layanan Anda sebagai hasil dari mutasi GraphQL. Ini adalah mekanisme yang kuat untuk pencarian dan tujuan lain. Karena Anda dapat menggunakan langganan GraphQL untuk membuat data Anda real-time, ini berfungsi sebagai mekanisme untuk memberi tahu klien tentang pembaruan data di domain Layanan Anda. OpenSearch

Kembali ke halaman Skema di AWS AppSync konsol dan pilih Lampirkan resolver untuk mutasi. `addPost()` Pilih sumber data OpenSearch Layanan lagi dan gunakan template pemetaan respons berikut untuk `Posts` skema:

```
{
    "version":"2017-02-28",
    "operation":"PUT",
    "path": $util.toJson("/post/_doc/$context.arguments.id"),
```

```
    "params":{
        "headers":{},
        "queryString":{},
        "body":{
            "id": $util.toJson($context.arguments.id),
            "author": $util.toJson($context.arguments.author),
            "ups": $util.toJson($context.arguments.ups),
            "downs": $util.toJson($context.arguments.downs),
            "url": $util.toJson($context.arguments.url),
            "content": $util.toJson($context.arguments.content),
            "title": $util.toJson($context.arguments.title)
        }
    }
}
```

Seperti sebelumnya, ini adalah contoh bagaimana data Anda mungkin terstruktur. Jika Anda memiliki nama bidang atau indeks yang berbeda, Anda perlu memperbarui `path` dan sesuai `body` kebutuhan. Contoh ini juga menunjukkan cara menggunakan `$context.arguments` untuk mengisi template dari argumen mutasi GraphQL Anda.

Sebelum melanjutkan, gunakan template pemetaan respons berikut, yang akan mengembalikan hasil operasi mutasi atau informasi kesalahan sebagai output:

```
#if($context.error)
    $util.toJson($ctx.error)
#else
    $util.toJson($context.result)
#end
```

Mengambil Dokumen Tunggal

Terakhir, jika Anda ingin menggunakan `getPost(id:ID)` kueri dalam skema Anda untuk mengembalikan dokumen individual, temukan kueri ini di editor skema AWS AppSync konsol dan pilih Lampirkan resolver. Pilih sumber data OpenSearch Layanan lagi dan gunakan template pemetaan berikut:

```
{
    "version":"2017-02-28",
    "operation":"GET",
    "path": $util.toJson("post/_doc/$context.arguments.id"),
    "params":{
```

```
        "headers":{},
        "queryString":{},
        "body":{}
    }
}
```

Karena di `path` atas menggunakan `id` argumen dengan badan kosong, ini mengembalikan dokumen tunggal. Namun, Anda perlu menggunakan template pemetaan respons berikut, karena sekarang Anda mengembalikan satu item dan bukan daftar:

```
$utils.toJson($context.result.get("_source"))
```

Lakukan Kueri dan Mutasi

Anda sekarang harus dapat melakukan operasi GraphQL terhadap OpenSearch domain Layanan Anda. Arahkan ke tab Kueri AWS AppSync konsol dan tambahkan catatan baru:

```
mutation addPost {
    addPost (
        id:"12345"
        author: "Fred"
        title: "My first book"
        content: "This will be fun to write!"
        url: "publisher website",
        ups: 100,
        downs:20
       )
}
```

Anda akan melihat hasil mutasi di sebelah kanan. Demikian pula, Anda sekarang dapat menjalankan `searchPosts` kueri terhadap domain OpenSearch Layanan Anda:

```
query searchPosts {
    searchPosts {
        id
        title
        author
        content
    }
}
```

Praktik Terbaik

- OpenSearch Layanan harus untuk kueri data, bukan sebagai basis data utama Anda. Anda mungkin ingin menggunakan OpenSearch Layanan bersama dengan Amazon DynamoDB sebagaimana diuraikan dalam Menggabungkan GraphQL Resolvers.
- Hanya berikan akses ke domain Anda dengan mengizinkan peran AWS AppSync layanan mengakses klaster.
- Anda dapat memulai dari yang kecil dalam pengembangan, dengan cluster berbiaya terendah, dan kemudian pindah ke cluster yang lebih besar dengan ketersediaan tinggi (HA) saat Anda pindah ke produksi.

Tutorial: Resolver Lokal

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync memungkinkan Anda menggunakan sumber data yang didukung (AWS Lambda, Amazon DynamoDB, atau OpenSearch Amazon Service) untuk melakukan berbagai operasi. Namun, dalam skenario tertentu, panggilan ke sumber data yang didukung mungkin tidak diperlukan.

Di sinilah resolver lokal berguna. Alih-alih memanggil sumber data jarak jauh, resolver lokal hanya akan meneruskan hasil template pemetaan permintaan ke template pemetaan respons. Resolusi lapangan tidak akan pergi AWS AppSync.

Resolver lokal berguna untuk beberapa kasus penggunaan. Kasus penggunaan yang paling populer adalah mempublikasikan notifikasi tanpa memicu panggilan sumber data. Untuk mendemonstrasikan kasus penggunaan ini, mari buat aplikasi paging; di mana pengguna dapat saling berhalaman. Contoh ini memanfaatkan Langganan, jadi jika Anda tidak terbiasa dengan Langganan, Anda dapat mengikuti tutorial Data Waktu Nyata.

Buat Aplikasi Paging

Dalam aplikasi paging kami, klien dapat berlangganan kotak masuk, dan mengirim halaman ke klien lain. Setiap halaman berisi pesan. Berikut adalah skema:

```
schema {
    query: Query
    mutation: Mutation
    subscription: Subscription
}

type Subscription {
    inbox(to: String!): Page
    @aws_subscribe(mutations: ["page"])
}

type Mutation {
    page(body: String!, to: String!): Page!
}

type Page {
    from: String
    to: String!
    body: String!
    sentAt: String!
}

type Query {
    me: String
}
```

Mari kita lampirkan resolver di lapangan. `Mutation.page` Di panel Schema, klik Lampirkan Resolver di sebelah definisi bidang di panel kanan. Buat sumber data baru tipe None dan beri nama PageDataSource.

Untuk template pemetaan permintaan, masukkan:

```
{
  "version": "2017-02-28",
  "payload": {
    "body": $util.toJson($context.arguments.body),
    "from": $util.toJson($context.identity.username),
    "to":  $util.toJson($context.arguments.to),
    "sentAt": "$util.time.nowISO8601()"
  }
}
```

Dan untuk template pemetaan respons, pilih default Teruskan hasilnya. Simpan resolver Anda. Aplikasi Anda sekarang siap, mari kita halaman!

Kirim dan berlangganan halaman

Agar klien dapat menerima halaman, mereka harus terlebih dahulu berlangganan kotak masuk.

Di panel Queries mari kita jalankan langganan: `inbox`

```
subscription Inbox {
    inbox(to: "Nadia") {
        body
        to
        from
        sentAt
    }
}
```

Nadia akan menerima halaman setiap kali `Mutation.page` mutasi dipanggil. Mari kita panggil mutasi dengan mengeksekusi mutasi:

```
mutation Page {
    page(to: "Nadia", body: "Hello, World!") {
        body
        to
        from
        sentAt
    }
}
```

Kami baru saja mendemonstrasikan penggunaan resolver lokal, dengan mengirim Halaman dan menerimanya tanpa pergi. AWS AppSync

Tutorial: Menggabungkan GraphQL Resolvers

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Resolver dan bidang dalam skema GraphQL memiliki hubungan 1:1 dengan tingkat fleksibilitas yang besar. Karena sumber data dikonfigurasi pada resolver secara independen dari skema, Anda memiliki kemampuan untuk jenis GraphQL untuk diselesaikan atau dimanipulasi melalui sumber data yang berbeda, pencampuran dan pencocokan pada skema untuk memenuhi kebutuhan Anda.

Contoh skenario berikut menunjukkan cara mencampur dan mencocokkan sumber data dalam skema Anda. Sebelum Anda mulai, kami sarankan Anda terbiasa dengan pengaturan sumber data dan resolver untukAWS Lambda, Amazon DynamoDB, dan OpenSearch Amazon Service seperti yang dijelaskan dalam tutorial sebelumnya.

Contoh Skema

Skema berikut memiliki jenis `Post` dengan 3 `Query` operasi dan 3 `Mutation` operasi didefinisikan:

```
type Post {
    id: ID!
    author: String!
    title: String
    content: String
    url: String
    ups: Int
    downs: Int
    version: Int!
}

type Query {
    allPost: [Post]
    getPost(id: ID!): Post
    searchPosts: [Post]
}

type Mutation {
    addPost(
        id: ID!,
        author: String!,
        title: String,
        content: String,
        url: String
    ): Post
    updatePost(
        id: ID!,
        author: String!,
```

```
        title: String,
        content: String,
        url: String,
        ups: Int!,
        downs: Int!,
        expectedVersion: Int!
    ): Post
    deletePost(id: ID!): Post
}
```

Dalam contoh ini Anda akan memiliki total 6 resolver untuk dilampirkan. Salah satu cara yang mungkin adalah agar semua ini berasal dari tabel Amazon DynamoDB, yang `Posts` disebut, `AllPosts` di mana menjalankan pemindaian `searchPosts` dan menjalankan kueri, seperti yang diuraikan dalam Referensi Template Pemetaan DynamoDB Resolver. Namun, ada alternatif untuk memenuhi kebutuhan bisnis Anda, seperti menyelesaikan kueri GraphQL ini dari Lambda atau Layanan. OpenSearch

Mengubah Data Melalui Resolver

Anda mungkin perlu mengembalikan hasil dari database seperti DynamoDB (atau Amazon Aurora) ke klien dengan beberapa atribut diubah. Ini mungkin karena pemformatan tipe data, seperti perbedaan stempel waktu pada klien, atau untuk menangani masalah kompatibilitas mundur. Untuk tujuan ilustrasi, dalam contoh berikut, AWS Lambda fungsi memanipulasi up-votes dan down-votes untuk posting blog dengan menetapkan nomor acak setiap kali GraphQL resolver dipanggil:

```
'use strict';
const doc = require('dynamodb-doc');
const dynamo = new doc.DynamoDB();

exports.handler = (event, context, callback) => {
    const payload = {
        TableName: 'Posts',
        Limit: 50,
        Select: 'ALL_ATTRIBUTES',
    };

    dynamo.scan(payload, (err, data) => {
        const result = { data: data.Items.map(item =>{
            item.ups = parseInt(Math.random() * (50 - 10) + 10, 10);
            item.downs = parseInt(Math.random() * (20 - 0) + 0, 10);
            return item;
```

```
        }) };
        callback(err, result.data);
    });
};
```

Ini adalah fungsi Lambda yang benar-benar valid dan dapat dilampirkan ke `AllPosts` bidang dalam skema GraphQL sehingga setiap kueri yang mengembalikan semua hasil mendapat angka acak untuk naik/turun.

DynamoDB dan Layanan OpenSearch

Untuk beberapa aplikasi, Anda mungkin melakukan mutasi atau kueri pencarian sederhana terhadap DynamoDB, dan memiliki proses latar belakang mentransfer dokumen ke Layanan. OpenSearch Anda kemudian dapat melampirkan `searchPosts` Resolver ke sumber data OpenSearch Layanan dan mengembalikan hasil pencarian (dari data yang berasal dari DynamoDB) menggunakan query GraphQL. Ini bisa sangat kuat saat menambahkan operasi pencarian lanjutan ke aplikasi Anda seperti kata kunci, kecocokan kata kabur atau bahkan pencarian geospasial. Mentransfer data dari DynamoDB dapat dilakukan melalui proses ETL atau sebagai alternatif Anda dapat melakukan streaming dari DynamoDB menggunakan Lambda. Anda dapat meluncurkan contoh lengkap ini menggunakan AWS CloudFormation tumpukan berikut di Wilayah AS Barat 2 (Oregon) di AWS akun Anda:

Skema dalam contoh ini memungkinkan Anda menambahkan posting menggunakan DynamoDB resolver sebagai berikut:

```
mutation add {
    putPost(author:"Nadia"
        title:"My first post"
        content:"This is some test content"
        url:"https://aws.amazon.com/appsync/"
    ){
        id
        title
    }
}
```

Ini menulis data ke DynamoDB yang kemudian mengalirkan data melalui Lambda OpenSearch ke Amazon Service yang dapat Anda cari untuk semua posting dengan bidang yang berbeda. Misalnya, karena data ada di Amazon OpenSearch Service, Anda dapat mencari bidang penulis atau konten dengan teks bentuk bebas, bahkan dengan spasi, sebagai berikut:

```
query searchName{
    searchAuthor(name:"   Nadia   "){
        id
        title
        content
    }
}

query searchContent{
    searchContent(text:"test"){
        id
        title
        content
    }
}
```

Karena data ditulis langsung ke DynamoDB, Anda masih dapat melakukan operasi pencarian daftar atau item yang efisien terhadap tabel dengan dan kueri. `allPosts{...}` `singlePost{...}` Tumpukan ini menggunakan kode contoh berikut untuk aliran DynamoDB:

Catatan: Kode ini misalnya saja.

```
var AWS = require('aws-sdk');
var path = require('path');
var stream = require('stream');

var esDomain = {
    endpoint: 'https://opensearch-domain-name.REGION.es.amazonaws.com',
    region: 'REGION',
    index: 'id',
    doctype: 'post'
};

var endpoint = new AWS.Endpoint(esDomain.endpoint)
var creds = new AWS.EnvironmentCredentials('AWS');

function postDocumentToES(doc, context) {
```

```
    var req = new AWS.HttpRequest(endpoint);

    req.method = 'POST';
    req.path = '/_bulk';
    req.region = esDomain.region;
    req.body = doc;
    req.headers['presigned-expires'] = false;
    req.headers['Host'] = endpoint.host;

    // Sign the request (Sigv4)
    var signer = new AWS.Signers.V4(req, 'es');
    signer.addAuthorization(creds, new Date());

    // Post document to ES
    var send = new AWS.NodeHttpClient();
    send.handleRequest(req, null, function (httpResp) {
        var body = '';
        httpResp.on('data', function (chunk) {
            body += chunk;
        });
        httpResp.on('end', function (chunk) {
            console.log('Successful', body);
            context.succeed();
        });
    }, function (err) {
        console.log('Error: ' + err);
        context.fail();
    });
}

exports.handler = (event, context, callback) => {
    console.log("event => " + JSON.stringify(event));
    var posts = '';

    for (var i = 0; i < event.Records.length; i++) {
        var eventName = event.Records[i].eventName;
        var actionType = '';
        var image;
        var noDoc = false;
        switch (eventName) {
            case 'INSERT':
                actionType = 'create';
                image = event.Records[i].dynamodb.NewImage;
                break;
```

```
            case 'MODIFY':
                actionType = 'update';
                image = event.Records[i].dynamodb.NewImage;
                break;
            case 'REMOVE':
            actionType = 'delete';
                image = event.Records[i].dynamodb.OldImage;
                noDoc = true;
                break;
        }

        if (typeof image !== "undefined") {
            var postData = {};
            for (var key in image) {
                if (image.hasOwnProperty(key)) {
                    if (key === 'postId') {
                        postData['id'] = image[key].S;
                    } else {
                        var val = image[key];
                        if (val.hasOwnProperty('S')) {
                            postData[key] = val.S;
                        } else if (val.hasOwnProperty('N')) {
                            postData[key] = val.N;
                        }
                    }
                }
            }

            var action = {};
            action[actionType] = {};
            action[actionType]._index = 'id';
            action[actionType]._type = 'post';
            action[actionType]._id = postData['id'];
            posts += [
                JSON.stringify(action),
            ].concat(noDoc?[]:[JSON.stringify(postData)]).join('\n') + '\n';
        }
    }
    console.log('posts:',posts);
    postDocumentToES(posts, context);
};
```

Anda kemudian dapat menggunakan aliran DynamoDB untuk melampirkan ini ke tabel DynamoDB dengan kunci `id` utama, dan setiap perubahan pada sumber DynamoDB akan mengalir ke domain Layanan Anda. OpenSearch Untuk informasi selengkapnya tentang mengonfigurasi ini, lihat dokumentasi DynamoDB Streams.

Tutorial: Penyelesai Batch DynamoDB

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync mendukung penggunaan operasi batch Amazon DynamoDB di satu atau beberapa tabel dalam satu wilayah. Operasi yang didukung adalah`BatchGetItem`,`BatchPutItem`, dan`BatchDeleteItem`. Dengan menggunakan fitur-fitur ini di AWS AppSync, Anda dapat melakukan tugas-tugas seperti:

- Lewati daftar kunci dalam satu kueri dan kembalikan hasil dari tabel
- Membaca catatan dari satu atau beberapa tabel dalam satu kueri
- Tulis catatan secara massal ke satu atau lebih tabel
- Menulis atau menghapus catatan secara kondisional dalam beberapa tabel yang mungkin memiliki hubungan

Menggunakan operasi batch dengan DynamoDB AWS AppSync in adalah teknik canggih yang membutuhkan sedikit pemikiran dan pengetahuan ekstra tentang operasi backend dan struktur tabel Anda. Selain itu, operasi batch AWS AppSync memiliki dua perbedaan utama dari operasi non-batch:

- Peran sumber data harus memiliki izin ke semua tabel yang akan diakses oleh resolver.
- Spesifikasi tabel untuk resolver adalah bagian dari template pemetaan.

Izin

Seperti resolver lainnya, Anda perlu membuat sumber data AWS AppSync dan membuat peran atau menggunakan yang sudah ada. Karena operasi batch memerlukan izin yang berbeda pada tabel DynamoDB, Anda perlu memberikan izin peran yang dikonfigurasi untuk tindakan baca atau tulis:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:region:account:table/TABLENAME",
                "arn:aws:dynamodb:region:account:table/TABLENAME/*"
            ]
        }
    ]
}
```

Catatan: Peran terkait dengan sumber data di AWS AppSync, dan resolver pada bidang dipanggil terhadap sumber data. Sumber data yang dikonfigurasi untuk mengambil terhadap DynamoDB hanya memiliki satu tabel yang ditentukan, untuk menjaga konfigurasi tetap sederhana. Oleh karena itu, saat melakukan operasi batch terhadap beberapa tabel dalam satu resolver, yang merupakan tugas yang lebih maju, Anda harus memberikan peran pada akses sumber data tersebut ke tabel mana pun yang akan berinteraksi dengan resolver. Ini akan dilakukan di bidang Sumber Daya dalam kebijakan IAM di atas. Konfigurasi tabel untuk membuat panggilan batch terhadap dilakukan dalam template resolver, yang kami jelaskan di bawah ini.

Sumber Data

Demi kesederhanaan, kita akan menggunakan sumber data yang sama untuk semua resolver yang digunakan dalam tutorial ini. Pada tab Sumber data, buat sumber data DynamoDB baru dan beri nama. BatchTutorial Nama tabel dapat berupa apa saja karena nama tabel ditentukan sebagai bagian dari template pemetaan permintaan untuk operasi batch. Kami akan memberikan nama tabel`empty`.

Untuk tutorial ini, peran apa pun dengan kebijakan inline berikut akan berfungsi:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:region:account:table/Posts",
                "arn:aws:dynamodb:region:account:table/Posts/*",
                "arn:aws:dynamodb:region:account:table/locationReadings",
                "arn:aws:dynamodb:region:account:table/locationReadings/*",
                "arn:aws:dynamodb:region:account:table/temperatureReadings",
                "arn:aws:dynamodb:region:account:table/temperatureReadings/*"
            ]
        }
    ]
}
```

Batch Tabel Tunggal

Untuk contoh ini, misalkan Anda memiliki satu tabel bernama Posts yang ingin Anda tambahkan dan hapus item dengan operasi batch. Gunakan skema berikut, perhatikan bahwa untuk kueri, kami akan meneruskan daftar ID:

```
type Post {
    id: ID!
    title: String
}

input PostInput {
    id: ID!
    title: String
}

type Query {
    batchGet(ids: [ID]): [Post]
}

type Mutation {
    batchAdd(posts: [PostInput]): [Post]
    batchDelete(ids: [ID]): [Post]
}

schema {
```

```
    query: Query
    mutation: Mutation
}
```

Lampirkan resolver ke `batchAdd()` bidang dengan Template Pemetaan Permintaan berikut. Ini secara otomatis mengambil setiap item dalam tipe `input PostInput` GraphQL dan membangun peta, yang diperlukan untuk operasi: `BatchPutItem`

```
#set($postsdata = [])
#foreach($item in ${ctx.args.posts})
    $util.qr($postsdata.add($util.dynamodb.toMapValues($item)))
#end

{
    "version" : "2018-05-29",
    "operation" : "BatchPutItem",
    "tables" : {
        "Posts": $utils.toJson($postsdata)
    }
}
```

Dalam hal ini, Response Mapping Template adalah passthrough sederhana, tetapi nama tabel ditambahkan `..data.Posts` ke objek konteks sebagai berikut:

```
$util.toJson($ctx.result.data.Posts)
```

Sekarang navigasikan ke halaman Kueri AWS AppSync konsol dan jalankan mutasi batchAdd berikut:

```
mutation add {
    batchAdd(posts:[{
            id: 1 title: "Running in the Park"},{
            id: 2 title: "Playing fetch"
        }]){
            id
            title
    }
}
```

Anda akan melihat hasil yang dicetak ke layar, dan dapat secara independen memvalidasi melalui konsol DynamoDB yang ditulis oleh kedua nilai ke tabel Posts.

Selanjutnya, lampirkan resolver ke `batchGet()` bidang dengan Template Pemetaan Permintaan berikut. Ini secara otomatis mengambil setiap item dalam tipe `ids:[]` GraphQL dan membangun peta yang diperlukan untuk operasi: `BatchGetItem`

```
#set($ids = [])
#foreach($id in ${ctx.args.ids})
    #set($map = {})
    $util.qr($map.put("id", $util.dynamodb.toString($id)))
    $util.qr($ids.add($map))
#end

{
    "version" : "2018-05-29",
    "operation" : "BatchGetItem",
    "tables" : {
        "Posts": {
            "keys": $util.toJson($ids),
            "consistentRead": true,
            "projection" : {
                "expression" : "#id, title",
                "expressionNames" : { "#id" : "id"}
                }
        }
    }
}
```

Template Pemetaan Respons sekali lagi merupakan passthrough sederhana, dengan sekali lagi nama tabel ditambahkan `..data.Posts` ke objek konteks:

```
$util.toJson($ctx.result.data.Posts)
```

Sekarang kembali ke halaman Queries AWS AppSync konsol, dan jalankan BatchGet Query berikut:

```
query get {
    batchGet(ids:[1,2,3]){
        id
        title
    }
}
```

Ini akan mengembalikan hasil untuk dua `id` nilai yang Anda tambahkan sebelumnya. Perhatikan bahwa `null` nilai yang dikembalikan untuk `id` dengan nilai`3`. Ini karena belum ada catatan di tabel Posts Anda dengan nilai itu. Perhatikan juga bahwa AWS AppSync mengembalikan hasil dalam urutan yang sama dengan kunci yang diteruskan ke kueri, yang merupakan fitur tambahan yang AWS AppSync dilakukan atas nama Anda. Jadi jika Anda beralih ke`batchGet(ids:[1,3,2)`, Anda akan melihat urutannya berubah. Anda juga akan tahu mana yang `id` mengembalikan `null` nilai.

Terakhir, lampirkan resolver ke `batchDelete()` bidang dengan Template Pemetaan Permintaan berikut. Ini secara otomatis mengambil setiap item dalam tipe `ids:[]` GraphQL dan membangun peta yang diperlukan untuk operasi: `BatchGetItem`

```
#set($ids = [])
#foreach($id in ${ctx.args.ids})
    #set($map = {})
    $util.qr($map.put("id", $util.dynamodb.toString($id)))
    $util.qr($ids.add($map))
#end

{
    "version" : "2018-05-29",
    "operation" : "BatchDeleteItem",
    "tables" : {
        "Posts": $util.toJson($ids)
    }
}
```

Template Pemetaan Respons sekali lagi merupakan passthrough sederhana, dengan sekali lagi nama tabel ditambahkan `..data.Posts` ke objek konteks:

```
$util.toJson($ctx.result.data.Posts)
```

Sekarang kembali ke halaman Queries AWS AppSync konsol, dan jalankan mutasi BatchDelete berikut:

```
mutation delete {
    batchDelete(ids:[1,2]){ id }
}
```

Catatan dengan `id` 1 dan sekarang 2 harus dihapus. Jika Anda menjalankan kembali `batchGet()` kueri dari sebelumnya, ini akan kembali`null`.

Batch Multi-Tabel

AWS AppSync juga memungkinkan Anda untuk melakukan operasi batch di seluruh tabel. Mari kita membangun aplikasi yang lebih kompleks. Bayangkan kita sedang membangun aplikasi Kesehatan Hewan Peliharaan, di mana sensor melaporkan lokasi hewan peliharaan dan suhu tubuh. Sensor bertenaga baterai dan mencoba untuk terhubung ke jaringan setiap beberapa menit. Ketika sensor membuat koneksi, ia mengirimkan bacaannya ke AWS AppSync API kami. Pemicu kemudian menganalisis data sehingga dasbor dapat disajikan kepada pemilik hewan peliharaan. Mari kita fokus pada merepresentasikan interaksi antara sensor dan penyimpanan data backend.

Sebagai prasyarat, pertama-tama mari kita buat dua tabel DynamoDB; LocationReadings akan menyimpan pembacaan lokasi sensor dan TemperatureReadings akan menyimpan pembacaan suhu sensor. Kedua tabel kebetulan berbagi struktur kunci utama yang sama: `sensorId (String)` menjadi kunci partisi, dan `timestamp (String)` kunci sortir.

Mari kita gunakan skema GraphQL berikut:

```
type Mutation {
    # Register a batch of readings
    recordReadings(tempReadings: [TemperatureReadingInput], locReadings:
 [LocationReadingInput]): RecordResult
    # Delete a batch of readings
    deleteReadings(tempReadings: [TemperatureReadingInput], locReadings:
 [LocationReadingInput]): RecordResult
}

type Query {
    # Retrieve all possible readings recorded by a sensor at a specific time
    getReadings(sensorId: ID!, timestamp: String!): [SensorReading]
}

type RecordResult {
    temperatureReadings: [TemperatureReading]
    locationReadings: [LocationReading]
}

interface SensorReading {
    sensorId: ID!
    timestamp: String!
}

# Sensor reading representing the sensor temperature (in Fahrenheit)
```

```
type TemperatureReading implements SensorReading {
    sensorId: ID!
    timestamp: String!
    value: Float
}

# Sensor reading representing the sensor location (lat,long)
type LocationReading implements SensorReading {
    sensorId: ID!
    timestamp: String!
    lat: Float
    long: Float
}

input TemperatureReadingInput {
    sensorId: ID!
    timestamp: String
    value: Float
}

input LocationReadingInput {
    sensorId: ID!
    timestamp: String
    lat: Float
    long: Float
}
```

BatchPutItem - Bacaan Sensor Perekaman

Sensor kami harus dapat mengirim bacaan mereka setelah mereka terhubung ke internet. `Mutation.recordReadings`Bidang GraphQL adalah API yang akan mereka gunakan untuk melakukannya. Mari lampirkan resolver untuk menghidupkan API kita.

Pilih Lampirkan di sebelah `Mutation.recordReadings` bidang. Pada layar berikutnya, pilih sumber `BatchTutorial` data yang sama yang dibuat di awal tutorial.

Mari tambahkan template pemetaan permintaan berikut:

Templat Pemetaan Permintaan

```
## Convert tempReadings arguments to DynamoDB objects
#set($tempReadings = [])
#foreach($reading in ${ctx.args.tempReadings})
```

```
    $util.qr($tempReadings.add($util.dynamodb.toMapValues($reading)))
#end

## Convert locReadings arguments to DynamoDB objects
#set($locReadings = [])
#foreach($reading in ${ctx.args.locReadings})
    $util.qr($locReadings.add($util.dynamodb.toMapValues($reading)))
#end

{
    "version" : "2018-05-29",
    "operation" : "BatchPutItem",
    "tables" : {
        "locationReadings": $utils.toJson($locReadings),
        "temperatureReadings": $utils.toJson($tempReadings)
    }
}
```

Seperti yang Anda lihat, `BatchPutItem` operasi memungkinkan kita untuk menentukan beberapa tabel.

Mari kita gunakan template pemetaan respons berikut.

Templat Pemetaan Respon

```
## If there was an error with the invocation
## there might have been partial results
#if($ctx.error)
    ## Append a GraphQL error for that field in the GraphQL response
    $utils.appendError($ctx.error.message, $ctx.error.message)
#end
## Also returns data for the field in the GraphQL response
$utils.toJson($ctx.result.data)
```

Dengan operasi batch, mungkin ada kesalahan dan hasil yang dikembalikan dari pemanggilan. Dalam hal ini, kami bebas melakukan beberapa penanganan kesalahan tambahan.

Catatan: Penggunaan `$utils.appendError()` mirip dengan`$util.error()`, dengan perbedaan utama bahwa itu tidak mengganggu evaluasi template pemetaan. Sebaliknya, itu menandakan ada kesalahan dengan bidang, tetapi memungkinkan template untuk dievaluasi dan akibatnya mengembalikan data kembali ke pemanggil. Kami menyarankan Anda menggunakan `$utils.appendError()` ketika aplikasi Anda perlu mengembalikan sebagian hasil.

Simpan resolver dan navigasikan ke halaman Kueri konsol. AWS AppSync Mari kita kirim beberapa pembacaan sensor!

Jalankan mutasi berikut:

```
mutation sendReadings {
  recordReadings(
    tempReadings: [
      {sensorId: 1, value: 85.5, timestamp: "2018-02-01T17:21:05.000+08:00"},
      {sensorId: 1, value: 85.7, timestamp: "2018-02-01T17:21:06.000+08:00"},
      {sensorId: 1, value: 85.8, timestamp: "2018-02-01T17:21:07.000+08:00"},
      {sensorId: 1, value: 84.2, timestamp: "2018-02-01T17:21:08.000+08:00"},
      {sensorId: 1, value: 81.5, timestamp: "2018-02-01T17:21:09.000+08:00"}
    ]
    locReadings: [
      {sensorId: 1, lat: 47.615063, long: -122.333551, timestamp:
 "2018-02-01T17:21:05.000+08:00"},
      {sensorId: 1, lat: 47.615163, long: -122.333552, timestamp:
 "2018-02-01T17:21:06.000+08:00"}
      {sensorId: 1, lat: 47.615263, long: -122.333553, timestamp:
 "2018-02-01T17:21:07.000+08:00"}
      {sensorId: 1, lat: 47.615363, long: -122.333554, timestamp:
 "2018-02-01T17:21:08.000+08:00"}
      {sensorId: 1, lat: 47.615463, long: -122.333555, timestamp:
 "2018-02-01T17:21:09.000+08:00"}
    ]) {
    locationReadings {
      sensorId
      timestamp
      lat
      long
    }
    temperatureReadings {
      sensorId
      timestamp
      value
    }
  }
}
```

Kami mengirim 10 pembacaan sensor dalam satu mutasi, dengan pembacaan dibagi menjadi dua tabel. Gunakan konsol DynamoDB untuk memvalidasi bahwa data muncul di tabel LocationReadings dan TemperatureReadings.

BatchDeleteItem - Menghapus Bacaan Sensor

Demikian pula, kita juga perlu menghapus batch pembacaan sensor. Mari kita gunakan bidang `Mutation.deleteReadings` GraphQL untuk tujuan ini. Pilih Lampirkan di sebelah `Mutation.recordReadings` bidang. Pada layar berikutnya, pilih sumber `BatchTutorial` data yang sama yang dibuat di awal tutorial.

Mari kita gunakan template pemetaan permintaan berikut.

Templat Pemetaan Permintaan

```
## Convert tempReadings arguments to DynamoDB primary keys
#set($tempReadings = [])
#foreach($reading in ${ctx.args.tempReadings})
    #set($pkey = {})
    $util.qr($pkey.put("sensorId", $reading.sensorId))
    $util.qr($pkey.put("timestamp", $reading.timestamp))
    $util.qr($tempReadings.add($util.dynamodb.toMapValues($pkey)))
#end

## Convert locReadings arguments to DynamoDB primary keys
#set($locReadings = [])
#foreach($reading in ${ctx.args.locReadings})
    #set($pkey = {})
    $util.qr($pkey.put("sensorId", $reading.sensorId))
    $util.qr($pkey.put("timestamp", $reading.timestamp))
    $util.qr($locReadings.add($util.dynamodb.toMapValues($pkey)))
#end

{
    "version" : "2018-05-29",
    "operation" : "BatchDeleteItem",
    "tables" : {
        "locationReadings": $utils.toJson($locReadings),
        "temperatureReadings": $utils.toJson($tempReadings)
    }
}
```

Template pemetaan respons sama dengan yang kami gunakan. `Mutation.recordReadings`

Templat Pemetaan Respon

```
## If there was an error with the invocation
```

```
## there might have been partial results
#if($ctx.error)
    ## Append a GraphQL error for that field in the GraphQL response
    $utils.appendError($ctx.error.message, $ctx.error.message)
#end
## Also return data for the field in the GraphQL response
$utils.toJson($ctx.result.data)
```

Simpan resolver dan navigasikan ke halaman Kueri konsol. AWS AppSync Sekarang, mari kita hapus beberapa pembacaan sensor!

Jalankan mutasi berikut:

```
mutation deleteReadings {
  # Let's delete the first two readings we recorded
  deleteReadings(
    tempReadings: [{sensorId: 1, timestamp: "2018-02-01T17:21:05.000+08:00"}]
    locReadings: [{sensorId: 1, timestamp: "2018-02-01T17:21:05.000+08:00"}]) {
    locationReadings {
      sensorId
      timestamp
      lat
      long
    }
    temperatureReadings {
      sensorId
      timestamp
      value
    }
  }
}
```

Validasi melalui konsol DynamoDB bahwa dua pembacaan ini telah dihapus dari tabel LocationReadings dan TemperatureReadings.

BatchGetItem - Ambil Bacaan

Operasi umum lainnya untuk aplikasi Kesehatan Hewan Peliharaan kami adalah mengambil pembacaan untuk sensor pada titik waktu tertentu. Mari kita lampirkan resolver ke bidang GraphQL pada `Query.getReadings` skema kita. Pilih Lampirkan, dan pada layar berikutnya pilih sumber `BatchTutorial` data yang sama yang dibuat di awal tutorial.

Mari tambahkan template pemetaan permintaan berikut.

Templat Pemetaan Permintaan

```
## Build a single DynamoDB primary key,
## as both locationReadings and tempReadings tables
## share the same primary key structure
#set($pkey = {})
$util.qr($pkey.put("sensorId", $ctx.args.sensorId))
$util.qr($pkey.put("timestamp", $ctx.args.timestamp))

{
    "version" : "2018-05-29",
    "operation" : "BatchGetItem",
    "tables" : {
        "locationReadings": {
            "keys": [$util.dynamodb.toMapValuesJson($pkey)],
            "consistentRead": true
        },
        "temperatureReadings": {
            "keys": [$util.dynamodb.toMapValuesJson($pkey)],
            "consistentRead": true
        }
    }
}
```

Perhatikan bahwa kita sekarang menggunakan BatchGetItemoperasi.

Template pemetaan respons kami akan sedikit berbeda karena kami memilih untuk mengembalikan `SensorReading` daftar. Mari kita memetakan hasil pemanggilan ke bentuk yang diinginkan.

Templat Pemetaan Respon

```
## Merge locationReadings and temperatureReadings
## into a single list
## __typename needed as schema uses an interface
#set($sensorReadings = [])

#foreach($locReading in $ctx.result.data.locationReadings)
    $util.qr($locReading.put("__typename", "LocationReading"))
    $util.qr($sensorReadings.add($locReading))
#end
```

```
#foreach($tempReading in $ctx.result.data.temperatureReadings)
    $util.qr($tempReading.put("__typename", "TemperatureReading"))
    $util.qr($sensorReadings.add($tempReading))
#end

$util.toJson($sensorReadings)
```

Simpan resolver dan navigasikan ke halaman Kueri konsol. AWS AppSync Sekarang, mari kita ambil pembacaan sensor!

Jalankan kueri berikut:

```
query getReadingsForSensorAndTime {
  # Let's retrieve the very first two readings
  getReadings(sensorId: 1, timestamp: "2018-02-01T17:21:06.000+08:00") {
    sensorId
    timestamp
    ...on TemperatureReading {
      value
    }
    ...on LocationReading {
      lat
      long
    }
  }
}
```

Kami telah berhasil mendemonstrasikan penggunaan operasi batch DynamoDB menggunakan. AWS AppSync

Penanganan Kesalahan

Dalam AWS AppSync, operasi sumber data terkadang dapat mengembalikan sebagian hasil. Hasil sebagian adalah istilah yang akan kita gunakan untuk menunjukkan ketika output dari suatu operasi terdiri dari beberapa data dan kesalahan. Karena penanganan kesalahan secara inheren spesifik aplikasi, AWS AppSync memberi Anda kesempatan untuk menangani kesalahan dalam template pemetaan respons. Kesalahan pemanggilan resolver, jika ada, tersedia dari konteks sebagai. `$ctx.error` Kesalahan pemanggilan selalu menyertakan pesan dan tipe, dapat diakses sebagai properti `$ctx.error.message` dan. `$ctx.error.type` Selama pemanggilan template pemetaan respons, Anda dapat menangani sebagian hasil dengan tiga cara:

1. menelan kesalahan pemanggilan hanya dengan mengembalikan data
2. meningkatkan kesalahan (menggunakan`$util.error(...)`) dengan menghentikan evaluasi template pemetaan respons, yang tidak akan mengembalikan data apa pun.
3. tambahkan kesalahan (menggunakan`$util.appendError(...)`) dan juga mengembalikan data

Mari kita tunjukkan masing-masing dari tiga poin di atas dengan operasi batch DynamoDB!

Operasi Batch DynamoDB

Dengan operasi batch DynamoDB, ada kemungkinan bahwa batch sebagian selesai. Artinya, ada kemungkinan bahwa beberapa item atau kunci yang diminta dibiarkan tidak diproses. Jika AWS AppSync tidak dapat menyelesaikan batch, item yang belum diproses dan kesalahan pemanggilan akan disetel pada konteksnya.

Kami akan menerapkan penanganan kesalahan menggunakan konfigurasi `Query.getReadings` bidang dari `BatchGetItem` operasi dari bagian sebelumnya dari tutorial ini. Kali ini, mari kita berpura-pura bahwa saat mengeksekusi `Query.getReadings` field, tabel DynamoDB `temperatureReadings` kehabisan throughput yang disediakan. DynamoDB mengangkat ProvisionedThroughputExceededExceptiona pada upaya kedua AWS AppSync dengan memproses elemen yang tersisa dalam batch.

JSON berikut mewakili konteks serial setelah pemanggilan batch DynamoDB tetapi sebelum template pemetaan respons dievaluasi.

```
{
  "arguments": {
    "sensorId": "1",
    "timestamp": "2018-02-01T17:21:05.000+08:00"
  },
  "source": null,
  "result": {
    "data": {
      "temperatureReadings": [
        null
      ],
      "locationReadings": [
        {
          "lat": 47.615063,
          "long": -122.333551,
```

```
          "sensorId": "1",
          "timestamp": "2018-02-01T17:21:05.000+08:00"
        }
      ]
    },
    "unprocessedKeys": {
      "temperatureReadings": [
        {
          "sensorId": "1",
          "timestamp": "2018-02-01T17:21:05.000+08:00"
        }
      ],
      "locationReadings": []
    }
  },
  "error": {
    "type": "DynamoDB:ProvisionedThroughputExceededException",
    "message": "You exceeded your maximum allowed provisioned throughput for a table or
 for one or more global secondary indexes. (...)"
  },
  "outErrors": []
}
```

Beberapa hal yang perlu diperhatikan pada konteksnya:

- kesalahan pemanggilan telah disetel pada konteks di **`$ctx.error`** by AWS AppSync, dan jenis kesalahan telah disetel ke DynamoDB:. ProvisionedThroughputExceededException
- hasil dipetakan per tabel di bawah`$ctx.result.data`, meskipun ada kesalahan
- kunci yang dibiarkan tidak diproses tersedia di`$ctx.result.data.unprocessedKeys`. Di sini, AWS AppSync tidak dapat mengambil item dengan kunci (sensorid:1, stempel waktu: 2018-02-01T 17:21:05.000 + 08:00) karena throughput tabel yang tidak mencukupi.

Catatan: Untuk`BatchPutItem`, itu`$ctx.result.data.unprocessedItems`. Untuk`BatchDeleteItem`, itu`$ctx.result.data.unprocessedKeys`.

Mari kita tangani kesalahan ini dengan tiga cara berbeda.

1. Menelan kesalahan pemanggilan

Mengembalikan data tanpa menangani kesalahan pemanggilan secara efektif menelan kesalahan, membuat hasil untuk bidang GraphQL yang diberikan selalu berhasil.

Template pemetaan respons yang kita tulis sudah tidak asing lagi dan hanya berfokus pada data hasil.

Templat pemetaan respons:

```
$util.toJson($ctx.result.data)
```

Tanggapan GraphQL:

```
{
  "data": {
    "getReadings": [
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00",
        "lat": 47.615063,
        "long": -122.333551
      },
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00",
        "value": 85.5
      }
    ]
  }
}
```

Tidak ada kesalahan yang akan ditambahkan ke respons kesalahan karena hanya data yang ditindaklanjuti.

2. Memunculkan kesalahan untuk membatalkan eksekusi template

Ketika kegagalan sebagian harus diperlakukan sebagai kegagalan total dari perspektif klien, Anda dapat membatalkan eksekusi template untuk mencegah pengembalian data. Metode `$util.error(...)` utilitas mencapai perilaku ini dengan tepat.

Templat pemetaan respons:

```
## there was an error let's mark the entire field
## as failed and do not return any data back in the response
#if ($ctx.error)
```

```
    $util.error($ctx.error.message, $ctx.error.type, null,
 $ctx.result.data.unprocessedKeys)
#end

$util.toJson($ctx.result.data)
```

Tanggapan GraphQL:

```
{
  "data": {
    "getReadings": null
  },
  "errors": [
    {
      "path": [
        "getReadings"
      ],
      "data": null,
      "errorType": "DynamoDB:ProvisionedThroughputExceededException",
      "errorInfo": {
        "temperatureReadings": [
          {
            "sensorId": "1",
            "timestamp": "2018-02-01T17:21:05.000+08:00"
          }
        ],
        "locationReadings": []
      },
      "locations": [
        {
          "line": 58,
          "column": 3
        }
      ],
      "message": "You exceeded your maximum allowed provisioned throughput for a table
 or for one or more global secondary indexes. (...)"
    }
  ]
}
```

Meskipun beberapa hasil mungkin telah dikembalikan dari operasi batch DynamoDB, kami memilih untuk memunculkan kesalahan sehingga bidang *`getReadings`* GraphQL adalah nol dan kesalahan telah ditambahkan ke blok kesalahan respons GraphQL.

3. Menambahkan kesalahan untuk mengembalikan data dan kesalahan

Dalam kasus tertentu, untuk memberikan pengalaman pengguna yang lebih baik, aplikasi dapat mengembalikan sebagian hasil dan memberi tahu klien mereka tentang item yang belum diproses. Klien dapat memutuskan untuk menerapkan coba lagi atau menerjemahkan kesalahan kembali ke pengguna akhir. Ini `$util.appendError(...)` adalah metode utilitas yang memungkinkan perilaku ini dengan membiarkan desainer aplikasi menambahkan kesalahan pada konteks tanpa mengganggu evaluasi template. Setelah mengevaluasi template, AWS AppSync akan memproses kesalahan konteks apa pun dengan menambahkannya ke blok kesalahan respons GraphQL.

Templat pemetaan respons:

```
#if ($ctx.error)
    ## pass the unprocessed keys back to the caller via the `errorInfo` field
    $util.appendError($ctx.error.message, $ctx.error.type, null,
 $ctx.result.data.unprocessedKeys)
#end

$util.toJson($ctx.result.data)
```

Kami meneruskan kesalahan pemanggilan dan elemen UnprocessedKeys di dalam blok kesalahan respons GraphQL. `getReadings`Bidang ini juga mengembalikan sebagian data dari tabel LocationReadings seperti yang Anda lihat pada respons di bawah ini.

Tanggapan GraphQL:

```
{
  "data": {
    "getReadings": [
      null,
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00",
        "value": 85.5
      }
    ]
  },
  "errors": [
    {
      "path": [
        "getReadings"
      ],
```

```
      "data": null,
      "errorType": "DynamoDB:ProvisionedThroughputExceededException",
      "errorInfo": {
        "temperatureReadings": [
          {
            "sensorId": "1",
            "timestamp": "2018-02-01T17:21:05.000+08:00"
          }
        ],
        "locationReadings": []
      },
      "locations": [
        {
          "line": 58,
          "column": 3
        }
      ],
      "message": "You exceeded your maximum allowed provisioned throughput for a table
 or for one or more global secondary indexes. (...)"
    }
  ]
}
```

Tutorial: Penyelesai Transaksi DynamoDB

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync mendukung penggunaan operasi transaksi Amazon DynamoDB di satu atau beberapa tabel dalam satu wilayah. Operasi yang didukung adalah `TransactGetItems` dan`TransactWriteItems`. Dengan menggunakan fitur-fitur ini di AWS AppSync, Anda dapat melakukan tugas-tugas seperti:

- Lewati daftar kunci dalam satu kueri dan kembalikan hasil dari tabel
- Membaca catatan dari satu atau beberapa tabel dalam satu kueri
- Menulis catatan dalam transaksi ke satu atau beberapa tabel dengan all-or-nothing cara
- Jalankan transaksi ketika beberapa kondisi terpenuhi

Izin

Seperti resolver lainnya, Anda perlu membuat sumber data AWS AppSync dan membuat peran atau menggunakan yang sudah ada. Karena operasi transaksi memerlukan izin yang berbeda pada tabel DynamoDB, Anda perlu memberikan izin peran yang dikonfigurasi untuk tindakan baca atau tulis:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:region:accountId:table/TABLENAME",
                "arn:aws:dynamodb:region:accountId:table/TABLENAME/*"
            ]
        }
    ]
}
```

Catatan: Peran terkait dengan sumber data di AWS AppSync, dan resolver pada bidang dipanggil terhadap sumber data. Sumber data yang dikonfigurasi untuk mengambil terhadap DynamoDB hanya memiliki satu tabel yang ditentukan, untuk menjaga konfigurasi tetap sederhana. Oleh karena itu, saat melakukan operasi transaksi terhadap beberapa tabel dalam satu resolver, yang merupakan tugas yang lebih maju, Anda harus memberikan peran pada akses sumber data tersebut ke tabel mana pun yang akan berinteraksi dengan resolver. Ini akan dilakukan di bidang Sumber Daya dalam kebijakan IAM di atas. Konfigurasi panggilan transaksi terhadap tabel dilakukan dalam template resolver, yang kami jelaskan di bawah ini.

Sumber Data

Demi kesederhanaan, kita akan menggunakan sumber data yang sama untuk semua resolver yang digunakan dalam tutorial ini. Pada tab Sumber data, buat sumber data DynamoDB baru dan beri nama. TransactTutorial Nama tabel dapat berupa apa saja karena nama tabel ditentukan sebagai

bagian dari template pemetaan permintaan untuk operasi transaksi. Kami akan memberikan nama tabel`empty`.

Kita akan memiliki dua tabel yang disebut SavingAccounts dan CheckingAccounts, `accountNumber` keduanya dengan kunci partisi, dan tabel TransactionHistory dengan as partition key. `transactionId`

Untuk tutorial ini, peran apa pun dengan kebijakan inline berikut akan berfungsi. Ganti `region` dan `accountId` dengan wilayah dan ID akun Anda:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:region:accountId:table/savingAccounts",
                "arn:aws:dynamodb:region:accountId:table/savingAccounts/*",
                "arn:aws:dynamodb:region:accountId:table/checkingAccounts",
                "arn:aws:dynamodb:region:accountId:table/checkingAccounts/*",
                "arn:aws:dynamodb:region:accountId:table/transactionHistory",
                "arn:aws:dynamodb:region:accountId:table/transactionHistory/*"
            ]
        }
    ]
}
```

Transaksi

Untuk contoh ini, konteksnya adalah transaksi perbankan klasik, di mana kita akan menggunakan `TransactWriteItems` untuk:

- Transfer uang dari rekening tabungan ke rekening giro
- Menghasilkan catatan transaksi baru untuk setiap transaksi

Dan kemudian kita akan menggunakan `TransactGetItems` untuk mengambil rincian dari rekening tabungan dan rekening giro.

Kami mendefinisikan skema GraphQL kami sebagai berikut:

```
type SavingAccount {
    accountNumber: String!
    username: String
    balance: Float
}

type CheckingAccount {
    accountNumber: String!
    username: String
    balance: Float
}

type TransactionHistory {
    transactionId: ID!
    from: String
    to: String
    amount: Float
}

type TransactionResult {
    savingAccounts: [SavingAccount]
    checkingAccounts: [CheckingAccount]
    transactionHistory: [TransactionHistory]
}

input SavingAccountInput {
    accountNumber: String!
    username: String
    balance: Float
}

input CheckingAccountInput {
    accountNumber: String!
    username: String
    balance: Float
}

input TransactionInput {
    savingAccountNumber: String!
```

```
    checkingAccountNumber: String!
    amount: Float!
}

type Query {
    getAccounts(savingAccountNumbers: [String], checkingAccountNumbers: [String]):
 TransactionResult
}

type Mutation {
    populateAccounts(savingAccounts: [SavingAccountInput], checkingAccounts:
 [CheckingAccountInput]): TransactionResult
    transferMoney(transactions: [TransactionInput]): TransactionResult
}

schema {
    query: Query
    mutation: Mutation
}
```

TransactWriteItems - Mengisi Akun

Untuk mentransfer uang antar akun, kita perlu mengisi tabel dengan detailnya. Kami akan menggunakan `Mutation.populateAccounts` operasi GraphQL untuk melakukannya.

Di bagian Skema, klik Lampirkan di sebelah `Mutation.populateAccounts` operasi. Buka Resolver Unit VTL, lalu pilih sumber data yang sama. `TransactTutorial`

Sekarang gunakan template pemetaan permintaan berikut:

Templat Pemetaan Permintaan

```
#set($savingAccountTransactPutItems = [])
#set($index = 0)
#foreach($savingAccount in ${ctx.args.savingAccounts})
    #set($keyMap = {})
    $util.qr($keyMap.put("accountNumber",
 $util.dynamodb.toString($savingAccount.accountNumber)))
    #set($attributeValues = {})
    $util.qr($attributeValues.put("username",
 $util.dynamodb.toString($savingAccount.username)))
    $util.qr($attributeValues.put("balance",
 $util.dynamodb.toNumber($savingAccount.balance)))
```

```
    #set($index = $index + 1)
    #set($savingAccountTransactPutItem = {"table": "savingAccounts",
        "operation": "PutItem",
        "key": $keyMap,
        "attributeValues": $attributeValues})
    $util.qr($savingAccountTransactPutItems.add($savingAccountTransactPutItem))
#end

#set($checkingAccountTransactPutItems = [])
#set($index = 0)
#foreach($checkingAccount in ${ctx.args.checkingAccounts})
    #set($keyMap = {})
    $util.qr($keyMap.put("accountNumber",
 $util.dynamodb.toString($checkingAccount.accountNumber)))
    #set($attributeValues = {})
    $util.qr($attributeValues.put("username",
 $util.dynamodb.toString($checkingAccount.username)))
    $util.qr($attributeValues.put("balance",
 $util.dynamodb.toNumber($checkingAccount.balance)))
    #set($index = $index + 1)
    #set($checkingAccountTransactPutItem = {"table": "checkingAccounts",
        "operation": "PutItem",
        "key": $keyMap,
        "attributeValues": $attributeValues})
    $util.qr($checkingAccountTransactPutItems.add($checkingAccountTransactPutItem))
#end

#set($transactItems = [])
$util.qr($transactItems.addAll($savingAccountTransactPutItems))
$util.qr($transactItems.addAll($checkingAccountTransactPutItems))

{
    "version" : "2018-05-29",
    "operation" : "TransactWriteItems",
    "transactItems" : $util.toJson($transactItems)
}
```

Dan template pemetaan respons berikut:

Templat Pemetaan Respon

```
#if ($ctx.error)
    $util.appendError($ctx.error.message, $ctx.error.type, null,
 $ctx.result.cancellationReasons)
```

```
#end

#set($savingAccounts = [])
#foreach($index in [0..2])
    $util.qr($savingAccounts.add(${ctx.result.keys[$index]}))
#end

#set($checkingAccounts = [])
#foreach($index in [3..5])
    $util.qr($checkingAccounts.add(${ctx.result.keys[$index]}))
#end

#set($transactionResult = {})
$util.qr($transactionResult.put('savingAccounts', $savingAccounts))
$util.qr($transactionResult.put('checkingAccounts', $checkingAccounts))

$util.toJson($transactionResult)
```

Simpan resolver dan navigasikan ke bagian Kueri AWS AppSync konsol untuk mengisi akun.

Jalankan mutasi berikut:

```
mutation populateAccounts {
  populateAccounts (
    savingAccounts: [
      {accountNumber: "1", username: "Tom", balance: 100},
      {accountNumber: "2", username: "Amy", balance: 90},
      {accountNumber: "3", username: "Lily", balance: 80},
    ]
    checkingAccounts: [
      {accountNumber: "1", username: "Tom", balance: 70},
      {accountNumber: "2", username: "Amy", balance: 60},
      {accountNumber: "3", username: "Lily", balance: 50},
    ]) {
    savingAccounts {
      accountNumber
    }
    checkingAccounts {
      accountNumber
    }
  }
}
```

Kami mengisi 3 rekening tabungan dan 3 rekening giro dalam satu mutasi.

Gunakan konsol DynamoDB untuk memvalidasi bahwa data muncul di tabel SavingAccounts dan CheckingAccounts.

TransactWriteItems - Transfer Uang

Lampirkan resolver ke `transferMoney` mutasi dengan Template Pemetaan Permintaan berikut. Perhatikan nilai-nilai`amounts`,`savingAccountNumbers`, `checkingAccountNumbers` dan sama.

```
#set($amounts = [])
#foreach($transaction in ${ctx.args.transactions})
    #set($attributeValueMap = {})
    $util.qr($attributeValueMap.put(":amount",
 $util.dynamodb.toNumber($transaction.amount)))
    $util.qr($amounts.add($attributeValueMap))
#end

#set($savingAccountTransactUpdateItems = [])
#set($index = 0)
#foreach($transaction in ${ctx.args.transactions})
    #set($keyMap = {})
    $util.qr($keyMap.put("accountNumber",
 $util.dynamodb.toString($transaction.savingAccountNumber)))
    #set($update = {})
    $util.qr($update.put("expression", "SET balance = balance - :amount"))
    $util.qr($update.put("expressionValues", $amounts[$index]))
    #set($index = $index + 1)
    #set($savingAccountTransactUpdateItem = {"table": "savingAccounts",
        "operation": "UpdateItem",
        "key": $keyMap,
        "update": $update})
    $util.qr($savingAccountTransactUpdateItems.add($savingAccountTransactUpdateItem))
#end

#set($checkingAccountTransactUpdateItems = [])
#set($index = 0)
#foreach($transaction in ${ctx.args.transactions})
    #set($keyMap = {})
    $util.qr($keyMap.put("accountNumber",
 $util.dynamodb.toString($transaction.checkingAccountNumber)))
    #set($update = {})
    $util.qr($update.put("expression", "SET balance = balance + :amount"))
```

```
    $util.qr($update.put("expressionValues", $amounts[$index]))
    #set($index = $index + 1)
    #set($checkingAccountTransactUpdateItem = {"table": "checkingAccounts",
        "operation": "UpdateItem",
        "key": $keyMap,
        "update": $update})

 $util.qr($checkingAccountTransactUpdateItems.add($checkingAccountTransactUpdateItem))
#end

#set($transactionHistoryTransactPutItems = [])
#foreach($transaction in ${ctx.args.transactions})
    #set($keyMap = {})
    $util.qr($keyMap.put("transactionId", $util.dynamodb.toString(${utils.autoId()})))
    #set($attributeValues = {})
    $util.qr($attributeValues.put("from",
 $util.dynamodb.toString($transaction.savingAccountNumber)))
    $util.qr($attributeValues.put("to",
 $util.dynamodb.toString($transaction.checkingAccountNumber)))
    $util.qr($attributeValues.put("amount",
 $util.dynamodb.toNumber($transaction.amount)))
    #set($transactionHistoryTransactPutItem = {"table": "transactionHistory",
        "operation": "PutItem",
        "key": $keyMap,
        "attributeValues": $attributeValues})

 $util.qr($transactionHistoryTransactPutItems.add($transactionHistoryTransactPutItem))
#end

#set($transactItems = [])
$util.qr($transactItems.addAll($savingAccountTransactUpdateItems))
$util.qr($transactItems.addAll($checkingAccountTransactUpdateItems))
$util.qr($transactItems.addAll($transactionHistoryTransactPutItems))

{
    "version" : "2018-05-29",
    "operation" : "TransactWriteItems",
    "transactItems" : $util.toJson($transactItems)
}
```

Kami akan memiliki 3 transaksi perbankan dalam satu `TransactWriteItems` operasi. Gunakan Template Pemetaan Respons berikut:

```
#if ($ctx.error)
    $util.appendError($ctx.error.message, $ctx.error.type, null,
 $ctx.result.cancellationReasons)
#end

#set($savingAccounts = [])
#foreach($index in [0..2])
    $util.qr($savingAccounts.add(${ctx.result.keys[$index]}))
#end

#set($checkingAccounts = [])
#foreach($index in [3..5])
    $util.qr($checkingAccounts.add(${ctx.result.keys[$index]}))
#end

#set($transactionHistory = [])
#foreach($index in [6..8])
    $util.qr($transactionHistory.add(${ctx.result.keys[$index]}))
#end

#set($transactionResult = {})
$util.qr($transactionResult.put('savingAccounts', $savingAccounts))
$util.qr($transactionResult.put('checkingAccounts', $checkingAccounts))
$util.qr($transactionResult.put('transactionHistory', $transactionHistory))

$util.toJson($transactionResult)
```

Sekarang arahkan ke bagian Kueri AWS AppSync konsol dan jalankan mutasi TransferMoney sebagai berikut:

```
mutation write {
  transferMoney(
    transactions: [
      {savingAccountNumber: "1", checkingAccountNumber: "1", amount: 7.5},
      {savingAccountNumber: "2", checkingAccountNumber: "2", amount: 6.0},
      {savingAccountNumber: "3", checkingAccountNumber: "3", amount: 3.3}
    ]) {
    savingAccounts {
      accountNumber
    }
    checkingAccounts {
      accountNumber
    }
```

```
    transactionHistory {
      transactionId
    }
  }
}
```

Kami mengirim 2 transaksi perbankan dalam satu mutasi. Gunakan konsol DynamoDB untuk memvalidasi data yang muncul di tabel SavingAccounts, CheckingAccounts, dan TransactionHistory.

TransactGetItems - Ambil Akun

Untuk mengambil detail dari rekening tabungan dan rekening giro dalam satu permintaan transaksional, kami akan melampirkan resolver ke operasi `Query.getAccounts` GraphQL pada skema kami. Pilih Lampirkan, pergi ke VTL Unit Resolvers, lalu pada layar berikutnya, pilih sumber `TransactTutorial` data yang sama yang dibuat di awal tutorial. Konfigurasikan template sebagai berikut:

Templat Pemetaan Permintaan

```
#set($savingAccountsTransactGets = [])
#foreach($savingAccountNumber in ${ctx.args.savingAccountNumbers})
    #set($savingAccountKey = {})
    $util.qr($savingAccountKey.put("accountNumber",
 $util.dynamodb.toString($savingAccountNumber)))
    #set($savingAccountTransactGet = {"table": "savingAccounts", "key":
 $savingAccountKey})
    $util.qr($savingAccountsTransactGets.add($savingAccountTransactGet))
#end

#set($checkingAccountsTransactGets = [])
#foreach($checkingAccountNumber in ${ctx.args.checkingAccountNumbers})
    #set($checkingAccountKey = {})
    $util.qr($checkingAccountKey.put("accountNumber",
 $util.dynamodb.toString($checkingAccountNumber)))
    #set($checkingAccountTransactGet = {"table": "checkingAccounts", "key":
 $checkingAccountKey})
    $util.qr($checkingAccountsTransactGets.add($checkingAccountTransactGet))
#end

#set($transactItems = [])
$util.qr($transactItems.addAll($savingAccountsTransactGets))
$util.qr($transactItems.addAll($checkingAccountsTransactGets))
```

```
{
    "version" : "2018-05-29",
    "operation" : "TransactGetItems",
    "transactItems" : $util.toJson($transactItems)
}
```

Templat Pemetaan Respon

```
#if ($ctx.error)
    $util.appendError($ctx.error.message, $ctx.error.type, null,
 $ctx.result.cancellationReasons)
#end

#set($savingAccounts = [])
#foreach($index in [0..2])
    $util.qr($savingAccounts.add(${ctx.result.items[$index]}))
#end

#set($checkingAccounts = [])
#foreach($index in [3..4])
    $util.qr($checkingAccounts.add($ctx.result.items[$index]))
#end

#set($transactionResult = {})
$util.qr($transactionResult.put('savingAccounts', $savingAccounts))
$util.qr($transactionResult.put('checkingAccounts', $checkingAccounts))

$util.toJson($transactionResult)
```

Simpan resolver dan navigasikan ke bagian Kueri konsol. AWS AppSync Untuk mengambil akun tabungan dan memeriksa akun, jalankan kueri berikut:

```
query getAccounts {
  getAccounts(
    savingAccountNumbers: ["1", "2", "3"],
    checkingAccountNumbers: ["1", "2"]
  ) {
    savingAccounts {
      accountNumber
      username
      balance
    }
    checkingAccounts {
```

```
      accountNumber
      username
      balance
    }
  }
}
```

Kami telah berhasil menunjukkan penggunaan transaksi DynamoDB menggunakan. AWS AppSync

Tutorial: Resolver HTTP

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync memungkinkan Anda menggunakan sumber data yang didukung (yaitu, Amazon DynamoDBAWS Lambda, Amazon Service, atau OpenSearch Amazon Aurora) untuk melakukan berbagai operasi, selain titik akhir HTTP arbitrer untuk menyelesaikan bidang GraphQL. Setelah titik akhir HTTP Anda tersedia, Anda dapat menghubungkannya menggunakan sumber data. Kemudian, Anda dapat mengonfigurasi resolver dalam skema untuk melakukan operasi GraphQL seperti kueri, mutasi, dan langganan. Tutorial ini memandu Anda melalui beberapa contoh umum.

Dalam tutorial ini Anda menggunakan REST API (dibuat menggunakan Amazon API Gateway dan Lambda) dengan titik akhir AWS AppSync GraphQL.

Pengaturan Satu-Klik

Jika ingin mengatur titik akhir GraphQL secara otomatis dengan titik akhir HTTP yang AWS AppSync dikonfigurasi (menggunakan Amazon API Gateway dan Lambda), Anda dapat menggunakan templat berikut: AWS CloudFormation

Membuat REST API

Anda dapat menggunakan AWS CloudFormation template berikut untuk menyiapkan titik akhir REST yang berfungsi untuk tutorial ini:

AWS CloudFormationTumpukan melakukan langkah-langkah berikut:

1. Menyiapkan fungsi Lambda yang berisi logika bisnis Anda untuk layanan mikro Anda.
2. Menyiapkan API REST API Gateway dengan kombinasi titik akhir/metode/tipe konten berikut:

Jalur Sumber Daya API	Metode HTTP	Jenis Konten yang Didukung
/v1/pengguna	POST	aplikasi/json
/v1/pengguna	DAPATKAN	aplikasi/json
/v1/pengguna/1	DAPATKAN	aplikasi/json
/v1/pengguna/1	PUT	aplikasi/json
/v1/pengguna/1	HAPUS	aplikasi/json

Membuat GraphQL API Anda

Untuk membuat GraphQL API di: AWS AppSync

- Buka AWS AppSync konsol dan pilih Buat API.
- Untuk nama API, ketik`UserData`.
- Pilih Skema kustom.
- Pilih Create (Buat).

AWS AppSync Konsol membuat API GraphQL baru untuk Anda menggunakan mode autentikasi kunci API. Anda dapat menggunakan konsol untuk mengatur sisa GraphQL API dan menjalankan kueri di atasnya untuk sisa tutorial ini.

Membuat Skema GraphQL

Sekarang Anda memiliki GraphQL API, mari buat skema GraphQL. Dari editor skema di AWS AppSync konsol, pastikan skema Anda cocok dengan skema berikut:

```
schema {
    query: Query
    mutation: Mutation
}

type Mutation {
    addUser(userInput: UserInput!): User
    deleteUser(id: ID!): User
}

type Query {
    getUser(id: ID): User
    listUser: [User!]!
}

type User {
    id: ID!
    username: String!
    firstname: String
    lastname: String
    phone: String
    email: String
}

input UserInput {
    id: ID!
    username: String!
    firstname: String
    lastname: String
    phone: String
    email: String
}
```

Konfigurasikan Sumber Data HTTP Anda

Untuk mengonfigurasi sumber data HTTP Anda, lakukan hal berikut:

- Pada DataSourcestab, pilih Baru, lalu ketik nama ramah untuk sumber data (misalnya,HTTP).
- Di tipe sumber data, pilih HTTP.
- Setel titik akhir ke titik akhir API Gateway yang dibuat. Pastikan Anda tidak menyertakan nama panggung sebagai bagian dari titik akhir.

Catatan: Saat ini hanya titik akhir publik yang didukung oleh AWS AppSync.

Catatan: Untuk informasi selengkapnya tentang otoritas sertifikasi yang diakui oleh AWS AppSync layanan, lihat Otoritas Sertifikat (CA) yang Diakui oleh AWS AppSync untuk Titik Akhir HTTPS.

Mengkonfigurasi Resolver

Pada langkah ini, Anda menghubungkan sumber data http ke kueri getUser.

Untuk mengatur resolver:

- Pilih tab Skema.
- Di panel tipe Data di sebelah kanan di bawah tipe Query, temukan bidang getUser dan pilih Lampirkan.
- Di nama sumber data, pilih HTTP.
- Di Konfigurasikan templat pemetaan permintaan, tempel kode berikut:

```
{
    "version": "2018-05-29",
    "method": "GET",
    "params": {
        "headers": {
            "Content-Type": "application/json"
        }
    },
    "resourcePath": $util.toJson("/v1/users/${ctx.args.id}")
}
```

- Di Konfigurasikan template pemetaan respons, tempel kode berikut:

```
## return the body
#if($ctx.result.statusCode == 200)
    ##if response is 200
    $ctx.result.body
#else
    ##if response is not 200, append the response to error block.
    $utils.appendError($ctx.result.body, "$ctx.result.statusCode")
#end
```

- Pilih tab Query, dan kemudian jalankan query berikut:

```
query GetUser{
    getUser(id:1){
        id
        username
    }
}
```

Ini akan mengembalikan respons berikut:

```
{
    "data": {
        "getUser": {
            "id": "1",
            "username": "nadia"
        }
    }
}
```

- Pilih tab Skema.
- Di panel tipe Data di sebelah kanan bawah Mutasi, temukan bidang AddUser dan pilih Lampirkan.
- Di nama sumber data, pilih HTTP.
- Di Konfigurasikan templat pemetaan permintaan, tempel kode berikut:

```
{
    "version": "2018-05-29",
    "method": "POST",
    "resourcePath": "/v1/users",
    "params":{
      "headers":{
        "Content-Type": "application/json",
      },
      "body": $util.toJson($ctx.args.userInput)
    }
}
```

- Di Konfigurasikan template pemetaan respons, tempel kode berikut:

```
## Raise a GraphQL field error in case of a datasource invocation error
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end
## if the response status code is not 200, then return an error. Else return the body
 **
#if($ctx.result.statusCode == 200)
    ## If response is 200, return the body.
    $ctx.result.body
#else
    ## If response is not 200, append the response to error block.
    $utils.appendError($ctx.result.body, "$ctx.result.statusCode")
#end
```

- Pilih tab Query, dan kemudian jalankan query berikut:

```
mutation addUser{
    addUser(userInput:{
        id:"2",
        username:"shaggy"
    }){
        id
        username
    }
}
```

Ini akan mengembalikan respons berikut:

```
{
    "data": {
        "getUser": {
        "id": "2",
        "username": "shaggy"
        }
    }
}
```

Layanan Pemanggilan AWS

Anda dapat menggunakan resolver HTTP untuk menyiapkan antarmuka GraphQL API untuk layanan. AWS Permintaan HTTP AWS harus ditandatangani dengan proses Signature Version 4 sehingga AWS dapat mengidentifikasi siapa yang mengirimnya. AWS AppSync menghitung tanda tangan atas nama Anda saat Anda mengaitkan peran IAM dengan sumber data HTTP.

Anda menyediakan dua komponen tambahan untuk memanggil AWS layanan dengan resolver HTTP:

- Peran IAM dengan izin untuk memanggil API layanan AWS
- Menandatangani konfigurasi di sumber data

Misalnya, jika Anda ingin memanggil ListGraphqlApis operasi dengan resolver HTTP, pertama-tama Anda membuat peran IAM yang AWS AppSync diasumsikan dengan kebijakan berikut yang dilampirkan:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "appsync:ListGraphqlApis"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Selanjutnya, buat sumber data HTTP untuk AWS AppSync. Dalam contoh ini, Anda menelepon AWS AppSync di Wilayah Barat AS (Oregon). Siapkan konfigurasi HTTP berikut dalam file bernama`http.json`, yang mencakup wilayah penandatanganan dan nama layanan:

```
{
    "endpoint": "https://appsync.us-west-2.amazonaws.com/",
    "authorizationConfig": {
        "authorizationType": "AWS_IAM",
        "awsIamConfig": {
            "signingRegion": "us-west-2",
            "signingServiceName": "appsync"
```

```
        }
    }
}
```

Kemudian, gunakan AWS CLI untuk membuat sumber data dengan peran terkait sebagai berikut:

```
aws appsync create-data-source --api-id <API-ID> \
                               --name AWSAppSync \
                               --type HTTP \
                               --http-config file:///http.json \
                               --service-role-arn <ROLE-ARN>
```

Saat Anda melampirkan resolver ke bidang dalam skema, gunakan templat pemetaan permintaan berikut untuk memanggil: AWS AppSync

```
{
    "version": "2018-05-29",
    "method": "GET",
    "resourcePath": "/v1/apis"
}
```

Saat Anda menjalankan kueri GraphQL untuk sumber data ini AWS AppSync , tandatangani permintaan menggunakan peran yang Anda berikan dan menyertakan tanda tangan dalam permintaan. Kueri menampilkan daftar API AWS AppSync GraphQL di akun Anda di Wilayah tersebut. AWS

Tutorial: Aurora Tanpa Server

AWS AppSyncmenyediakan sumber data untuk menjalankan perintah SQL terhadap klaster Amazon Aurora Tanpa Server yang telah diaktifkan dengan API Data. Anda dapat menggunakan AppSync resolver untuk mengeksekusi pernyataan SQL terhadap Data API dengan kueri, mutasi, dan langganan GraphQL.

Buat cluster

Sebelum menambahkan sumber data RDS ke AppSync Anda harus terlebih dahulu mengaktifkan API Data pada cluster Aurora Tanpa Server dan mengonfigurasi rahasia menggunakan. AWS Secrets Manager Anda dapat membuat cluster Aurora Tanpa Server terlebih dahulu dengan: AWS CLI

```
aws rds create-db-cluster --db-cluster-identifier http-endpoint-test  --master-username
 USERNAME \
--master-user-password COMPLEX_PASSWORD --engine aurora --engine-mode serverless \
--region us-east-1
```

Ini akan mengembalikan ARN untuk cluster.

Buat Rahasia melalui AWS Secrets Manager Konsol atau juga melalui CLI dengan file input seperti berikut menggunakan USERNAME dan COMPLEX_PASSWORD dari langkah sebelumnya:

```
{
    "username": "USERNAME",
    "password": "COMPLEX_PASSWORD"
}
```

Berikan ini sebagai parameter keAWS CLI:

```
aws secretsmanager create-secret --name HttpRDSSecret --secret-string file://creds.json
 --region us-east-1
```

Ini akan mengembalikan ARN untuk rahasianya.

Perhatikan ARN cluster Aurora Tanpa Server Anda dan Rahasia untuk digunakan nanti di AppSync konsol saat membuat sumber data.

Aktifkan API Data

Anda dapat mengaktifkan Data API pada klaster Anda dengan mengikuti petunjuk dalam dokumentasi RDS. Data API harus diaktifkan sebelum menambahkan sebagai sumber AppSync data.

Buat database dan tabel

Setelah Anda mengaktifkan API Data Anda, Anda dapat memastikannya berfungsi dengan `aws rds-data execute-statement` perintah diAWS CLI. Ini akan memastikan bahwa cluster Aurora Serverless Anda dikonfigurasi dengan benar sebelum menambahkannya ke API Anda. AppSync Pertama membuat database yang disebut TESTDB dengan `--sql` parameter seperti:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789000:cluster:http-endpoint-test" \
```

```
--schema "mysql"  --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789000:secret:testHttp2-AmNvc1"  \
--region us-east-1 --sql "create DATABASE TESTDB"
```

Jika ini berjalan tanpa kesalahan, tambahkan tabel dengan perintah create table:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789000:cluster:http-endpoint-test" \
 --schema "mysql"  --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789000:secret:testHttp2-AmNvc1" \
 --region us-east-1 \
 --sql "create table Pets(id varchar(200), type varchar(200), price float)" --database
 "TESTDB"
```

Jika semuanya berjalan tanpa masalah, Anda dapat melanjutkan untuk menambahkan cluster sebagai sumber data di AppSync API Anda.

GraphQL skema

Sekarang setelah Aurora Serverless Data API Anda aktif dan berjalan dengan tabel, kami akan membuat skema GraphQL dan melampirkan resolver untuk melakukan mutasi dan langganan. Buat API baru di AWS AppSync konsol dan arahkan ke halaman Skema, lalu masukkan yang berikut ini:

```
type Mutation {
    createPet(input: CreatePetInput!): Pet
    updatePet(input: UpdatePetInput!): Pet
    deletePet(input: DeletePetInput!): Pet
}

input CreatePetInput {
    type: PetType
    price: Float!
}

input UpdatePetInput {
id: ID!
    type: PetType
    price: Float!
}

input DeletePetInput {
```

```
    id: ID!
}

type Pet {
    id: ID!
    type: PetType
    price: Float
}

enum PetType {
    dog
    cat
    fish
    bird
    gecko
}

type Query {
    getPet(id: ID!): Pet
    listPets: [Pet]
    listPetsByPriceRange(min: Float, max: Float): [Pet]
}

schema {
    query: Query
    mutation: Mutation
}
```

Simpan skema Anda dan arahkan ke halaman Sumber Data dan buat sumber data baru. Pilih Database relasional untuk tipe sumber data, dan berikan nama yang ramah. Gunakan nama database yang Anda buat pada langkah terakhir, serta ARN Cluster tempat Anda membuatnya. Untuk Peran, Anda dapat AppSync membuat peran baru atau membuat peran dengan kebijakan yang mirip dengan di bawah ini:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-data:DeleteItems",
                "rds-data:ExecuteSql",
                "rds-data:ExecuteStatement",
```

```
                "rds-data:GetItems",
                "rds-data:InsertItems",
                "rds-data:UpdateItems"
            ],
            "Resource": [
                "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster",
                "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret",
                "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret:*"
            ]
        }
    ]
}
```

Perhatikan ada dua Pernyataan dalam kebijakan ini yang Anda berikan akses peran. Sumber daya pertama adalah cluster Aurora Tanpa Server Anda dan yang kedua adalah ARN Anda. AWS Secrets Manager Anda harus menyediakan KEDUA ARN dalam konfigurasi sumber AppSync data sebelum mengklik Buat.

Mengkonfigurasi Resolver

Sekarang kita memiliki skema GraphQL yang valid dan sumber data RDS, kita dapat melampirkan resolver ke bidang GraphQL pada skema kita. API kami akan menawarkan kemampuan berikut:

1. buat hewan peliharaan melalui bidang mutation.createPet
2. perbarui hewan peliharaan melalui bidang mutation.updatePet
3. hapus hewan peliharaan melalui bidang mutation.deletePet
4. dapatkan satu hewan peliharaan melalui bidang Query.getPet
5. daftar semua hewan peliharaan melalui bidang Query.listPets
6. daftar hewan peliharaan dalam kisaran harga melalui Query. listPetsByPriceRangelapangan

Mutasi.createPet

Dari editor skema di AWS AppSync konsol, di sisi kanan pilih Lampirkan Resolver untuk. `createPet(input: CreatePetInput!): Pet` Pilih sumber data RDS Anda. Di bagian template pemetaan permintaan, tambahkan template berikut:

```
#set($id=$utils.autoId())
{
"version": "2018-05-29",
    "statements": [
        "insert into Pets VALUES (:ID, :TYPE, :PRICE)",
        "select * from Pets WHERE id = :ID"
    ],
    "variableMap": {
        ":ID": "$ctx.args.input.id",
        ":TYPE": $util.toJson($ctx.args.input.type),
        ":PRICE": $util.toJson($ctx.args.input.price)
    }
}
```

Pernyataan SQL akan mengeksekusi secara berurutan, berdasarkan urutan dalam array pernyataan. Hasilnya akan kembali dalam urutan yang sama. Karena ini adalah mutasi, kami menjalankan pernyataan pilih setelah sisipan untuk mengambil nilai yang dikomit untuk mengisi template pemetaan respons GraphQL.

Di bagian template pemetaan respons, tambahkan templat berikut:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[1][0])
```

Karena pernyataan memiliki dua query SQL, kita perlu menentukan hasil kedua dalam matriks yang kembali dari database dengan:. `$utils.rds.toJsonString($ctx.result))[1][0])`

mutasi.updatePET

Dari editor skema di AWS AppSync konsol, di sisi kanan pilih Lampirkan Resolver untuk. `updatePet(input: UpdatePetInput!): Pet` Pilih sumber data RDS Anda. Di bagian template pemetaan permintaan, tambahkan template berikut:

```
{
"version": "2018-05-29",
```

```
    "statements": [
        $util.toJson("update Pets set type=:TYPE, price=:PRICE WHERE id=:ID"),
        $util.toJson("select * from Pets WHERE id = :ID")
    ],
    "variableMap": {
        ":ID": "$ctx.args.input.id",
        ":TYPE": $util.toJson($ctx.args.input.type),
        ":PRICE": $util.toJson($ctx.args.input.price)
    }
}
```

Di bagian template pemetaan respons, tambahkan templat berikut:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[1][0])
```

mutasi.deletePET

Dari editor skema di AWS AppSync konsol, di sisi kanan pilih Lampirkan Resolver untuk. `deletePet(input: DeletePetInput!): Pet` Pilih sumber data RDS Anda. Di bagian template pemetaan permintaan, tambahkan template berikut:

```
{
"version": "2018-05-29",
    "statements": [
        $util.toJson("select * from Pets WHERE id=:ID"),
        $util.toJson("delete from Pets WHERE id=:ID")
    ],
    "variableMap": {
        ":ID": "$ctx.args.input.id"
    }
}
```

Di bagian template pemetaan respons, tambahkan templat berikut:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0][0])
```

Query.getPet

Sekarang mutasi dibuat untuk skema Anda, kami akan menghubungkan tiga kueri untuk menampilkan cara mendapatkan item individual, daftar, dan menerapkan pemfilteran SQL. Dari editor

skema di AWS AppSync konsol, di sisi kanan pilih Lampirkan Resolver untuk. `getPet(id: ID!): Pet` Pilih sumber data RDS Anda. Di bagian template pemetaan permintaan, tambahkan template berikut:

```
{
"version": "2018-05-29",
        "statements": [
            $util.toJson("select * from Pets WHERE id=:ID")
    ],
    "variableMap": {
        ":ID": "$ctx.args.id"
    }
}
```

Di bagian template pemetaan respons, tambahkan templat berikut:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0][0])
```

Query.listpets

Dari editor skema di AWS AppSync konsol, di sisi kanan pilih Lampirkan Resolver untuk. `getPet(id: ID!): Pet` Pilih sumber data RDS Anda. Di bagian template pemetaan permintaan, tambahkan template berikut:

```
{
    "version": "2018-05-29",
    "statements": [
        "select * from Pets"
    ]
}
```

Di bagian template pemetaan respons, tambahkan templat berikut:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0])
```

Pertanyaan. listPetsByPriceRange

Dari editor skema di AWS AppSync konsol, di sisi kanan pilih Lampirkan Resolver untuk. `getPet(id: ID!): Pet` Pilih sumber data RDS Anda. Di bagian template pemetaan permintaan, tambahkan template berikut:

```
{
    "version": "2018-05-29",
    "statements": [
            "select * from Pets where price > :MIN and price < :MAX"
    ],

    "variableMap": {
        ":MAX": $util.toJson($ctx.args.max),
        ":MIN": $util.toJson($ctx.args.min)
    }
}
```

Di bagian template pemetaan respons, tambahkan templat berikut:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0])
```

Jalankan mutasi

Sekarang setelah Anda mengonfigurasi semua resolver Anda dengan pernyataan SQL dan menghubungkan GraphQL API Anda ke API Data Aurora Tanpa Server, Anda dapat mulai melakukan mutasi dan kueri. Di AWS AppSync konsol, pilih tab Kueri dan masukkan yang berikut ini untuk membuat Pet:

```
mutation add {
    createPet(input : { type:fish, price:10.0 }){
        id
        type
        price
    }
}
```

Respons harus berisi id, tipe, dan harga seperti:

```
{
  "data": {
    "createPet": {
      "id": "c6fedbbe-57ad-4da3-860a-ffe8d039882a",
      "type": "fish",
      "price": "10.0"
    }
  }
```

```
}
```

Anda dapat memodifikasi item ini dengan menjalankan mutasi updatePet:

```
mutation update {
    updatePet(input : {
        id: ID_PLACEHOLDER,
        type:bird,
        price:50.0
    }){
        id
        type
        price
    }
}
```

Perhatikan bahwa kami menggunakan id yang dikembalikan dari operasi createPet sebelumnya. Ini akan menjadi nilai unik untuk catatan Anda saat resolver dimanfaatkan. `$util.autoId()` Anda dapat menghapus catatan dengan cara yang sama:

```
mutation delete {
    deletePet(input : {id:ID_PLACEHOLDER}){
        id
        type
        price
    }
}
```

Buat beberapa catatan dengan mutasi pertama dengan nilai harga yang berbeda dan kemudian jalankan beberapa kueri.

Jalankan Kueri

Masih di tab Kueri konsol, gunakan pernyataan berikut untuk mencantumkan semua catatan yang telah Anda buat:

```
query allpets {
    listPets {
        id
        type
        price
```

```
    }
}
```

Ini bagus tapi mari kita manfaatkan predikat SQL WHERE yang ada *`where price > :MIN and price < :MAX`* di template pemetaan kami untuk Query. listPetsByPriceRangedengan query GraphQL berikut:

```
query petsByPriceRange {
    listPetsByPriceRange(min:1, max:11) {
        id
        type
        price
    }
}
```

Anda seharusnya hanya melihat catatan dengan harga lebih dari $1 atau kurang dari $10. Akhirnya, Anda dapat melakukan kueri untuk mengambil catatan individual sebagai berikut:

```
query onePet {
    getPet(id:ID_PLACEHOLDER){
        id
        type
        price
    }
}
```

Sanitasi Masukan

Kami menyarankan agar pengembang menggunakan `variableMap` untuk perlindungan terhadap serangan injeksi SQL. Jika peta variabel tidak digunakan, pengembang bertanggung jawab untuk membersihkan argumen operasi GraphQL mereka. Salah satu cara untuk melakukannya adalah dengan memberikan input langkah-langkah validasi spesifik dalam template pemetaan permintaan sebelum eksekusi pernyataan SQL terhadap API Data Anda. Mari kita lihat bagaimana kita dapat memodifikasi template pemetaan permintaan dari `listPetsByPriceRange` contoh. Alih-alih hanya mengandalkan input pengguna, Anda dapat melakukan hal berikut:

```
#set($validMaxPrice = $util.matches("\d{1,3}[,\\.]?(\\d{1,2})?",$ctx.args.maxPrice))

#set($validMinPrice = $util.matches("\d{1,3}[,\\.]?(\\d{1,2})?",$ctx.args.minPrice))
```

```
#if (!$validMaxPrice || !$validMinPrice)
    $util.error("Provided price input is not valid.")
#end
{
    "version": "2018-05-29",
    "statements": [
            "select * from Pets where price > :MIN and price < :MAX"
    ],

    "variableMap": {
        ":MAX": $util.toJson($ctx.args.maxPrice),
        ":MIN": $util.toJson($ctx.args.minPrice)
    }
}
```

Cara lain untuk melindungi terhadap masukan jahat saat mengeksekusi resolver terhadap API Data Anda adalah dengan menggunakan pernyataan yang disiapkan bersama dengan prosedur tersimpan dan input berparameter. Misalnya, dalam resolver untuk `listPets` menentukan prosedur berikut yang mengeksekusi pilih sebagai pernyataan yang disiapkan:

```
CREATE PROCEDURE listPets (IN type_param VARCHAR(200))
  BEGIN
     PREPARE stmt FROM 'SELECT * FROM Pets where type=?';
     SET @type = type_param;
     EXECUTE stmt USING @type;
     DEALLOCATE PREPARE stmt;
  END
```

Ini dapat dibuat di Aurora Serverless Instance Anda menggunakan perintah execute sql berikut:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:xxxxxxxxxxxx:cluster:http-endpoint-test" \
--schema "mysql"  --secret-arn "arn:aws:secretsmanager:us-
east-1:xxxxxxxxxxxx:secret:httpendpoint-xxxxxx"  \
--region us-east-1  --database "DB_NAME" \
--sql "CREATE PROCEDURE listPets (IN type_param VARCHAR(200)) BEGIN PREPARE stmt FROM
 'SELECT * FROM Pets where type=?'; SET @type = type_param; EXECUTE stmt USING @type;
 DEALLOCATE PREPARE stmt; END"
```

Kode resolver yang dihasilkan untuk ListPets disederhanakan karena kita sekarang cukup memanggil prosedur tersimpan. Minimal, setiap input string harus memiliki tanda kutip tunggal yang lolos.

```
#set ($validType = $util.isString($ctx.args.type) && !
$util.isNullOrBlank($ctx.args.type))
#if (!$validType)
    $util.error("Input for 'type' is not valid.", "ValidationError")
#end

{
    "version": "2018-05-29",
    "statements": [
        "CALL listPets(:type)"
    ]
    "variableMap": {
        ":type": $util.toJson($ctx.args.type.replace("'", "''"))
    }
}
```

String melarikan diri

Tanda kutip tunggal mewakili awal dan akhir literal string dalam pernyataan SQL, misalnya. `'some string value'`. Untuk memungkinkan nilai string dengan satu atau lebih karakter kutipan tunggal (`'`) untuk digunakan dalam string, masing-masing harus diganti dengan dua tanda kutip tunggal (`''`). Misalnya, jika string inputnya`Nadia's dog`, Anda akan menghindarinya untuk pernyataan SQL seperti

```
update Pets set type='Nadia''s dog' WHERE id='1'
```

Tutorial: Penyelesai Pipa

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync menyediakan cara sederhana untuk menghubungkan bidang GraphQL ke sumber data tunggal melalui resolver unit. Namun, menjalankan satu operasi mungkin tidak cukup. Pipeline resolvers menawarkan kemampuan untuk menjalankan operasi secara serial terhadap sumber data. Buat fungsi di API Anda dan lampirkan ke resolver pipeline. Setiap hasil eksekusi fungsi disalurkan ke yang berikutnya sampai tidak ada fungsi yang tersisa untuk dieksekusi. Dengan resolver pipeline,

Anda sekarang dapat membangun alur kerja yang lebih kompleks secara langsung. AWS AppSync Dalam tutorial ini, Anda membangun aplikasi tampilan gambar sederhana, di mana pengguna dapat memposting dan melihat gambar yang diposting oleh teman-teman mereka.

Pengaturan Satu-Klik

Jika Anda ingin secara otomatis mengatur titik akhir AWS AppSync GraphQL dengan semua resolver yang dikonfigurasi dan sumber daya yang AWS diperlukan, Anda dapat menggunakan templat berikut: AWS CloudFormation

Tumpukan ini membuat sumber daya berikut di akun Anda:

- Peran IAM AWS AppSync untuk mengakses sumber daya di akun Anda
- 2 tabel DynamoDB
- 1 kumpulan pengguna Amazon Cognito
- 2 grup kumpulan pengguna Amazon Cognito
- 3 pengguna kumpulan pengguna Amazon Cognito
- 1 AWS AppSync API

Di akhir proses pembuatan AWS CloudFormation tumpukan, Anda menerima satu email untuk masing-masing dari tiga pengguna Amazon Cognito yang dibuat. Setiap email berisi kata sandi sementara yang Anda gunakan untuk masuk sebagai pengguna Amazon Cognito ke konsol. AWS AppSync Simpan kata sandi untuk sisa tutorial.

Pengaturan Manual

Jika Anda lebih suka melalui step-by-step proses secara manual melalui AWS AppSync konsol, ikuti proses pengaturan di bawah ini.

Menyiapkan Non AWS AppSync Sumber Daya Anda

API berkomunikasi dengan dua tabel DynamoDB: tabel gambar yang menyimpan gambar dan tabel teman yang menyimpan hubungan antar pengguna. API dikonfigurasi untuk menggunakan kumpulan pengguna Amazon Cognito sebagai jenis otentikasi. AWS CloudFormationTumpukan berikut mengatur sumber daya ini di akun.

Di akhir proses pembuatan AWS CloudFormation tumpukan, Anda menerima satu email untuk masing-masing dari tiga pengguna Amazon Cognito yang dibuat. Setiap email berisi kata sandi sementara yang Anda gunakan untuk masuk sebagai pengguna Amazon Cognito ke konsol. AWS AppSync Simpan kata sandi untuk sisa tutorial.

Membuat GraphQL API Anda

Untuk membuat GraphQL API di: AWS AppSync

1. Buka AWS AppSync konsol dan pilih Build From Scratch dan pilih Start.
2. Tetapkan nama API ke`AppSyncTutorial-PicturesViewer`.
3. Pilih Create (Buat).

AWS AppSync Konsol membuat API GraphQL baru untuk Anda menggunakan mode autentikasi kunci API. Anda dapat menggunakan konsol untuk mengatur sisa GraphQL API dan menjalankan kueri terhadapnya selama sisa tutorial ini.

Mengkonfigurasi API GraphQL

Anda perlu mengonfigurasi AWS AppSync API dengan kumpulan pengguna Amazon Cognito yang baru saja Anda buat.

1. Pilih tab Pengaturan.
2. Di bawah bagian Jenis Otorisasi, pilih Kumpulan Pengguna Amazon Cognito.
3. Di bawah Konfigurasi Pool Pengguna, pilih US-WEST-2 untuk Wilayah. AWS
4. Pilih AppSyncTutorial- kumpulan UserPool pengguna.
5. Pilih DENY sebagai Default Action.
6. Biarkan bidang regex AppId klien kosong.
7. Pilih Save (Simpan).

API sekarang disiapkan untuk menggunakan kumpulan pengguna Amazon Cognito sebagai jenis otorisasi.

Mengkonfigurasi Sumber Data untuk Tabel DynamoDB

Setelah tabel DynamoDB dibuat, navigasikan ke AWS AppSync GraphQL API Anda di konsol dan pilih tab Sumber Data. Sekarang, Anda akan membuat sumber data AWS AppSync untuk setiap tabel DynamoDB yang baru saja Anda buat.

1. Pilih tab Sumber data.
2. Pilih Baru untuk membuat sumber data baru.
3. Untuk nama sumber data, masukkan`PicturesDynamoDBTable`.
4. Untuk tipe sumber data, pilih tabel Amazon DynamoDB.
5. Untuk wilayah, pilih US-WEST-2.
6. Dari daftar tabel, pilih tabel AppSyncTutorial-Pictures DynamoDB.
7. Di bagian Buat atau gunakan peran yang ada, pilih Peran yang ada.
8. Pilih peran yang baru saja dibuat dari CloudFormation template. Jika Anda tidak mengubah ResourceNamePrefix, nama peran harus AppSyncTutorial-DynamodBrole.
9. Pilih Create (Buat).

Ulangi proses yang sama untuk tabel teman, nama tabel DynamoDB AppSyncTutorialharus - Friends jika Anda tidak mengubah parameter ResourceNamePrefixpada saat membuat tumpukan. CloudFormation

Membuat Skema GraphQL

Sekarang sumber data terhubung ke tabel DynamoDB Anda, mari buat skema GraphQL. Dari editor skema di AWS AppSync konsol, pastikan skema Anda cocok dengan skema berikut:

```
schema {
    query: Query
    mutation: Mutation
}

type Mutation {
    createPicture(input: CreatePictureInput!): Picture!
    @aws_auth(cognito_groups: ["Admins"])
    createFriendship(id: ID!, target: ID!): Boolean
    @aws_auth(cognito_groups: ["Admins"])
}
```

```
type Query {
    getPicturesByOwner(id: ID!): [Picture]
    @aws_auth(cognito_groups: ["Admins", "Viewers"])
}

type Picture {
    id: ID!
    owner: ID!
    src: String
}

input CreatePictureInput {
    owner: ID!
    src: String!
}
```

Pilih Simpan Skema untuk menyimpan skema Anda.

Beberapa bidang skema telah dianotasi dengan direktif @aws_auth. Karena konfigurasi tindakan default API disetel ke DENY, API menolak semua pengguna yang bukan anggota grup yang disebutkan di dalam direktif @aws_auth. Untuk informasi selengkapnya tentang cara mengamankan API, Anda dapat membaca halaman Keamanan. Dalam hal ini, hanya pengguna admin yang memiliki akses ke bidang mutation.createPicture dan mutation.createFriendship, sementara pengguna yang merupakan anggota grup Admin atau Pemirsa dapat mengakses Kueri. getPicturesByBidang pemilik. Semua pengguna lain tidak memiliki akses.

Mengkonfigurasi Resolver

Sekarang Anda memiliki skema GraphQL yang valid dan dua sumber data, Anda dapat melampirkan resolver ke bidang GraphQL pada skema. API menawarkan kemampuan berikut:

- Buat gambar melalui bidang mutation.createPicture
- Buat persahabatan melalui bidang mutation.createFriendship
- Mengambil gambar melalui bidang Query.getPicture

mutasi.createPicture

Dari editor skema di AWS AppSync konsol, di sisi kanan pilih Lampirkan Resolver untuk. `createPicture(input: CreatePictureInput!): Picture!` Pilih sumber data

PicturesDynamoDynamoDB DbTable. Di bagian template pemetaan permintaan, tambahkan template berikut:

```
#set($id = $util.autoId())

{
    "version" : "2018-05-29",

    "operation" : "PutItem",

    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($id),
        "owner": $util.dynamodb.toDynamoDBJson($ctx.args.input.owner)
    },

    "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args.input)
}
```

Di bagian template pemetaan respons, tambahkan templat berikut:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end
$util.toJson($ctx.result)
```

Fungsionalitas membuat gambar selesai. Anda menyimpan gambar di tabel Gambar, menggunakan UUID yang dihasilkan secara acak sebagai id gambar, dan menggunakan nama pengguna Cognito sebagai pemilik gambar.

mutation.createFriendship

Dari editor skema di AWS AppSync konsol, di sisi kanan pilih Lampirkan Resolver untuk. `createFriendship(id: ID!, target: ID!): Boolean` Pilih sumber data FriendsDynamoDynamoDB DbTable. Di bagian template pemetaan permintaan, tambahkan template berikut:

```
#set($userToFriendFriendship = { "userId" : "$ctx.args.id", "friendId":
 "$ctx.args.target" })
#set($friendToUserFriendship = { "userId" : "$ctx.args.target", "friendId":
 "$ctx.args.id" })
#set($friendsItems = [$util.dynamodb.toMapValues($userToFriendFriendship),
 $util.dynamodb.toMapValues($friendToUserFriendship)])
```

```
{
    "version" : "2018-05-29",
    "operation" : "BatchPutItem",
    "tables" : {
        ## Replace 'AppSyncTutorial-' default below with the ResourceNamePrefix you
 provided in the CloudFormation template
        "AppSyncTutorial-Friends": $util.toJson($friendsItems)
    }
}
```

Penting: Dalam template BatchPutItempermintaan, nama yang tepat dari tabel DynamoDB harus ada. Nama tabel default adalah AppSyncTutorial-Friends. Jika Anda menggunakan nama tabel yang salah, Anda mendapatkan kesalahan saat AppSync mencoba mengambil peran yang disediakan.

Demi kesederhanaan dalam tutorial ini, lanjutkan seolah-olah permintaan pertemanan telah disetujui dan simpan entri hubungan langsung ke AppSyncTutorialFriendstabel.

Secara efektif, Anda menyimpan dua item untuk setiap pertemanan karena hubungan itu dua arah. Untuk detail selengkapnya tentang praktik terbaik Amazon DynamoDB untuk many-to-many mewakili hubungan, lihat Praktik Terbaik DynamoDB.

Di bagian template pemetaan respons, tambahkan templat berikut:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end
true
```

Catatan: Pastikan template permintaan Anda berisi nama tabel yang tepat. Nama default adalah AppSyncTutorial-Friends, tetapi nama tabel Anda mungkin berbeda jika Anda mengubah CloudFormation ResourceNamePrefixparameter.

Pertanyaan. getPicturesByPemilik

Sekarang setelah Anda memiliki pertemanan dan gambar, Anda perlu memberikan kemampuan bagi pengguna untuk melihat foto teman-teman mereka. Untuk memenuhi persyaratan ini, Anda harus terlebih dahulu memeriksa apakah pemohon berteman dengan pemilik, dan akhirnya meminta gambar.

Karena fungsi ini memerlukan dua operasi sumber data, Anda akan membuat dua fungsi. Fungsi pertama, IsFriend, memeriksa apakah pemohon dan pemiliknya adalah teman. Fungsi kedua,

getPicturesByOwner, mengambil gambar yang diminta diberikan ID pemilik. Mari kita lihat alur eksekusi di bawah ini untuk resolver yang diusulkan pada Query. getPicturesByBidang pemilik:

1. Sebelum memetakan template: Siapkan konteks dan argumen masukan bidang.
2. Fungsi isFriend: Memeriksa apakah pemohon adalah pemilik gambar. Jika tidak, ia memeriksa apakah pengguna pemohon dan pemilik adalah teman dengan melakukan operasi GetItem DynamoDB di meja teman.
3. getPicturesByFungsi pemilik: Mengambil gambar dari tabel Gambar menggunakan operasi Query DynamoDB pada indeks pemilik Indeks Sekunder Global.
4. Setelah memetakan template: Memetakan hasil gambar sehingga DynamoDB mengatribusikan peta dengan benar ke bidang tipe GraphQL yang diharapkan.

Pertama mari kita buat fungsinya.

Fungsi isFriend

1. Pilih tab Fungsi.
2. Pilih Create Function untuk membuat fungsi.
3. Untuk nama sumber data, masukkan`FriendsDynamoDBTable`.
4. Untuk nama fungsi, masukkan isFriend.
5. Di dalam area teks template pemetaan permintaan, tempel template berikut:

```
#set($ownerId = $ctx.prev.result.owner)
#set($callerId = $ctx.prev.result.callerId)

## if the owner is the caller, no need to make the check
#if($ownerId == $callerId)
    #return($ctx.prev.result)
#end

{
    "version" : "2018-05-29",

    "operation" : "GetItem",

    "key" : {
        "userId" : $util.dynamodb.toDynamoDBJson($callerId),
        "friendId" : $util.dynamodb.toDynamoDBJson($ownerId)
    }
```

```
}
```

6. Di dalam area teks template pemetaan respons, tempel template berikut:

```
#if($ctx.error)
    $util.error("Unable to retrieve friend mapping message: ${ctx.error.message}",
 $ctx.error.type)
#end

## if the users aren't friends
#if(!$ctx.result)
    $util.unauthorized()
#end

$util.toJson($ctx.prev.result)
```

7. Pilih Buat Fungsi.

Hasil: Anda telah membuat fungsi isFriend.

getPicturesByFungsi pemilik

1. Pilih tab Fungsi.
2. Pilih Create Function untuk membuat fungsi.
3. Untuk nama sumber data, masukkan`PicturesDynamoDBTable`.
4. Untuk nama fungsi, masukkan`getPicturesByOwner`.
5. Di dalam area teks template pemetaan permintaan, tempel template berikut:

```
{
    "version" : "2018-05-29",

    "operation" : "Query",

    "query" : {
        "expression": "#owner = :owner",
        "expressionNames": {
            "#owner" : "owner"
        },
        "expressionValues" : {
            ":owner" : $util.dynamodb.toDynamoDBJson($ctx.prev.result.owner)
        }
```

```
    },

    "index": "owner-index"
}
```

6. Di dalam area teks template pemetaan respons, tempel template berikut:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end

$util.toJson($ctx.result)
```

7. Pilih Buat Fungsi.

Hasil: Anda telah membuat fungsi getPicturesByPemilik. Sekarang fungsi telah dibuat, lampirkan resolver pipeline ke Query. getPicturesByBidang pemilik.

Dari editor skema di AWS AppSync konsol, di sisi kanan pilih Lampirkan Resolver untuk. `Query.getPicturesByOwner(id: ID!): [Picture]` Pada halaman berikut, pilih tautan Convert to pipeline resolver yang muncul di bawah daftar drop-down sumber data. Gunakan yang berikut ini untuk template sebelum pemetaan:

```
#set($result = { "owner": $ctx.args.id, "callerId": $ctx.identity.username })
$util.toJson($result)
```

Di bagian template setelah pemetaan, gunakan yang berikut ini:

```
#foreach($picture in $ctx.result.items)
    ## prepend "src://" to picture.src property
    #set($picture['src'] = "src://${picture['src']}")
#end
$util.toJson($ctx.result.items)
```

Pilih Buat Resolver. Anda telah berhasil memasang resolver pipa pertama Anda. Pada halaman yang sama, tambahkan dua fungsi yang Anda buat sebelumnya. Di bagian fungsi, pilih Tambahkan Fungsi dan kemudian pilih atau ketik nama fungsi pertama, IsFriend. Tambahkan fungsi kedua dengan mengikuti proses yang sama untuk fungsi getPicturesByPemilik. Pastikan fungsi isFriend muncul pertama kali dalam daftar diikuti oleh fungsi getPicturesByOwner. Anda dapat menggunakan panah atas dan bawah untuk mengatur ulang urutan pelaksanaan fungsi dalam pipa.

Sekarang setelah pipeline resolver dibuat dan Anda telah melampirkan fungsinya, mari kita uji GraphQL API yang baru dibuat.

Menguji GraphQL API Anda

Pertama, Anda perlu mengisi gambar dan pertemanan dengan mengeksekusi beberapa mutasi menggunakan pengguna admin yang Anda buat. Di sisi kiri AWS AppSync konsol, pilih tab Kueri.

Mutasi CreatePicture

1. Di AWS AppSync konsol, pilih tab Kueri.
2. Pilih Login Dengan User Pools.
3. Pada modal, masukkan Cognito Sample Client ID yang dibuat oleh CloudFormation stack misalnya, 37solo6mmhh7k4v63cqdfgdg5d).
4. Masukkan nama pengguna yang Anda berikan sebagai parameter ke CloudFormation tumpukan. Defaultnya adalah nadia.
5. Gunakan kata sandi sementara yang dikirim ke email yang Anda berikan sebagai parameter ke CloudFormation tumpukan (misalnya, UserPoolUserEmail).
6. Pilih Login. Anda sekarang akan melihat tombol berganti nama menjadi Logout nadia, atau nama pengguna apa pun yang Anda pilih saat membuat CloudFormation tumpukan (yaitu,). UserPoolUsername

Mari kita kirim beberapa mutasi CreatePicture untuk mengisi tabel gambar. Jalankan query GraphQL berikut di dalam konsol:

```
mutation {
  createPicture(input:{
    owner: "nadia"
    src: "nadia.jpg"
  }) {
    id
    owner
    src
  }
}
```

Responsnya akan terlihat seperti di bawah ini:

```
{
  "data": {
    "createPicture": {
      "id": "c6fedbbe-57ad-4da3-860a-ffe8d039882a",
      "owner": "nadia",
      "src": "nadia.jpg"
    }
  }
}
```

Mari tambahkan beberapa gambar lagi:

```
mutation {
  createPicture(input:{
    owner: "shaggy"
    src: "shaggy.jpg"
  }) {
    id
    owner
    src
  }
}
```

```
mutation {
  createPicture(input:{
    owner: "rex"
    src: "rex.jpg"
  }) {
    id
    owner
    src
  }
}
```

Anda telah menambahkan tiga gambar menggunakan nadia sebagai pengguna admin.

Mutasi CreateFriendship

Mari tambahkan entri persahabatan. Jalankan mutasi berikut di konsol.

Catatan: Anda masih harus masuk sebagai pengguna admin (pengguna admin default adalah nadia).

```
mutation {
  createFriendship(id: "nadia", target: "shaggy")
}
```

Responsnya akan terlihat seperti:

```
{
  "data": {
    "createFriendship": true
  }
}
```

Nadia dan shaggy adalah teman. Rex tidak berteman dengan siapa pun.

getPicturesByPertanyaan Pemilik

Untuk langkah ini, masuk sebagai pengguna nadia menggunakan Cognito User Pools, menggunakan kredensil yang diatur di awal tutorial ini. Sebagai nadia, ambil foto-foto yang dimiliki oleh shaggy.

```
query {
    getPicturesByOwner(id: "shaggy") {
        id
        owner
        src
    }
}
```

Karena nadia dan shaggy adalah teman, kueri harus mengembalikan gambar yang sesuai.

```
{
  "data": {
    "getPicturesByOwner": [
      {
        "id": "05a16fba-cc29-41ee-a8d5-4e791f4f1079",
        "owner": "shaggy",
        "src": "src://shaggy.jpg"
      }
    ]
  }
}
```

Demikian pula, jika nadia mencoba mengambil fotonya sendiri, itu juga berhasil. Pipeline resolver telah dioptimalkan untuk menghindari menjalankan GetItem operasi IsFriend dalam kasus itu. Coba kueri berikut:

```
query {
    getPicturesByOwner(id: "nadia") {
        id
        owner
        src
    }
}
```

Jika Anda mengaktifkan logging pada API Anda (di panel Pengaturan), mengatur tingkat debug ke ALL, dan menjalankan kueri yang sama lagi, itu akan mengembalikan log untuk eksekusi bidang. Dengan melihat log, Anda dapat menentukan apakah fungsi isFriend kembali lebih awal pada tahap Template Pemetaan Permintaan:

```
{
  "errors": [],
  "mappingTemplateType": "Request Mapping",
  "path": "[getPicturesByOwner]",
  "resolverArn": "arn:aws:appsync:us-west-2:XXXX:apis/XXXX/types/Query/fields/
getPicturesByOwner",
  "functionArn": "arn:aws:appsync:us-west-2:XXXX:apis/XXXX/functions/
o2f42p2jrfdl3dw7s6xub2csdfs",
  "functionName": "isFriend",
  "earlyReturnedValue": {
    "owner": "nadia",
    "callerId": "nadia"
  },
  "context": {
    "arguments": {
      "id": "nadia"
    },
    "prev": {
      "result": {
        "owner": "nadia",
        "callerId": "nadia"
      }
    },
    "stash": {},
    "outErrors": []
```

```
  },
  "fieldInError": false
}
```

earlyReturnedValueKunci mewakili data yang dikembalikan oleh direktif #return.

Akhirnya, meskipun rex adalah anggota Grup UserPool Cognito Pemirsa, dan karena rex tidak berteman dengan siapa pun, dia tidak akan dapat mengakses salah satu gambar yang dimiliki oleh shaggy atau nadia. Jika Anda masuk sebagai rex di konsol dan menjalankan kueri berikut:

```
query {
    getPicturesByOwner(id: "nadia") {
        id
        owner
        src
    }
}
```

Anda mendapatkan kesalahan tidak sah berikut:

```
{
  "data": {
    "getPicturesByOwner": null
  },
  "errors": [
    {
      "path": [
        "getPicturesByOwner"
      ],
      "data": null,
      "errorType": "Unauthorized",
      "errorInfo": null,
      "locations": [
        {
          "line": 2,
          "column": 9,
          "sourceName": null
        }
      ],
      "message": "Not Authorized to access getPicturesByOwner on type Query"
    }
  ]
```

```
}
```

Anda telah berhasil menerapkan otorisasi kompleks menggunakan resolver pipa.

Tutorial: Sinkronisasi Delta

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Aplikasi klien dalam AWS AppSync menyimpan data dengan menyimpan respons GraphQL secara lokal ke disk dalam aplikasi seluler/web. Sumber dan `Sync` operasi data berversi memberi pelanggan kemampuan untuk melakukan proses sinkronisasi menggunakan resolver tunggal. Ini memungkinkan klien untuk menghidrasi cache lokal mereka dengan hasil dari satu kueri dasar yang mungkin memiliki banyak catatan, dan kemudian hanya menerima data yang diubah sejak kueri terakhir mereka (pembaruan delta). Dengan mengizinkan klien untuk melakukan hidrasi dasar cache dengan permintaan awal dan pembaruan tambahan di tempat lain, Anda dapat memindahkan perhitungan dari aplikasi klien Anda ke backend. Ini jauh lebih efisien untuk aplikasi klien yang sering beralih antara status online dan offline.

Untuk mengimplementasikan Delta Sync, `Sync` kueri menggunakan `Sync` operasi pada sumber data berversi. Ketika AWS AppSync mutasi mengubah item dalam sumber data berversi, catatan perubahan itu akan disimpan di tabel Delta juga. Anda dapat memilih untuk menggunakan tabel Delta yang berbeda (misalnya satu per jenis, satu per area domain) untuk sumber data berversi lainnya atau tabel Delta tunggal untuk API Anda. AWS AppSync merekomendasikan untuk tidak menggunakan tabel Delta tunggal untuk beberapa API guna menghindari tabrakan kunci utama.

Selain itu, klien Delta Sync juga dapat menerima langganan sebagai argumen, dan kemudian klien mengoordinasikan langganan menghubungkan kembali dan menulis antara transisi offline ke online. Delta Sync melakukan ini dengan melanjutkan langganan secara otomatis (termasuk backoff eksponensial dan coba lagi dengan jitter melalui skenario kesalahan jaringan yang berbeda), dan menyimpan peristiwa dalam antrian. Kueri delta atau basis yang sesuai kemudian dijalankan sebelum menggabungkan peristiwa apa pun dari antrian, dan akhirnya memproses langganan seperti biasa.

Dokumentasi untuk opsi konfigurasi klien, termasuk Amplify DataStore, tersedia di situs web Amplify Framework. Dokumentasi ini menguraikan cara mengatur sumber `Sync` dan operasi data DynamoDB berversi untuk bekerja dengan klien Delta Sync untuk akses data yang optimal.

Pengaturan Satu-Klik

Untuk secara otomatis mengatur titik akhir AWS AppSync GraphQL dengan semua resolver yang dikonfigurasi dan sumber daya yang diperlukan, gunakan templat iniAWS: AWS CloudFormation

Tumpukan ini membuat sumber daya berikut di akun Anda:

- 2 tabel DynamoDB (Basis dan Delta)
- 1 AWS AppSync API dengan kunci API
- 1 Peran IAM dengan kebijakan untuk tabel DynamoDB

Dua tabel digunakan untuk mempartisi kueri sinkronisasi Anda ke tabel kedua yang bertindak sebagai jurnal peristiwa yang tidak terjawab saat klien sedang offline. Untuk menjaga agar kueri tetap efisien pada tabel delta, Amazon DynamoDB TTL digunakan untuk secara otomatis mengatur acara seperlunya. Waktu TTL dapat dikonfigurasi untuk kebutuhan Anda pada sumber data (Anda mungkin menginginkan ini sebagai 1 jam, 1 hari, dll.).

Skema

Untuk mendemonstrasikan Delta Sync, aplikasi sampel membuat skema Posts yang didukung oleh tabel Base dan Delta di DynamoDB. AWS AppSync secara otomatis menulis mutasi ke kedua tabel. Kueri sinkronisasi menarik catatan dari tabel Base atau Delta sebagaimana mestinya, dan satu langganan didefinisikan untuk menunjukkan bagaimana klien dapat memanfaatkan ini dalam logika rekoneksi mereka.

```
input CreatePostInput {
    author: String!
    title: String!
    content: String!
    url: String
    ups: Int
    downs: Int
    _version: Int
```

```
}

interface Connection {
  nextToken: String
  startedAt: AWSTimestamp!
}

type Mutation {
    createPost(input: CreatePostInput!): Post
    updatePost(input: UpdatePostInput!): Post
    deletePost(input: DeletePostInput!): Post
}

type Post {
    id: ID!
    author: String!
    title: String!
    content: String!
    url: AWSURL
    ups: Int
    downs: Int
    _version: Int
    _deleted: Boolean
    _lastChangedAt: AWSTimestamp!
}

type PostConnection implements Connection {
    items: [Post!]!
    nextToken: String
    startedAt: AWSTimestamp!
}

type Query {
    getPost(id: ID!): Post
    syncPosts(limit: Int, nextToken: String, lastSync: AWSTimestamp): PostConnection!
}

type Subscription {
    onCreatePost: Post
        @aws_subscribe(mutations: ["createPost"])
    onUpdatePost: Post
        @aws_subscribe(mutations: ["updatePost"])
    onDeletePost: Post
        @aws_subscribe(mutations: ["deletePost"])
```

```
}

input DeletePostInput {
    id: ID!
    _version: Int!
}

input UpdatePostInput {
    id: ID!
    author: String
    title: String
    content: String
    url: String
    ups: Int
    downs: Int
    _version: Int!
}

schema {
    query: Query
    mutation: Mutation
    subscription: Subscription
}
```

Skema GraphQL adalah standar, tetapi beberapa hal layak dipanggil sebelum bergerak maju. Pertama, semua mutasi secara otomatis pertama menulis ke tabel Base dan kemudian ke tabel Delta. Tabel dasar adalah sumber utama kebenaran untuk negara sedangkan tabel Delta adalah jurnal Anda. Jika Anda tidak meneruskan*`lastSync: AWSTimestamp`*, *`syncPosts`* kueri berjalan terhadap tabel Base dan menghidrasi cache serta berjalan pada waktu berkala sebagai proses penangkapan global untuk kasus tepi saat klien offline lebih lama dari waktu TTL yang dikonfigurasi di tabel Delta. Jika Anda meneruskan`lastSync: AWSTimestamp`, `syncPosts` kueri berjalan terhadap tabel Delta Anda dan digunakan oleh klien untuk mengambil peristiwa yang diubah sejak terakhir offline. Amplify klien secara otomatis meneruskan `lastSync: AWSTimestamp` nilai, dan bertahan ke disk dengan tepat.

Bidang _deleted pada Post digunakan untuk operasi DELETE. Saat klien offline dan catatan dihapus dari tabel Dasar, atribut ini memberi tahu klien yang melakukan sinkronisasi untuk mengusir item dari cache lokal mereka. Dalam kasus di mana klien offline untuk jangka waktu yang lebih lama dan item telah dihapus sebelum klien dapat mengambil nilai ini dengan kueri Delta Sync, peristiwa pengejaran global dalam kueri dasar (dapat dikonfigurasi di klien) berjalan dan menghapus item dari cache.

Bidang ini ditandai opsional karena hanya mengembalikan nilai saat menjalankan kueri sinkronisasi yang telah menghapus item yang ada.

Mutasi

Untuk semua mutasi, AWS AppSync lakukan operasi Buat/Perbarui/Hapus standar di tabel Dasar dan juga mencatat perubahan dalam tabel Delta secara otomatis. Anda dapat mengurangi atau memperpanjang waktu untuk menyimpan catatan dengan memodifikasi `DeltaSyncTableTTL` nilai pada sumber data. Untuk organisasi dengan kecepatan data yang tinggi, mungkin masuk akal untuk membuat ini singkat. Atau, jika klien Anda offline untuk jangka waktu yang lebih lama, mungkin bijaksana untuk menyimpannya lebih lama.

Pertanyaan Sinkronisasi

Kueri dasar adalah operasi DynamoDB Sync tanpa `lastSync` nilai yang ditentukan. Bagi banyak organisasi, ini berfungsi karena kueri dasar hanya berjalan pada saat startup dan secara berkala setelahnya.

Query delta adalah operasi DynamoDB Sync dengan nilai yang ditentukan. `lastSync` Kueri delta dijalankan setiap kali klien kembali online dari status offline (selama waktu periodik kueri dasar belum dipicu untuk dijalankan). Klien secara otomatis melacak kapan terakhir kali mereka berhasil menjalankan kueri untuk menyinkronkan data.

Ketika kueri delta dijalankan, resolver kueri menggunakan `ds_pk` dan `ds_sk` untuk melakukan kueri hanya untuk catatan yang telah berubah sejak terakhir kali klien melakukan sinkronisasi. Klien menyimpan respons GraphQL yang sesuai.

Untuk informasi selengkapnya tentang menjalankan Kueri Sinkronisasi, lihat dokumentasi Operasi Sinkronisasi.

Contoh

Mari kita mulai dulu dengan memanggil `createPost` mutasi untuk membuat item:

```
mutation create {
  createPost(input: {author: "Nadia", title: "My First Post", content: "Hello World"})
 {
    id
    author
    title
    content
```

```
    _version
    _lastChangedAt
    _deleted
  }
}
```

Nilai pengembalian mutasi ini akan terlihat sebagai berikut:

```
{
  "data": {
    "createPost": {
      "id": "81d36bbb-1579-4efe-92b8-2e3f679f628b",
      "author": "Nadia",
      "title": "My First Post",
      "content": "Hello World",
      "_version": 1,
      "_lastChangedAt": 1574469356331,
      "_deleted": null
    }
  }
}
```

Jika Anda memeriksa isi tabel Base, Anda akan melihat catatan yang terlihat seperti:

```
{
  "_lastChangedAt": {
    "N": "1574469356331"
  },
  "_version": {
    "N": "1"
  },
  "author": {
    "S": "Nadia"
  },
  "content": {
    "S": "Hello World"
  },
  "id": {
    "S": "81d36bbb-1579-4efe-92b8-2e3f679f628b"
  },
  "title": {
    "S": "My First Post"
  }
```

```
}
```

Jika Anda memeriksa isi tabel Delta, Anda akan melihat catatan yang terlihat seperti:

```
{
  "_lastChangedAt": {
    "N": "1574469356331"
  },
  "_ttl": {
    "N": "1574472956"
  },
  "_version": {
    "N": "1"
  },
  "author": {
    "S": "Nadia"
  },
  "content": {
    "S": "Hello World"
  },
  "ds_pk": {
    "S": "AppSync-delta-sync-post:2019-11-23"
  },
  "ds_sk": {
    "S": "00:35:56.331:81d36bbb-1579-4efe-92b8-2e3f679f628b:1"
  },
  "id": {
    "S": "81d36bbb-1579-4efe-92b8-2e3f679f628b"
  },
  "title": {
    "S": "My First Post"
  }
}
```

Sekarang kita dapat mensimulasikan kueri Base yang akan dijalankan klien untuk menghidrasi penyimpanan data lokalnya menggunakan `syncPosts` kueri seperti:

```
query baseQuery {
  syncPosts(limit: 100, lastSync: null, nextToken: null) {
    items {
      id
      author
      title
```

```
      content
      _version
      _lastChangedAt
    }
    startedAt
    nextToken
  }
}
```

Nilai pengembalian kueri Base ini akan terlihat sebagai berikut:

```
{
  "data": {
    "syncPosts": {
      "items": [
        {
          "id": "81d36bbb-1579-4efe-92b8-2e3f679f628b",
          "author": "Nadia",
          "title": "My First Post",
          "content": "Hello World",
          "_version": 1,
          "_lastChangedAt": 1574469356331
        }
      ],
      "startedAt": 1574469602238,
      "nextToken": null
    }
  }
}
```

Kita akan menyimpan `startedAt` nilainya nanti untuk mensimulasikan kueri Delta, tapi pertama-tama kita perlu membuat perubahan pada tabel kita. Mari gunakan `updatePost` mutasi untuk memodifikasi Posting kami yang ada:

```
mutation updatePost {
  updatePost(input: {id: "81d36bbb-1579-4efe-92b8-2e3f679f628b", _version: 1, title:
 "Actually this is my Second Post"}) {
    id
    author
    title
    content
    _version
```

```
    _lastChangedAt
    _deleted
  }
}
```

Nilai pengembalian mutasi ini akan terlihat sebagai berikut:

```
{
  "data": {
    "updatePost": {
      "id": "81d36bbb-1579-4efe-92b8-2e3f679f628b",
      "author": "Nadia",
      "title": "Actually this is my Second Post",
      "content": "Hello World",
      "_version": 2,
      "_lastChangedAt": 1574469851417,
      "_deleted": null
    }
  }
}
```

Jika Anda memeriksa isi tabel Base sekarang, Anda akan melihat item yang diperbarui:

```
{
  "_lastChangedAt": {
    "N": "1574469851417"
  },
  "_version": {
    "N": "2"
  },
  "author": {
    "S": "Nadia"
  },
  "content": {
    "S": "Hello World"
  },
  "id": {
    "S": "81d36bbb-1579-4efe-92b8-2e3f679f628b"
  },
  "title": {
    "S": "Actually this is my Second Post"
  }
```

```
}
```

Jika Anda memeriksa isi tabel Delta sekarang, Anda akan melihat dua catatan:

1. Catatan saat item dibuat
2. Catatan kapan item diperbarui.

Item baru akan terlihat seperti:

```
{
  "_lastChangedAt": {
    "N": "1574469851417"
  },
  "_ttl": {
    "N": "1574473451"
  },
  "_version": {
    "N": "2"
  },
  "author": {
    "S": "Nadia"
  },
  "content": {
    "S": "Hello World"
  },
  "ds_pk": {
    "S": "AppSync-delta-sync-post:2019-11-23"
  },
  "ds_sk": {
    "S": "00:44:11.417:81d36bbb-1579-4efe-92b8-2e3f679f628b:2"
  },
  "id": {
    "S": "81d36bbb-1579-4efe-92b8-2e3f679f628b"
  },
  "title": {
    "S": "Actually this is my Second Post"
  }
}
```

Sekarang kita dapat mensimulasikan kueri Delta untuk mengambil modifikasi yang terjadi ketika klien sedang offline. Kami akan menggunakan `startedAt` nilai yang dikembalikan dari kueri Base kami untuk membuat permintaan:

```
query delta {
  syncPosts(limit: 100, lastSync: 1574469602238, nextToken: null) {
    items {
      id
      author
      title
      content
      _version
    }
    startedAt
    nextToken
  }
}
```

Nilai pengembalian kueri Delta ini akan terlihat sebagai berikut:

```
{
  "data": {
    "syncPosts": {
      "items": [
        {
          "id": "81d36bbb-1579-4efe-92b8-2e3f679f628b",
          "author": "Nadia",
          "title": "Actually this is my Second Post",
          "content": "Hello World",
          "_version": 2
        }
      ],
      "startedAt": 1574470400808,
      "nextToken": null
    }
  }
}
```

Konfigurasi dan pengaturan

AWS AppSyncmemungkinkan Anda untuk:

- Data cache yang sering diminta tetapi tidak mungkin berubah dari permintaan ke permintaan. Ini dapat mengurangi beban pada resolver Anda. Untuk informasi selengkapnya, lihat the section called “Caching dan kompresi”.
- Version GraphQL objek untuk menangani dan menghindari konflik di antara beberapa klien. Untuk informasi selengkapnya, lihat the section called “Deteksi dan Sinkronisasi Konflik”.
- Gunakan nama domain khusus untuk mengonfigurasi domain tunggal yang mudah diingat yang berfungsi untuk GraphQL dan API real-time Anda. Untuk informasi selengkapnya, lihat Mengonfigurasi nama domain kustom.
- Izinkan akses ke GraphQL API Anda melalui VPC. Untuk informasi selengkapnya, lihat Menggunakan API AWS AppSync Pribadi.
- Aktifkan introspeksi dan atur kedalaman kueri dan batas resolver per kueri. Untuk informasi selengkapnya, lihat Batas konfigurasi.

Selain itu, AWS AppSync termasuk AWS alat standar berikut untuk logging, monitoring, dan tracing:

- Masuk AWS CloudTrail
- Pemantauan dengan Amazon CloudWatch
- Menelusuri dengan AWS X-Ray

Caching dan kompresi

AWS AppSyncKemampuan caching data sisi server membuat data tersedia dalam cache dalam memori berkecepatan tinggi, meningkatkan kinerja, dan mengurangi latensi. Ini mengurangi kebutuhan untuk mengakses sumber data secara langsung. Caching tersedia untuk resolver unit dan pipeline.

AWS AppSyncjuga memungkinkan Anda untuk mengompres respons API sehingga konten muatan dimuat dan diunduh lebih cepat. Ini berpotensi mengurangi ketegangan pada aplikasi Anda sekaligus berpotensi mengurangi biaya transfer data Anda. Perilaku kompresi dapat dikonfigurasi dan dapat diatur sesuai kebijaksanaan Anda sendiri.

Lihat bagian ini untuk membantu menentukan perilaku caching dan kompresi sisi server yang diinginkan di API Anda. AWS AppSync

Tipe instans

AWS AppSyncmeng-host Amazon ElastiCache untuk instans Redis di AWS akun dan AWS Wilayah yang sama dengan API Anda AWS AppSync .

Berikut ini ElastiCache untuk jenis instans Redis tersedia:

small

1 vCPU, RAM 1,5 GiB, kinerja jaringan rendah hingga sedang

medium

2 vCPU, RAM 3 GiB, kinerja jaringan rendah hingga sedang

large

2 vCPU, 12,3 GiB RAM, hingga 10 Gigabit kinerja jaringan

xlarge

4 vCPU, 25,05 GiB RAM, hingga 10 Gigabit kinerja jaringan

2xlarge

8 vCPU, 50,47 GiB RAM, hingga 10 Gigabit kinerja jaringan

4xlarge

16 vCPU, 101.38 GiB RAM, hingga 10 Gigabit kinerja jaringan

8xlarge

32 vCPU, 203.26 GiB RAM, kinerja jaringan 10 Gigabit (tidak tersedia di semua Wilayah)

12xlarge

48 vCPU, 317,77 GiB RAM, kinerja jaringan 10 Gigabit

> **Note**
>
> Secara historis, Anda menentukan jenis instance tertentu (sepert`it2.medium`). Mulai Juli 2020, jenis instans lama ini terus tersedia, tetapi penggunaannya tidak digunakan lagi dan

tidak disarankan. Kami menyarankan Anda menggunakan jenis instance generik yang dijelaskan di sini.

Perilaku caching

Berikut ini adalah perilaku yang terkait dengan caching:

Tidak ada

Tidak ada caching sisi server.

Caching permintaan penuh

Jika data tidak dalam cache, itu diambil dari sumber data dan mengisi cache sampai waktu untuk hidup (TTL) kedaluwarsa. Semua permintaan berikutnya ke API Anda dikembalikan dari cache. Ini berarti bahwa sumber data tidak dihubungi secara langsung kecuali TTL kedaluwarsa. Dalam pengaturan ini, kami menggunakan konten `context.arguments` dan `context.identity` peta sebagai kunci caching.

Caching per penyelesai

Dengan pengaturan ini, setiap resolver harus secara eksplisit memilih untuk men-cache respons. Anda dapat menentukan tombol TTL dan caching pada resolver. Kunci cache yang dapat Anda tentukan adalah peta tingkat atas, dan `context.arguments` `context.sourcecontext.identity`, dan/atau bidang string dari peta ini. Nilai TTL adalah wajib, tetapi kunci caching adalah opsional. Jika Anda tidak menentukan kunci caching apa pun, defaultnya adalah isi dari`context.arguments`,, `context.source` dan peta. `context.identity`

Misalnya, Anda dapat menggunakan kombinasi berikut:

- context.arguments dan context.source
- context.arguments dan context.identity.sub
- context.arguments.id atau context.arguments. InputType.id
- context.source.id dan context.identity.sub
- context.identity.claims.username

Bila Anda hanya menentukan TTL dan tidak ada kunci caching, perilaku resolver sama dengan caching permintaan penuh.

Waktu cache untuk hidup

Pengaturan ini menentukan jumlah waktu untuk menyimpan entri yang di-cache dalam memori. TTL maksimum adalah 3.600 detik (1 jam), setelah itu entri dihapus secara otomatis.

Enkripsi cache

Enkripsi cache hadir dalam dua rasa berikut. Ini mirip dengan pengaturan yang ElastiCache memungkinkan Redis. Anda dapat mengaktifkan pengaturan enkripsi hanya ketika pertama kali mengaktifkan caching untuk API Anda AWS AppSync .

- Enkripsi dalam perjalanan — Permintaan antara AWS AppSync, cache, dan sumber data (kecuali sumber data HTTP yang tidak aman) dienkripsi di tingkat jaringan. Karena ada beberapa pemrosesan yang diperlukan untuk mengenkripsi dan mendekripsi data di titik akhir, enkripsi dalam transit dapat memengaruhi kinerja.
- Enkripsi saat istirahat — Data yang disimpan ke disk dari memori selama operasi swap dienkripsi pada instance cache. Pengaturan ini juga memengaruhi kinerja.

Untuk membatalkan entri cache, Anda dapat membuat panggilan API cache flush menggunakan AWS AppSync konsol atau (). AWS Command Line Interface AWS CLI

Untuk informasi selengkapnya, lihat tipe ApiCachedata di Referensi AWS AppSync API.

Penggusuran cache

Saat Anda mengatur AWS AppSync caching sisi server, Anda dapat mengonfigurasi TTL maksimum. Nilai ini mendefinisikan jumlah waktu entri cache disimpan dalam memori. Dalam situasi di mana Anda harus menghapus entri tertentu dari cache Anda, Anda dapat menggunakan AWS AppSync utilitas `evictFromApiCache` ekstensi dalam permintaan atau respons resolver Anda. (Misalnya, ketika data Anda di sumber data Anda telah berubah, dan entri cache Anda sekarang basi.) Untuk mengusir item dari cache, Anda harus tahu kuncinya. Untuk alasan ini, jika Anda harus mengusir item secara dinamis, sebaiknya gunakan caching per-resolver dan secara eksplisit mendefinisikan kunci yang akan digunakan untuk menambahkan entri ke cache Anda.

Mengusir entri cache

Untuk mengusir item dari cache, gunakan utilitas `evictFromApiCache` ekstensi. Tentukan nama tipe dan nama bidang, lalu berikan objek item nilai kunci untuk membangun kunci entri yang ingin

Anda usir. Dalam objek, setiap kunci mewakili entri yang valid dari `context` objek yang digunakan dalam daftar resolver cache. `cachingKey` Setiap nilai adalah nilai aktual yang digunakan untuk membangun nilai kunci. Anda harus meletakkan item dalam objek dalam urutan yang sama dengan kunci caching dalam daftar resolver cache. `cachingKey`

Misalnya, lihat skema berikut:

```
type Note {
  id: ID!
  title: String
  content: String!
}

type Query {
  getNote(id: ID!): Note
}

type Mutation {
  updateNote(id: ID!, content: String!): Note
}
```

Dalam contoh ini, Anda dapat mengaktifkan caching per-resolver, lalu mengaktifkannya untuk kueri. `getNote` Kemudian, Anda dapat mengonfigurasi kunci caching untuk terdiri dari. `[context.arguments.id]`

Ketika Anda mencoba untuk mendapatkan`Note`, untuk membangun kunci cache, AWS AppSync melakukan pencarian di cache sisi server menggunakan `id` argumen kueri. `getNote`

Ketika Anda memperbarui`Note`, Anda harus mengusir entri untuk catatan tertentu untuk memastikan bahwa permintaan berikutnya mengambilnya dari sumber data backend. Untuk melakukan ini, Anda harus membuat penangan permintaan.

Contoh berikut menunjukkan salah satu cara untuk menangani penggusuran menggunakan metode ini:

```
import { util, Context } from '@aws-appsync/utils';
import { update } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 extensions.evictFromApiCache('Query', 'getNote', { 'ctx.args.id': ctx.args.id });
 return update({ key: { id: ctx.args.id }, update: { context: ctx.args.content } });
```

```
}

export const response = (ctx) => ctx.result;
```

Atau, Anda juga dapat menangani penggusuran di penangan respons.

Ketika `updateNote` mutasi diproses, cobalah untuk AWS AppSync mengusir entri. Jika entri berhasil dihapus, respons berisi `apiCacheEntriesDeleted` nilai dalam `extensions` objek yang menunjukkan berapa banyak entri yang dihapus:

```
"extensions": {  "apiCacheEntriesDeleted": 1}
```

Mengusir entri cache berdasarkan identitas

Anda dapat membuat kunci caching berdasarkan beberapa nilai dari `context` objek.

Misalnya, ambil skema berikut yang menggunakan kumpulan pengguna Amazon Cognito sebagai mode autentikasi default dan didukung oleh sumber data Amazon DynamoDB:

```
type Note {
  id: ID! # a slug; e.g.: "my-first-note-on-graphql"
  title: String
  content: String!
}

type Query {
  getNote(id: ID!): Note
}

type Mutation {
  updateNote(id: ID!, content: String!): Note
}
```

Jenis `Note` objek disimpan dalam tabel DynamoDB. Tabel memiliki kunci komposit yang menggunakan nama pengguna Amazon Cognito sebagai kunci utama dan `id` (siput) `Note` sebagai kunci partisi. Ini adalah sistem multi-penyewa yang memungkinkan banyak pengguna untuk meng-host dan memperbarui `Note` objek pribadi mereka, yang tidak pernah dibagikan.

Karena ini adalah sistem read-heavy, `getNote` kueri di-cache menggunakan caching per-resolver, dengan kunci caching terdiri dari. `[context.identity.username, context.arguments.id]`

Ketika a `Note` diperbarui, Anda dapat mengusir entri untuk spesifik itu. `Note` Anda harus menambahkan komponen dalam objek dalam urutan yang sama dengan yang ditentukan dalam daftar resolver Anda. `cachingKeys`

Contoh berikut menunjukkan ini:

```
import { util, Context } from '@aws-appsync/utils';
import { update } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 extensions.evictFromApiCache('Query', 'getNote', {
  'ctx.identity.username': ctx.identity.username,
  'ctx.args.id': ctx.args.id,
 });
 return update({ key: { id: ctx.args.id }, update: { context: ctx.args.content } });
}

export const response = (ctx) => ctx.result;
```

Sistem backend juga dapat memperbarui `Note` dan mengusir entri. Misalnya, ambil mutasi ini:

```
type Mutation {
  updateNoteFromBackend(id: ID!, content: String!, username: ID!): Note @aws_iam
}
```

Anda dapat mengusir entri, tetapi menambahkan komponen kunci caching ke objek. `cachingKeys`

Dalam contoh berikut, penggusuran terjadi sebagai respons penyelesai:

```
import { util, Context } from '@aws-appsync/utils';
import { update } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 extensions.evictFromApiCache('Query', 'getNote', {
  'ctx.identity.username': ctx.args.username,
  'ctx.args.id': ctx.args.id,
 });
 return update({ key: { id: ctx.args.id }, update: { context: ctx.args.content } });
}

export const response = (ctx) => ctx.result;
```

Dalam kasus di mana data backend Anda telah diperbarui di luar AWS AppSync, Anda dapat mengusir item dari cache dengan memanggil mutasi yang menggunakan sumber data. `NONE`

Mengompresi respons API

AWS AppSync memungkinkan klien untuk meminta muatan terkompresi. Jika diminta, respons API dikompresi dan dikembalikan sebagai tanggapan atas permintaan yang menunjukkan bahwa konten terkompresi lebih disukai. Respons API terkompresi dimuat lebih cepat, konten diunduh lebih cepat, dan biaya transfer data Anda juga dapat dikurangi.

> **Note**
>
> Kompresi tersedia di semua API baru yang dibuat setelah 1 Juni 2020.
> AWS AppSync mengompres objek dengan upaya terbaik. Dalam kasus yang jarang terjadi, AWS AppSync dapat melewatkan kompresi berdasarkan berbagai faktor, termasuk kapasitas saat ini.

AWS AppSync dapat mengompres ukuran muatan kueri GraphQL antara 1.000 hingga 10.000.000 byte. Untuk mengaktifkan kompresi, klien harus mengirim `Accept-Encoding` header dengan nilai`gzip`. Kompresi dapat diverifikasi dengan memeriksa nilai `Content-Encoding` header dalam respon (`gzip`).

Penjelajah kueri di AWS AppSync konsol secara otomatis menetapkan nilai header dalam permintaan secara default. Jika Anda menjalankan kueri yang memiliki respons yang cukup besar, kompresi dapat dikonfirmasi menggunakan alat pengembang browser Anda.

Mengkonfigurasi nama domain kustom

denganAWS AppSync, Anda dapat menggunakan nama domain khusus untuk mengonfigurasi satu domain yang mudah diingat yang berfungsi untuk API GraphQL dan real-time Anda.

Dengan kata lain, Anda dapat memanfaatkan URL endpoint yang sederhana dan mudah diingat dengan nama domain pilihan Anda dengan membuat nama domain khusus yang Anda kaitkan denganAWS AppSyncAPI di akun Anda.

Saat Anda mengonfigurasiAWS AppSyncAPI, dua titik akhir disediakan:

AWS AppSyncTitik akhir GraphQL:

```
https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql
```

AWS AppSynctitik akhir waktu nyata:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/
graphql
```

Dengan nama domain khusus, Anda dapat berinteraksi dengan kedua titik akhir menggunakan satu domain. Misalnya, jika Anda mengkonfigurasi`api.example.com`sebagai domain kustom, Anda dapat berinteraksi dengan GraphQL dan titik akhir real-time menggunakan URL berikut:

AWS AppSynctitik akhir GraphQL domain kustom:

```
https://api.example.com/graphql
```

AWS AppSynctitik akhir real-time domain kustom:

```
wss://api.example.com/graphql/realtime
```

> **Note**
>
> AWS AppSyncAPI hanya mendukung TLS 1.2 dan TLS 1.3 untuk nama domain khusus.

Mendaftarkan dan mengonfigurasi nama domain

Untuk mengatur nama domain kustom untukAWS AppSyncAPI, Anda harus memiliki nama domain internet terdaftar. Anda dapat mendaftarkan domain internet menggunakanAmazon Route 53 domain registrationatau registrar domain pihak ketiga pilihan Anda. Untuk informasi lebih lanjut tentang Route 53, lihatApa itu Amazon Route 53?diPanduan Pengembang Amazon Route 53.

Nama domain kustom API dapat berupa nama subdomain atau domain root (juga dikenal sebagai “zone apex”) dari domain internet terdaftar. Setelah Anda membuat nama domain kustom diAWS AppSync, Anda harus membuat atau memperbarui catatan sumber daya penyedia DNS Anda untuk dipetakan ke titik akhir API Anda. Tanpa pemetaan ini, permintaan API yang terikat untuk nama domain kustom tidak dapat mencapaiAWS AppSync.

Membuat nama domain khusus diAWS AppSync

Membuat nama domain khusus untukAWS AppSyncAPI menyiapkanAmazon CloudFrontdistribusi. Anda harus menyiapkan catatan DNS untuk memetakan nama domain kustom keCloudFrontnama domain distribusi. Pemetaan ini diperlukan untuk merutekan permintaan API yang terikat untuk nama domain kustomAWS AppSyncmelalui yang dipetakanCloudFrontdistribusi. Anda juga harus memberikan sertifikat untuk nama domain kustom.

Untuk mengatur nama domain kustom atau memperbarui sertifikatnya, Anda harus memiliki izin untuk memperbaruiCloudFrontDistribusi dan menggambarkanAWS Certificate Manager(ACM) sertifikat yang Anda rencanakan untuk digunakan. Untuk memberikan izin ini, lampirkan yang berikutAWS Identity and Access Management(IAM) pernyataan kebijakan kepada pengguna, grup, atau peran IAM di akun Anda:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowUpdateDistributionForAppSyncCustomDomainName",
      "Effect": "Allow",
      "Action": ["cloudfront:updateDistribution"],
      "Resource": ["*"]
    },
    {
      "Sid": "AllowDescribeCertificateForAppSyncCustomDomainName",
      "Effect": "Allow",
      "Action": "acm:DescribeCertificate",
      "Resource": "arn:aws:acm:<region>:<account-id>:certificate/<certificate-id>"
    }
  ]
}
```

AWS AppSyncmendukung nama domain kustom dengan memanfaatkan Server Name Indication (SNI) padaCloudFrontdistribusi. Untuk informasi selengkapnya tentang menggunakan nama domain kustom diCloudFrontdistribusi, termasuk format sertifikat yang diperlukan dan panjang kunci sertifikat maksimum, lihatMenggunakan HTTPS denganCloudFrontdiAmazonCloudFrontPanduan Pengembang.

Untuk menyiapkan nama domain kustom sebagai nama host API, pemilik API harus memberikan sertifikat SSL/TLS untuk nama domain kustom. Untuk memberikan sertifikat, lakukan salah satu hal berikut:

- Minta sertifikat baru di ACM, atau impor sertifikat yang dikeluarkan oleh otoritas sertifikat pihak ketiga ke ACM di`us-east-1` AWSWilayah (AS Timur (Virginia N.)). Untuk informasi selengkapnya tentang ACM, lihatApa ituAWS Certificate Manager?diAWS Certificate ManagerPanduan Pengguna.
- Berikan sertifikat server IAM. Untuk informasi lebih lanjut, lihatMengelola sertifikat server di IAMdiPanduan Pengguna IAM.

Nama domain kustom wildcard diAWS AppSync

AWS AppSyncmendukung nama domain kustom wildcard. Untuk mengonfigurasi nama domain khusus wildcard, tentukan karakter wildcard (`*`) sebagai subdomain pertama dari domain kustom. Ini mewakili semua kemungkinan subdomain dari domain root. Misalnya, nama domain kustom wildcard`*.example.com`menghasilkan subdomain sepertia`.example.com`,`b.example.com`, dan`c.example.com`. Semua subdomain ini merutekan ke domain yang sama.

Untuk menggunakan nama domain kustom wildcard diAWS AppSync, Anda harus memberikan sertifikat yang dikeluarkan oleh ACM yang berisi nama wildcard yang dapat melindungi beberapa situs dalam domain yang sama. Untuk informasi lebih lanjut, lihatKarakteristik sertifikat ACMdiAWS Certificate ManagerPanduan Pengguna.

Deteksi dan Sinkronisasi Konflik

Sumber Data Berversioning

AWS AppSync saat ini mendukung versi pada sumber data DynamoDB. Operasi Deteksi Konflik, Resolusi Konflik, dan Sinkronisasi memerlukan sumber`Versioned` data. Ketika Anda mengaktifkan versi pada sumber data,AWS AppSync akan secara otomatis:

- Tingkatkan item dengan metadata versi objek.
- Rekam perubahan yang dilakukan pada item denganAWS AppSync mutasi ke tabel Delta.
- Pertahankan item yang dihapus di tabel Dasar dengan “batu nisan” untuk jumlah waktu yang dapat dikonfigurasi.

Konfigurasi Sumber Data Berversi

Saat Anda mengaktifkan versioning di sumber data DynamoDB, Anda menentukan versioning berikut:

`BaseTableTTL`

Jumlah menit untuk mempertahankan item yang dihapus di tabel Dasar dengan “batu nisan” - bidang metadata yang menunjukkan bahwa item tersebut telah dihapus. Anda dapat mengatur nilai ini ke 0 jika Anda ingin item segera dihapus saat dihapus. Bidang ini wajib diisi.

`DeltaSyncTableName`

Nama tabel tempat perubahan yang dilakukan pada item denganAWS AppSync mutasi disimpan. Bidang ini wajib diisi.

`DeltaSyncTableTTL`

Jumlah menit untuk menyimpan item di meja Delta. Bidang ini wajib diisi.

Tabel Sinkronisasi Delta

AWS AppSync saat ini mendukung Delta Sync Logging untuk mutasi menggunakan`PutItem`,`UpdateItem`, dan operasi`DeleteItem` DynamoDB.

KetikaAWS AppSync mutasi mengubah item dalam sumber data berversi, catatan perubahan itu akan disimpan dalam tabel Delta yang dioptimalkan untuk pembaruan tambahan. Anda dapat memilih untuk menggunakan tabel Delta yang berbeda (misalnya satu per jenis, satu per area domain) untuk sumber data versi lain atau satu tabel Delta untuk API Anda. AWS AppSync merekomendasikan agar tidak menggunakan tabel Delta tunggal untuk beberapa API untuk menghindari tabrakan kunci primer.

Skema yang diperlukan untuk tabel ini adalah sebagai berikut:

`ds_pk`

Sebuah nilai string yang digunakan sebagai kunci partisi. Ini dibangun dengan menggabungkan nama sumber data Base dan format ISO 8601 dari tanggal perubahan terjadi (misalnya`Comments:2019-01-01`).

Ketika`customPartitionKey` bendera dari template pemetaan VTL ditetapkan sebagai nama kolom kunci partisi (lihat Referensi Template Pemetaan Resolver untuk DynamoDB di

PanduanAWS AppSync Pengembang), formatds_pk perubahan, dan string dibuat dengan menambahkan nilai kunci partisi dalam catatan baru di tabel Dasar. Misalnya, jika catatan dalam tabel Base memiliki nilai kunci partisi1a dan nilai kunci semacam2b, nilai baru dari string akan:Comments:2019-01-01:1a.

ds_sk

Sebuah nilai string yang digunakan sebagai kunci semacam. Ini dibangun dengan menggabungkan format ISO 8601 dari waktu perubahan terjadi, kunci utama item, dan versi item. Kombinasi bidang ini menjamin keunikan untuk setiap entri dalam tabel Delta (misalnya untuk waktu09:30:00 dan ID dari1a dan versi2, ini akan09:30:00:1a:2).

KetikacustomPartitionKey bendera dari template pemetaan VTL diatur ke nama kolom kunci partisi (lihat Referensi Template Pemetaan Resolver untuk DynamoDB di PanduanAWS AppSync Pengembang), formatds_sk perubahan, dan string dibuat dengan mengganti nilai tombol kombinasi dengan nilai kunci sortir di tabel dasar. Menggunakan contoh sebelumnya di atas, jika catatan dalam tabel Base memiliki nilai kunci partisi1a dan nilai kunci semacam2b, nilai baru dari string akan:09:30:00:2b:3.

_ttl

Nilai numerik yang menyimpan stempel waktu, dalam detik epoch, ketika item harus dihapus dari tabel Delta. Nilai ini ditentukan dengan menambahkanDeltaSyncTableTTL nilai yang dikonfigurasi pada sumber data ke momen ketika perubahan terjadi. Bidang ini harus dikonfigurasi sebagai Atribut TTL DynamoDB.

Peran IAM yang dikonfigurasi untuk digunakan dengan tabel Dasar juga harus berisi izin untuk beroperasi pada tabel Delta. Dalam contoh ini, kebijakan perizinan untuk tabel Base disebutComments dan tabel DeltaChangeLog disebut ditampilkan:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
```

```
                "dynamodb:UpdateItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:000000000000:table/Comments",
                "arn:aws:dynamodb:us-east-1:000000000000:table/Comments/*",
                "arn:aws:dynamodb:us-east-1:000000000000:table/ChangeLog",
                "arn:aws:dynamodb:us-east-1:000000000000:table/ChangeLog/*"
            ]
        }
    ]
}
```

Metadata Sumber Data Berversi

AWS AppSync mengelola bidang metadata pada sumber`Versioned` data atas nama Anda. Memodifikasi bidang ini sendiri dapat menyebabkan kesalahan dalam aplikasi atau kehilangan data Anda. Bidang ini meliputi:

`_version`

Penghitung yang meningkat secara monoton yang diperbarui setiap saat perubahan terjadi pada item.

`_lastChangedAt`

Nilai numerik yang menyimpan stempel waktu, dalam milidetik epoch, saat item terakhir diubah.

`_deleted`

Sebuah Boolean “batu nisan” nilai yang menunjukkan bahwa item telah dihapus. Ini dapat digunakan oleh aplikasi untuk mengusir item yang dihapus dari toko data lokal.

`_ttl`

Nilai numerik yang menyimpan stempel waktu, dalam hitungan detik, ketika item harus dihapus dari sumber data yang mendasarinya.

`ds_pk`

Sebuah nilai string yang digunakan sebagai kunci partisi untuk tabel Delta.

`ds_sk`

Sebuah nilai string yang digunakan sebagai kunci semacam untuk tabel Delta.

`gsi_ds_pk`

Atribut nilai string yang dihasilkan untuk mendukung indeks sekunder global sebagai kunci partisi. Ini akan disertakan hanya jika keduapopulateIndexFields benderacustomPartitionKey dan diaktifkan dalam template pemetaan VTL (lihat Referensi Template Pemetaan Resolver untuk DynamoDB di PanduanAWS AppSync Pengembang). Jika diaktifkan, nilai akan dibangun dengan menggabungkan nama sumber data dasar dan format ISO 8601 dari tanggal di mana perubahan terjadi (misalnya jika tabel Dasar diberi nama Komentar, catatan ini akan ditetapkan sebagaiComments:2019-01-01).

`gsi_ds_sk`

Atribut nilai string yang dihasilkan untuk mendukung indeks sekunder global sebagai kunci semacam. Ini akan disertakan hanya jika keduapopulateIndexFields benderacustomPartitionKey dan diaktifkan dalam template pemetaan VTL (lihat Referensi Template Pemetaan Resolver untuk DynamoDB di PanduanAWS AppSync Pengembang). Jika diaktifkan, nilai akan dibangun dengan menggabungkan format ISO 8601 dari waktu di mana perubahan terjadi, kunci partisi item dalam tabel Dasar, kunci semacam item dalam tabel Dasar, dan versi item (misalnya untuk waktu09:30:00, nilai kunci partisi1a, nilai kunci semacam2b, dan versi3, ini akan09:30:00:1a#2b:3).

Bidang metadata ini akan memengaruhi ukuran keseluruhan item di sumber data yang mendasari. AWS AppSync merekomendasikan pemesanan 500 byte+Max Primary Key Size penyimpanan untuk metadata sumber data berversi saat merancang aplikasi Anda. Untuk menggunakan metadata ini dalam aplikasi klien, sertakan_version,_lastChangedAt, dan_deleted bidang pada jenis GraphQL Anda dan dalam pemilihan yang ditetapkan untuk mutasi.

Deteksi dan Resolusi Konflik

Ketika penulisan bersamaan terjadiAWS AppSync, Anda dapat mengonfigurasi strategi Deteksi Konflik dan Penyelesaian Konflik untuk menangani pembaruan dengan tepat. Deteksi Konflik menentukan apakah mutasi bertentangan dengan item tertulis yang sebenarnya di sumber data. Deteksi Konflik diaktifkan dengan menetapkan nilai SyncConfig diconflictDetection bidang untukVERSION.

Resolusi Konflik adalah tindakan yang diambil jika konflik terdeteksi. Hal ini ditentukan dengan menetapkan bidang Conflict Handler di SyncConfig. Ada tiga strategi Resolusi Konflik:

- OPTIMISTIK_CONCURRENCY

- AUTOMERGE
- LAMBDA

Masing-masing strategi Resolusi Konflik ini dirinci secara mendalam di bawah ini.

Versi secara otomatis bertambah AppSync selama operasi tulis dan tidak boleh dimodifikasi oleh klien atau di luar resolver yang dikonfigurasi dengan sumber data yang diaktifkan versi. Melakukannya akan mengubah perilaku konsistensi sistem dan dapat mengakibatkan kehilangan data.

Konkurensi Optimis

Konkurensi Optimis adalah strategi resolusi konflik yangAWS AppSync menyediakan sumber data berversi. Ketika resolver konflik diatur ke Optimistic Concurrency, jika mutasi masuk terdeteksi memiliki versi yang berbeda dari versi sebenarnya dari objek, pengendali konflik hanya akan menolak permintaan yang masuk. Di dalam respons GraphQL, item yang ada di server yang memiliki versi terbaru akan disediakan. Klien kemudian diharapkan untuk menangani konflik ini secara lokal dan mencoba lagi mutasi dengan versi terbaru dari item.

Automerge

Automerge menyediakan pengembang cara mudah untuk mengkonfigurasi strategi resolusi konflik tanpa menulis logika sisi klien untuk menggabungkan konflik secara manual yang tidak dapat ditangani oleh strategi lain. Automerge mematuhi aturan ketat yang ditetapkan saat menggabungkan data untuk menyelesaikan konflik. Prinsip Automerge berputar di sekitar tipe data yang mendasari bidang GraphQL. Mereka adalah sebagai berikut:

- Konflik pada bidang skalar: skalar GraphQL atau bidang apa pun yang bukan koleksi (yaitu List, Set, Map). Tolak nilai yang masuk untuk bidang skalar dan pilih nilai yang ada di server.
- Konflik pada daftar: Jenis GraphQL dan tipe database adalah daftar. Menggabungkan daftar masuk dengan daftar yang ada di server. Nilai daftar dalam mutasi yang masuk akan ditambahkan ke akhir daftar di server. Nilai duplikat akan dipertahankan.
- Konflik pada set: Jenis GraphQL adalah daftar dan jenis database adalah Set. Terapkan serikat set menggunakan masuk set dan set yang ada di server. Ini menganut properti Set, yang berarti tidak ada entri duplikat.
- Ketika mutasi masuk menambahkan bidang baru untuk item atau dibuat terhadap bidang dengan nilai`null`, menggabungkan yang pada item yang ada.

- Konflik pada peta: Ketika tipe data yang mendasari dalam database adalah Peta (yaitu dokumen kunci-nilai), terapkan aturan di atas saat mem-parsing dan memproses setiap properti Peta.

Automerge dirancang untuk secara otomatis mendeteksi, menggabungkan, dan mencoba kembali permintaan dengan versi terbaru, membebaskan klien dari kebutuhan untuk menggabungkan data yang saling bertentangan secara manual.

Untuk menunjukkan contoh bagaimana Automerge menangani Konflik pada tipe Skalar. Kami akan menggunakan catatan berikut sebagai titik awal kami.

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "_version" : 4
}
```

Sekarang mutasi yang masuk mungkin mencoba memperbarui item tetapi dengan versi yang lebih lama karena klien belum disinkronkan dengan server. Terlihat seperti ini:

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 55,
  "_version" : 2
}
```

Perhatikan versi 2 yang sudah ketinggalan zaman dalam permintaan yang masuk. Selama alur ini, Automerge akan menggabungkan data dengan menolak pembaruan bidang 'jersey' ke '55' dan menjaga nilai pada '5' sehingga menghasilkan gambar berikut dari item yang disimpan di server.

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "_version" : 5 # version is incremented every time automerge performs a merge that is
 stored on the server.
}
```

Mengingat keadaan item yang ditunjukkan di atas pada versi 5, sekarang misalkan mutasi masuk yang mencoba untuk bermutasi item dengan gambar berikut:

```
{
  "id" : 1,
  "name" : "Shaggy",
  "jersey" : 5,
  "interests" : ["breakfast", "lunch", "dinner"] # underlying data type is a Set
  "points": [24, 30, 27] # underlying data type is a List
  "_version" : 3
}
```

Ada tiga poin yang menarik dalam mutasi yang masuk. Nama, skalar, telah diubah tetapi dua bidang baru “kepentingan”, Set, dan “poin”, Daftar, telah ditambahkan. Dalam skenario ini, konflik akan terdeteksi karena ketidakcocokan versi. Automerge mematuhi sifat-sifatnya dan menolak perubahan nama karena itu menjadi skalar dan menambahkan pada bidang non-konflik. Ini menghasilkan item yang disimpan di server muncul sebagai berikut.

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "interests" : ["breakfast", "lunch", "dinner"] # underlying data type is a Set
  "points": [24, 30, 27] # underlying data type is a List
  "_version" : 6
}
```

Dengan gambar item yang diperbarui dengan versi 6, sekarang misalkan mutasi yang masuk (dengan ketidakcocokan versi lain) mencoba mengubah item menjadi yang berikut:

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "interests" : ["breakfast", "lunch", "brunch"] # underlying data type is a Set
  "points": [30, 35] # underlying data type is a List
  "_version" : 5
}
```

Di sini kita amati bahwa bidang masuk untuk “kepentingan” memiliki satu nilai duplikat yang ada di server dan dua nilai baru. Dalam hal ini, karena tipe data yang mendasarinya adalah Set, Automerge

akan menggabungkan nilai-nilai yang ada di server dengan orang-orang dalam permintaan masuk dan strip keluar duplikat apapun. Demikian pula ada konflik pada bidang “poin” di mana ada satu nilai duplikat dan satu nilai baru. Tapi karena tipe data yang mendasari di sini adalah Daftar, Automerge hanya akan menambahkan semua nilai dalam permintaan yang masuk ke akhir nilai yang sudah ada di server. Gambar gabungan yang dihasilkan disimpan di server akan muncul sebagai berikut:

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "interests" : ["breakfast", "lunch", "dinner", "brunch"] # underlying data type is a
 Set
  "points": [24, 30, 27, 30, 35] # underlying data type is a List
  "_version" : 7
}
```

Sekarang mari kita asumsikan item yang disimpan di server muncul sebagai berikut, pada versi 8.

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "interests" : ["breakfast", "lunch", "dinner", "brunch"] # underlying data type is a
 Set
  "points": [24, 30, 27, 30, 35] # underlying data type is a List
  "stats": {
      "ppg": "35.4",
      "apg": "6.3"
  }
  "_version" : 8
}
```

Tetapi permintaan masuk mencoba memperbarui item dengan gambar berikut, sekali lagi dengan ketidakcocokan versi:

```
{
  "id" : 1,
  "name" : "Nadia",
  "stats": {
      "ppg": "25.7",
      "rpg": "6.9"
  }
```

```
  "_version" : 3
}
```

Sekarang dalam skenario ini, kita dapat melihat bahwa bidang yang sudah ada di server hilang (minat, poin, jersey). Selain itu, nilai untuk “PPG” dalam peta “statistik” sedang diedit, nilai baru “rpg” sedang ditambahkan, dan “apg” dihilangkan. Automerge melestarikan bidang yang telah dihilangkan (catatan: jika bidang dimaksudkan untuk dihapus, maka permintaan harus dicoba lagi dengan versi yang cocok), dan sehingga mereka tidak akan hilang. Ini juga akan menerapkan aturan yang sama untuk bidang dalam peta dan oleh karena itu perubahan ke “PPG” akan ditolak sedangkan “apg” dipertahankan dan “rpg”, bidang baru”, ditambahkan pada. Item yang dihasilkan disimpan di server sekarang akan muncul sebagai:

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "interests" : ["breakfast", "lunch", "dinner", "brunch"] # underlying data type is a
 Set
  "points": [24, 30, 27, 30, 35] # underlying data type is a List
  "stats": {
      "ppg": "35.4",
      "apg": "6.3",
      "rpg": "6.9"
  }
  "_version" : 9
}
```

Lambda

Opsi Penyelesaian Konflik:

- `RESOLVE`: Ganti item yang ada dengan item baru yang disediakan dalam payload respons. Anda hanya dapat mencoba kembali operasi yang sama di satu item dalam satu waktu. Saat ini didukung untuk DynamoDB`PutItem` &`UpdateItem`.
- `REJECT`: Tolak mutasi dan kembalikan kesalahan dengan item yang ada dalam respons GraphQL. Saat ini didukung untuk DynamoDB`PutItem`,`UpdateItem`, &`DeleteItem`.
- `REMOVE`: Hapus item yang sudah ada. Saat ini didukung untuk DynamoDB`DeleteItem`.

Permintaan Doa Lambda

ResolverAWS AppSync DynamoDB memanggil fungsi Lambda yang ditentukan dalam`LambdaConflictHandlerArn`. Menggunakan sama`service-role-arn` dikonfigurasi pada sumber data. Payload doa memiliki struktur berikut:

```
{
    "newItem": { ... },
    "existingItem": {... },
    "arguments": { ... },
    "resolver": { ... },
    "identity": { ... }
}
```

Bidang ditentukan sebagai berikut:

`newItem`

Item pratinjau, jika mutasi berhasil.

`existingItem`

Item yang saat ini berada di tabel DynamoDB.

`arguments`

Argumen dari mutasi GraphQL.

`resolver`

Informasi tentangAWS AppSync resolver.

`identity`

Informasi tentang penelepon. Bidang ini diatur ke null, jika akses dengan kunci API.

Contoh muatan:

```
{
    "newItem": {
        "id": "1",
        "author": "Jeff",
        "title": "Foo Bar",
        "rating": 5,
        "comments": ["hello world"],
```

```
    },
    "existingItem": {
        "id": "1",
        "author": "Foo",
        "rating": 5,
        "comments": ["old comment"]
    },
    "arguments": {
        "id": "1",
        "author": "Jeff",
        "title": "Foo Bar",
        "comments": ["hello world"]
    },
    "resolver": {
        "tableName": "post-table",
        "awsRegion": "us-west-2",
        "parentType": "Mutation",
        "field": "updatePost"
    },
    "identity": {
         "accountId": "123456789012",
         "sourceIp": "x.x.x.x",
         "username": "AIDAAAAAAAAAAAAAAAAA",
         "userArn": "arn:aws:iam::123456789012:user/appsync"
    }
}
```

Respon Doa Lambda

UntukPutItem dan resolusiUpdateItem konflik

RESOLVEmutasi. Respons harus dalam format berikut.

```
{
    "action": "RESOLVE",
    "item": { ... }
}
```

itemBidang mewakili objek yang akan digunakan untuk mengganti item yang ada di sumber data yang mendasarinya. Kunci primer dan metadata sinkronisasi akan diabaikan jika disertakanitem.

REJECTmutasi. Respons harus dalam format berikut.

```
{
    "action": "REJECT"
}
```

Untuk resolusi`DeleteItem` konflik

`REMOVE`item. Respons harus dalam format berikut.

```
{
    "action": "REMOVE"
}
```

`REJECT`mutasi. Respons harus dalam format berikut.

```
{
    "action": "REJECT"
}
```

Contoh fungsi Lambda di bawah memeriksa siapa yang membuat panggilan dan nama resolver. Jika dibuat oleh`jeffTheAdmin`,`REMOVE` objek untuk DeletePost resolver atau`RESOLVE` konflik dengan item baru untuk resolver Update/Put. Jika tidak, mutasi adalah`REJECT`.

```
exports.handler = async (event, context, callback) => {
    console.log("Event: "+ JSON.stringify(event));

    // Business logic goes here.
    var response;
    if ( event.identity.user == "jeffTheAdmin" ) {
        let resolver = event.resolver.field;

        switch(resolver) {
            case "deletePost":
                response = {
                    "action" : "REMOVE"
                }
                break;

            case "updatePost":
            case "createPost":
                response = {
                    "action" : "RESOLVE",
```

```
                    "item": event.newItem
                }
                break;
            default:
                response = { "action" : "REJECT" };
        }
    } else {
        response = { "action" : "REJECT" };
    }

    console.log("Response: "+ JSON.stringify(response));
    return response;
}
```

Kesalahan

`ConflictUnhandled`

Deteksi konflik menemukan ketidakcocokan versi dan pengendali konflik menolak mutasi.

Contoh: Resolusi konflik dengan pengendali konflik Konkurensi Optimis. Atau, Lambda konflik handler kembali dengan`REJECT`.

`ConflictError`

Terjadi kesalahan internal saat mencoba untuk menyelesaikan konflik.

Contoh: Penangan konflik Lambda mengembalikan respons yang salah. Atau, tidak dapat memanggil Lambda konflik handler karena sumber daya yang`LambdaConflictHandlerArn` disediakan tidak ditemukan.

`MaxConflicts`

Upaya coba ulang maks dicapai untuk penyelesaian konflik.

Contoh: Terlalu banyak permintaan bersamaan pada objek yang sama. Sebelum konflik diselesaikan, objek diperbarui ke versi baru oleh klien lain.

`BadRequest`

Klien mencoba memperbarui bidang metadata (`_version`,`_ttl`,`_lastChangedAt`,`_deleted`).

Contoh: Klien mencoba memperbarui _version objek dengan mutasi pembaruan.

`DeltaSyncWriteError`

Gagal menulis rekaman sinkronisasi delta.

Contoh: Mutasi berhasil, tetapi terjadi kesalahan internal saat mencoba menulis ke tabel sinkronisasi delta.

`InternalFailure`

Terjadi eror internal.

CloudWatch Log

JikaAWS AppSync API telah mengaktifkan CloudWatch Log dengan pengaturan logging disetel ke Log Tingkat Lapangan`enabled` dan tingkat log untuk Log Tingkat Lapangan yang disetel`ALL`, makaAWS AppSync akan memancarkan informasi Deteksi Konflik dan Resolusi ke grup log. Untuk informasi tentang format pesan log, lihat dokumentasi untuk Deteksi Konflik dan Pencatatan Sinkronisasi.

Sinkronisasi Operasi

Sumber data berversi mendukung`Sync` operasi yang memungkinkan Anda mengambil semua hasil dari tabel DynamoDB dan kemudian hanya menerima data yang diubah sejak kueri terakhir Anda (pembaruan delta). KetikaAWS AppSync menerima permintaan untuk`Sync` operasi, ia menggunakan bidang yang ditentukan dalam permintaan untuk menentukan apakah tabel Dasar atau tabel Delta harus diakses.

- Jika`lastSync` bidang tidak ditentukan,`Scan` pada tabel Base dilakukan.
- Jika`lastSync` bidang ditentukan, tetapi nilai sebelum`current moment - DeltaSyncTTL`,`Scan` pada tabel Base dilakukan.
- Jika`lastSync` bidang ditentukan, dan nilai pada atau setelah`current moment - DeltaSyncTTL`,`Query` pada tabel Delta dilakukan.

AWS AppSync mengembalikan`startedAt` bidang ke template pemetaan respon untuk semua`Sync` operasi. `startedAt`Bidang adalah momen, dalam milidetik epoch, ketika`Sync` operasi dimulai yang dapat Anda simpan secara lokal dan gunakan dalam permintaan lain. Jika token pagination disertakan dalam permintaan, nilai ini akan sama dengan yang dikembalikan oleh permintaan untuk halaman pertama hasil.

Untuk informasi tentang format templateSync pemetaan, lihat referensi template pemetaan.

Pemantauan dan pencatatan

Untuk memantau AWS AppSync GraphQL API dan membantu men-debug masalah yang terkait dengan permintaan, Anda dapat mengaktifkan logging ke Amazon Logs. CloudWatch

Penyiapan dan konfigurasi

Untuk mengaktifkan logging otomatis pada GraphQL API, gunakan konsol. AWS AppSync

1. Masuk ke AWS Management Console dan buka AppSynckonsol.
2. Pada halaman API, pilih nama GraphQL API.
3. Di beranda API Anda, di panel navigasi, pilih Pengaturan.
4. Di bawah Logging, lakukan hal berikut:
   a. Aktifkan Aktifkan Log.
   b. Untuk pencatatan tingkat permintaan terperinci, pilih kotak centang di bawah Sertakan konten verbose. (opsional)
   c. Di bawah Level log penyelesai bidang, pilih level logging tingkat bidang pilihan Anda (Tidak Ada, Kesalahan, atau Semua). (opsional)
   d. Di bawah Buat atau gunakan peran yang ada, pilih Peran baru untuk membuat baru AWS Identity and Access Management (IAM) yang memungkinkan AWS AppSync untuk menulis log. CloudWatch Atau, pilih Peran yang ada untuk memilih Nama Sumber Daya Amazon (ARN) peran IAM yang ada di akun Anda. AWS
5. Pilih Simpan.

Konfigurasi peran IAM manual

Jika Anda memilih untuk menggunakan peran IAM yang ada, peran tersebut harus memberikan izin AWS AppSync yang diperlukan untuk menulis log. CloudWatch Untuk mengonfigurasi ini secara manual, Anda harus menyediakan ARN peran layanan sehingga AWS AppSync dapat mengambil peran saat menulis log.

Di konsol IAM, buat kebijakan baru dengan nama AWSAppSyncPushToCloudWatchLogsPolicy yang memiliki definisi berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

Selanjutnya, buat peran baru dengan nama AWSAppSyncPushToCloudWatchLogsRole, dan lampirkan kebijakan yang baru dibuat ke peran tersebut. Edit hubungan kepercayaan untuk peran ini menjadi berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "appsync.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}
```

Salin peran ARN dan gunakan saat menyiapkan logging untuk AWS AppSync GraphQL API.

CloudWatch metrik

Anda dapat menggunakan CloudWatch metrik untuk memantau dan memberikan peringatan tentang peristiwa tertentu yang dapat menghasilkan kode status HTTP atau dari latensi. Metrik berikut dipancarkan:

Daftar metrik

`4XXError`

Kesalahan yang dihasilkan dari permintaan yang tidak valid karena konfigurasi klien yang salah. Biasanya, kesalahan ini terjadi di mana saja di luar pemrosesan GraphQL. Misalnya, kesalahan ini dapat terjadi ketika permintaan menyertakan payload JSON yang salah atau kueri yang salah, saat layanan dibatasi, atau ketika pengaturan otorisasi salah dikonfigurasi.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total kejadian kesalahan ini.

`5XXError`

Kesalahan yang ditemui selama menjalankan kueri GraphQL. Misalnya, ini dapat terjadi saat menjalankan kueri untuk skema kosong atau salah. Ini juga dapat terjadi ketika ID atau AWS Wilayah kumpulan pengguna Amazon Cognito tidak valid. Atau, ini juga bisa terjadi AWS AppSync jika mengalami masalah selama pemrosesan permintaan.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total kejadian kesalahan ini.

`Latency`

Waktu antara ketika AWS AppSync menerima permintaan dari klien dan ketika mengembalikan respons ke klien. Ini tidak termasuk latensi jaringan yang ditemui untuk respons untuk mencapai perangkat akhir.

Satuan: Millidetik. Gunakan statistik Rata-rata untuk mengevaluasi latensi yang diharapkan.

`Requests`

Jumlah permintaan (kueri+mutasi) yang telah diproses oleh semua API di akun Anda, menurut Wilayah.

Satuan: Hitung. Jumlah semua permintaan yang diproses di Wilayah tertentu.

`TokensConsumed`

Token dialokasikan `Requests` berdasarkan jumlah sumber daya (waktu pemrosesan dan memori yang digunakan) yang `Request` dikonsumsi. Biasanya, masing-masing `Request` mengkonsumsi satu token. Namun, a `Request` yang mengkonsumsi sejumlah besar sumber daya dialokasikan token tambahan sesuai kebutuhan.

Satuan: Hitung. Jumlah token yang dialokasikan untuk permintaan yang diproses di Wilayah tertentu.

`NetworkBandwidthOutAllowanceExceeded`

> **Note**
>
> Di AWS AppSync konsol, pada halaman pengaturan cache, opsi Metrik Kesehatan Cache memungkinkan Anda mengaktifkan metrik kesehatan terkait cache ini.

Paket jaringan turun karena throughput melebihi batas bandwidth agregat. Ini berguna untuk mendiagnosis kemacetan dalam konfigurasi cache. Data direkam untuk API tertentu dengan menentukan `API_Id` dalam `appsyncCacheNetworkBandwidthOutAllowanceExceeded` metrik.

Satuan: Hitung. Jumlah paket turun setelah melebihi batas bandwidth untuk API yang ditentukan oleh ID.

`EngineCPUUtilization`

> **Note**
>
> Di AWS AppSync konsol, pada halaman pengaturan cache, opsi Metrik Kesehatan Cache memungkinkan Anda mengaktifkan metrik kesehatan terkait cache ini.

Pemanfaatan CPU (persentase) dialokasikan untuk proses Redis. Ini berguna untuk mendiagnosis kemacetan dalam konfigurasi cache. Data direkam untuk API tertentu dengan menentukan `API_Id` dalam `appsyncCacheEngineCPUUtilization` metrik.

Satuan: Persen. Persentase CPU yang saat ini digunakan oleh proses Redis untuk API yang ditentukan oleh ID.

Langganan waktu nyata

Semua metrik dipancarkan dalam satu dimensi: GraphqlaPiid. Ini berarti bahwa semua metrik digabungkan dengan ID API GraphQL. Metrik berikut terkait dengan langganan GraphQL di atas murni: WebSockets

Daftar metrik

`ConnectRequests`

Jumlah permintaan WebSocket koneksi yang dibuat untuk AWS AppSync, termasuk upaya yang berhasil dan tidak berhasil.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total permintaan koneksi.

`ConnectSuccess`

Jumlah WebSocket koneksi yang berhasil ke AWS AppSync. Dimungkinkan untuk memiliki koneksi tanpa langganan.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan kejadian total dari koneksi yang berhasil.

`ConnectClientError`

Jumlah WebSocket koneksi yang ditolak oleh AWS AppSync karena kesalahan sisi klien. Ini dapat menyiratkan bahwa layanan dibatasi atau bahwa pengaturan otorisasi salah dikonfigurasi.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan kejadian total kesalahan koneksi sisi klien.

`ConnectServerError`

Jumlah kesalahan yang berasal dari AWS AppSync saat memproses koneksi. Ini biasanya terjadi ketika masalah sisi server yang tidak terduga terjadi.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan kejadian total kesalahan koneksi sisi server.

`DisconnectSuccess`

Jumlah WebSocket pemutusan yang berhasil dari AWS AppSync.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan kejadian total dari pemutusan yang berhasil.

`DisconnectClientError`

Jumlah kesalahan klien yang berasal dari AWS AppSync saat memutuskan koneksi WebSocket.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan kejadian total dari kesalahan pemutusan.

`DisconnectServerError`

Jumlah kesalahan server yang berasal dari AWS AppSync saat memutuskan koneksi WebSocket.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan kejadian total dari kesalahan pemutusan.

`SubscribeSuccess`

Jumlah langganan yang berhasil didaftarkan AWS AppSync melalui WebSocket. Dimungkinkan untuk memiliki koneksi tanpa langganan, tetapi tidak mungkin memiliki langganan tanpa koneksi.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total kemunculan langganan yang berhasil.

`SubscribeClientError`

Jumlah langganan yang ditolak oleh AWS AppSync karena kesalahan sisi klien. Ini dapat terjadi ketika payload JSON salah, layanan dibatasi, atau pengaturan otorisasi salah dikonfigurasi.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total kejadian kesalahan langganan sisi klien.

`SubscribeServerError`

Jumlah kesalahan yang berasal dari AWS AppSync saat memproses langganan. Ini biasanya terjadi ketika masalah sisi server yang tidak terduga terjadi.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total kejadian kesalahan langganan sisi server.

`UnsubscribeSuccess`

Jumlah permintaan berhenti berlangganan yang berhasil diproses.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total kemunculan permintaan berhenti berlangganan yang berhasil.

`UnsubscribeClientError`

Jumlah permintaan berhenti berlangganan yang ditolak oleh AWS AppSync karena kesalahan sisi klien.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total kejadian kesalahan permintaan berhenti berlangganan sisi klien.

`UnsubscribeServerError`

Jumlah kesalahan yang berasal dari AWS AppSync saat memproses permintaan berhenti berlangganan. Ini biasanya terjadi ketika masalah sisi server yang tidak terduga terjadi.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total kejadian kesalahan permintaan berhenti berlangganan sisi server.

`PublishDataMessageSuccess`

Jumlah pesan acara berlangganan yang berhasil dipublikasikan.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total pesan acara berlangganan yang berhasil dipublikasikan.

`PublishDataMessageClientError`

Jumlah pesan acara langganan yang gagal dipublikasikan karena kesalahan sisi klien.

`Unit`: Hitung. Gunakan statistik Jumlah untuk mendapatkan total kejadian kesalahan peristiwa berlangganan penerbitan sisi klien.

`PublishDataMessageServerError`

Jumlah kesalahan yang berasal dari AWS AppSync saat menerbitkan pesan acara berlangganan. Ini biasanya terjadi ketika masalah sisi server yang tidak terduga terjadi.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total kejadian kesalahan peristiwa berlangganan penerbitan sisi server.

`PublishDataMessageSize`

Ukuran pesan acara berlangganan yang diterbitkan.

Satuan: Byte.

`ActiveConnections`

Jumlah WebSocket koneksi bersamaan dari klien ke AWS AppSync dalam 1 menit.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total koneksi yang terbuka.

`ActiveSubscriptions`

Jumlah langganan bersamaan dari klien dalam 1 menit.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan total langganan aktif.

`ConnectionDuration`

Jumlah waktu koneksi tetap terbuka.

Satuan: Milidetik. Gunakan statistik Rata-rata untuk mengevaluasi durasi koneksi.

`OutboundMessages`

Jumlah pesan terukur berhasil dipublikasikan. Satu pesan terukur sama dengan 5 kB data yang dikirimkan.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total pesan terukur yang berhasil dipublikasikan.

`InboundMessageSuccess`

Jumlah pesan masuk berhasil diproses. Setiap jenis langganan yang dipanggil oleh mutasi menghasilkan satu pesan masuk.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total pesan masuk yang berhasil diproses.

`InboundMessageError`

Jumlah pesan masuk yang gagal diproses karena permintaan API tidak valid, seperti melebihi batas ukuran payload langganan 240 kB.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total pesan masuk dengan kegagalan pemrosesan terkait API.

`InboundMessageFailure`

Jumlah pesan masuk yang gagal diproses karena kesalahan dari AWS AppSync.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total pesan masuk dengan kegagalan pemrosesan AWS AppSync terkait.

`InboundMessageDelayed`

Jumlah pesan masuk yang tertunda. Pesan masuk dapat ditunda ketika kuota rasio pesan masuk atau kuota rasio pesan keluar dilanggar.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total pesan masuk yang tertunda.

InboundMessageDropped

Jumlah pesan masuk yang jatuh. Pesan masuk dapat dihapus ketika kuota rasio pesan masuk atau kuota rasio pesan keluar dilanggar.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total pesan masuk yang dijatuhkan.

InvalidationSuccess

Jumlah langganan yang berhasil dibatalkan (berhenti berlangganan) dengan mutasi dengan. `$extensions.invalidateSubscriptions()`

Satuan: Hitung. Gunakan statistik Jumlah untuk mengambil jumlah total langganan yang berhasil berhenti berlangganan.

InvalidationRequestSuccess

Jumlah permintaan pembatalan berhasil diproses.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total permintaan pembatalan yang berhasil diproses.

InvalidationRequestError

Jumlah permintaan pembatalan yang gagal diproses karena permintaan API tidak valid.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total permintaan pembatalan dengan kegagalan pemrosesan terkait API.

InvalidationRequestFailure

Jumlah permintaan pembatalan yang gagal diproses karena kesalahan dari. AWS AppSync

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total permintaan pembatalan dengan kegagalan pemrosesan AWS AppSync terkait.

InvalidationRequestDropped

Jumlah permintaan pembatalan turun ketika kuota permintaan pembatalan terlampaui.

Satuan: Hitung. Gunakan statistik Jumlah untuk mendapatkan jumlah total permintaan pembatalan yang dijatuhkan.

Membandingkan pesan masuk dan keluar

Saat mutasi dijalankan, bidang langganan dengan direktif @aws_subscribe untuk mutasi tersebut dipanggil. Setiap pemanggilan langganan menghasilkan satu pesan masuk. Misalnya, jika dua bidang langganan menentukan mutasi yang sama di @aws_subscribe, maka dua pesan masuk dihasilkan saat mutasi itu dipanggil.

Satu pesan keluar sama dengan 5 kB data yang dikirimkan ke WebSocket klien. Misalnya, mengirim 15 kB data ke 10 klien menghasilkan 30 pesan keluar (15 kB* 10 klien/ 5 kB per pesan = 30 pesan).

Anda dapat meminta peningkatan kuota untuk pesan masuk atau keluar. Untuk informasi selengkapnya, lihat AWS AppSync titik akhir dan kuota dalam panduan Referensi AWS Umum dan petunjuk untuk Meminta peningkatan kuota dalam Panduan Pengguna Service Quotas.

Metrik yang disempurnakan

Metrik yang disempurnakan memancarkan data terperinci tentang penggunaan dan kinerja API seperti jumlah AWS AppSync permintaan dan kesalahan, latensi, dan hits/misses cache. Semua data metrik yang disempurnakan dikirim ke CloudWatch akun Anda, dan Anda dapat mengonfigurasi jenis data yang akan dikirim.

> **Note**
>
> Biaya tambahan dikenakan saat menggunakan metrik yang disempurnakan. Untuk informasi selengkapnya, lihat tingkat penetapan harga pemantauan terperinci di CloudWatchharga Amazon.

Metrik ini dapat ditemukan di berbagai halaman pengaturan di AWS AppSync konsol. Pada halaman pengaturan API, bagian Metrik yang Ditingkatkan memungkinkan Anda mengaktifkan atau menonaktifkan item berikut:

1. Perilaku metrik penyelesai: Opsi ini mengontrol cara metrik tambahan untuk resolver dikumpulkan. Anda dapat memilih untuk mengaktifkan metrik resolver permintaan lengkap (metrik diaktifkan untuk semua resolver dalam permintaan) atau metrik per-resolver (metrik hanya diaktifkan untuk resolver yang konfigurasi disetel ke diaktifkan). Pilihan berikut tersedia:

Metrik	Dimensi Metrik	Nama Metrik	Satuan	Deskripsi

Kesalahan GraphQL per resolver	API_id, Penyelesai	GraphQLError	Hitungan	Jumlah kesalahan GraphQL yang terjadi per resolver.
Permintaan per resolver	API_id, Penyelesai	Permintaan	Hitungan	Jumlah pemanggil an yang terjadi selama permintaa n. Ini dicatat berdasarkan per-resolver.
Latensi per resolver	API_id, Penyelesai	Latensi	Millidetik	Waktu untuk menyelesaikan doa resolver. Latensi diukur dalam milidetik dan dicatat berdasarkan per-resolver.
Cache hits per resolver	API_id, Penyelesai	CacheHit	Hitungan	Jumlah cache yang muncul selama permintaa n. Ini hanya akan dipancark an jika cache digunakan. Hit cache direkam berdasarkan per-resolver.

Cache meleset per resolver	API_id, Penyelesai	CacheMiss	Hitungan	Jumlah cache hilang selama permintaan. Ini hanya akan dipancarkan jika cache digunakan. Kesalahan cache dicatat berdasarkan per-resolver.

2. Perilaku metrik sumber data: Opsi ini mengontrol cara metrik tambahan untuk sumber data dikumpulkan. Anda dapat memilih untuk mengaktifkan metrik sumber data permintaan lengkap (metrik diaktifkan untuk semua sumber data dalam permintaan) atau metrik sumber per data (metrik hanya diaktifkan untuk sumber data yang konfigurasi disetel ke diaktifkan). Pilihan berikut tersedia:

Metrik	Dimensi Metrik	Nama Metrik	Satuan	Deskripsi
Permintaan per sumber data	API_id, Sumber Data	Permintaan	Hitungan	Jumlah pemanggilan yang terjadi selama permintaan. Permintaan dicatat berdasarkan sumber per data. Jika permintaan penuh diaktifkan, setiap sumber data akan memiliki

				entri sendiri CloudWatch.
Latensi per sumber data	API_id, Sumber Data	Latensi	Millidetik	Waktu untuk menyelesaikan pemanggilan sumber data. Latensi dicatat berdasarkan sumber per data.
Kesalahan per sumber data	API_id, Sumber Data	GraphQLError	Hitungan	Jumlah kesalahan yang terjadi selama pemanggilan sumber data.

3. Metrik operasi: Mengaktifkan metrik tingkat operasi GraphQL.

Metrik	Dimensi Metrik	Nama Metrik	Satuan	Deskripsi
Permintaan per operasi	API_id, Operasi	Permintaan	Hitungan	Berapa kali operasi GraphQL tertentu dipanggil.
Kesalahan GraphQL per operasi	API_id, Operasi	GraphQLError	Hitungan	Jumlah kesalahan GraphQL yang terjadi selama operasi GraphQL tertentu.

CloudWatch log

Anda dapat mengonfigurasi dua jenis logging pada GraphQL API baru atau yang sudah ada: tingkat permintaan dan tingkat bidang.

Log tingkat permintaan

Ketika pencatatan tingkat permintaan (Sertakan konten verbose) dikonfigurasi, informasi berikut dicatat:

- Jumlah token yang dikonsumsi
- Header HTTP permintaan dan respons
- Query GraphQL yang berjalan dalam permintaan
- Ringkasan operasi keseluruhan
- Langganan GraphQL baru dan yang sudah ada yang terdaftar

Log tingkat lapangan

Ketika logging tingkat lapangan dikonfigurasi, informasi berikut dicatat:

- Pemetaan permintaan yang dihasilkan dengan sumber dan argumen untuk setiap bidang
- Pemetaan respons yang ditransformasikan untuk setiap bidang, yang mencakup data sebagai hasil penyelesaian bidang tersebut
- Menelusuri informasi untuk setiap bidang

Jika Anda mengaktifkan logging, AWS AppSync mengelola CloudWatch Log. Prosesnya termasuk membuat grup log dan aliran log, dan melaporkan ke aliran log dengan log ini.

Saat Anda mengaktifkan logging pada GraphQL API dan membuat permintaan AWS AppSync , buat grup log dan aliran log di bawah grup log. Grup log diberi nama mengikuti `/aws/appsync/apis/{graphql_api_id}` format. Dalam setiap grup log, log dibagi lagi menjadi aliran log. Ini diurutkan berdasarkan Waktu Acara Terakhir karena data yang dicatat dilaporkan.

Setiap peristiwa log ditandai dengan x-amzn- RequestId dari permintaan itu. Ini membantu Anda memfilter peristiwa log CloudWatch untuk mendapatkan semua informasi yang dicatat tentang permintaan itu. Anda bisa mendapatkan dari header respons RequestId dari setiap permintaan GraphQL AWS AppSync .

Pencatatan tingkat bidang dikonfigurasi dengan level log berikut:

- Tidak ada - Tidak ada log tingkat lapangan yang ditangkap.
- Kesalahan - Mencatat informasi berikut hanya untuk bidang yang salah:
    - Bagian kesalahan dalam respon server
    - Kesalahan tingkat lapangan
    - Fungsi permintaan/respons yang dihasilkan yang diselesaikan untuk bidang kesalahan
- Semua - Log informasi berikut untuk semua bidang dalam kueri:
    - Informasi penelusuran tingkat lapangan
    - Fungsi permintaan/respons yang dihasilkan yang diselesaikan untuk setiap bidang

Manfaat pemantauan

Anda dapat menggunakan logging dan metrik untuk mengidentifikasi, memecahkan masalah, dan mengoptimalkan kueri GraphQL Anda. Misalnya, ini akan membantu Anda men-debug masalah latensi menggunakan informasi penelusuran yang dicatat untuk setiap bidang dalam kueri. Untuk mendemonstrasikan ini, misalkan Anda menggunakan satu atau lebih resolver yang bersarang dalam kueri GraphQL. Operasi bidang sampel di CloudWatch Log mungkin terlihat mirip dengan yang berikut ini:

```
{
    "path": [
        "singlePost",
        "authors",
        0,
        "name"
    ],
    "parentType": "Post",
    "returnType": "String!",
    "fieldName": "name",
    "startOffset": 416563350,
    "duration": 11247
}
```

Ini mungkin sesuai dengan skema GraphQL, mirip dengan yang berikut:

```
type Post {
  id: ID!
```

```
  name: String!
  authors: [Author]
}

type Author {
  id: ID!
  name: String!
}

type Query {
  singlePost(id:ID!): Post
}
```

Dalam hasil log sebelumnya, path menampilkan satu item dalam data Anda yang dikembalikan dari menjalankan kueri bernama. `singlePost()` Dalam contoh ini, ini mewakili bidang nama pada indeks pertama (0). StartOffset memberikan offset dari awal operasi query GraphQL. Durasi adalah total waktu untuk menyelesaikan bidang. Nilai-nilai ini dapat berguna untuk memecahkan masalah mengapa data dari sumber data tertentu mungkin berjalan lebih lambat dari yang diharapkan, atau jika bidang tertentu memperlambat seluruh kueri. Misalnya, Anda dapat memilih untuk meningkatkan throughput yang disediakan untuk tabel Amazon DynamoDB, atau menghapus bidang tertentu dari kueri yang menyebabkan keseluruhan operasi berkinerja buruk.

Per 8 Mei 2019, AWS AppSync menghasilkan peristiwa log sebagai JSON yang terstruktur sepenuhnya. Ini dapat membantu Anda menggunakan layanan analisis CloudWatch log seperti Wawasan Log dan OpenSearch Layanan Amazon untuk memahami kinerja permintaan GraphQL dan karakteristik penggunaan bidang skema Anda. Misalnya, Anda dapat dengan mudah mengidentifikasi resolver dengan latensi besar yang mungkin menjadi akar penyebab masalah kinerja. Anda juga dapat mengidentifikasi bidang yang paling sering dan paling jarang digunakan dalam skema Anda dan menilai dampak dari menghentikan bidang GraphQL.

Deteksi konflik dan pencatatan sinkronisasi

Jika AWS AppSync API telah login ke CloudWatch Log yang dikonfigurasi dengan tingkat log penyelesai bidang yang disetel ke Semua, maka akan AWS AppSync memancarkan deteksi konflik dan informasi resolusi ke grup log. Ini memberikan wawasan terperinci tentang bagaimana AWS AppSync API merespons konflik. Untuk membantu Anda menafsirkan respons, informasi berikut disediakan di log:

Daftar metrik

`conflictType`

Merinci apakah konflik terjadi karena ketidakcocokan versi atau kondisi yang disediakan pelanggan.

`conflictHandlerConfigured`

Menyatakan penangan konflik yang dikonfigurasi pada resolver pada saat permintaan.

`message`

Memberikan informasi tentang bagaimana konflik terdeteksi dan diselesaikan.

`syncAttempt`

Jumlah percobaan server mencoba untuk menyinkronkan data sebelum akhirnya menolak permintaan.

`data`

Jika pengendali konflik dikonfigurasi`Automerge`, bidang ini diisi untuk menunjukkan keputusan apa yang `Automerge` diambil untuk setiap bidang. Tindakan yang diberikan dapat berupa:

- DITOLAK - Ketika `Automerge` menolak nilai bidang masuk yang mendukung nilai di server.
- TAMBAH - Ketika `Automerge` menambahkan pada bidang masuk karena tidak ada nilai yang sudah ada sebelumnya di server.
- DITAMBAHKAN - Ketika `Automerge` menambahkan nilai masuk ke nilai-nilai untuk Daftar yang ada di server.
- MERGED - Ketika `Automerge` menggabungkan nilai-nilai yang masuk ke nilai-nilai untuk Set yang ada di server.

Menggunakan jumlah token untuk mengoptimalkan permintaan Anda

Permintaan yang mengkonsumsi kurang dari atau sama dengan 1.500 Kb-detik memori dan waktu vCPU dialokasikan satu token. Permintaan dengan konsumsi sumber daya lebih dari 1.500 Kb-detik menerima token tambahan. Misalnya, jika permintaan menghabiskan 3.350 KB-detik, AWS AppSync mengalokasikan tiga token (dibulatkan ke nilai integer berikutnya) ke permintaan. Secara default, AWS AppSync mengalokasikan maksimum 2.000 token permintaan per detik ke API di akun Anda, per AWS Wilayah. Jika API Anda masing-masing menggunakan rata-rata dua token per detik, Anda akan dibatasi hingga 1.000 permintaan per detik. Jika Anda membutuhkan lebih banyak token per

detik dari jumlah yang dialokasikan, Anda dapat mengajukan permintaan untuk meningkatkan kuota default untuk tarif token permintaan. Untuk informasi selengkapnya, lihat AWS AppSynctitik akhir dan kuota dalam Referensi Umum AWS panduan dan Meminta peningkatan kuota dalam Panduan Pengguna Service Quotas.

Jumlah token per permintaan yang tinggi dapat menunjukkan bahwa ada peluang untuk mengoptimalkan permintaan Anda dan meningkatkan kinerja API Anda. Faktor-faktor yang dapat meningkatkan jumlah token per permintaan Anda meliputi:

- Ukuran dan kompleksitas skema GraphQL Anda.
- Kompleksitas template pemetaan permintaan dan respons.
- Jumlah pemanggilan resolver per permintaan.
- Jumlah data yang dikembalikan dari resolver.
- Latensi sumber data hilir.
- Desain skema dan kueri yang memerlukan panggilan sumber data berturut-turut (sebagai lawan dari panggilan paralel atau batch).
- Konfigurasi logging, terutama konten log tingkat lapangan dan verbose.

> **Note**
>
> Selain AWS AppSync metrik dan log, klien dapat mengakses jumlah token yang dikonsumsi dalam permintaan melalui header `x-amzn-appsync-TokensConsumed` respons.

Referensi tipe log

RequestSummary

- RequeSTid: Pengidentifikasi unik untuk permintaan.
- GraphQlapiid: ID dari GraphQL API yang membuat permintaan.
- StatusCode: respons kode status HTTP.
- latensi: E nd-to-end latensi permintaan, dalam nanodetik, sebagai bilangan bulat.

```
{
     "logType": "RequestSummary",
```

```
     "requestId": "dbe87af3-c114-4b32-ae79-8af11f3f96f1",
     "graphQLAPIId": "pmo28inf75eepg63qxq4ekoeg4",
     "statusCode": 200,
     "latency": 242000000
}
```

ExecutionSummary

- RequeSTid: Pengidentifikasi unik untuk permintaan.
- GraphQlapiid: ID dari GraphQL API yang membuat permintaan.
- StartTime: Stempel waktu awal pemrosesan GraphQL untuk permintaan, dalam format RFC 3339.
- EndTime: Stempel waktu akhir pemrosesan GraphQL untuk permintaan, dalam format RFC 3339.
- durasi: Total waktu pemrosesan GraphQL yang telah berlalu, dalam nanodetik, sebagai bilangan bulat.
- versi: Versi skema dari. ExecutionSummary
- penguraian:
    - startOffset: Offset awal untuk parsing, dalam nanodetik, relatif terhadap pemanggilan, sebagai bilangan bulat.
    - durasi: Waktu yang dihabiskan parsing, dalam nanodetik, sebagai bilangan bulat.
- validasi:
    - startOffset: Offset awal untuk validasi, dalam nanodetik, relatif terhadap pemanggilan, sebagai bilangan bulat.
    - durasi: Waktu yang dihabiskan untuk melakukan validasi, dalam nanodetik, sebagai bilangan bulat.

```
{
    "duration": 217406145,
    "logType": "ExecutionSummary",
    "requestId": "dbe87af3-c114-4b32-ae79-8af11f3f96f1",
    "startTime": "2019-01-01T06:06:18.956Z",
    "endTime": "2019-01-01T06:06:19.174Z",
    "parsing": {
        "startOffset": 49033,
        "duration": 34784
    },
    "version": 1,
```

```
    "validation": {
        "startOffset": 129048,
        "duration": 69126
    },
    "graphQLAPIId": "pmo28inf75eepg63qxq4ekoeg4"
}
```

Pelacakan

- RequeSTid: Pengidentifikasi unik untuk permintaan.
- GraphQlapiid: ID dari GraphQL API yang membuat permintaan.
- StartOffset: Offset awal untuk resolusi bidang, dalam nanodetik, relatif terhadap pemanggilan, sebagai bilangan bulat.
- durasi: Waktu yang dihabiskan untuk menyelesaikan bidang, dalam nanodetik, sebagai bilangan bulat.
- fieldName: Nama bidang yang sedang diselesaikan.
- ParentType: Jenis induk dari bidang yang sedang diselesaikan.
- ReturnType: Jenis kembali bidang yang sedang diselesaikan.
- path: Daftar segmen jalur, dimulai dari akar respons dan diakhiri dengan bidang yang diselesaikan.
- resolverARN: ARN dari resolver yang digunakan untuk resolusi lapangan. Mungkin tidak ada di bidang bersarang.

```
{
    "duration": 216820346,
    "logType": "Tracing",
    "path": [
        "putItem"
    ],
    "fieldName": "putItem",
    "startOffset": 178156,
    "resolverArn": "arn:aws:appsync:us-east-1:111111111111:apis/
pmo28inf75eepg63qxq4ekoeg4/types/Mutation/fields/putItem",
    "requestId": "dbe87af3-c114-4b32-ae79-8af11f3f96f1",
    "parentType": "Mutation",
    "returnType": "Item",
    "graphQLAPIId": "pmo28inf75eepg63qxq4ekoeg4"
}
```

Menganalisis log Anda dengan Wawasan CloudWatch Log

Berikut ini adalah contoh kueri yang dapat Anda jalankan untuk mendapatkan wawasan yang dapat ditindaklanjuti tentang kinerja dan kesehatan operasi GraphQL Anda. Contoh ini tersedia sebagai contoh kueri di konsol Wawasan CloudWatch Log. Di CloudWatchkonsol, pilih Wawasan Log, pilih grup AWS AppSync log untuk GraphQL API Anda, lalu AWS AppSync pilih kueri di bawah Kueri sampel.

Kueri berikut mengembalikan 10 permintaan GraphQL teratas dengan token maksimum yang dikonsumsi:

```
filter @message like "Tokens Consumed"
| parse @message "* Tokens Consumed: *" as requestId, tokens
| sort tokens desc
| display requestId, tokens
| limit 10
```

Kueri berikut mengembalikan 10 resolver teratas dengan latensi maksimum:

```
  fields resolverArn, duration
| filter logType = "Tracing"
| limit 10
| sort duration desc
```

Kueri berikut mengembalikan resolver yang paling sering dipanggil:

```
  fields ispresent(resolverArn) as isRes
| stats count() as invocationCount by resolverArn
| filter isRes and logType = "Tracing"
| limit 10
| sort invocationCount desc
```

Kueri berikut mengembalikan resolver dengan kesalahan terbanyak dalam template pemetaan:

```
  fields ispresent(resolverArn) as isRes
| stats count() as errorCount by resolverArn, logType
| filter isRes and (logType = "RequestMapping" or logType = "ResponseMapping") and
 fieldInError
| limit 10
| sort errorCount desc
```

Kueri berikut mengembalikan statistik latensi resolver:

```
  fields ispresent(resolverArn) as isRes
| stats min(duration), max(duration), avg(duration) as avg_dur by resolverArn
| filter isRes and logType = "Tracing"
| limit 10
| sort avg_dur desc
```

Kueri berikut mengembalikan statistik latensi bidang:

```
  stats min(duration), max(duration), avg(duration) as avg_dur
  by concat(parentType, '/', fieldName) as fieldKey
| filter logType = "Tracing"
| limit 10
| sort avg_dur desc
```

Hasil kueri Wawasan CloudWatch Log dapat diekspor ke dasbor. CloudWatch

Analisis log Anda dengan OpenSearch Layanan

Anda dapat mencari, menganalisis, dan memvisualisasikan AWS AppSync log Anda dengan Amazon OpenSearch Service untuk mengidentifikasi kemacetan kinerja dan akar penyebab masalah operasional. Anda dapat mengidentifikasi resolver dengan latensi dan kesalahan maksimum. Selain itu, Anda dapat menggunakan OpenSearch Dasbor untuk membuat dasbor dengan visualisasi yang kuat. OpenSearch Dasbor adalah alat visualisasi dan eksplorasi data sumber terbuka yang tersedia di Layanan. OpenSearch Menggunakan OpenSearch Dasbor, Anda dapat terus memantau kinerja dan kesehatan operasi GraphQL Anda. Misalnya, Anda dapat membuat dasbor untuk memvisualisasikan latensi P90 dari permintaan GraphQL Anda dan menelusuri latensi P90 dari setiap resolver.

Saat menggunakan OpenSearch Layanan, gunakan “cwl*” sebagai pola filter untuk mencari indeks. OpenSearch OpenSearch Layanan mengindeks log yang dialirkan dari CloudWatch Log dengan awalan “cwl -”. Untuk membedakan log AWS AppSync API dari CloudWatch log lain yang dikirim ke OpenSearch Layanan, sebaiknya tambahkan ekspresi filter tambahan `graphQLAPIID.keyword=`*`YourGraphQLAPIID`* ke penelusuran Anda.

Migrasi format log

Peristiwa log yang AWS AppSync dihasilkan pada atau setelah 8 Mei 2019 diformat sebagai JSON yang sepenuhnya terstruktur. Untuk menganalisis permintaan GraphQL sebelum 8 Mei 2019, Anda

dapat memigrasikan log lama ke JSON yang sepenuhnya terstruktur menggunakan skrip yang tersedia di Sampel. GitHub Jika Anda perlu menggunakan format log sebelum 8 Mei 2019, buat tiket dukungan dengan pengaturan berikut: setel Jenis ke Manajemen Akun, lalu setel Kategori ke Pertanyaan Akun Umum.

Anda juga dapat menggunakan filter metrik CloudWatch untuk mengubah data log menjadi CloudWatch metrik numerik, sehingga Anda dapat membuat grafik atau mengatur alarm pada mereka.

Menelusuri denganAWS X-Ray

Anda dapat menggunakanAWS X-Rayuntuk melacak permintaan karena mereka dieksekusi diAWSAppSync. Anda dapat menggunakan X-Ray denganAWSAppSync di semuaAWSDaerah di mana X-Ray tersedia. X-Ray memberi Anda gambaran rinci tentang seluruh permintaan GraphQL. Hal ini memungkinkan Anda menganalisis latensi di API Anda dan resolver dan sumber data yang mendasarinya. Anda dapat menggunakan peta layanan X-Ray untuk melihat latensi permintaan, termasuk layanan AWS yang terintegrasi dengan X-Ray. Anda juga dapat mengonfigurasi aturan pengambilan sampel untuk memberi tahu X-Ray mengenai permintaan yang dicatat, tingkat pengambilan sampel, sesuai kriteria yang Anda tentukan.

Untuk informasi selengkapnya tentang pengambilan sampel di X-Ray, lihatMengkonfigurasi Aturan Sampling diAWS X-RayKonsol.

Penyiapan dan Konfigurasi

Anda dapat mengaktifkan penelusuran X-Ray untuk API GraphQL melaluiAWSKonsol AppSync.

1. Masuk keAWSKonsol AppSync.
2. PilihPengaturandari panel navigasi.
3. Di bawahX-Ray, nyalakanAktifkan X-Ray.
4. Pilih Save (Simpan). Pelacakan X-Ray sekarang diaktifkan untuk API Anda.

Jika Anda menggunakanAWS CLIatauAWS CloudFormation, Anda juga dapat mengaktifkan pelacakan X-Ray saat Anda membuat yang baruAWSAppSync API, atau memperbarui yang sudah adaAWSAppSync API, dengan menetapkan`xrayEnabled`properti untuk`true`.

Saat pelacakan X-Ray diaktifkan untukAWSAppSync API, sebuahAWS Identity and Access Management peran yang terhubung dengan layanandibuat secara otomatis di akun Anda dengan izin

yang sesuai. Hal ini mengizinkan AppSync AWS untuk mengirim penelusuran ke X-Ray dengan cara yang aman.

Menelusuri API Anda dengan X-Ray

Pengambilan sampel

Dengan menggunakan aturan pengambilan sampel, Anda dapat mengontrol jumlah data yang Anda catatAWSAppSync, dan dapat mengubah perilaku pengambilan sampel dengan cepat tanpa mengubah atau men-deploy ulang kode Anda. Misalnya, contoh aturan ini meminta ke API GraphQL dengan ID API`3n572shhcpfokwhdnq1ogu59v6`.

- Nama aturan—`test-sample`
- Prioritas—`10`
- Ukuran reservoir—`10`
- Tingkat tetap—`10`
- Nama layanan—*
- Jenis layanan—`AWS::AppSync::GraphQLAPI`
- Metode HTTP—*
- Sumber daya ARN—`arn:aws:appsync:us-west-2:123456789012:apis/3n572shhcpfokwhdnq1ogu59v6`
- Host—*

Memahami Jejak

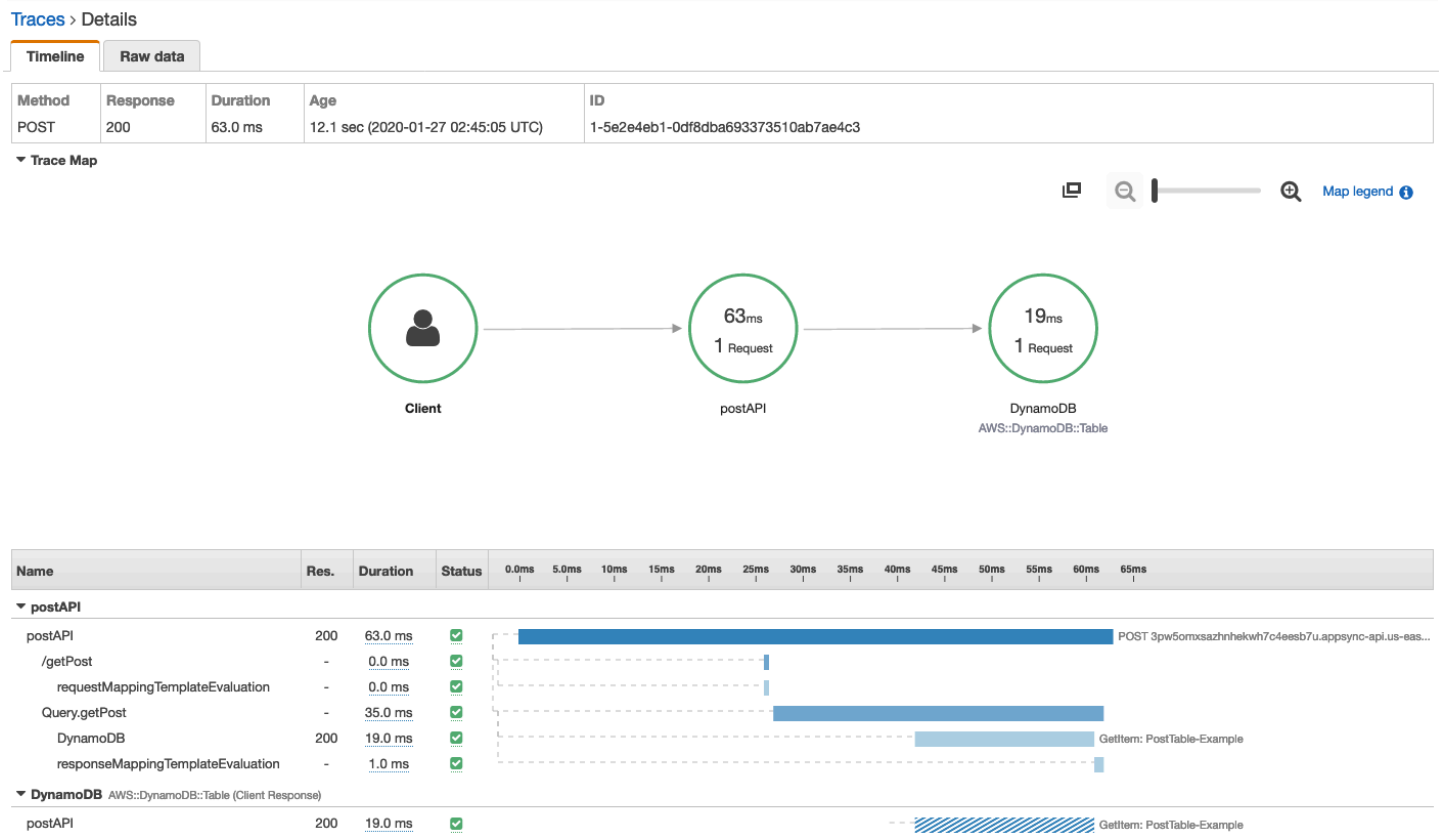
Bila Anda mengaktifkan pelacakan X-Ray untuk API GraphQL, Anda dapat menggunakan halaman detail pelacakan X-Ray untuk memeriksa informasi latensi terperinci tentang permintaan yang dibuat ke API Anda. Contoh berikut menunjukkan tampilan jejak bersama dengan peta layanan untuk permintaan spesifik ini. Permintaan dibuat ke API yang disebut`postAPI`dengan tipe Post, yang datanya terkandung dalam tabel Amazon DynamoDB yang disebut`PostTable-Example`.

Gambar jejak berikut sesuai dengan query GraphQL berikut:

```
query getPost {
    getPost(id: "1") {
```

```
      id
      title
    }
}
```

Resolver untuk`getPost`query menggunakan sumber data DynamoDB yang mendasari. Tampilan jejak berikut menunjukkan panggilan ke DynamoDB, serta latensi berbagai bagian eksekusi query:



- Pada gambar sebelumnya,`/getPost`mewakili path lengkap untuk elemen yang sedang diselesaikan. Dalam kasus ini, karena`getPost`adalah bidang pada akar`Query`ketik, itu muncul langsung setelah akar jalan.
- `requestMappingTemplateEvaluation`mewakili waktu yang dihabiskan olehAWSAppSync mengevaluasi template pemetaan permintaan untuk elemen ini dalam kueri.
- `Query.getPost`mewakili jenis dan bidang (di`Type.field`format). Hal ini dapat berisi beberapa subsegment, tergantung pada struktur API dan permintaan yang ditelusuri.
  - `DynamoDB`mewakili sumber data yang melekat pada resolver ini. Ini berisi latensi untuk panggilan jaringan ke DynamoDB untuk menyelesaikan bidang.
  - `responseMappingTemplateEvaluation`mewakili waktu yang dihabiskan olehAWSAppSync mengevaluasi template pemetaan respon untuk elemen ini dalam kueri.

Bila Anda melihat jejak di X-Ray, Anda bisa mendapatkan informasi kontekstual dan metadata tambahan tentang subsegmen diAWSSegmen AppSync dengan memilih subsegment dan menjelajahi tampilan rinci.

Untuk kueri tertentu yang sangat bersarang atau kompleks, perhatikan bahwa segmen dikirim ke X-Ray olehAWSAppSync bisa lebih besar dari ukuran maksimum yang diizinkan untuk dokumen segmen, seperti yang didefinisikan dalamAWS X-RayDokumen segmen. X-Ray tidak menampilkan segmen yang melebihi batas.

Mencatat panggilan API AWS AppSync menggunakan AWS CloudTrail

AWS AppSync terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau layanan AWS di AWS AppSync. CloudTrailmenangkap semua panggilan API untukAWS AppSyncsebagai peristiwa. Panggilan yang direkam mencakup panggilan dari konsol AWS AppSync tersebut dan dari panggilan kode ke AWS AppSync API. Anda dapat menggunakan informasi yang dikumpulkan olehCloudTrailuntuk menentukan permintaan yang dibuat untukAWS AppSync, alamat IP pemohon, yang membuat permintaan, saat permintaan dibuat, dan detail tambahan.

Anda dapat membuatjejakuntuk memungkinkan pengiriman terus menerusCloudTrailperistiwa ke bucket Amazon Simple Storage Service (Amazon S3), termasuk acara untukAWS AppSync. Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru diCloudTrailkonsol.

> **Important**
>
> Tidak semua tindakan GraphQL saat ini dicatat. AppSynctidak mencatat tindakan Kueri dan Mutasi keCloudTrail.

Untuk informasi selengkapnya tentang CloudTrail, lihat AWS CloudTrailPanduan Pengguna .

Informasi AWS AppSync di CloudTrail

CloudTrail diaktifkan pada akun AWS Anda saat Anda membuat akun tersebut. DiCloudTrailkonsol diRiwayat acara, Anda dapat melihat, mencari, dan mengunduh acara terbaru diAWSakun. Untuk informasi lebih lanjut, lihatMelihat Acara denganCloudTrailRiwayat AcaradiAWS CloudTrailPanduan Pengguna.

Untuk catatan berkelanjutan tentang peristiwa di akun AWS Anda, termasuk peristiwa untuk AWS AppSync, buat jejak. Secara default, saat Anda membuat jejak di dalam konsol tersebut, jejak diterapkan ke semua Wilayah AWS. Jejak mencatat peristiwa dari semua Wilayah di partisi AWS dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi layanan AWS lainnya untuk menganalisis lebih lanjut dan bertindak berdasarkan data kejadian yang dikumpulkan di log CloudTrail. Untuk informasi selengkapnya, pelajari topik berikut di Panduan Pengguna AWS CloudTrail:

- Membuat Jejak Untuk AndaAWSAkun
- AWSIntegrasi Layanan DenganCloudTrailLog
- Mengkonfigurasi Notifikasi Amazon SNS untukCloudTrail
- MenerimaCloudTrailFile Log dari Beberapa Wilayah
- MenerimaCloudTrailLog File dari Beberapa Akun

CloudTraillog semuaAWS AppSyncOperasi API. Misalnya, panggilan ke`CreateGraphqlApi`,`CreateDataSource`, dan`ListResolvers`API menghasilkan entri diCloudTrailfile log. Operasi ini dan lainnya didokumentasikan dalamAWS AppSyncReferensi API.

Setiap entri peristiwa atau log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan:

- Bahwa permintaan dibuat dengan kredensial pengguna root atau pengguna AWS Identity and Access Management (IAM).
- Bahwa permintaan tersebut dibuat dengan kredensial keamanan sementara untuk peran atau pengguna gabungan.
- Bahwa permintaan dibuat oleh layanan AWS lain.

Untuk informasi lebih lanjut, lihatCloudTrailElemen UserIdentitydiAWS CloudTrailPanduan Pengguna.

Memahami entri file log AWS AppSync

CloudTrailmengirimkan peristiwa sebagai file log yang berisi satu atau lebih entri log. Peristiwa mewakili satu permintaan dari sumber mana pun dan mencakup informasi tentang operasi yang diminta, tanggal dan waktu operasi, parameter permintaan, dan sebagainya. Karena file log ini bukan jejak tumpukan yang diurutkan dari panggilan API publik, file tersebut tidak muncul dalam urutan tertentu.

Contoh berikutCloudTrailentri log menunjukkan`CreateApiKey`operasi.

```
{
  "Records": [{
    "eventVersion": "1.05",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "A1B2C3D4E5F6G7EXAMPLE",
      "arn": "arn:aws:iam::111122223333:user/Alice",
      "accountId": "111122223333",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "userName": "Alice"
    },
    "eventTime": "2018-01-31T21:49:09Z",
    "eventSource": "appsync.amazonaws.com",
    "eventName": "CreateApiKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.2.0.1",
    "userAgent": "aws-cli/1.11.72 Python/2.7.11 Darwin/16.7.0 botocore/1.5.35",
    "requestParameters": {
      "apiId": "a1b2c3d4e5f6g7h8i9jexample"
    },
    "responseElements": {
      "apiKey": {
      "id": "***",
      "expires": 1518037200000
      }
    },
    "requestID": "99999999-9999-9999-9999-999999999999",
    "eventID": "99999999-9999-9999-9999-999999999999",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
    }
  ]
}
```

Contoh berikutCloudTrailentri log menunjukkan`ListApiKeys`operasi.

```
{
  "Records": [{
    "eventVersion": "1.05",
    "userIdentity": {
```

```
      "type": "IAMUser",
      "principalId": "A1B2C3D4E5F6G7EXAMPLE",
      "arn": "arn:aws:iam::111122223333:user/Alice",
      "accountId": "111122223333",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "userName": "Alice"
    },
    "eventTime": "2018-01-31T21:49:09Z",
    "eventSource": "appsync.amazonaws.com",
    "eventName": "ListApiKeys",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.2.0.1",
    "userAgent": "aws-cli/1.11.72 Python/2.7.11 Darwin/16.7.0 botocore/1.5.35",
    "requestParameters": {
      "apiId": "a1b2c3d4e5f6g7h8i9jexample"
    },
    "responseElements": {
      "apiKeys": [
              {
                    "id": "***",
                    "expires": 1517954400000
              },
              {
                    "id": "***",
                    "expires": 1518037200000
              },
            ]
    },
    "requestID": "99999999-9999-9999-9999-999999999999",
    "eventID": "99999999-9999-9999-9999-999999999999",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
    }
  ]
}
```

Contoh berikutCloudTrailentri log menunjukkan`DeleteApiKey`operasi.

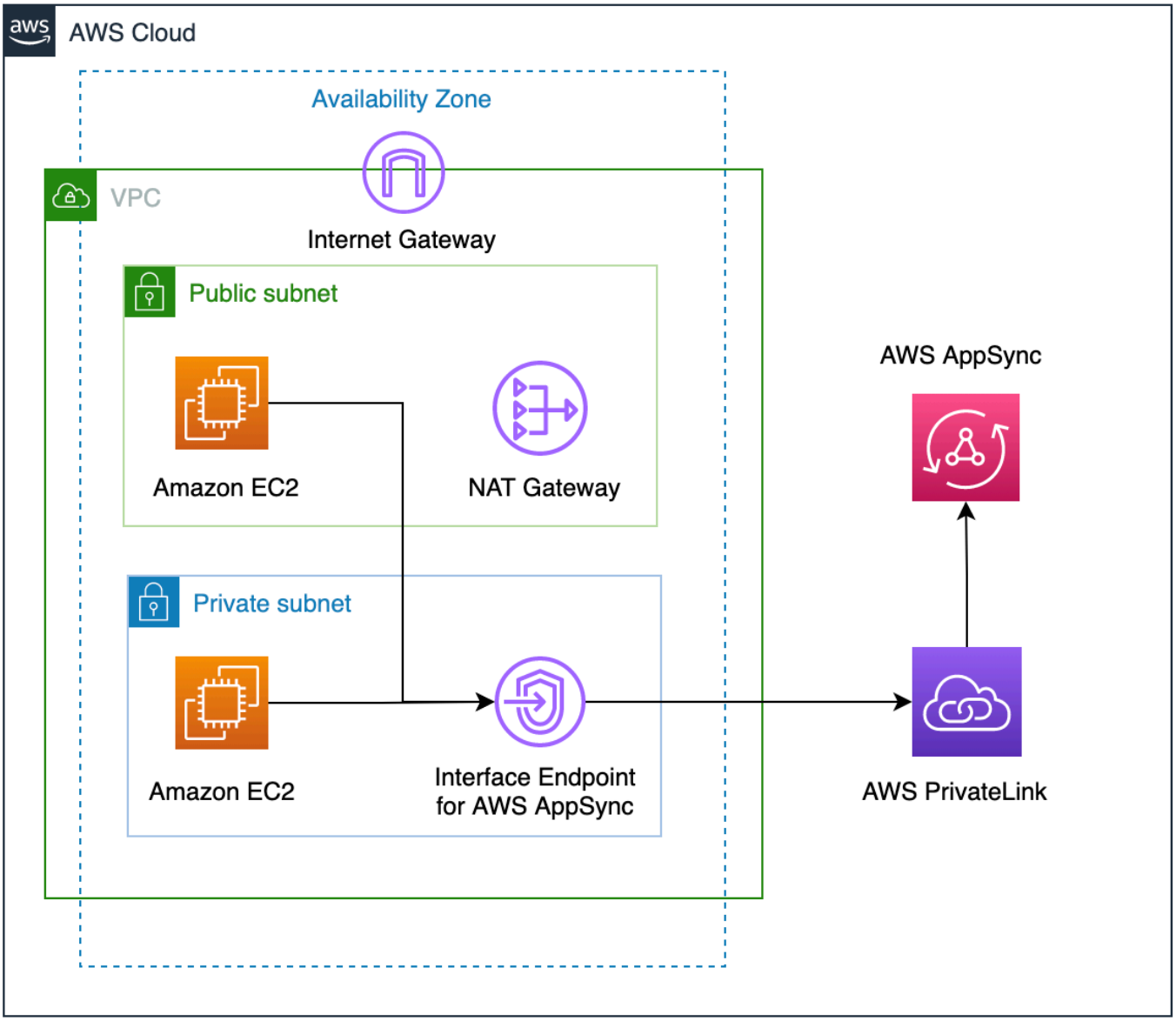
```
{
  "Records": [{
    "eventVersion": "1.05",
    "userIdentity": {
```

```
      "type": "IAMUser",
      "principalId": "A1B2C3D4E5F6G7EXAMPLE",
      "arn": "arn:aws:iam::111122223333:user/Alice",
      "accountId": "111122223333",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "userName": "Alice"
    },
    "eventTime": "2018-01-31T21:49:09Z",
    "eventSource": "appsync.amazonaws.com",
    "eventName": "DeleteApiKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.2.0.1",
    "userAgent": "aws-cli/1.11.72 Python/2.7.11 Darwin/16.7.0 botocore/1.5.35",
    "requestParameters": {
      "id": "***",
      "apiId": "a1b2c3d4e5f6g7h8i9jexample"
    },
    "responseElements": null,
    "requestID": "99999999-9999-9999-9999-999999999999",
    "eventID": "99999999-9999-9999-9999-999999999999",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
    }
  ]
}
```

MenggunakanAWS AppSyncAPI pribadi

Jika Anda menggunakan Amazon Virtual Private Cloud (Amazon VPC), Anda dapat membuatAWS AppSyncAPI pribadi, yaitu API yang hanya dapat diakses dari VPC. Dengan API Pribadi, Anda dapat membatasi akses API ke aplikasi internal Anda dan terhubung ke titik akhir GraphQL dan Realtime tanpa mengekspos data secara publik.

Untuk membuat koneksi pribadi antara VPC Anda danAWS AppSynclayanan, Anda harus membuatantarmuka titik akhir VPC. Endpoint antarmuka didukung olehAWS PrivateLink, yang memungkinkan Anda untuk mengakses secara pribadiAWS AppSyncAPI tanpa gateway internet, perangkat NAT, koneksi VPN, atauAWS Direct Connectkoneksi. Instans dalam VPC Anda tidak memerlukan alamat IP publik untuk berkomunikasi dengan API AWS AppSync. Lalu lintas antara VPC Anda danAWS AppSynctidak meninggalkanAWSjaringan.



Ada beberapa faktor tambahan yang perlu dipertimbangkan sebelum mengaktifkan fitur API Pribadi:

- Menyiapkan titik akhir antarmuka VPC untukAWS AppSyncdengan fitur DNS Pribadi diaktifkan akan mencegah sumber daya di VPC agar tidak dapat memanggil yang lainAWS AppSyncAPI publik menggunakanAWS AppSyncURL API yang dihasilkan. Ini karena permintaan ke API publik dirutekan melalui titik akhir antarmuka, yang tidak diizinkan untuk API publik. Untuk menjalankan API publik dalam skenario ini, disarankan untuk mengonfigurasi nama domain khusus pada API publik, yang kemudian dapat digunakan oleh sumber daya di VPC untuk memanggil API publik.

- AndaAWS AppSyncAPI pribadi hanya akan tersedia dari VPC Anda. TheAWS AppSyncEditor kueri konsol hanya akan dapat menjangkau API Anda jika konfigurasi jaringan browser Anda dapat merutekan lalu lintas ke VPC Anda (mis., Koneksi melalui VPN atau lebihAWS Direct Connect).
- Dengan titik akhir antarmuka VPC untukAWS AppSync, Anda dapat mengakses API Pribadi apa pun dalam hal yang samaAWSakun dan Wilayah. Untuk lebih membatasi akses ke API Pribadi, Anda dapat mempertimbangkan opsi berikut:
  - Memastikan hanya administrator yang diperlukan yang dapat membuat antarmuka titik akhir VPC untukAWS AppSync.
  - Menggunakan kebijakan kustom titik akhir VPC untuk membatasi API mana yang dapat dipanggil dari sumber daya di VPC.
  - Untuk sumber daya di VPC, kami sarankan Anda menggunakan otorisasi IAM untuk memanggilAWS AppSyncAPI dengan memastikan bahwa sumber daya diberikan peran tercakup ke bawah ke API.
- Saat membuat atau menggunakan kebijakan yang membatasi prinsip IAM, Anda harus menyetel`authorizationType`dari metode untuk`AWS_IAM`atau`NONE`.

MenciptakanAWS AppSyncAPI pribadi

Langkah-langkah berikut menunjukkan kepada Anda cara membuat API Pribadi diAWS AppSynclayanan.

> **Warning**
>
> Anda dapat mengaktifkan fitur API Pribadi hanya selama pembuatan API. Pengaturan ini tidak dapat diubah padaAWS AppSyncAPI atauAWS AppSyncAPI pribadi setelah dibuat.

1. Masuk keAWS Management Consoledan bukaAppSynckonsol.
   - DiDasbor, pilihBuat API.
2. PilihDesain API dari awal, lalu pilihBerikutnya.
3. DiAPI Pribadibagian, pilihGunakan fitur API Pribadi.
4. Konfigurasikan opsi lainnya, tinjau data API Anda, lalu pilihBuat.

Sebelum Anda dapat menggunakanAWS AppSyncAPI pribadi, Anda harus mengonfigurasi titik akhir antarmuka untukAWS AppSyncdi VPC Anda. Perhatikan bahwa API Pribadi dan VPC harus samaAWSakun dan Wilayah.

Membuat titik akhir antarmuka untukAWS AppSync

Anda dapat membuat titik akhir antarmuka untukAWS AppSyncmenggunakan konsol Amazon VPC atauAWS Command Line Interface(AWS CLI). Untuk informasi lebih lanjut, lihat Membuat titik akhir antarmuka di Panduan Pengguna Amazon VPC.

Console

1. Masuk keAWS Management Consoledan bukaTitik akhirhalaman konsol Amazon VPC.
2. Pilih Buat Titik Akhir.
   a. DiKategori layananbidang, verifikasi bahwaAWSjasadipilih.
   b. DiLayananmeja, pilih`com.amazonaws.{region}.appsync-api`. Verifikasi bahwaJenisnilai kolom adalah`Interface`.
   c. DiVPCbidang, pilih VPC dan subnetnya.
   d. Untuk mengaktifkan fitur DNS pribadi untuk titik akhir antarmuka, centangAktifkan Nama DNSkotak centang.
   e. DiKelompok keamananbidang, pilih satu atau lebih grup keamanan.
3. Pilih Buat Titik Akhir.

CLI

Gunakan perintah `create-vpc-endpoint` dan tentukan ID VPC, tipe VPC endpoint (antarmuka), nama layanan, subnet yang akan menggunakan titik akhir, dan grup keamanan yang dikaitkan dengan antarmuka jaringan titik akhir. Misalnya:

```
$ aws ec2 create-vpc-endpoint –vpc-id vpc-ec43eb89 \
  –vpc-endpoint-type Interface \
  –service-name com.amazonaws.{region}.appsync-api \
  –subnet-id subnet-abababab –security-group-id sg-1a2b3c4d
```

Untuk menggunakan opsi DNS pribadi, Anda harus mengatur`enableDnsHostnames`dan`enableDnsSupport`atribut`es`nilai VPC Anda. Untuk

informasi selengkapnya, lihat Melihat dan memperbarui dukungan DNS untuk VPC Anda di Panduan Pengguna Amazon VPC. Jika Anda mengaktifkan fitur DNS pribadi untuk titik akhir antarmuka, Anda dapat membuat permintaanAWS AppSyncAPI GraphQL dan titik akhir Real-time menggunakan titik akhir DNS publik defaultnya menggunakan format di bawah ini:

```
https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql
```

Untuk informasi selengkapnya tentang titik akhir layanan, lihatTitik akhir dan kuota layanandiAWSReferensi Umum.

Untuk informasi selengkapnya tentang interaksi layanan dengan titik akhir antarmuka, lihatMengakses layanan melalui titik akhir antarmukadiPanduan Pengguna Amazon VPC.

Untuk informasi tentang membuat dan mengonfigurasi titik akhir menggunakanAWS CloudFormation, lihatAWS::EC2: :VPcendPointsumber daya diAWS CloudFormationPanduan Pengguna.

Contoh lanjutan

Jika Anda mengaktifkan fitur DNS pribadi untuk titik akhir antarmuka, Anda dapat membuat permintaanAWS AppSyncAPI GraphQL dan titik akhir Real-time menggunakan titik akhir DNS publik defaultnya menggunakan format di bawah ini:

```
https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql
```

Menggunakan antarmuka VPC endpoint public DNS hostname, URL dasar untuk memanggil API akan dalam format berikut:

```
https://{vpc_endpoint_id}-{endpoint_dns_identifier}.appsync-api.
{region}.vpce.amazonaws.com/graphql
```

Anda juga dapat menggunakan nama host DNS khusus AZ jika Anda telah menerapkan titik akhir di AZ:

```
https://{vpc_endpoint_id}-{endpoint_dns_identifier}-{az_id}.appsync-api.
{region}.vpce.amazonaws.com/graphql.
```

Menggunakan nama DNS publik titik akhir VPC akan membutuhkanAWS AppSyncNama host titik akhir API untuk diteruskan sebagaiHostatau sebagai `x-appsync-domain`header untuk

permintaan. Contoh-contoh ini menggunakanTodoAPIyang diciptakan diLuncurkan skema sampelpanduan:

```
curl https://{vpc_endpoint_id}-{endpoint_dns_identifier}.appsync-api.
{region}.vpce.amazonaws.com/graphql \
-H "Content-Type:application/graphql" \
-H "x-api-key:da2-{xxxxxxxxxxxxxxxxxxxxxxxxxx}" \
-H "Host:{api_url_identifier}.appsync-api.{region}.amazonaws.com" \
-d '{"query":"mutation add($createtodoinput: CreateTodoInput!) {\n createTodo(input:
 $createtodoinput) {\n id\n name\n where\n when\n description\n }\n}","variables":
{"createtodoinput":{"name":"My first GraphQL task","when":"Friday Night","where":"Day
 1","description":"Learn more about GraphQL"}}}'
```

Dalam contoh berikut, kita akan menggunakanTodoaplikasi yang dihasilkan diLuncurkan skema sampelpanduan. Untuk menguji contoh Todo API, kita akan menggunakan DNS Pribadi untuk menjalankan API. Anda dapat menggunakan alat baris perintah pilihan Anda; contoh ini menggunakanmeringkukuntuk mengirim kueri dan mutasi danwscatuntuk mengatur langganan. Untuk meniru contoh kita, ganti nilai dalam tanda kurung{ }dalam perintah di bawah ini dengan nilai yang sesuai dari AndaAWSakun.

Pengujian Operasi Mutasi —**`createTodo`**Permintaan

```
curl https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql \
-H "Content-Type:application/graphql" \
-H "x-api-key:da2-{xxxxxxxxxxxxxxxxxxxxxxxxxx}" \
-d '{"query":"mutation add($createtodoinput: CreateTodoInput!) {\n createTodo(input:
 $createtodoinput) {\n id\n name\n where\n when\n description\n }\n}","variables":
{"createtodoinput":{"name":"My first GraphQL task","when":"Friday Night","where":"Day
 1","description":"Learn more about GraphQL"}}}'
```

Pengujian Operasi Mutasi —**`createTodo`**Respon

```
{
    "data": {
        "createTodo": {
            "id": "<todo-id>",
            "name": "My first GraphQL task",
            "where": "Day 1",
            "when": "Friday Night",
            "description": "Learn more about GraphQL"
        }
    }
```

```
}
```

Operasi Kueri Pengujian -**`listTodos`**Permintaan

```
curl https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql \
-H "Content-Type:application/graphql" \
-H "x-api-key:da2-{xxxxxxxxxxxxxxxxxxxxxxxxxx}" \
-d '{"query":"query ListTodos {\n listTodos {\n items {\n description\n id\n name
\n when\n where\n }\n }\n}\n","variables":{"createtodoinput":{"name":"My first
 GraphQL task","when":"Friday Night","where":"Day 1","description":"Learn more about
 GraphQL"}}}'
```

Operasi Kueri Pengujian -**`listTodos`**Permintaan

```
{
  "data": {
    "listTodos": {
      "items": [
        {
          "description": "Learn more about GraphQL",
          "id": "<todo-id>",
          "name": "My first GraphQL task",
          "when": "Friday night",
          "where": "Day 1"
        }
      ]
    }
  }
}
```

Menguji Operasi Berlangganan - Berlangganan**`createTodo`**mutasi

Untuk mengatur langganan GraphQL diAWS AppSync, lihatMembangun waktu nyataWebSocketklien. Dari instans Amazon EC2 di VPC, Anda dapat mengujiAWS AppSyncTitik akhir langganan API pribadi menggunakanwscat. Contoh di bawah ini menggunakan`API KEY`untuk otorisasi.

```
$ header=`echo '{"host":"{api_url_identifier}.appsync-api.{region}.amazonaws.com","x-
api-key":"da2-{xxxxxxxxxxxxxxxxxxxxxxxxxx}"}' | base64 | tr -d '\n'`
$ wscat -p 13 -s graphql-ws -c  "wss://{api_url_identifier}.appsync-realtime-api.us-
west-2.amazonaws.com/graphql?header=$header&payload=e30="
Connected (press CTRL+C to quit)
```

```
> {"type": "connection_init"}
< {"type":"connection_ack","payload":{"connectionTimeoutMs":300000}}
< {"type":"ka"}
> {"id":"f7a49717","payload":{"data":"{\"query\":\"subscription
 onCreateTodo {onCreateTodo {description id name where when}}\",
\"variables\":{}}","extensions":{"authorization":{"x-api-key":"da2-
{xxxxxxxxxxxxxxxxxxxxxxxxxx}","host":"{api_url_identifier}.appsync-api.
{region}.amazonaws.com"}}},"type":"start"}
< {"id":"f7a49717","type":"start_ack"}
```

Atau, gunakan nama domain titik akhir VPC sambil memastikan untuk menentukanTuan rumahheader di`wscat`perintah untuk membuat websocket:

```
$ header=`echo '{"host":"{api_url_identifier}.appsync-api.{region}.amazonaws.com","x-
api-key":"da2-{xxxxxxxxxxxxxxxxxxxxxxxxxx}"}' | base64 | tr -d '\n'`
$ wscat -p 13 -s graphql-ws -c  "wss://{vpc_endpoint_id}-
{endpoint_dns_identifier}.appsync-api.{region}.vpce.amazonaws.com/graphql?header=
$header&payload=e30=" --header Host:{api_url_identifier}.appsync-realtime-api.us-
west-2.amazonaws.com
Connected (press CTRL+C to quit)
> {"type": "connection_init"}
< {"type":"connection_ack","payload":{"connectionTimeoutMs":300000}}
< {"type":"ka"}
> {"id":"f7a49717","payload":{"data":"{\"query\":\"subscription
 onCreateTodo {onCreateTodo {description id priority title}}\",
\"variables\":{}}","extensions":{"authorization":{"x-api-key":"da2-
{xxxxxxxxxxxxxxxxxxxxxxxxxx}","host":"{api_url_identifier}.appsync-api.
{region}.amazonaws.com"}}},"type":"start"}
< {"id":"f7a49717","type":"start_ack"}
```

Jalankan kode mutasi di bawah ini:

```
curl https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql \
-H "Content-Type:application/graphql" \
-H "x-api-key:da2-{xxxxxxxxxxxxxxxxxxxxxxxxxx}" \
-d '{"query":"mutation add($createtodoinput: CreateTodoInput!) {\n createTodo(input:
 $createtodoinput) {\n id\n name\n where\n when\n description\n }\n}","variables":
{"createtodoinput":{"name":"My first GraphQL task","when":"Friday Night","where":"Day
 1","description":"Learn more about GraphQL"}}}'
```

Setelah itu, langganan dipicu, dan pemberitahuan pesan muncul seperti yang ditunjukkan di bawah ini:

```
< {"id":"f7a49717","type":"data","payload":{"data":{"onCreateTodo":{"description":"Go
 to the shops","id":"169ce516-b7e8-4a6a-88c1-ab840184359f","priority":5,"title":"Go to
 the shops"}}}}
```

Menggunakan kebijakan IAM untuk membatasi pembuatan API publik

AWS AppSyncmendukung IAM`Condition`pernyataanuntuk digunakan dengan API Pribadi. The`visibility`bidang dapat disertakan dengan pernyataan kebijakan IAM untuk`appsync:CreateGraphqlApi`operasi untuk mengontrol peran IAM dan pengguna mana yang dapat membuat API pribadi dan publik. Ini memberi administrator IAM kemampuan untuk menentukan kebijakan IAM yang hanya akan memungkinkan pengguna untuk membuat API GraphQL Pribadi. Pengguna yang mencoba membuat API publik akan menerima pesan yang tidak sah.

Misalnya, administrator IAM dapat membuat pernyataan kebijakan IAM berikut untuk memungkinkan pembuatan API Pribadi:

```
{
    "Sid": "AllowPrivateAppSyncApis",
    "Effect": "Allow",
    "Action": "appsync:CreateGraphqlApi",
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "appsync:Visibility": "PRIVATE"
        }
    }
}
```

Administrator IAM juga dapat menambahkan yang berikutkebijakan kontrol layananuntuk memblokir semua pengguna diAWSorganisasi dari menciptakanAWS AppSyncAPI selain API Pribadi:

```
{
    "Sid": "BlockNonPrivateAppSyncApis",
    "Effect": "Deny",
    "Action": "appsync:CreateGraphqlApi",
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringNotEquals": {
            "appsync:Visibility": "PRIVATE"
```

```
        }
    }
}
```

Mengkonfigurasi GraphQL menjalankan kompleksitas, kedalaman kueri, dan introspeksi dengan AWS AppSync

AWS AppSyncmemungkinkan Anda untuk mengaktifkan atau menonaktifkan fitur introspeksi dan menetapkan batas jumlah level bersarang dan resolver dalam satu kueri.

Menggunakan fitur introspeksi

> **Tip**
>
> Untuk informasi lebih lanjut tentang introspeksi di GraphQL, lihat artikel ini di situs web GraphQL foundation.

Secara default, GraphQL memungkinkan Anda menggunakan introspeksi untuk menanyakan skema itu sendiri untuk menemukan jenis, bidang, kueri, mutasi, langganan, dll. Ini adalah fitur penting untuk mempelajari bagaimana data dibentuk dan diproses oleh layanan GraphQL Anda. Namun, ada beberapa hal yang perlu dipertimbangkan ketika berhadapan dengan introspeksi. Anda mungkin memiliki kasus penggunaan yang akan mendapat manfaat dari introspeksi yang dinonaktifkan, seperti kasus di mana nama bidang mungkin sensitif atau tersembunyi atau skema API lengkap dimaksudkan untuk dibiarkan tidak terdokumentasi bagi konsumen. Dalam kasus ini, penerbitan data skema melalui introspeksi dapat mengakibatkan kebocoran data pribadi yang disengaja.

Untuk mencegah hal ini terjadi, Anda dapat menonaktifkan introspeksi. Ini akan mencegah pihak yang tidak berwenang menggunakan bidang introspeksi pada skema Anda. Namun, penting untuk dicatat bahwa introspeksi berguna bagi tim pengembangan untuk mempelajari bagaimana data dalam layanan mereka diproses. Secara internal, mungkin membantu untuk tetap mengaktifkan introspeksi saat menonaktifkannya dalam kode produksi sebagai lapisan keamanan tambahan. Cara lain untuk menangani ini adalah dengan menambahkan metode otorisasi, yang AWS AppSync juga menyediakan. Untuk informasi selengkapnya, lihat otorisasi.

AWS AppSyncmemungkinkan Anda untuk mengaktifkan atau menonaktifkan introspeksi di tingkat API. Untuk mengaktifkan atau menonaktifkan introspeksi, lakukan hal berikut:

1. Masuk ke AWS Management Console dan buka AppSynckonsol.
2. Pada halaman API, pilih nama GraphQL API.
3. Di beranda API Anda, di panel navigasi, pilih Pengaturan.
4. Dalam konfigurasi API, pilih Edit.
5. Di bawah kueri Introspeksi, lakukan hal berikut:
   - Mengaktifkan atau menonaktifkan Aktifkan kueri introspeksi.
6. Pilih Simpan.

Ketika introspeksi diaktifkan (perilaku default), menggunakan sistem introspeksi akan bekerja secara normal. Misalnya, gambar di bawah ini menunjukkan `__schema` bidang yang memproses semua jenis yang tersedia dalam skema:

Saat menonaktifkan fitur ini, kesalahan validasi akan muncul di respons sebagai gantinya:

Mengkonfigurasi batas kedalaman kueri

Ada kalanya Anda mungkin menginginkan kontrol yang lebih terperinci atas bagaimana API berfungsi selama operasi. Salah satu kontrol tersebut adalah menambahkan batas jumlah level bersarang yang dapat diproses oleh kueri. Secara default, kueri dapat memproses tingkat bersarang dalam jumlah tak terbatas. Membatasi kueri ke jumlah level bersarang tertentu memiliki implikasi potensial terhadap kinerja dan fleksibilitas proyek Anda. Ambil kueri berikut:

```
query MyQuery {
  L1: nextLayer {
    L2: nextLayer {
      L3: nextLayer {
        L4: value
      }
    }
  }
}
```

Proyek Anda mungkin meminta untuk membatasi kueri untuk L1 atau L2 untuk beberapa tujuan. Secara default, seluruh kueri dari L1 to L4 akan diproses tanpa cara untuk mengontrolnya. Dengan menetapkan batas, Anda dapat mencegah kueri mengakses apa pun yang melewati level yang ditentukan.

Untuk menambahkan batas kedalaman kueri, lakukan hal berikut:

1. Masuk ke AWS Management Console dan buka AppSynckonsol.

2. Pada halaman API, pilih nama GraphQL API.
3. Di beranda API Anda, di panel navigasi, pilih Pengaturan.
4. Dalam konfigurasi API, pilih Edit.
5. Di bawah Query depth, lakukan hal berikut:
   a. Nyalakan atau nonaktifkan Aktifkan kedalaman kueri.
   b. Dalam kedalaman maksimum, atur batas kedalaman. Ini bisa antara 1 dan75.
6. Pilih Simpan.

Ketika batas ditetapkan, melewati batas atasnya akan menghasilkan `QueryDepthLimitReached` kesalahan. Misalnya, gambar di bawah ini menunjukkan kueri dengan batas kedalaman 2 melewati batas ke level ketiga (`L3`) dan keempat (`L4`):

Perhatikan bahwa bidang masih dapat ditandai sebagai nullable atau non-nullable dalam skema. Jika bidang yang tidak dapat dibatalkan menerima `QueryDepthLimitReached` kesalahan, kesalahan itu akan dilemparkan ke bidang induk nullable pertama.

Mengkonfigurasi batas jumlah resolver

Anda juga dapat mengontrol berapa banyak resolver yang dapat diproses setiap kueri. Seperti kedalaman kueri, Anda dapat menetapkan batas untuk jumlah ini. Ambil kueri berikut yang berisi tiga resolver:

```
query MyQuery {
  resolver1: resolver
```

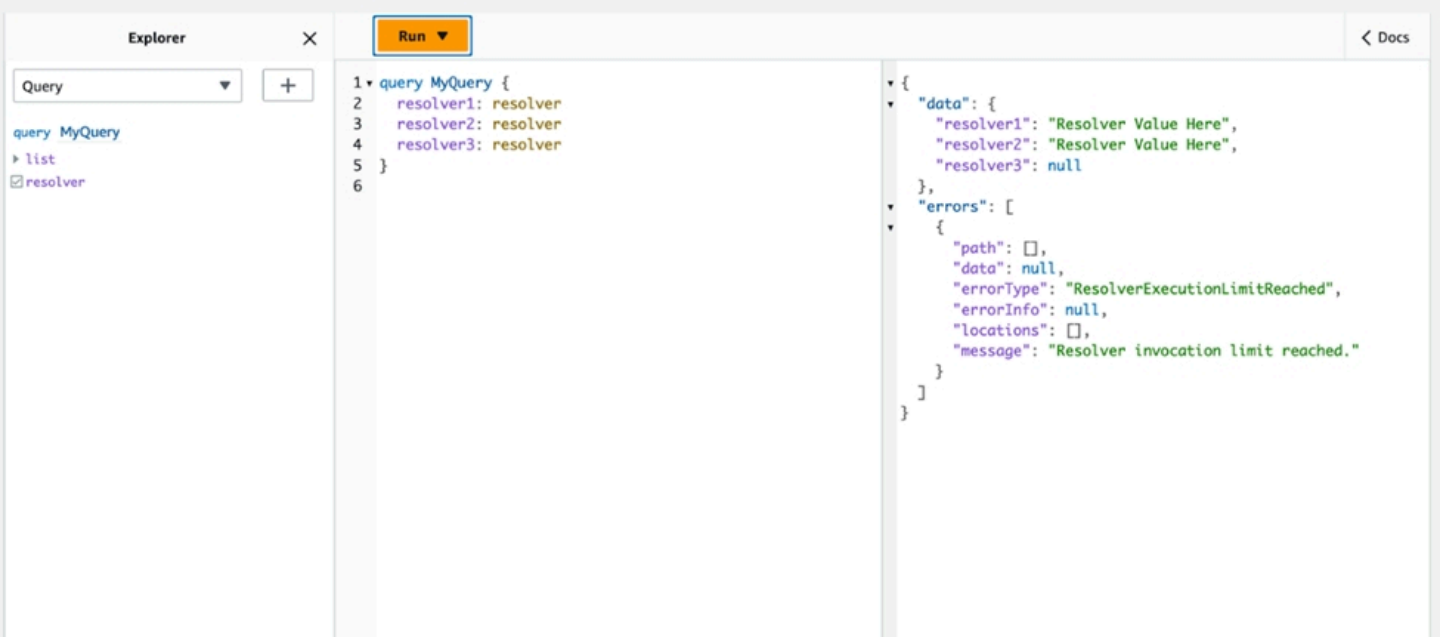
```
  resolver2: resolver
  resolver3: resolver
}
```

Secara default, setiap kueri dapat memproses hingga 10000 resolver. Pada contoh di atas, `resolver1resolver2`,, dan `resolver3` akan diproses. Namun, proyek Anda mungkin meminta untuk membatasi setiap kueri untuk menangani satu atau dua resolver secara total. Dengan menetapkan batas, Anda dapat memberi tahu kueri untuk tidak menangani resolver apa pun melewati nomor tertentu seperti resolver first (`resolver1`) atau second (`resolver2`).

Untuk menambahkan batas hitungan resolver, lakukan hal berikut:

1. Masuk ke AWS Management Console dan buka AppSynckonsol.
2. Pada halaman API, pilih nama GraphQL API.
3. Di beranda API Anda, di panel navigasi, pilih Pengaturan.
4. Dalam konfigurasi API, pilih Edit.
5. Di bawah batas jumlah Resolver, lakukan hal berikut:
   a. Aktifkan Aktifkan jumlah resolver.
   b. Dalam Hitungan resolver maksimum, atur batas hitungan. Ini bisa antara 1 dan`10000`.
6. Pilih Simpan.

Seperti batas kedalaman kueri, melebihi batas resolver yang dikonfigurasi menyebabkan kueri diakhiri dengan `ResolverExecutionLimitReached` kesalahan pada resolver tambahan. Pada gambar di bawah ini, kueri dengan batas jumlah resolver 2 mencoba memproses tiga resolver. Karena batasnya, resolver ketiga melempar kesalahan dan tidak berjalan.



Menggunakan variabel lingkungan di AWS AppSync

Anda dapat menggunakan variabel lingkungan untuk menyesuaikan perilaku AWS AppSync resolver dan fungsi Anda tanpa memperbarui kode Anda. Variabel lingkungan adalah pasangan string yang disimpan dengan konfigurasi API Anda yang tersedia untuk resolver dan fungsi Anda untuk dimanfaatkan saat runtime. Mereka sangat berguna untuk situasi di mana Anda harus mereferensikan data konfigurasi yang hanya tersedia selama penyiapan awal tetapi perlu digunakan oleh resolver dan fungsi Anda selama dijalankan. Variabel lingkungan mengekspos data konfigurasi dalam kode Anda, sehingga mengurangi kebutuhan untuk hard-code nilai-nilai tersebut.

> **Note**
>
> Untuk meningkatkan keamanan database, sebaiknya gunakan Secrets Manager atau AWS Systems Manager Parameter Store, bukan variabel lingkungan untuk menyimpan kredensi atau informasi sensitif. Untuk memanfaatkan fitur ini, lihat Memanggil AWS layanan dengan sumber data AWS AppSync HTTP.

Variabel lingkungan harus mengikuti beberapa perilaku dan aturan agar berfungsi dengan baik:

- Baik JavaScript resolver/fungsi dan template VTL mendukung variabel lingkungan.
- Variabel lingkungan tidak dievaluasi sebelum pemanggilan fungsi.

- Variabel lingkungan hanya mendukung nilai string.
- Setiap nilai yang ditentukan dalam variabel lingkungan dianggap sebagai string literal dan tidak diperluas.
- Evaluasi variabel idealnya harus dilakukan dalam kode fungsi.

Mengkonfigurasi variabel lingkungan (konsol)

Anda dapat mengonfigurasi variabel lingkungan untuk AWS AppSync GraphQL API Anda dengan membuat variabel dan menentukan pasangan nilai kunci-nya. Resolver dan fungsi Anda akan menggunakan nama kunci variabel lingkungan untuk mengambil nilai saat runtime. Untuk mengatur variabel lingkungan di AWS AppSync konsol:

1. Masuk ke AWS Management Console dan buka AppSynckonsol.
2. Pada halaman API, pilih nama GraphQL API.
3. Di beranda API Anda, di panel navigasi, pilih Pengaturan.
4. Di bawah variabel Lingkungan, pilih Tambahkan variabel lingkungan.
5. Pilih Tambahkan variabel lingkungan.
6. Masukkan kunci dan nilai.
7. Jika perlu, ulangi langkah 5 dan 6 untuk menambahkan lebih banyak nilai kunci. Jika Anda perlu menghapus nilai kunci, pilih opsi Hapus dan kunci untuk dihapus.
8. Pilih Kirim.

> **Tip**
>
> Ada beberapa aturan yang harus Anda ikuti saat membuat kunci dan nilai:
>
> - Kunci harus dimulai dengan surat.
> - Kunci harus memiliki panjang setidaknya dua karakter.
> - Tombol hanya dapat berisi huruf, angka, dan karakter garis bawah (_).
> - Nilainya bisa mencapai 512 karakter.
> - Anda dapat mengonfigurasi hingga 50 pasangan nilai kunci dalam GraphQL API.

Mengkonfigurasi variabel lingkungan (API)

Untuk menyetel variabel lingkungan menggunakan API, Anda dapat menggunakan`PutGraphqlApiEnvironmentVariables`. Perintah CLI yang sesuai adalah. `put-graphql-api-environment-variables`

Untuk mengambil variabel lingkungan menggunakan API, Anda dapat menggunakan`GetGraphqlApiEnvironmentVariables`. Perintah CLI yang sesuai adalah. `get-graphql-api-environment-variables`

Perintah harus berisi ID API dan daftar variabel lingkungan:

```
aws appsync put-graphql-api-environment-variables \
  --api-id "<api-id>" \
  --environment-variables '{"key1":"value1","key2":"value2", …}'
```

Contoh berikut menetapkan dua variabel lingkungan dalam API dengan ID `abcdefghijklmnopqrstuvwxyz` menggunakan `put-graphql-api-environment-variables` perintah:

```
aws appsync put-graphql-api-environment-variables \
  --api-id "abcdefghijklmnopqrstuvwxyz" \
  --environment-variables '{"USER_TABLE":"users_prod","DEBUG":"true"}'
```

Perhatikan bahwa ketika Anda menerapkan variabel lingkungan dengan `put-graphql-api-environment-variables` perintah, isi struktur variabel lingkungan ditimpa; ini berarti variabel lingkungan yang ada akan hilang. Untuk mempertahankan variabel lingkungan yang ada saat menambahkan yang baru, sertakan semua pasangan nilai kunci yang ada bersama dengan yang baru dalam permintaan Anda. Menggunakan contoh di atas, jika Anda ingin menambahkan`"EMPTY":""`, Anda dapat melakukan hal berikut:

```
aws appsync put-graphql-api-environment-variables \
  --api-id "abcdefghijklmnopqrstuvwxyz" \
  --environment-variables '{"USER_TABLE":"users_prod","DEBUG":"true", "EMPTY":""}'
```

Untuk mengambil konfigurasi saat ini, gunakan `get-graphql-api-environment-variables` perintah:

```
aws appsync get-graphql-api-environment-variables --api-id "<api-id>"
```

Dengan menggunakan contoh di atas, Anda dapat menggunakan perintah berikut:

```
aws appsync get-graphql-api-environment-variables --api-id "abcdefghijklmnopqrstuvwxyz"
```

Hasilnya akan menampilkan daftar variabel lingkungan bersama dengan nilai-nilai kunci mereka:

```
{
    "environmentVariables": {
        "USER_TABLE": "users_prod",
        "DEBUG": "true",
        "EMPTY": ""
    }
}
```

Mengkonfigurasi variabel lingkungan (CFN)

Anda dapat menggunakan template di bawah ini untuk membuat variabel lingkungan:

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  GraphQLApiWithEnvVariables:
    Type: "AWS::AppSync::GraphQLApi"
    Properties:
      Name: "MyApiWithEnvVars"
      AuthenticationType: "AWS_IAM"
      EnvironmentVariables:
        EnvKey1: "non-empty"
        EnvKey2: ""
```

variabel lingkungan dan API gabungan

Variabel lingkungan yang ditentukan dalam API Sumber juga tersedia di API Gabungan Anda. Variabel lingkungan dalam API Gabungan bersifat hanya-baca dan tidak dapat diperbarui. Perhatikan bahwa kunci variabel lingkungan Anda harus unik di semua API Sumber agar penggabungan Anda berhasil; kunci duplikat akan selalu mengakibatkan kegagalan penggabungan.

Mengambil variabel lingkungan

Untuk mengambil variabel lingkungan dalam kode fungsi Anda, ambil nilai dari `ctx.env` objek di resolver dan fungsi Anda. Di bawah ini adalah beberapa contoh dari tindakan ini.

Publishing to Amazon SNS

Dalam contoh ini, resolver HTTP kami mengirimkan pesan ke topik Amazon SNS. ARN topik hanya diketahui setelah tumpukan yang mendefinisikan GraphQL API dan topik telah diterapkan.

```
/**
 * Sends a publish request to the SNS topic
 */
export function request(ctx) {
    const TOPIC_ARN = ctx.env.TOPIC_ARN;
    const { input: values } = ctx.args;
    // this custom function sends values to the SNS topic
    return publishToSNSRequest(TOPIC_ARN, values);
}
```

Transactions with DynamoDB

Dalam contoh ini, nama tabel DynamoDB berbeda jika API digunakan untuk pementasan atau sudah dalam produksi. Kode resolver tidak perlu diubah. Nilai variabel lingkungan diperbarui berdasarkan di mana API digunakan.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { authorId, postId } = ctx.args;
  return {
    operation: 'TransactWriteItems',
    transactItems: [
      {
        table: ctx.env.POST_TABLE,
        operation: 'PutItem',
        key: util.dynamodb.toMapValues({ postId }),
        // rest of the configuration
      },
      {
        table: ctx.env.AUTHOR_TABLE,
        operation: 'UpdateItem',
        key: util.dynamodb.toMapValues({ authorId }),
        // rest of the configuration
      },
    ],
  };
}
```

Otorisasi dan otentikasi

Bagian ini menjelaskan opsi untuk mengonfigurasi keamanan dan perlindungan data untuk aplikasi Anda.

Jenis otorisasi

Ada lima cara Anda dapat mengotorisasi aplikasi untuk berinteraksi dengan AWS AppSync GraphQL API Anda. Anda menentukan jenis otorisasi yang Anda gunakan dengan menentukan salah satu nilai jenis otorisasi berikut dalam panggilan API AWS AppSync atau CLI Anda:

- **`API_KEY`**

  Untuk menggunakan kunci API.
- **`AWS_LAMBDA`**

  Untuk menggunakan suatu AWS Lambda fungsi.
- **`AWS_IAM`**

  Untuk menggunakan izin AWS Identity and Access Management (IAM).
- **`OPENID_CONNECT`**

  Untuk menggunakan penyedia OpenID Connect Anda.
- **`AMAZON_COGNITO_USER_POOLS`**

  Untuk menggunakan kumpulan pengguna Amazon Cognito.

Jenis otorisasi dasar ini berfungsi untuk sebagian besar pengembang. Untuk kasus penggunaan yang lebih lanjut, Anda dapat menambahkan mode otorisasi tambahan melalui konsol, CLI, dan. AWS CloudFormation Untuk mode otorisasi tambahan, AWS AppSync berikan jenis otorisasi yang mengambil nilai yang tercantum di atas (yaitu,,`API_KEY`, `AWS_LAMBDA` `AWS_IAMOPENID_CONNECT`, dan`AMAZON_COGNITO_USER_POOLS`).

Saat Anda menentukan`API_KEY`,`AWS_LAMBDA`, atau `AWS_IAM` sebagai jenis otorisasi utama atau default, Anda tidak dapat menentukannya lagi sebagai salah satu mode otorisasi tambahan. Demikian pula, Anda tidak dapat menduplikasi`API_KEY`, `AWS_LAMBDA` atau `AWS_IAM` di dalam mode otorisasi tambahan. Anda dapat menggunakan beberapa Amazon Cognito User Pools dan

penyedia OpenID Connect. Namun, Anda tidak dapat menggunakan duplikat Amazon Cognito User Pools atau penyedia OpenID Connect antara mode otorisasi default dan salah satu mode otorisasi tambahan. Anda dapat menentukan klien yang berbeda untuk Kumpulan Pengguna Amazon Cognito atau penyedia OpenID Connect menggunakan ekspresi reguler konfigurasi yang sesuai.

Otorisasi API_KEY

API yang tidak diautentikasi memerlukan pembatasan yang lebih ketat daripada API yang diautentikasi. Salah satu cara untuk mengontrol pelambatan untuk titik akhir GraphQL yang tidak diautentikasi adalah melalui penggunaan kunci API. Kunci API adalah nilai hard-code dalam aplikasi Anda yang dihasilkan oleh AWS AppSync layanan saat Anda membuat titik akhir GraphQL yang tidak diautentikasi. Anda dapat memutar kunci API dari konsol, dari CLI, atau dari referensi AWS AppSync API.

Console

1. Masuk ke AWS Management Console dan buka AppSynckonsol.

   a. Di dasbor API, pilih GraphQL API Anda.
   b. Di Sidebar, pilih Pengaturan.
2. Di bawah mode otorisasi default, pilih kunci API.
3. Dalam tabel kunci API, pilih Tambahkan kunci API.

   Kunci API baru akan dihasilkan dalam tabel.

   - Untuk menghapus kunci API lama, pilih kunci API di tabel lalu pilih Hapus.
4. Pilih Simpan di bagian bawah halaman.

CLI

1. Jika Anda belum melakukannya, konfigurasikan akses Anda ke AWS CLI. Untuk informasi selengkapnya, lihat Dasar-dasar konfigurasi.
2. Buat objek GraphQL API dengan menjalankan perintah. `update-graphql-api`

   Anda harus mengetikkan dua parameter untuk perintah khusus ini:

   1. `api-id`GraphQL API Anda.
   2. Yang baru `name` dari API Anda. Anda dapat menggunakan hal yang sama`name`.

3. Yang `authentication-type` akan menjadi`API_KEY`.

> **Note**
>
> Ada parameter lain seperti `Region` yang harus dikonfigurasi tetapi biasanya akan default ke nilai konfigurasi CLI Anda.

Contoh perintah mungkin terlihat seperti ini:

```
aws appsync update-graphql-api --api-id abcdefghijklmnopqrstuvwxyz --name
 TestAPI --authentication-type API_KEY
```

Output akan dikembalikan dalam CLI. Berikut adalah contoh di JSON:

```
{
    "graphqlApi": {
        "xrayEnabled": false,
        "name": "TestAPI",
        "authenticationType": "API_KEY",
        "tags": {},
        "apiId": "abcdefghijklmnopqrstuvwxyz",
        "uris": {
            "GRAPHQL": "https://s8i3kk3ufhe9034ujnv73r513e.appsync-api.us-
west-2.amazonaws.com/graphql",
            "REALTIME": "wss://s8i3kk3ufhe9034ujnv73r513e.appsync-realtime-
api.us-west-2.amazonaws.com/graphql"
        },
        "arn": "arn:aws:appsync:us-west-2:348581070237:apis/
abcdefghijklmnopqrstuvwxyz"
    }
}
```

Kunci API dapat dikonfigurasi hingga 365 hari, dan Anda dapat memperpanjang tanggal kedaluwarsa yang ada hingga 365 hari lagi sejak hari itu. Kunci API direkomendasikan untuk tujuan pengembangan atau kasus penggunaan di mana aman untuk mengekspos API publik.

Pada klien, kunci API ditentukan oleh header`x-api-key`.

Misalnya, jika ya`'ABC123'`, Anda `API_KEY` dapat mengirim kueri `curl` GraphQL melalui sebagai berikut:

```
$ curl -XPOST -H "Content-Type:application/graphql" -H "x-api-key:ABC123" -d
 '{ "query": "query { movies { id } }" }' https://YOURAPPSYNCENDPOINT/graphql
```

Otorisasi AWS_LAMBDA

Anda dapat menerapkan logika otorisasi API Anda sendiri menggunakan AWS Lambda fungsi. Anda dapat menggunakan fungsi Lambda untuk otorisasi primer atau sekunder, tetapi mungkin hanya ada satu fungsi otorisasi Lambda per API. Saat menggunakan fungsi Lambda untuk otorisasi, berikut ini berlaku:

- Jika API mengaktifkan mode `AWS_IAM` otorisasi `AWS_LAMBDA` dan, tanda tangan SigV4 tidak dapat digunakan sebagai token otorisasi. `AWS_LAMBDA`
- Jika API memiliki mode `AWS_LAMBDA` dan `OPENID_CONNECT` otorisasi atau mode `AMAZON_COGNITO_USER_POOLS` otorisasi diaktifkan, maka token OIDC tidak dapat digunakan sebagai token otorisasi. `AWS_LAMBDA` Perhatikan bahwa token OIDC dapat menjadi skema Pembawa.
- Fungsi Lambda tidak boleh mengembalikan lebih dari 5MB data kontekstual untuk resolver.

Misalnya, jika token otorisasi Anda`'ABC123'`, Anda dapat mengirim kueri GraphQL melalui curl sebagai berikut:

```
$ curl -XPOST -H "Content-Type:application/graphql" -H "Authorization:ABC123" -d
 '{ "query":
        "query { movies { id } }" }' https://YOURAPPSYNCENDPOINT/graphql
```

Fungsi Lambda dipanggil sebelum setiap kueri atau mutasi. Nilai pengembalian dapat di-cache berdasarkan ID API dan token otentikasi. Secara default, caching tidak diaktifkan, tetapi ini dapat diaktifkan pada tingkat API atau dengan menetapkan `ttlOverride` nilai dalam nilai pengembalian fungsi.

Ekspresi reguler yang memvalidasi token otorisasi sebelum fungsi dipanggil dapat ditentukan jika diinginkan. Ekspresi reguler ini digunakan untuk memvalidasi bahwa token otorisasi memiliki format

yang benar sebelum fungsi Anda dipanggil. Setiap permintaan menggunakan token yang tidak cocok dengan ekspresi reguler ini akan ditolak secara otomatis.

Fungsi Lambda yang digunakan untuk otorisasi memerlukan kebijakan utama `appsync.amazonaws.com` untuk diterapkan pada mereka untuk memungkinkan AWS AppSync untuk memanggil mereka. Tindakan ini dilakukan secara otomatis di AWS AppSync konsol; AWS AppSync Konsol tidak menghapus kebijakan. Untuk informasi selengkapnya tentang melampirkan kebijakan ke fungsi Lambda, lihat Kebijakan berbasis sumber daya di Panduan Pengembang. AWS Lambda

Fungsi Lambda yang Anda tentukan akan menerima acara dengan bentuk berikut:

```
{
    "authorizationToken": "ExampleAUTHtoken123123123",
    "requestContext": {
        "apiId": "aaaaaa123123123example123",
        "accountId": "111122223333",
        "requestId": "f4081827-1111-4444-5555-5cf4695f339f",
        "queryString": "mutation CreateEvent {...}\n\nquery MyQuery {...}\n",
        "operationName": "MyQuery",
        "variables": {}
    }
    "requestHeaders": {
        application request headers
    }
}
```

`event`Objek berisi header yang dikirim dalam permintaan dari klien aplikasi ke AWS AppSync.

Fungsi otorisasi harus mengembalikan setidaknya`isAuthorized`, boolean yang menunjukkan jika permintaan diotorisasi. AWS AppSync mengenali kunci berikut yang dikembalikan dari fungsi otorisasi Lambda:

Daftar fungsi

`isAuthorized`(boolean, diperlukan)

Nilai boolean yang menunjukkan apakah nilai di `authorizationToken` diotorisasi untuk melakukan panggilan ke GraphQL API.

Jika nilai ini benar, eksekusi GraphQL API berlanjut. Jika nilai ini salah, an `UnauthorizedException` dinaikkan

`deniedFields`(daftar string, opsional)

Daftar yang secara paksa diubah menjadi`null`, bahkan jika nilai dikembalikan dari resolver.

Setiap item adalah bidang ARN yang memenuhi syarat penuh dalam bentuk `arn:aws:appsync:`*`us-east-1`*`:`*`111122223333`*`:apis/`*`GraphQLApiId`*`/types/`*`TypeName`*`/fields/`*`FieldName` atau bentuk pendek. *`TypeName`*`.`*`FieldName` Formulir ARN lengkap harus digunakan ketika dua API berbagi otorisasi fungsi Lambda dan mungkin ada ambiguitas antara tipe dan bidang umum antara dua API.

`resolverContext`(Objek JSON, opsional)

Sebuah objek JSON terlihat seperti `$ctx.identity.resolverContext` pada template resolver. Misalnya, jika struktur berikut dikembalikan oleh resolver:

```
{
  "isAuthorized":true
  "resolverContext": {
    "banana":"very yellow",
    "apple":"very green"
  }
}
```

Nilai `ctx.identity.resolverContext.apple` dalam template resolver akan menjadi ""`very green`. `resolverContext`Objek hanya mendukung pasangan kunci-nilai. Kunci bersarang tidak didukung.

> **Warning**
>
> Ukuran total objek JSON ini tidak boleh melebihi 5MB.

`ttlOverride`(bilangan bulat, opsional)

Jumlah detik respons harus di-cache. Jika tidak ada nilai yang dikembalikan, nilai dari API akan digunakan. Jika ini 0, responsnya tidak di-cache.

Otorisasi Lambda memiliki batas waktu 10 detik. Kami merekomendasikan merancang fungsi untuk dijalankan dalam waktu sesingkat mungkin untuk menskalakan kinerja API Anda.

Beberapa AWS AppSync API dapat berbagi satu fungsi Lambda otentikasi. Penggunaan otorisasi lintas akun tidak diizinkan.

Saat berbagi fungsi otorisasi di antara beberapa API, ketahuilah bahwa nama bidang bentuk pendek (*typename*.*fieldname*) mungkin secara tidak sengaja menyembunyikan bidang. Untuk membedakan bidang di`deniedFields`, Anda dapat menentukan bidang ARN yang tidak ambigu dalam bentuk. `arn:aws:appsync:`*region*`:`*accountId*`:apis/`*GraphQLApiId*`/types/`*typeName*`/fields/`*fieldName*

Untuk menambahkan fungsi Lambda sebagai mode otorisasi default di: AWS AppSync

Console

1. Masuk ke AWS AppSync Konsol dan navigasikan ke API yang ingin Anda perbarui.
2. Arahkan ke halaman Pengaturan untuk API Anda.

   Ubah otorisasi tingkat API menjadi. AWS Lambda
3. Pilih Wilayah AWS dan Lambda ARN untuk mengotorisasi panggilan API terhadap.

   > **Note**
   >
   > Kebijakan utama yang sesuai akan ditambahkan secara otomatis, memungkinkan AWS AppSync untuk memanggil fungsi Lambda Anda.
4. Secara opsional, atur respons TTL dan ekspresi reguler validasi token.

AWS CLI

1. Lampirkan kebijakan berikut ke fungsi Lambda yang digunakan:

   ```
   aws lambda add-permission --function-name "my-function" --statement-id "appsync"
    --principal appsync.amazonaws.com --action lambda:InvokeFunction --output text
   ```

   > **Important**
   >
   > Jika Anda ingin kebijakan fungsi dikunci ke satu GraphQL API, Anda dapat menjalankan perintah ini:

```
aws lambda add-permission --function-name “my-function” --
statement-id “appsync” --principal appsync.amazonaws.com --action
 lambda:InvokeFunction --source-arn “<my AppSync API ARN>” --output text
```

2. Perbarui AWS AppSync API Anda untuk menggunakan ARN fungsi Lambda yang diberikan sebagai otorisasi:

```
aws appsync update-graphql-api --api-id example2f0ur2oid7acexample --
name exampleAPI --authentication-type AWS_LAMBDA --lambda-authorizer-config
 authorizerUri="arn:aws:lambda:us-east-2:111122223333:function:my-function"
```

> **Note**
>
> Anda juga dapat menyertakan opsi konfigurasi lain seperti ekspresi reguler token.

Contoh berikut menjelaskan fungsi Lambda yang menunjukkan berbagai otentikasi dan status kegagalan yang dapat dimiliki fungsi Lambda saat digunakan sebagai mekanisme otorisasi: AWS AppSync

```
def handler(event, context):
  # This is the authorization token passed by the client
  token = event.get('authorizationToken')
  # If a lambda authorizer throws an exception, it will be treated as unauthorized.
  if 'Fail' in token:
    raise Exception('Purposefully thrown exception in Lambda Authorizer.')

  if 'Authorized' in token and 'ReturnContext' in token:
    return {
      'isAuthorized': True,
      'resolverContext': {
        'key': 'value'
      }
    }

  # Authorized with no f
  if 'Authorized' in token:
    return {
```

```
    'isAuthorized': True
  }
# Partial authorization
if 'Partial' in token:
  return {
    'isAuthorized': True,
    'deniedFields':['user.favoriteColor']
  }
if 'NeverCache' in token:
  return {
    'isAuthorized': True,
    'ttlOverride': 0
  }
if 'Unauthorized' in token:
  return {
    'isAuthorized': False
  }
# if nothing is returned, then the authorization fails.
return {}
```

Menghindari batasan otorisasi token SiGv4 dan OIDC

Metode berikut dapat digunakan untuk menghindari masalah tidak dapat menggunakan tanda tangan SigV4 atau token OIDC Anda sebagai token otorisasi Lambda Anda ketika mode otorisasi tertentu diaktifkan.

Jika Anda ingin menggunakan tanda tangan Sigv4 sebagai token otorisasi Lambda saat mode `AWS_IAM` dan `AWS_LAMBDA` otorisasi diaktifkan untuk API, lakukan AWS AppSync hal berikut:

- Untuk membuat token otorisasi Lambda baru, tambahkan sufiks dan/atau awalan acak ke tanda tangan SigV4.
- Untuk mengambil tanda tangan SiGv4 asli, perbarui fungsi Lambda Anda dengan menghapus awalan acak dan/atau sufiks dari token otorisasi Lambda. Kemudian, gunakan tanda tangan SiGv4 asli untuk otentikasi.

Jika Anda ingin menggunakan token OIDC sebagai token otorisasi Lambda saat mode otorisasi atau mode `OPENID_CONNECT` otorisasi `AMAZON_COGNITO_USER_POOLS` dan diaktifkan untuk `AWS_LAMBDA` AWS AppSync API, lakukan hal berikut:

- Untuk membuat token otorisasi Lambda baru, tambahkan sufiks dan/atau awalan acak ke token OIDC. Token otorisasi Lambda tidak boleh berisi awalan skema Pembawa.
- Untuk mengambil token OIDC asli, perbarui fungsi Lambda Anda dengan menghapus awalan acak dan/atau sufiks dari token otorisasi Lambda. Kemudian, gunakan token OIDC asli untuk otentikasi.

Otorisasi AWS_IAM

Jenis otorisasi ini memberlakukan proses penandatanganan versi AWS tanda tangan 4 pada GraphQL API. Anda dapat mengaitkan kebijakan akses Identity and Access Management (IAM) dengan jenis otorisasi ini. Aplikasi Anda dapat memanfaatkan asosiasi ini dengan menggunakan kunci akses (yang terdiri dari ID kunci akses dan kunci akses rahasia) atau dengan menggunakan kredensi sementara yang berumur pendek yang disediakan oleh Amazon Cognito Federated Identities.

Jika Anda menginginkan peran yang memiliki akses untuk melakukan semua operasi data:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "appsync:GraphQL"
         ],
         "Resource": [
            "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/*"
         ]
      }
   ]
}
```

Anda dapat menemukan `YourGraphQLApiId` dari halaman daftar API utama di AppSync konsol, langsung di bawah nama API Anda. Atau Anda dapat mengambilnya dengan CLI: `aws appsync list-graphql-apis`

Jika Anda ingin membatasi akses ke operasi GraphQL tertentu saja, Anda dapat melakukan ini untuk `Query` root`Mutation`,, dan bidang. `Subscription`

```
{
   "Version": "2012-10-17",
```

```
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "appsync:GraphQL"
         ],
         "Resource": [
            "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/types/Query/
fields/<Field-1>",
            "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/types/Query/
fields/<Field-2>",
            "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/types/
Mutation/fields/<Field-1>",
            "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/types/
Subscription/fields/<Field-1>"
         ]
     }
   ]
}
```

Misalnya, Anda memiliki skema berikut dan Anda ingin membatasi akses untuk mendapatkan semua posting:

```
schema {
   query: Query
   mutation: Mutation
}

type Query {
   posts:[Post!]!
}

type Mutation {
   addPost(id:ID!, title:String!):Post!
}
```

Kebijakan IAM terkait untuk suatu peran (yang dapat Anda lampirkan ke kumpulan identitas Amazon Cognito, misalnya) akan terlihat seperti berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
            "appsync:GraphQL"
            ],
            "Resource": [
                "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/types/
Query/fields/posts"
            ]
        }
    ]
}
```

Otorisasi OPENID_CONNECT

Jenis otorisasi ini memberlakukan token OpenID connect (OIDC) yang disediakan oleh layanan yang sesuai dengan OIDC. Aplikasi Anda dapat memanfaatkan pengguna dan hak istimewa yang ditentukan oleh penyedia OIDC Anda untuk mengontrol akses.

URL Penerbit adalah satu-satunya nilai konfigurasi wajib yang Anda berikan AWS AppSync (misalnya,`https://auth.example.com`). URL ini harus dapat dialamatkan melalui HTTPS. AWS AppSync menambahkan `/.well-known/openid-configuration` ke URL penerbit dan menempatkan konfigurasi OpenID sesuai spesifikasi OpenID Connect `https://auth.example.com/.well-known/openid-configuration` Discovery. Ia mengharapkan untuk mengambil dokumen JSON yang sesuai dengan RFC5785 di URL ini. Dokumen JSON ini harus berisi `jwks_uri` kunci, yang menunjuk ke dokumen JSON Web Key Set (JWKS) dengan kunci penandatanganan. AWS AppSync mengharuskan JWKS berisi bidang JSON dan. `kty` `kid`

AWS AppSync mendukung berbagai algoritma penandatanganan.

Algoritme penandatanganan
RS256
RS384
RS512
PS256
PS384

Algoritme penandatanganan
PS512
HS256
HS384
HS512
ES256
ES384
ES512

Kami menyarankan Anda menggunakan algoritma RSA. Token yang dikeluarkan oleh penyedia harus mencakup waktu di mana token dikeluarkan (`iat`) dan dapat mencakup waktu di mana token itu diautentikasi (`auth_time`). Anda dapat memberikan nilai TTL untuk waktu yang dikeluarkan (`iatTTL`) dan waktu otentikasi (`authTTL`) dalam konfigurasi OpenID Connect untuk validasi tambahan. Jika penyedia Anda mengotorisasi beberapa aplikasi, Anda juga dapat memberikan ekspresi reguler (`clientId`) yang digunakan untuk mengotorisasi oleh ID klien. Ketika `clientId` ada dalam konfigurasi OpenID Connect Anda, AWS AppSync validasi klaim dengan mewajibkan `clientId` untuk mencocokkan dengan `azp` klaim `aud` atau dalam token.

Untuk memvalidasi beberapa ID klien, gunakan operator pipeline (“|”) yang merupakan “atau” dalam ekspresi reguler. Misalnya, jika aplikasi OIDC Anda memiliki empat klien dengan ID klien seperti 0A1S2D, 1F4G9H, 1J6L4B, 6GS5MG, untuk memvalidasi hanya tiga ID klien pertama, Anda akan menempatkan 1F4G9H | 1J6L4B | 6GS5MG di bidang ID klien.

Otorisasi AMAZON_COGNITO_USER_POOLS

Jenis otorisasi ini memberlakukan token OIDC yang disediakan oleh Amazon Cognito User Pools. Aplikasi Anda dapat memanfaatkan pengguna dan grup di kumpulan pengguna Anda dan mengaitkannya dengan bidang GraphQL untuk mengontrol akses.

Saat menggunakan Kumpulan Pengguna Amazon Cognito, Anda dapat membuat grup yang menjadi milik pengguna. Informasi ini dikodekan dalam token JWT yang dikirimkan aplikasi Anda AWS

AppSync dalam header otorisasi saat mengirim operasi GraphQL. Anda dapat menggunakan arahan GraphQL pada skema untuk mengontrol grup mana yang dapat memanggil resolver mana di lapangan, sehingga memberikan akses yang lebih terkontrol ke pelanggan Anda.

Misalnya, Anda memiliki skema GraphQL berikut:

```
schema {
   query: Query
   mutation: Mutation
}

type Query {
   posts:[Post!]!
}

type Mutation {
   addPost(id:ID!, title:String!):Post!
}
...
```

Jika Anda memiliki dua grup di Amazon Cognito User Pools - blogger dan pembaca - dan Anda ingin membatasi pembaca sehingga mereka tidak dapat menambahkan entri baru, maka skema Anda akan terlihat seperti ini:

```
schema {
   query: Query
   mutation: Mutation
}
```

```
type Query {
   posts:[Post!]!
   @aws_auth(cognito_groups: ["Bloggers", "Readers"])
}

type Mutation {
   addPost(id:ID!, title:String!):Post!
   @aws_auth(cognito_groups: ["Bloggers"])
}
...
```

Perhatikan bahwa Anda dapat menghilangkan `@aws_auth` arahan jika Anda ingin default ke grant-or-deny strategi akses tertentu. Anda dapat menentukan grant-or-deny strategi dalam konfigurasi kumpulan pengguna saat membuat GraphQL API melalui konsol atau melalui perintah CLI berikut:

```
$ aws appsync --region us-west-2 create-graphql-api --authentication-
type AMAZON_COGNITO_USER_POOLS  --name userpoolstest --user-pool-config
 '{ "userPoolId":"test", "defaultEffect":"ALLOW", "awsRegion":"us-west-2"}'
```

Menggunakan mode otorisasi tambahan

Saat Anda menambahkan mode otorisasi tambahan, Anda dapat langsung mengonfigurasi pengaturan otorisasi di level API AWS AppSync GraphQL (yaitu, `authenticationType` bidang yang dapat Anda konfigurasikan langsung pada `GraphqlApi` objek) dan bertindak sebagai default pada skema. Ini berarti bahwa jenis apa pun yang tidak memiliki arahan khusus harus melewati pengaturan otorisasi tingkat API.

Pada tingkat skema, Anda dapat menentukan mode otorisasi tambahan menggunakan arahan pada skema. Anda dapat menentukan mode otorisasi pada bidang individual dalam skema. Misalnya, untuk `API_KEY` otorisasi yang akan Anda gunakan `@aws_api_key` pada definisi/bidang tipe objek skema. Arahan berikut didukung pada bidang skema dan definisi tipe objek:

- `@aws_api_key`- Untuk menentukan bidang `API_KEY` diotorisasi.
- `@aws_iam`- Untuk menentukan bahwa bidang tersebut `AWS_IAM` diotorisasi.
- `@aws_oidc`- Untuk menentukan bahwa bidang tersebut `OPENID_CONNECT` diotorisasi.
- `@aws_cognito_user_pools`- Untuk menentukan bahwa bidang tersebut `AMAZON_COGNITO_USER_POOLS` diotorisasi.
- `@aws_lambda`- Untuk menentukan bahwa bidang tersebut `AWS_LAMBDA` diotorisasi.

Anda tidak dapat menggunakan `@aws_auth` arahan bersama dengan mode otorisasi tambahan. `@aws_auth`hanya berfungsi dalam konteks `AMAZON_COGNITO_USER_POOLS` otorisasi tanpa mode otorisasi tambahan. Namun, Anda dapat menggunakan `@aws_cognito_user_pools` direktif sebagai pengganti `@aws_auth` direktif, menggunakan argumen yang sama. Perbedaan utama antara keduanya adalah bahwa Anda dapat menentukan `@aws_cognito_user_pools` pada setiap bidang dan definisi jenis objek.

Untuk memahami bagaimana mode otorisasi tambahan bekerja dan bagaimana mereka dapat ditentukan pada skema, mari kita lihat skema berikut:

```
schema {
   query: Query
   mutation: Mutation
}

type Query {
   getPost(id: ID): Post
   getAllPosts(): [Post]
   @aws_api_key
}

type Mutation {
   addPost(
      id: ID!
      author: String!
      title: String!
      content: String!
      url: String!
   ): Post!
}

type Post @aws_api_key @aws_iam {
   id: ID!
   author: String
   title: String
   content: String
   url: String
   ups: Int!
   downs: Int!
   version: Int!
}
...
```

Untuk skema ini, asumsikan bahwa itu `AWS_IAM` adalah jenis otorisasi default pada AWS AppSync GraphQL API. Ini berarti bahwa bidang yang tidak memiliki arahan dilindungi menggunakan`AWS_IAM`. Misalnya, itulah kasus untuk `getPost` bidang pada `Query` tipe. Arahan skema memungkinkan Anda untuk menggunakan lebih dari satu mode otorisasi. Misalnya, Anda dapat `API_KEY` mengonfigurasi sebagai mode otorisasi tambahan pada AWS AppSync GraphQL API, dan Anda dapat menandai bidang menggunakan arahan (misalnya, `@aws_api_key` dalam contoh `getAllPosts` ini). Arahan bekerja di tingkat lapangan sehingga Anda perlu memberikan `API_KEY` akses ke `Post` jenis juga.

Anda dapat melakukan ini dengan menandai setiap bidang dalam `Post` tipe dengan direktif, atau dengan menandai `Post` tipe dengan `@aws_api_key` direktif.

Untuk lebih membatasi akses ke bidang dalam `Post` jenis, Anda dapat menggunakan arahan terhadap bidang individual dalam `Post` jenis seperti yang ditunjukkan berikut.

Misalnya, Anda dapat menambahkan `restrictedContent` bidang ke `Post` jenis dan membatasi akses ke sana dengan menggunakan `@aws_iam` direktif. `AWS_IAM`Permintaan yang diautentikasi dapat mengaksesrestrictedContent, namun `API_KEY` permintaan tidak akan dapat mengaksesnya.

```
type Post @aws_api_key @aws_iam{
   id: ID!
   author: String
   title: String
   content: String
   url: String
   ups: Int!
   downs: Int!
   version: Int!
   restrictedContent: String!
   @aws_iam
}
...
```

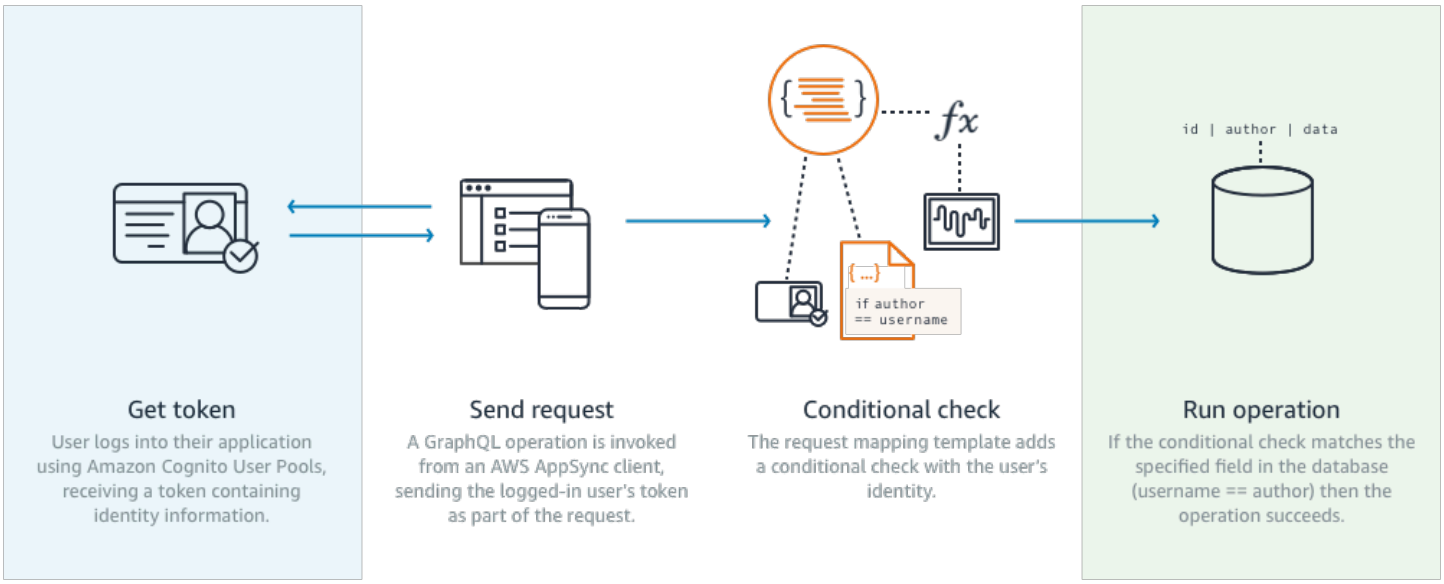
Kontrol akses detail

Informasi sebelumnya menunjukkan cara membatasi atau memberikan akses ke bidang GraphQL tertentu. Jika Anda ingin mengatur kontrol akses pada data berdasarkan kondisi tertentu (misalnya, berdasarkan pengguna yang melakukan panggilan dan apakah pengguna memiliki data), Anda dapat menggunakan templat pemetaan di resolver Anda. Anda juga dapat melakukan logika bisnis yang lebih kompleks, yang kami jelaskan dalam Memfilter Informasi.

Bagian ini menunjukkan cara mengatur kontrol akses pada data Anda menggunakan template pemetaan DynamoDB resolver.

Sebelum melanjutkan lebih jauh, jika Anda tidak terbiasa dengan template pemetaan di AWS AppSync, Anda mungkin ingin meninjau referensi template pemetaan Resolver dan referensi template pemetaan Resolver untuk DynamoDB.

Dalam contoh berikut menggunakan DynamoDB, misalkan Anda menggunakan skema posting blog sebelumnya, dan hanya pengguna yang membuat posting yang diizinkan untuk mengeditnya. Proses evaluasi akan bagi pengguna untuk mendapatkan kredensil dalam aplikasi mereka, menggunakan Amazon Cognito User Pools misalnya, dan kemudian meneruskan kredensil ini sebagai bagian dari operasi GraphQL. Template pemetaan kemudian akan menggantikan nilai dari kredensional (seperti nama pengguna) dalam pernyataan bersyarat yang kemudian akan dibandingkan dengan nilai dalam database Anda.



Untuk menambahkan fungsi ini, tambahkan bidang `editPost` GraphQL sebagai berikut:

```
schema {
   query: Query
   mutation: Mutation
}

type Query {
   posts:[Post!]!
}

type Mutation {
   editPost(id:ID!, title:String, content:String):Post
   addPost(id:ID!, title:String!):Post!
}
...
```

Template pemetaan resolver untuk `editPost` (ditampilkan dalam contoh di akhir bagian ini) perlu melakukan pemeriksaan logis terhadap penyimpanan data Anda untuk mengizinkan hanya pengguna

yang membuat posting untuk mengeditnya. Karena ini adalah operasi edit, ini sesuai dengan DynamoDB `UpdateItem` di. Anda dapat melakukan pemeriksaan bersyarat sebelum melakukan tindakan ini, menggunakan konteks yang diteruskan untuk validasi identitas pengguna. Ini disimpan dalam `Identity` objek yang memiliki nilai-nilai berikut:

```
{
   "accountId" : "12321434323",
   "cognitoIdentityPoolId" : "",
   "cognitoIdentityId" : "",
   "sourceIP" : "",
   "caller" : "ThisistheprincipalARN",
   "username" : "username",
   "userArn" : "Sameasabove"
}
```

Untuk menggunakan objek ini dalam panggilan `UpdateItem` DynamoDB, Anda perlu menyimpan informasi identitas pengguna dalam tabel untuk perbandingan. Pertama, `addPost` mutasi Anda perlu menyimpan pencipta. Kedua, `editPost` mutasi Anda perlu melakukan pemeriksaan bersyarat sebelum memperbarui.

Berikut adalah contoh kode resolver `addPost` yang menyimpan identitas pengguna sebagai `Author` kolom:

```
import { util, Context } from '@aws-appsync/utils';
import { put } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const { id: postId, ...item } = ctx.args;
 return put({
  key: { postId },
  item: { ...item, Author: ctx.identity.username },
  condition: { postId: { attributeExists: false } },
 });
}

export const response = (ctx) => ctx.result;
```

Perhatikan bahwa `Author` atribut diisi dari `Identity` objek, yang berasal dari aplikasi.

Terakhir, berikut adalah contoh kode resolver untuk`editPost`, yang hanya memperbarui konten posting blog jika permintaan berasal dari pengguna yang membuat posting:

```
import { util, Context } from '@aws-appsync/utils';
import { put } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const { id, ...item } = ctx.args;
 return put({
  key: { id },
  item,
  condition: { author: { contains: ctx.identity.username } },
 });
}

export const response = (ctx) => ctx.result;
```

Contoh ini menggunakan a `PutItem` yang menimpa semua nilai daripada`UpdateItem`, tetapi konsep yang sama berlaku pada blok `condition` pernyataan.

Memfilter informasi

Mungkin ada kasus di mana Anda tidak dapat mengontrol respons dari sumber data Anda, tetapi Anda tidak ingin mengirim informasi yang tidak perlu kepada klien saat berhasil menulis atau membaca ke sumber data. Dalam kasus ini, Anda dapat memfilter informasi dengan menggunakan template pemetaan respons.

Misalnya, Anda tidak memiliki indeks yang sesuai pada tabel DynamoDB posting blog Anda (seperti indeks pada). `Author` Anda dapat menggunakan resolver berikut:

```
import { util, Context } from '@aws-appsync/utils';
import { get } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 return get({ key: { ctx.args.id } });
}

export function response(ctx) {
 if (ctx.result.author === ctx.identity.username) {
  return ctx.result;
 }
 return null;
}
```

Handler permintaan mengambil item bahkan jika pemanggil bukan penulis yang membuat posting. Untuk mencegah hal ini mengembalikan semua data, penangan respons memeriksa untuk memastikan pemanggil cocok dengan penulis item. Jika pemanggil tidak cocok dengan pemeriksaan ini, hanya respons nol yang dikembalikan.

Akses sumber data

AWS AppSync berkomunikasi dengan sumber data menggunakan peran Identity and Access Management (IAM) dan kebijakan akses. Jika Anda menggunakan peran yang ada, Kebijakan Kepercayaan perlu ditambahkan AWS AppSync agar dapat mengambil peran tersebut. Hubungan kepercayaan akan terlihat seperti di bawah ini:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "appsync.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Penting untuk mengurangi kebijakan akses pada peran agar hanya memiliki izin untuk bertindak berdasarkan kumpulan sumber daya minimal yang diperlukan. Saat menggunakan AppSync konsol untuk membuat sumber data dan membuat peran, ini dilakukan secara otomatis untuk Anda. Namun saat menggunakan templat sampel bawaan dari konsol IAM untuk membuat peran di luar AWS AppSync konsol, izin tidak akan dicakup secara otomatis pada sumber daya dan Anda harus melakukan tindakan ini sebelum memindahkan aplikasi ke produksi.

Kasus penggunaan otorisasi

Di bagian Keamanan, Anda mempelajari tentang mode Otorisasi yang berbeda untuk melindungi API Anda dan pengantar diberikan pada mekanisme Fine Grained Authorization untuk memahami konsep dan alur. KarenaAWS AppSync memungkinkan Anda untuk melakukan operasi penuh logika pada data melalui penggunaan template Pemetaan Resolver GraphQL, Anda dapat melindungi data

saat membaca atau menulis dengan cara yang sangat fleksibel menggunakan kombinasi identitas pengguna, kondisional, dan injeksi data.

Jika Anda tidak terbiasa dengan mengeditAWS AppSync Resolvers, tinjau panduan pemrograman.

Gambaran Umum

Pemberian akses ke data dalam sistem secara tradisional dilakukan melalui matriks kontrol akses di mana persimpangan baris (sumber daya) dan kolom (pengguna/peran) adalah izin yang diberikan.

AWS AppSync menggunakan sumber daya di akun Anda sendiri dan informasi identitas benang (pengguna/peran) ke dalam permintaan GraphQL dan respons sebagai objek konteks, yang dapat Anda gunakan dalam resolver. Ini berarti bahwa izin dapat diberikan dengan tepat baik pada operasi tulis atau baca berdasarkan logika resolver. Jika logika ini berada pada tingkat sumber daya, misalnya hanya pengguna atau grup tertentu yang dapat membaca/menulis ke baris database tertentu, maka “metadata otorisasi” harus disimpan. AWS AppSync tidak menyimpan data apa pun sehingga Anda harus menyimpan metadata otorisasi ini dengan sumber daya sehingga izin dapat dihitung. Metadata otorisasi biasanya merupakan atribut (kolom) dalam tabel DynamoDB, seperti pemilik atau daftar pengguna/grup. Misalnya mungkin ada atribut Pembaca dan Penulis.

Dari tingkat tinggi, artinya ini adalah bahwa jika Anda membaca item individual dari sumber data, Anda melakukan`#if () ... #end` pernyataan kondisional dalam template respons setelah resolver membaca dari sumber data. Pemeriksaan biasanya akan menggunakan nilai pengguna atau grup`$context.identity` untuk pemeriksaan keanggotaan terhadap metadata otorisasi yang dikembalikan dari operasi baca. Untuk beberapa catatan, seperti daftar yang dikembalikan dari tabel`Scan` atau`Query`, Anda akan mengirimkan pemeriksaan kondisi sebagai bagian dari operasi ke sumber data menggunakan nilai pengguna atau grup yang serupa.

Demikian pula ketika menulis data Anda akan menerapkan pernyataan kondisional untuk tindakan (seperti`PutItem` atau`UpdateItem` untuk melihat apakah pengguna atau kelompok membuat mutasi memiliki izin. Bersyarat lagi akan berkali-kali menggunakan nilai`$context.identity` untuk membandingkan dengan metadata otorisasi pada sumber daya itu. Untuk template permintaan dan respons, Anda juga dapat menggunakan header khusus dari klien untuk melakukan pemeriksaan validasi.

Membaca Data

Seperti yang diuraikan di atas metadata otorisasi untuk melakukan pemeriksaan harus disimpan dengan sumber daya atau diteruskan ke permintaan GraphQL (identitas, header, dll.). Untuk menunjukkan ini misalkan Anda memiliki tabel DynamoDB di bawah ini:

ID	Data	PeopleCanAccess	GroupsCanAccess	Owner
123	{my: data,...}	[Mary, Joe]	[Admins, Editors]	Nadia

Kunci utama adalah`id` dan data yang akan diakses adalah`Data`. Kolom lainnya adalah contoh pemeriksaan yang dapat Anda lakukan untuk otorisasi. `Owner`akan menjadi`String` sementara`PeopleCanAccess` dan`GroupsCanAccess` akan`String Sets` seperti yang diuraikan dalam referensi template pemetaan Resolver untuk DynamoDB.

Dalam gambaran template pemetaan resolver diagram menunjukkan bagaimana template respon tidak hanya berisi objek konteks tetapi juga hasil dari sumber data. Untuk kueri GraphQL dari masing-masing item, Anda dapat menggunakan template respons untuk memeriksa apakah pengguna diizinkan untuk melihat hasil ini atau mengembalikan pesan kesalahan otorisasi. Ini terkadang disebut sebagai “Filter Otorisasi”. Untuk permintaan GraphQL kembali daftar, menggunakan Scan atau Query, itu lebih performant untuk melakukan pemeriksaan pada permintaan template dan kembali data hanya jika kondisi otorisasi puas. Implementasinya kemudian:

1. GetItem - pemeriksaan otorisasi untuk catatan individu. Selesai menggunakan`#if() ... #end` pernyataan.
2. operasi Scan/Query - cek otorisasi adalah`"filter":{"expression":...}` pernyataan. Pemeriksaan umum adalah kesetaraan (`attribute = :input`) atau memeriksa apakah nilai ada dalam daftar (`contains(attribute, :input)`).

Dalam #2`attribute` dalam kedua pernyataan mewakili nama kolom catatan dalam tabel, seperti`owner` dalam contoh kami di atas. Anda dapat alias ini dengan# tanda dan penggunaan`"expressionNames":{...}` tetapi itu tidak wajib. `:input`Ini adalah referensi ke nilai yang Anda bandingkan dengan atribut database, yang akan Anda tentukan`"expressionValues": {...}`. Anda akan melihat contoh-contoh di bawah ini.

Kasus penggunaan: pemilik dapat membaca

Menggunakan tabel di atas, jika Anda hanya ingin mengembalikan data jika`Owner == Nadia` untuk operasi baca individu (`GetItem`) template Anda akan terlihat seperti:

```
#if($context.result["Owner"] == $context.identity.username)
    $utils.toJson($context.result)
#else
    $utils.unauthorized()
#end
```

Beberapa hal untuk disebutkan di sini yang akan digunakan kembali di bagian yang tersisa. Pertama, pemeriksaan menggunakan nama pendaftaran pengguna`$context.identity.username` yang ramah jika kumpulan pengguna Amazon Cognito digunakan dan akan menjadi identitas pengguna jika IAM digunakan (termasuk Amazon Cognito Federated Identities). Ada nilai lain yang harus disimpan untuk pemilik seperti nilai unik “Amazon Cognito identity”, yang berguna saat federasi login dari beberapa lokasi, dan Anda harus meninjau opsi yang tersedia di Referensi Konteks Template Pemetaan Resolver.

Kedua, pemeriksaan lain bersyarat yang merespons sepenuhnya opsional tetapi direkomendasikan sebagai praktik terbaik saat merancang API GraphQL Anda.`$util.unauthorized()`

Kasus penggunaan: akses khusus hardcode

```
// This checks if the user is part of the Admin group and makes the call
#foreach($group in $context.identity.claims.get("cognito:groups"))
    #if($group == "Admin")
        #set($inCognitoGroup = true)
    #end
#end
#if($inCognitoGroup)
{
    "version" : "2017-02-28",
    "operation" : "UpdateItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },
    "attributeValues" : {
        "owner" : $util.dynamodb.toDynamoDBJson($context.identity.username)
        #foreach( $entry in $context.arguments.entrySet() )
            ,"${entry.key}" : $util.dynamodb.toDynamoDBJson($entry.value)
```

```
        #end
    }
}
#else
    $utils.unauthorized()
#end
```

Kasus penggunaan: memfilter daftar hasil

Pada contoh sebelumnya, Anda dapat melakukan pemeriksaan terhadap`$context.result` secara langsung saat mengembalikan satu item, namun beberapa operasi seperti pemindaian akan mengembalikan beberapa item di`$context.result.items` mana Anda perlu melakukan filter otorisasi dan hanya mengembalikan hasil yang diizinkan dilihat pengguna. Misalkan`owner` bidang memiliki Amazon Cognito IdentityID kali ini ditetapkan pada catatan, Anda kemudian dapat menggunakan template pemetaan respons berikut untuk memfilter agar hanya menampilkan catatan yang dimiliki pengguna:

```
#set($myResults = [])
#foreach($item in $context.result.items)
    ##For userpools use $context.identity.username instead
    #if($item.Owner == $context.identity.cognitoIdentityId)
        #set($added = $myResults.add($item))
    #end
#end
$utils.toJson($myResults)
```

Kasus penggunaan: banyak orang dapat membaca

Opsi otorisasi populer lainnya adalah memungkinkan sekelompok orang untuk dapat membaca data. Dalam contoh di bawah ini`"filter":{"expression":...}` hanya mengembalikan nilai-nilai dari scan tabel jika pengguna menjalankan query GraphQL tercantum dalam set untuk`PeopleCanAccess`.

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "limit": #if(${context.arguments.count}) $util.toJson($context.arguments.count)
 #else 20 #end,
    "nextToken": #if(${context.arguments.nextToken})
 $util.toJson($context.arguments.nextToken) #else null #end,
```

```
    "filter":{
        "expression": "contains(#peopleCanAccess, :value)",
        "expressionNames": {
                "#peopleCanAccess": "peopleCanAccess"
        },
        "expressionValues": {
                ":value": $util.dynamodb.toDynamoDBJson($context.identity.username)
        }
    }
}
```

Kasus penggunaan: grup dapat membaca

Mirip dengan kasus penggunaan terakhir, mungkin hanya orang dalam satu kelompok atau lebih yang memiliki hak untuk membaca item tertentu dalam database. Penggunaan`"expression": "contains()"` operasi serupa namun logis-atau dari semua kelompok yang pengguna mungkin menjadi bagian dari yang perlu dipertanggungjawabkan dalam keanggotaan set. Dalam hal ini kita membangun sebuah`$expression` pernyataan di bawah ini untuk setiap kelompok pengguna berada dan kemudian lulus ini ke filter:

```
#set($expression = "")
#set($expressionValues = {})
#foreach($group in $context.identity.claims.get("cognito:groups"))
    #set( $expression = "${expression} contains(groupsCanAccess, :var
$foreach.count )" )
    #set( $val = {})
    #set( $test = $val.put("S", $group))
    #set( $values = $expressionValues.put(":var$foreach.count", $val))
    #if ( $foreach.hasNext )
    #set( $expression = "${expression} OR" )
    #end
#end
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "limit": #if(${context.arguments.count}) $util.toJson($context.arguments.count)
 #else 20 #end,
    "nextToken": #if(${context.arguments.nextToken})
 $util.toJson($context.arguments.nextToken) #else null #end,
    "filter":{
        "expression": "$expression",
        "expressionValues": $utils.toJson($expressionValues)
```

```
    }
}
```

Menulis Data

Menulis data pada mutasi selalu dikontrol pada template pemetaan permintaan. Dalam kasus sumber data DynamoDB, kuncinya adalah menggunakan yang sesuai`"condition": {"expression"...}` yang melakukan validasi terhadap metadata otorisasi dalam tabel itu. Dalam Keamanan, kami memberikan contoh yang dapat Anda gunakan untuk memeriksa`Author` bidang dalam tabel. Kasus penggunaan di bagian ini mengeksplorasi lebih banyak kasus penggunaan.

Kasus penggunaan: beberapa pemilik

Menggunakan diagram tabel contoh dari sebelumnya, misalkan`PeopleCanAccess` daftar

```
{
    "version" : "2017-02-28",
    "operation" : "UpdateItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },
    "update" : {
        "expression" : "SET meta = :meta",
        "expressionValues": {
            ":meta" : $util.dynamodb.toDynamoDBJson($ctx.args.meta)
        }
    },
    "condition" : {
        "expression"       : "contains(Owner,:expectedOwner)",
        "expressionValues" : {
            ":expectedOwner" :
 $util.dynamodb.toDynamoDBJson($context.identity.username)
        }
    }
}
```

Kasus penggunaan: grup dapat membuat catatan baru

```
#set($expression = "")
#set($expressionValues = {})
#foreach($group in $context.identity.claims.get("cognito:groups"))
```

```
    #set( $expression = "${expression} contains(groupsCanAccess, :var
$foreach.count )" )
    #set( $val = {})
    #set( $test = $val.put("S", $group))
    #set( $values = $expressionValues.put(":var$foreach.count", $val))
    #if ( $foreach.hasNext )
    #set( $expression = "${expression} OR" )
    #end
#end
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key" : {
        ## If your table's hash key is not named 'id', update it here. **
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
        ## If your table has a sort key, add it as an item here. **
    },
    "attributeValues" : {
        ## Add an item for each field you would like to store to Amazon DynamoDB. **
        "title" : $util.dynamodb.toDynamoDBJson($ctx.args.title),
        "content": $util.dynamodb.toDynamoDBJson($ctx.args.content),
        "owner": $util.dynamodb.toDynamoDBJson($context.identity.username)
    },
    "condition" : {
        "expression": $util.toJson("attribute_not_exists(id) AND $expression"),
        "expressionValues": $utils.toJson($expressionValues)
    }
}
```

Kasus penggunaan: grup dapat memperbarui rekaman yang ada

```
#set($expression = "")
#set($expressionValues = {})
#foreach($group in $context.identity.claims.get("cognito:groups"))
    #set( $expression = "${expression} contains(groupsCanAccess, :var
$foreach.count )" )
    #set( $val = {})
    #set( $test = $val.put("S", $group))
    #set( $values = $expressionValues.put(":var$foreach.count", $val))
    #if ( $foreach.hasNext )
    #set( $expression = "${expression} OR" )
    #end
#end
```

```
{
    "version" : "2017-02-28",
    "operation" : "UpdateItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },
    "update":{
                "expression" : "SET title = :title, content = :content",
        "expressionValues": {
            ":title" : $util.dynamodb.toDynamoDBJson($ctx.args.title),
            ":content" : $util.dynamodb.toDynamoDBJson($ctx.args.content)
        }
    },
    "condition" : {
        "expression": $util.toJson($expression),
        "expressionValues": $utils.toJson($expressionValues)
    }
}
```

Catatan publik pribadi

Dengan filter kondisional Anda juga dapat memilih untuk menandai data sebagai pribadi, publik atau beberapa pemeriksaan Boolean lainnya. Ini kemudian dapat digabungkan sebagai bagian dari filter otorisasi di dalam template respons. Menggunakan cek ini adalah cara yang bagus untuk menyembunyikan sementara data atau menghapusnya dari tampilan tanpa mencoba mengontrol keanggotaan grup.

Misalnya, Anda menambahkan atribut pada setiap item dalam tabel DynamoDB Anda yang disebut`public` dengan nilai`yes` atau`no`. Template respons berikut dapat digunakan pada`GetItem` panggilan untuk hanya menampilkan data jika pengguna berada dalam grup yang memiliki akses DAN jika data tersebut ditandai sebagai publik:

```
#set($permissions = $context.result.GroupsCanAccess)
#set($claimPermissions = $context.identity.claims.get("cognito:groups"))

#foreach($per in $permissions)
    #foreach($cgroups in $claimPermissions)
        #if($cgroups == $per)
            #set($hasPermission = true)
        #end
    #end
#end
```

```
#if($hasPermission && $context.result.public == 'yes')
    $utils.toJson($context.result)
#else
    $utils.unauthorized()
#end
```

Kode di atas juga bisa menggunakan OR logis (`||`) untuk memungkinkan orang membaca jika mereka memiliki izin untuk catatan atau jika itu publik:

```
#if($hasPermission || $context.result.public == 'yes')
    $utils.toJson($context.result)
#else
    $utils.unauthorized()
#end
```

Secara umum, Anda akan menemukan operator standar`==`,`!=`,`&&`, dan`||` membantu saat melakukan pemeriksaan otorisasi.

Data waktu nyata

Anda dapat menerapkan Kontrol Akses Berbutir Halus ke langganan GraphQL pada saat klien berlangganan, menggunakan teknik yang sama yang dijelaskan sebelumnya dalam dokumentasi ini. Anda melampirkan resolver ke bidang langganan, di mana titik Anda dapat melakukan kueri data dari sumber data dan melakukan logika kondisional baik dalam template pemetaan permintaan atau respons. Anda juga dapat mengembalikan data tambahan ke klien, seperti hasil awal dari langganan, selama struktur data cocok dengan jenis yang dikembalikan dalam langganan GraphQL Anda.

Kasus penggunaan: pengguna hanya dapat berlangganan percakapan tertentu

Kasus penggunaan umum untuk data real-time dengan langganan GraphQL adalah membangun aplikasi pesan atau obrolan pribadi. Saat membuat aplikasi obrolan yang memiliki banyak pengguna, percakapan dapat terjadi antara dua orang atau di antara beberapa orang. Ini mungkin dikelompokkan ke dalam “kamar”, yang bersifat pribadi atau publik. Dengan demikian, Anda hanya ingin mengizinkan pengguna untuk berlangganan percakapan (yang bisa menjadi satu lawan satu atau di antara grup) yang telah mereka berikan akses. Untuk tujuan demonstrasi, contoh di bawah ini menunjukkan kasus penggunaan sederhana dari satu pengguna yang mengirim pesan pribadi ke yang lain. Penyiapan memiliki dua tabel Amazon DynamoDB:

- Tabel pesan: (kunci utama)`toUser`, (kunci sortir)`id`

- Tabel izin: (kunci utama)`username`

Tabel Pesan menyimpan pesan aktual yang dikirim melalui mutasi GraphQL. Tabel Izin diperiksa oleh langganan GraphQL untuk otorisasi pada waktu koneksi klien. Contoh di bawah mengasumsikan Anda menggunakan skema GraphQL berikut:

```
input CreateUserPermissionsInput {
    user: String!
    isAuthorizedForSubscriptions: Boolean
}

type Message {
    id: ID
    toUser: String
    fromUser: String
    content: String
}

type MessageConnection {
    items: [Message]
    nextToken: String
}

type Mutation {
    sendMessage(toUser: String!, content: String!): Message
    createUserPermissions(input: CreateUserPermissionsInput!): UserPermissions
    updateUserPermissions(input: UpdateUserPermissionInput!): UserPermissions
}

type Query {
    getMyMessages(first: Int, after: String): MessageConnection
    getUserPermissions(user: String!): UserPermissions
}

type Subscription {
    newMessage(toUser: String!): Message
        @aws_subscribe(mutations: ["sendMessage"])
}

input UpdateUserPermissionInput {
    user: String!
    isAuthorizedForSubscriptions: Boolean
```

```
}

type UserPermissions {
    user: String
    isAuthorizedForSubscriptions: Boolean
}

schema {
    query: Query
    mutation: Mutation
    subscription: Subscription
}
```

Beberapa operasi standar, seperti`createUserPermissions()`, tidak tercakup di bawah ini untuk menggambarkan resolver langganan, tetapi merupakan implementasi standar resolver DynamoDB. Sebagai gantinya, kami akan fokus pada alur otorisasi berlangganan dengan resolver. Untuk mengirim pesan dari satu pengguna ke pengguna lain, lampirkan resolver ke`sendMessage()` bidang dan pilih sumber data tabel pesan dengan template permintaan berikut:

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key" : {
        "toUser" : $util.dynamodb.toDynamoDBJson($ctx.args.toUser),
        "id" : $util.dynamodb.toDynamoDBJson($util.autoId())
    },
    "attributeValues" : {
        "fromUser" : $util.dynamodb.toDynamoDBJson($context.identity.username),
        "content" : $util.dynamodb.toDynamoDBJson($ctx.args.content),
    }
}
```

Dalam contoh ini, kami menggunakan `$context.identity.username`. Ini mengembalikan informasi pengguna untukAWS Identity and Access Management atau pengguna Amazon Cognito. Template respon adalah passthrough sederhana`$util.toJson($ctx.result)`. Simpan dan kembali ke halaman skema. Kemudian lampirkan resolver untuk`newMessage()` langganan, menggunakan tabel Izin sebagai sumber data dan template pemetaan permintaan berikut:

```
{
    "version": "2018-05-29",
    "operation": "GetItem",
```

```
    "key": {
        "username": $util.dynamodb.toDynamoDBJson($ctx.identity.username),
    },
}
```

Kemudian gunakan template pemetaan respons berikut untuk melakukan pemeriksaan otorisasi Anda menggunakan data dari tabel Izin:

```
#if(! ${context.result})
    $utils.unauthorized()
#elseif(${context.identity.username} != ${context.arguments.toUser})
    $utils.unauthorized()
#elseif(! ${context.result.isAuthorizedForSubscriptions})
    $utils.unauthorized()
#else
##User is authorized, but we return null to continue
    null
#end
```

Dalam hal ini, Anda melakukan tiga pemeriksaan otorisasi. Yang pertama memastikan bahwa hasilnya dikembalikan. Yang kedua memastikan bahwa pengguna tidak berlangganan pesan yang dimaksudkan untuk orang lain. Yang ketiga memastikan bahwa pengguna diizinkan untuk berlangganan bidang apa pun, dengan memeriksa atribut DynamoDB yang`isAuthorizedForSubscriptions` disimpan sebagai `aB00L`.

Untuk menguji semuanya, Anda dapat masuk keAWS AppSync konsol menggunakan kumpulan pengguna Amazon Cognito dan pengguna bernama “Nadia”, lalu jalankan langganan GraphQL berikut:

```
subscription AuthorizedSubscription {
    newMessage(toUser: "Nadia") {
        id
        toUser
        fromUser
        content
    }
}
```

Jika dalam tabel Izin ada catatan untuk atribut`username` kunci`Nadia` dengan`isAuthorizedForSubscriptions` set ke`true`, Anda akan melihat respons yang berhasil.

Jika Anda mencoba yang berbeda`username` dalam`newMessage()` query di atas, kesalahan akan dikembalikan.

MenggunakanAWS WAFuntuk melindungi API Anda

AWS WAFadalah firewall aplikasi web yang membantu melindungi aplikasi web dan API dari serangan. Ini memungkinkan Anda untuk mengonfigurasi seperangkat aturan, yang disebut daftar kontrol akses web (web ACL), yang memungkinkan, memblokir, atau memantau (menghitung) permintaan web berdasarkan aturan dan kondisi keamanan web yang dapat disesuaikan yang Anda tentukan. Ketika Anda mengintegrasikanAWS AppSyncAPI denganAWS WAF, Anda mendapatkan lebih banyak kontrol dan visibilitas ke lalu lintas HTTP yang diterima oleh API Anda. Untuk mempelajari lebih lanjut tentangAWS WAF, lihatBagaimanaAWS WAFBekerjadiAWS WAFPanduan Pengembang.

Anda dapat menggunakanAWS WAFuntuk melindungi AndaAppSyncAPI dari eksploitasi web umum, seperti injeksi SQL dan serangan cross-site scripting (XSS). Ini dapat memengaruhi ketersediaan dan kinerja API, membahayakan keamanan, atau mengkonsumsi sumber daya yang berlebihan. Misalnya, Anda dapat membuat aturan untuk mengizinkan atau memblokir permintaan dari rentang alamat IP tertentu, permintaan dari blok CIDR, permintaan yang berasal dari negara atau wilayah tertentu, permintaan yang berisi kode SQL berbahaya, atau permintaan yang berisi skrip berbahaya.

Anda juga dapat membuat aturan yang cocok dengan string tertentu atau pola ekspresi reguler di header HTTP, metode, string kueri, URI, dan badan permintaan (terbatas pada 8 KB pertama). Selain itu, Anda dapat membuat aturan untuk memblokir serangan dari agen pengguna tertentu, bot buruk, dan pencakar konten. Misalnya, Anda dapat menggunakan aturan berbasis tarif untuk menentukan jumlah permintaan web yang diizinkan oleh setiap IP klien dalam periode 5 menit yang terus diperbarui.

Untuk mempelajari lebih lanjut tentang jenis aturan yang didukung dan tambahanAWS WAFfitur, lihatAWS WAFPanduan PengembangdanAWS WAFReferensi API.

> **Important**
>
> AWS WAFadalah garis pertahanan pertama Anda terhadap eksploitasi web. KapanAWS WAFdiaktifkan pada API,AWS WAFaturan dievaluasi sebelum fitur kontrol akses lainnya, seperti otorisasi kunci API, kebijakan IAM, token OIDC, dan kumpulan pengguna Amazon Cognito.

Integrasikan sebuahAppSyncAPI denganAWS WAF

Anda dapat mengintegrasikan API Appsync denganAWS WAFmenggunakanAWS Management Console, yangAWS CLI,AWS CloudFormation, atau klien lain yang kompatibel.

Untuk mengintegrasikan sebuahAWS AppSyncAPI denganAWS WAF

1. Buat sebuahAWS WAFweb ACL. Untuk langkah-langkah rinci menggunakanAWS WAFKonsol, lihatMembuat web ACL.
2. Tentukan aturan untuk ACL web. Aturan atau aturan didefinisikan dalam proses pembuatan ACL web. Untuk informasi tentang cara menyusun aturan, lihatAWS WAFaturan. Untuk contoh aturan berguna yang dapat Anda tentukan untukAWS AppSyncAPI, lihatMembuat aturan untuk ACL web.
3. Kaitkan ACL web denganAWS AppSyncAPI. Anda dapat melakukan langkah ini diAWS WAFKonsolatau diAppSyncKonsol.
   - Untuk mengaitkan ACL web denganAWS AppSyncAPI diAWS WAFKonsol, ikuti instruksi untukMengaitkan atau memisahkan ACL Web denganAWSsumber dayadiAWS WAFPanduan Pengembang.
   - Untuk mengaitkan ACL web denganAWS AppSyncAPI diAWS AppSyncKonsol
     a. Masuk keAWS Management Consoledan bukaAppSyncKonsol.
     b. Pilih API yang ingin Anda kaitkan dengan ACL web.
     c. Di panel navigasi, pilih Pengaturan.
     d. DalamFirewall aplikasi webbagian, nyalakanAktifkanAWS WAF.
     e. DalamWeb ACLdaftar dropdown, pilih nama ACL web untuk dikaitkan dengan API Anda.
     f. PilihSimpanuntuk mengaitkan ACL web dengan API Anda.

> **Note**
>
> Setelah Anda membuat ACL web diAWS WAFKonsol, dibutuhkan beberapa menit agar ACL web baru tersedia. Jika Anda tidak melihat ACL web yang baru dibuat diFirewall aplikasi webmenu, tunggu beberapa menit dan coba lagi langkah-langkah untuk mengaitkan ACL web dengan API Anda.

> **Note**
>
> AWS WAFIntegrasi hanya mendukung`Subscription registration message`acara untuk titik akhir waktu nyata.AWS AppSyncakan merespons dengan pesan kesalahan, bukan`start_ack`pesan untuk apapun`Subscription registration message`diblokir olehAWS WAF.

Setelah Anda mengaitkan ACL web denganAWS AppSyncAPI, Anda akan mengelola ACL web menggunakanAWS WAFAPI. Anda tidak perlu mengaitkan kembali ACL web denganAWS AppSyncAPI kecuali Anda ingin mengaitkanAWS AppSyncAPI dengan ACL web yang berbeda.

Membuat aturan untuk ACL web

Aturan menentukan cara memeriksa permintaan web dan apa yang harus dilakukan ketika permintaan web cocok dengan kriteria inspeksi. Aturan tidak ada diAWS WAFsendiri. Anda dapat mengakses aturan berdasarkan nama di grup aturan atau di ACL web tempat aturan ditentukan. Untuk informasi lebih lanjut, lihatAWS WAFaturan. Contoh berikut menunjukkan bagaimana mendefinisikan dan mengaitkan aturan yang berguna untuk melindungiAppSyncAPI.

Example aturan ACL web untuk membatasi ukuran badan permintaan

Berikut ini adalah contoh aturan yang membatasi ukuran badan permintaan. Ini akan dimasukkan ke dalamEditor aturan JSONsaat membuat ACL web diAWS WAFKonsol.

```
{
    "Name": "BodySizeRule",
    "Priority": 1,
    "Action": {
        "Block": {}
    },
    "Statement": {
        "SizeConstraintStatement": {
            "ComparisonOperator": "GE",
            "FieldToMatch": {
                "Body": {}
            },
            "Size": 1024,
            "TextTransformations": [
                {
                    "Priority": 0,
```

```
                    "Type": "NONE"
                }
            ]
        }
    },
    "VisibilityConfig": {
        "CloudWatchMetricsEnabled": true,
        "MetricName": "BodySizeRule",
        "SampledRequestsEnabled": true
     }
}
```

Setelah Anda membuat ACL web Anda menggunakan aturan contoh sebelumnya, Anda harus mengaitkannya denganAppSyncAPI. Sebagai alternatif untuk menggunakanAWS Management Console, Anda dapat melakukan langkah ini diAWS CLIdengan menjalankan perintah berikut.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Diperlukan beberapa menit agar perubahan menyebar, tetapi setelah menjalankan perintah ini, permintaan yang berisi badan yang lebih besar dari 1024 byte akan ditolak olehAWS AppSync.

> **Note**
>
> Setelah Anda membuat ACL web baru diAWS WAFKonsol, dibutuhkan beberapa menit agar ACL web tersedia untuk diasosiasikan dengan API. Jika Anda menjalankan perintah CLI dan mendapatkan`WAFUnavailableEntityException`kesalahan, tunggu beberapa menit dan coba lagi menjalankan perintah.

Example aturan ACL web untuk membatasi permintaan dari satu alamat IP

Berikut ini adalah contoh aturan yang membatasiAppSyncAPI untuk 100 permintaan dari satu alamat IP. Ini akan dimasukkan ke dalamEditor aturan JSONsaat membuat ACL web dengan aturan berbasis tarif diAWS WAFKonsol.

```
{
  "Name": "Throttle",
  "Priority": 0,
  "Action": {
    "Block": {}
```

```
  },
  "VisibilityConfig": {
    "SampledRequestsEnabled": true,
    "CloudWatchMetricsEnabled": true,
    "MetricName": "Throttle"
  },
  "Statement": {
    "RateBasedStatement": {
      "Limit": 100,
      "AggregateKeyType": "IP"
    }
  }
}
```

Setelah Anda membuat ACL web Anda menggunakan aturan contoh sebelumnya, Anda harus mengaitkannya denganAppSyncAPI. Anda dapat melakukan langkah ini diAWS CLIdengan menjalankan perintah berikut.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Example aturan ACL web untuk mencegah kueri introspeksi GraphQL __schema ke API

Berikut ini adalah contoh aturan yang mencegah kueri introspeksi GraphQL __schema ke API. Setiap badan HTTP yang menyertakan string “__schema” akan diblokir. Ini akan dimasukkan ke dalamEditor aturan JSONsaat membuat ACL web diAWS WAFKonsol.

```
{
  "Name": "BodyRule",
  "Priority": 5,
  "Action": {
    "Block": {}
  },
  "VisibilityConfig": {
    "SampledRequestsEnabled": true,
    "CloudWatchMetricsEnabled": true,
    "MetricName": "BodyRule"
  },
  "Statement": {
    "ByteMatchStatement": {
      "FieldToMatch": {
        "Body": {}
      },
```

```
      "PositionalConstraint": "CONTAINS",
      "SearchString": "__schema",
      "TextTransformations": [
        {
          "Type": "NONE",
          "Priority": 0
        }
      ]
    }
  }
}
```

Setelah Anda membuat ACL web Anda menggunakan aturan contoh sebelumnya, Anda harus mengaitkannya denganAppSyncAPI. Anda dapat melakukan langkah ini diAWS CLIdengan menjalankan perintah berikut.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Keamanan di AWS AppSync

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menjelaskan hal ini sebagai keamanan dari cloud dan keamanan dalam cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Auditor pihak ketiga secara teratur menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari Program AWS Kepatuhan Program AWS Kepatuhan . Untuk mempelajari tentang program kepatuhan yang berlaku AWS AppSync, lihat AWS Layanan dalam Lingkup oleh AWS Layanan Program Kepatuhan .
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain, yang mencakup sensitivitas data Anda, persyaratan perusahaan Anda, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan AWS AppSync. Topik berikut menunjukkan cara mengonfigurasi AWS AppSync untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga belajar cara menggunakan AWS layanan lain yang membantu Anda memantau dan mengamankan AWS AppSync sumber daya Anda.

Topik

Perlindungan data di AWS AppSync

Model tanggung jawab AWS bersama model berlaku untuk perlindungan data di AWS AppSync. Seperti yang dijelaskan dalam model AWS ini, bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk mempertahankan kendali atas konten yang di-host pada infrastruktur ini. Anda juga bertanggung jawab atas tugas-tugas konfigurasi dan manajemen keamanan untuk Layanan AWS yang Anda gunakan. Lihat informasi yang lebih lengkap tentang privasi data dalam Pertanyaan Umum Privasi Data. Lihat informasi tentang perlindungan data di Eropa di pos blog Model Tanggung Jawab Bersama dan GDPR AWS di Blog Keamanan AWS .

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi Akun AWS kredensyal dan mengatur pengguna individu dengan AWS IAM Identity Center atau AWS Identity and Access Management (IAM). Dengan cara itu, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugasnya. Kami juga menyarankan supaya Anda mengamankan data dengan cara-cara berikut:

- Gunakan autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan sumber daya. AWS Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Siapkan API dan pencatatan aktivitas pengguna dengan AWS CloudTrail.
- Gunakan solusi AWS enkripsi, bersama dengan semua kontrol keamanan default di dalamnya Layanan AWS.
- Gunakan layanan keamanan terkelola lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.
- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-2 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi yang lebih lengkap tentang titik akhir FIPS yang tersedia di Standar Pemrosesan Informasi Federal (FIPS) 140-2.

Kami sangat merekomendasikan agar Anda tidak pernah memasukkan informasi identifikasi yang sensitif, seperti nomor rekening pelanggan Anda, ke dalam tanda atau bidang isian bebas seperti bidang Nama. Ini termasuk saat Anda bekerja dengan AWS AppSync atau lainnya Layanan AWS menggunakan konsol, API AWS CLI, atau AWS SDK. Data apa pun yang Anda masukkan ke dalam tanda atau bidang isian bebas yang digunakan untuk nama dapat digunakan untuk log penagihan atau log diagnostik. Saat Anda memberikan URL ke server eksternal, kami sangat menganjurkan

supaya Anda tidak menyertakan informasi kredensial di dalam URL untuk memvalidasi permintaan Anda ke server itu.

Enkripsi bergerak

AWS AppSync, seperti semua AWS layanan, memanfaatkan TLS1.2 dan seterusnya untuk komunikasi saat menggunakan API dan SDK yang AWS dipublikasikan.

Menggunakan AWS AppSync dengan AWS layanan lain seperti Amazon DynamoDB memastikan enkripsi dalam perjalanan: AWS Semua layanan menggunakan TLS 1.2 dan seterusnya untuk berkomunikasi satu sama lain kecuali ditentukan lain. Untuk resolver yang menggunakan Amazon EC2 atau CloudFront, Anda bertanggung jawab untuk memverifikasi bahwa TLS (HTTPS) dikonfigurasi dan aman. Untuk informasi tentang mengonfigurasi HTTPS di Amazon EC2, lihat Mengonfigurasi SSL/TLS di Amazon Linux 2 di panduan pengguna Amazon EC2. Untuk informasi tentang mengonfigurasi HTTPS CloudFront, lihat HTTPS di Amazon CloudFront di panduan CloudFront pengguna.

Validasi kepatuhan untuk AWS AppSync

Auditor pihak ketiga menilai keamanan dan kepatuhan AWS AppSync sebagai bagian dari beberapa program AWS kepatuhan. AWS AppSync sesuai dengan program SOC, PCI, HIPAA/HIPAA BAA, IRAP, C5, ENS High, OSPAR, dan HITRUST CSF.

Untuk mempelajari apakah an Layanan AWS berada dalam lingkup program kepatuhan tertentu, lihat Layanan AWS di Lingkup oleh Program Kepatuhan Layanan AWS dan pilih program kepatuhan yang Anda minati. Untuk informasi umum, lihat Program AWS Kepatuhan Program AWS .

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Mengunduh Laporan di AWS Artifact .

Tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, dan hukum dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk membantu kepatuhan:

- Panduan Memulai Cepat Keamanan dan Kepatuhan — Panduan penerapan ini membahas pertimbangan arsitektur dan memberikan langkah-langkah untuk menerapkan lingkungan dasar AWS yang berfokus pada keamanan dan kepatuhan.

- Arsitektur untuk Keamanan dan Kepatuhan HIPAA di Amazon Web Services — Whitepaper ini menjelaskan bagaimana perusahaan dapat menggunakan AWS untuk membuat aplikasi yang memenuhi syarat HIPAA.

> **Note**
>
> Tidak semua memenuhi Layanan AWS syarat HIPAA. Untuk informasi selengkapnya, lihat Referensi Layanan yang Memenuhi Syarat HIPAA.

- AWS Sumber Daya AWS — Kumpulan buku kerja dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.
- AWS Panduan Kepatuhan Pelanggan - Memahami model tanggung jawab bersama melalui lensa kepatuhan. Panduan ini merangkum praktik terbaik untuk mengamankan Layanan AWS dan memetakan panduan untuk kontrol keamanan di berbagai kerangka kerja (termasuk Institut Standar dan Teknologi Nasional (NIST), Dewan Standar Keamanan Industri Kartu Pembayaran (PCI), dan Organisasi Internasional untuk Standardisasi (ISO)).
- Mengevaluasi Sumber Daya dengan Aturan dalam Panduan AWS Config Pengembang — AWS Config Layanan menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- AWS Security Hub— Ini Layanan AWS memberikan pandangan komprehensif tentang keadaan keamanan Anda di dalamnya AWS. Security Hub menggunakan kontrol keamanan untuk mengevaluasi sumber daya AWS Anda dan memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat Referensi kontrol Security Hub.
- AWS Audit ManagerIni Layanan AWS membantu Anda terus mengaudit AWS penggunaan Anda untuk menyederhanakan cara Anda mengelola risiko dan kepatuhan terhadap peraturan dan standar industri.

Keamanan infrastruktur di AWS AppSync

Sebagai layanan terkelola, AWS AppSync dilindungi oleh keamanan jaringan AWS global. Untuk informasi tentang layanan AWS keamanan dan cara AWS melindungi infrastruktur, lihat Keamanan AWS Cloud. Untuk mendesain AWS lingkungan Anda menggunakan praktik terbaik untuk keamanan infrastruktur, lihat Perlindungan Infrastruktur dalam Kerangka Kerja yang AWS Diarsiteksikan dengan Baik Pilar Keamanan.

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses AWS AppSync melalui jaringan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Sandi cocok dengan sistem kerahasiaan maju sempurna (perfect forward secrecy, PFS) seperti DHE (Ephemeral Diffie-Hellman) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani dengan menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan pengguna utama IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk menghasilkan kredensial keamanan sementara untuk menandatangani permintaan.

Ketahanan di AWS AppSync

Infrastruktur AWS global dibangun di sekitar AWS Wilayah dan Zona Ketersediaan. AWS Wilayah menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan fail over di antara zona tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur pusat data tunggal atau multi tradisional.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat Infrastruktur AWS Global.

Selain infrastruktur AWS global, AWS AppSync memungkinkan sebagian besar sumber daya didefinisikan menggunakan AWS CloudFormation templat; untuk contoh menggunakan AWS CloudFormation templat untuk mendeklarasikan AWS AppSync sumber daya, lihat Kasus penggunaan praktis untuk AWS AppSync Pipeline Resolvers di AWS blog dan Panduan Pengguna.AWS CloudFormation

Identitas dan manajemen akses untuk AWS AppSync

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang

dapat diautentikasi (masuk) dan diberi wewenang (memiliki izin) untuk menggunakan sumber daya. AWS AppSync IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan. AWS AppSync

Pengguna layanan — Jika Anda menggunakan AWS AppSync layanan untuk melakukan pekerjaan Anda, maka administrator Anda memberi Anda kredensyal dan izin yang Anda butuhkan. Saat Anda menggunakan lebih banyak AWS AppSync fitur untuk melakukan pekerjaan Anda, Anda mungkin memerlukan izin tambahan. Memahami cara akses dikelola dapat membantu Anda meminta izin yang tepat dari administrator Anda. Jika Anda tidak dapat mengakses fitur AWS AppSync, lihatMemecahkan masalah AWS AppSync identitas dan akses.

Administrator layanan — Jika Anda bertanggung jawab atas AWS AppSync sumber daya di perusahaan Anda, Anda mungkin memiliki akses penuh ke AWS AppSync. Tugas Anda adalah menentukan AWS AppSync fitur dan sumber daya mana yang harus diakses pengguna layanan Anda. Kemudian, Anda harus mengirimkan permintaan kepada administrator IAM Anda untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep Basic IAM. Untuk mempelajari lebih lanjut tentang bagaimana perusahaan Anda dapat menggunakan IAM AWS AppSync, lihatBagaimana AWS AppSync bekerja dengan IAM.

Administrator IAM - Jika Anda seorang administrator IAM, Anda mungkin ingin mempelajari detail tentang cara menulis kebijakan untuk mengelola akses. AWS AppSync Untuk melihat contoh kebijakan AWS AppSync berbasis identitas yang dapat Anda gunakan di IAM, lihat. Kebijakan berbasis identitas untuk AWS AppSync

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensyal identitas Anda. Anda harus diautentikasi (masuk ke AWS) sebagai Pengguna root akun AWS, sebagai pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk AWS sebagai identitas federasi dengan menggunakan kredensil yang disediakan melalui sumber identitas. AWS IAM Identity Center Pengguna (IAM Identity Center), autentikasi masuk tunggal perusahaan Anda, dan kredensyal Google atau Facebook Anda adalah contoh identitas federasi. Saat Anda masuk sebagai identitas gabungan, administrator Anda sebelumnya menyiapkan federasi identitas menggunakan peran IAM. Ketika Anda mengakses AWS dengan menggunakan federasi, Anda secara tidak langsung mengambil peran.

Bergantung pada jenis pengguna Anda, Anda dapat masuk ke AWS Management Console atau portal AWS akses. Untuk informasi selengkapnya tentang masuk AWS, lihat Cara masuk ke Panduan AWS Sign-In Pengguna Anda Akun AWS.

Jika Anda mengakses AWS secara terprogram, AWS sediakan kit pengembangan perangkat lunak (SDK) dan antarmuka baris perintah (CLI) untuk menandatangani permintaan Anda secara kriptografis dengan menggunakan kredensyal Anda. Jika Anda tidak menggunakan AWS alat, Anda harus menandatangani permintaan sendiri. Untuk informasi selengkapnya tentang penggunaan metode yang disarankan untuk menandatangani permintaan sendiri, lihat Menandatangani permintaan AWS API di Panduan Pengguna IAM.

Apa pun metode autentikasi yang digunakan, Anda mungkin diminta untuk menyediakan informasi keamanan tambahan. Misalnya, AWS merekomendasikan agar Anda menggunakan otentikasi multi-faktor (MFA) untuk meningkatkan keamanan akun Anda. Untuk mempelajari selengkapnya, lihat Autentikasi multi-faktor dalam Panduan Pengguna AWS IAM Identity Center dan Menggunakan autentikasi multi-faktor (MFA) di AWS dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya di akun. Identitas ini disebut pengguna Akun AWS root dan diakses dengan masuk dengan alamat email dan kata sandi yang Anda gunakan untuk membuat akun. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari Anda. Lindungi kredensial pengguna root Anda dan gunakan kredensial tersebut untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar lengkap tugas

yang mengharuskan Anda masuk sebagai pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Identitas terfederasi

Sebagai praktik terbaik, mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS dengan menggunakan kredensyal sementara.

Identitas federasi adalah pengguna dari direktori pengguna perusahaan Anda, penyedia identitas web, direktori Pusat Identitas AWS Directory Service, atau pengguna mana pun yang mengakses Layanan AWS dengan menggunakan kredensil yang disediakan melalui sumber identitas. Ketika identitas federasi mengakses Akun AWS, mereka mengambil peran, dan peran memberikan kredensi sementara.

Untuk manajemen akses terpusat, kami sarankan Anda menggunakan AWS IAM Identity Center. Anda dapat membuat pengguna dan grup di Pusat Identitas IAM, atau Anda dapat menghubungkan dan menyinkronkan ke sekumpulan pengguna dan grup di sumber identitas Anda sendiri untuk digunakan di semua aplikasi Akun AWS dan aplikasi Anda. Untuk informasi tentang Pusat Identitas IAM, lihat Apa itu Pusat Identitas IAM? dalam Panduan Pengguna AWS IAM Identity Center .

Pengguna dan grup IAM

Pengguna IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus untuk satu orang atau aplikasi. Jika memungkinkan, sebaiknya andalkan kredensial sementara daripada membuat pengguna IAM yang memiliki kredensial jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan khusus yang memerlukan kredensial jangka panjang dengan pengguna IAM, kami sarankan Anda merotasi kunci akses. Untuk informasi selengkapnya, lihat Rotasikan kunci akses secara rutin untuk kasus penggunaan yang memerlukan kredensial jangka panjang dalam Panduan Pengguna IAM.

Grup IAM adalah identitas yang menentukan kumpulan pengguna IAM. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin untuk beberapa pengguna sekaligus. Grup membuat izin lebih mudah dikelola untuk sekelompok besar pengguna. Misalnya, Anda dapat memiliki grup yang bernama IAMAdmins dan memberikan izin kepada grup tersebut untuk mengelola sumber daya IAM.

Pengguna berbeda dari peran. Pengguna secara unik terkait dengan satu orang atau aplikasi, tetapi peran dimaksudkan untuk dapat digunakan oleh siapa pun yang membutuhkannya. Pengguna memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk

mempelajari selengkapnya, silakan lihat Kapan harus membuat pengguna IAM (bukan peran) dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus. Peran ini mirip dengan pengguna IAM, tetapi tidak terkait dengan orang tertentu. Anda dapat mengambil peran IAM untuk sementara AWS Management Console dengan beralih peran. Anda dapat mengambil peran dengan memanggil operasi AWS CLI atau AWS API atau dengan menggunakan URL kustom. Untuk informasi selengkapnya tentang cara menggunakan peran, lihat Menggunakan peran IAM dalam Panduan Pengguna IAM.

Peran IAM dengan kredensial sementara berguna dalam situasi berikut:

- Akses pengguna gabungan – Untuk menetapkan izin ke sebuah identitas gabungan, Anda dapat membuat peran dan menentukan izin untuk peran tersebut. Saat identitas terfederasi mengautentikasi, identitas tersebut akan dikaitkan dengan peran dan diberi izin yang ditentukan oleh peran tersebut. Untuk informasi tentang peran-peran untuk federasi, lihat Membuat peran untuk Penyedia Identitas pihak ketiga dalam Panduan Pengguna IAM. Jika Anda menggunakan Pusat Identitas IAM, Anda perlu mengonfigurasi set izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah identitas tersebut diautentikasi, Pusat Identitas IAM mengorelasikan izin yang diatur ke peran dalam IAM. Untuk informasi tentang rangkaian izin, lihat Rangkaian izin dalam Panduan Pengguna AWS IAM Identity Center .
- Izin pengguna IAM sementara – Pengguna atau peran IAM dapat mengambil peran IAM guna mendapatkan berbagai izin secara sementara untuk tugas tertentu.
- Akses lintas akun – Anda dapat menggunakan peran IAM untuk mengizinkan seseorang (pengguna utama tepercaya) dengan akun berbeda untuk mengakses sumber daya yang ada di akun Anda. Peran adalah cara utama untuk memberikan akses lintas akun. Namun, dengan beberapa Layanan AWS, Anda dapat melampirkan kebijakan secara langsung ke sumber daya (alih-alih menggunakan peran sebagai proxy). Untuk mempelajari perbedaan antara kebijakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Bagaimana peran IAM berbeda dari kebijakan berbasis sumber daya dalam Panduan Pengguna IAM.
- Akses lintas layanan — Beberapa Layanan AWS menggunakan fitur lain Layanan AWS. Sebagai contoh, ketika Anda melakukan panggilan dalam suatu layanan, biasanya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Sebuah layanan mungkin melakukannya menggunakan izin prinsipal yang memanggil, menggunakan peran layanan, atau peran terkait layanan.

- Sesi akses teruskan (FAS) — Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Saat Anda menggunakan beberapa layanan, Anda mungkin melakukan tindakan yang kemudian memulai tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan saat membuat permintaan FAS, lihat Teruskan sesi akses.
- Peran layanan – Peran layanan adalah peran IAM yang diambil oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, memodifikasi, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.
- Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan dapat menggunakan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

- Aplikasi yang berjalan di Amazon EC2 — Anda dapat menggunakan peran IAM untuk mengelola kredensi sementara untuk aplikasi yang berjalan pada instans EC2 dan membuat atau permintaan API. AWS CLI AWS Cara ini lebih dianjurkan daripada menyimpan kunci akses dalam instans EC2. Untuk menetapkan AWS peran ke instans EC2 dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instance yang dilampirkan ke instance. Profil instans berisi peran dan memungkinkan program yang berjalan di instans EC2 mendapatkan kredensial sementara. Untuk informasi selengkapnya, lihat Menggunakan peran IAM untuk memberikan izin ke aplikasi yang berjalan di instans Amazon EC2 dalam Panduan Pengguna IAM.

Untuk mempelajari apakah kita harus menggunakan peran IAM atau pengguna IAM, lihat Kapan harus membuat peran IAM (bukan pengguna) dalam Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan adalah objek AWS yang, ketika dikaitkan dengan identitas atau sumber daya, menentukan izinnya. AWS mengevaluasi kebijakan ini ketika prinsipal (pengguna, pengguna root, atau sesi peran) membuat permintaan. Izin dalam kebijakan dapat menentukan permintaan yang diizinkan atau ditolak. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk

informasi selengkapnya tentang struktur dan konten dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, pengguna utama manakah yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Untuk memberikan izin kepada pengguna untuk melakukan tindakan pada sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat menjalankan peran.

Kebijakan IAM mendefinisikan izin untuk suatu tindakan terlepas dari metode yang Anda gunakan untuk operasi. Sebagai contoh, anggap saja Anda memiliki kebijakan yang mengizinkan tindakan `iam:GetRole`. Pengguna dengan kebijakan tersebut bisa mendapatkan informasi peran dari AWS Management Console, API AWS CLI, atau AWS API.

Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan pengguna dan peran, di sumber daya mana, dan dengan ketentuan apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Membuat kebijakan IAM dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan terkelola. Kebijakan inline disematkan langsung ke satu pengguna, grup, atau peran. Kebijakan terkelola adalah kebijakan mandiri yang dapat Anda lampirkan ke beberapa pengguna, grup, dan peran dalam. Akun AWS Kebijakan AWS terkelola mencakup kebijakan terkelola dan kebijakan yang dikelola pelanggan. Untuk mempelajari cara memilih antara kebijakan terkelola atau kebijakan inline, lihat Memilih antara kebijakan terkelola dan kebijakan inline dalam Panduan Pengguna IAM.

Kebijakan berbasis sumber daya

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya yang dilampiri kebijakan tersebut, kebijakan ini menentukan jenis tindakan yang dapat dilakukan oleh pengguna utama tertentu di sumber daya tersebut dan apa ketentuannya.

Anda harus menentukan pengguna utama dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Kebijakan berbasis sumber daya merupakan kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Daftar kontrol akses (ACL)

Daftar kontrol akses (ACL) mengendalikan pengguna utama mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL sama dengan kebijakan berbasis sumber daya, meskipun tidak menggunakan format dokumen kebijakan JSON.

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung ACL. Untuk mempelajari ACL selengkapnya, silakan lihat Gambaran umum daftar kontrol akses (ACL) di Panduan Developer Layanan Penyimpanan Ringkas Amazon.

Tipe kebijakan lain

AWS mendukung jenis kebijakan tambahan yang kurang umum. Tipe-tipe kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda berdasarkan tipe kebijakan yang lebih umum.

- Batasan izin – Batasan izin adalah fitur lanjutan di mana Anda menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM (pengguna atau peran IAM). Anda dapat menetapkan batasan izin untuk suatu entitas. Izin yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas milik entitas dan batasan izinnya. Kebijakan berbasis sumber daya yang menentukan pengguna atau peran dalam bidang `Principal` tidak dibatasi oleh batasan izin. Penolakan secara eksplisit terhadap salah satu kebijakan ini akan mengesampingkan izin tersebut. Untuk informasi selengkapnya tentang batasan izin, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCP) — SCP adalah kebijakan JSON yang menentukan izin maksimum untuk organisasi atau unit organisasi (OU) di. AWS Organizations AWS Organizations adalah layanan untuk mengelompokkan dan mengelola secara terpusat beberapa Akun AWS yang dimiliki bisnis Anda. Jika Anda mengaktifkan semua fitur di sebuah organisasi, maka Anda dapat menerapkan kebijakan kontrol layanan (SCP) ke salah satu atau semua akun Anda. SCP membatasi izin untuk entitas di akun anggota, termasuk masing-masing. Pengguna root akun AWS Untuk informasi selengkapnya tentang Organisasi dan SCP, lihat Cara kerja SCP dalam Panduan Pengguna AWS Organizations .
- Kebijakan sesi – Kebijakan sesi adalah kebijakan lanjutan yang Anda teruskan sebagai parameter saat Anda membuat sesi sementara secara terprogram untuk peran atau pengguna gabungan. Izin

sesi yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas pengguna atau peran dan kebijakan sesi. Izin juga bisa datang dari kebijakan berbasis sumber daya. Penolakan eksplisit dalam salah satu kebijakan ini membatalkan izin. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

Beberapa jenis kebijakan

Ketika beberapa jenis kebijakan berlaku untuk sebuah permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan Pengguna IAM.

Bagaimana AWS AppSync bekerja dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses AWS AppSync, pelajari fitur IAM yang tersedia untuk digunakan. AWS AppSync

Fitur IAM yang dapat Anda gunakan AWS AppSync

Fitur IAM	AWS AppSync dukungan
Kebijakan berbasis identitas	Ya
Kebijakan berbasis sumber daya	Tidak
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
Kunci persyaratan kebijakan	Tidak
ACL	Tidak
ABAC (tanda dalam kebijakan)	Parsial
Kredensial sementara	Ya
Sesi akses teruskan (FAS)	Parsial
Peran layanan	Tidak

Fitur IAM	AWS AppSync dukungan
Peran terkait layanan	Parsial

Untuk mendapatkan tampilan tingkat tinggi tentang cara AWS AppSync dan AWS layanan lain bekerja dengan sebagian besar fitur IAM, lihat AWS layanan yang bekerja dengan IAM di Panduan Pengguna IAM.

Kebijakan berbasis identitas untuk AWS AppSync

Mendukung kebijakan berbasis identitas	Ya

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan pengguna dan peran, di sumber daya mana, dan dengan ketentuan apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Membuat kebijakan IAM dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan tindakan dan sumber daya yang diizinkan atau ditolak, serta ketentuan terkait jenis tindakan yang diizinkan atau ditolak. Anda tidak dapat menentukan pengguna utama dalam kebijakan berbasis identitas karena kebijakan ini berlaku untuk pengguna atau peran yang dilampiri kebijakan. Untuk mempelajari semua elemen yang dapat digunakan dalam kebijakan JSON, lihat Referensi elemen kebijakan JSON IAM dalam Panduan Pengguna IAM.

Contoh kebijakan berbasis identitas untuk AWS AppSync

Untuk melihat contoh kebijakan AWS AppSync berbasis identitas, lihat. Kebijakan berbasis identitas untuk AWS AppSync

Kebijakan berbasis sumber daya dalam AWS AppSync

Mendukung kebijakan berbasis sumber daya	Tidak

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan

kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya yang dilampiri kebijakan tersebut, kebijakan ini menentukan jenis tindakan yang dapat dilakukan oleh pengguna utama tertentu di sumber daya tersebut dan apa ketentuannya. Anda harus menentukan pengguna utama dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan seluruh akun atau entitas IAM di akun lain sebagai pengguna utama dalam kebijakan berbasis sumber daya. Menambahkan pengguna utama lintas akun ke kebijakan berbasis sumber daya bagian dari membangun hubungan kepercayaan. Ketika prinsipal dan sumber daya berbeda Akun AWS, administrator IAM di akun tepercaya juga harus memberikan izin entitas utama (pengguna atau peran) untuk mengakses sumber daya. Izin diberikan dengan melampirkan kebijakan berbasis identitas ke entitas tersebut. Namun, jika kebijakan berbasis sumber daya memberikan akses kepada pengguna utama dalam akun yang sama, kebijakan berbasis identitas lainnya tidak diperlukan. Untuk informasi selengkapnya, lihat Perbedaan peran IAM dengan kebijakan berbasis sumber daya di Panduan Pengguna IAM.

Tindakan kebijakan untuk AWS AppSync

Mendukung tindakan kebijakan	Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Tindakan kebijakan biasanya memiliki nama yang sama dengan operasi AWS API terkait. Ada beberapa pengecualian, misalnya tindakan hanya izin yang tidak memiliki operasi API yang cocok. Ada juga beberapa operasi yang memerlukan beberapa tindakan dalam suatu kebijakan. Tindakan tambahan ini disebut tindakan dependen.

Menyertakan tindakan dalam suatu kebijakan untuk memberikan izin melakukan operasi terkait.

Untuk melihat daftar AWS AppSync tindakan, lihat Tindakan yang ditentukan oleh AWS AppSync dalam Referensi Otorisasi Layanan.

Tindakan kebijakan AWS AppSync menggunakan awalan berikut sebelum tindakan:

```
appsync
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan-tindakan tersebut dengan koma.

```
"Action": [
      "appsync:action1",
      "appsync:action2"
         ]
```

Untuk melihat contoh kebijakan AWS AppSync berbasis identitas, lihat. Kebijakan berbasis identitas untuk AWS AppSync

Sumber daya kebijakan untuk AWS AppSync

Mendukung sumber daya kebijakan	Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen kebijakan JSON `Resource` menentukan objek atau beberapa objek yang menjadi target penerapan tindakan. Pernyataan harus menyertakan elemen `Resource` atau `NotResource`. Praktik terbaiknya, tentukan sumber daya menggunakan Amazon Resource Name (ARN). Anda dapat melakukan ini untuk tindakan yang mendukung jenis sumber daya tertentu, yang dikenal sebagai izin tingkat sumber daya.

Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, misalnya operasi pencantuman, gunakan wildcard (*) untuk mengindikasikan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*"
```

Untuk melihat daftar jenis AWS AppSync sumber daya dan ARNnya, lihat Sumber daya yang ditentukan oleh AWS AppSync dalam Referensi Otorisasi Layanan. Untuk mempelajari tindakan mana yang dapat Anda tentukan ARN dari setiap sumber daya, lihat Tindakan yang ditentukan oleh. AWS AppSync

Untuk melihat contoh kebijakan AWS AppSync berbasis identitas, lihat. Kebijakan berbasis identitas untuk AWS AppSync

Kunci kondisi kebijakan untuk AWS AppSync

Mendukung kunci kondisi kebijakan spesifik layanan	Tidak

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen `Condition` (atau blok `Condition`) memungkinkan Anda menentukan kondisi di mana suatu pernyataan akan diterapkan. Elemen `Condition` bersifat opsional. Anda dapat membuat ekspresi kondisional yang menggunakan operator kondisi, misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta.

Jika Anda menentukan beberapa elemen `Condition` dalam satu pernyataan, atau beberapa kunci dalam satu elemen `Condition`, AWS akan mengevaluasinya dengan menggunakan operasi `AND` logis. Jika Anda menentukan beberapa nilai untuk satu kunci kondisi, AWS mengevaluasi kondisi menggunakan `OR` operasi logis. Semua kondisi harus dipenuhi sebelum izin pernyataan diberikan.

Anda juga dapat menggunakan variabel placeholder saat menentukan kondisi. Misalnya, Anda dapat memberikan izin kepada pengguna IAM untuk mengakses sumber daya hanya jika izin tersebut mempunyai tag yang sesuai dengan nama pengguna IAM mereka. Untuk informasi selengkapnya, lihat Elemen kebijakan IAM: variabel dan tag di Panduan Pengguna IAM.

AWS mendukung kunci kondisi global dan kunci kondisi khusus layanan. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan Pengguna IAM.

Untuk melihat daftar kunci AWS AppSync kondisi, lihat Kunci kondisi untuk AWS AppSync dalam Referensi Otorisasi Layanan. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan kunci kondisi, lihat Tindakan yang ditentukan oleh AWS AppSync.

Untuk melihat contoh kebijakan AWS AppSync berbasis identitas, lihat. Kebijakan berbasis identitas untuk AWS AppSync

Daftar kontrol akses (ACL) di AWS AppSync

Mendukung ACL	Tidak

Daftar kontrol akses (ACL) mengontrol pengguna utama (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL sama dengan kebijakan berbasis sumber daya, meskipun tidak menggunakan format dokumen kebijakan JSON.

Kontrol akses berbasis atribut (ABAC) dengan AWS AppSync

Mendukung ABAC (tanda dalam kebijakan)	Parsial

Kontrol akses berbasis atribut (ABAC) adalah strategi otorisasi yang menentukan izin berdasarkan atribut. Dalam AWS, atribut ini disebut tag. Anda dapat melampirkan tag ke entitas IAM (pengguna atau peran) dan ke banyak AWS sumber daya. Pemberian tanda ke entitas dan sumber daya adalah langkah pertama dari ABAC. Kemudian, rancanglah kebijakan ABAC untuk mengizinkan operasi saat tag milik pengguna utama cocok dengan tag yang ada di sumber daya yang ingin diakses.

ABAC sangat berguna di lingkungan yang berkembang dengan cepat dan berguna di situasi di mana pengelolaan kebijakan menjadi rumit.

Untuk mengendalikan akses berdasarkan tag, berikan informasi tentang tag di elemen kondisi dari kebijakan dengan menggunakan kunci kondisi `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, atau `aws:TagKeys`.

Jika sebuah layanan mendukung ketiga kunci kondisi untuk setiap jenis sumber daya, nilainya adalah Ya untuk layanan tersebut. Jika suatu layanan mendukung ketiga kunci kondisi hanya untuk beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat Apa itu ABAC? di Panduan Pengguna IAM. Untuk melihat tutorial terkait langkah-langkah penyiapan ABAC, lihat Menggunakan kontrol akses berbasis atribut (ABAC) di Panduan Pengguna IAM.

Menggunakan kredensyal sementara dengan AWS AppSync

Mendukung kredensial sementara	Ya

Beberapa Layanan AWS tidak berfungsi saat Anda masuk menggunakan kredensyal sementara. Untuk informasi tambahan, termasuk yang Layanan AWS bekerja dengan kredensyal sementara, lihat Layanan AWS yang bekerja dengan IAM di Panduan Pengguna IAM.

Anda menggunakan kredensyal sementara jika Anda masuk AWS Management Console menggunakan metode apa pun kecuali nama pengguna dan kata sandi. Misalnya, ketika Anda mengakses AWS menggunakan tautan masuk tunggal (SSO) perusahaan Anda, proses tersebut secara otomatis membuat kredensyal sementara. Anda juga akan membuat kredensial sementara secara otomatis saat masuk ke konsol sebagai pengguna dan kemudian beralih peran. Untuk informasi selengkapnya tentang cara beralih peran, lihat Beralih peran (konsol) di Panduan Pengguna IAM.

Anda dapat membuat kredensil sementara secara manual menggunakan API AWS CLI atau AWS . Anda kemudian dapat menggunakan kredensyal sementara tersebut untuk mengakses. AWS AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat Kredensial keamanan sementara di IAM.

Teruskan sesi akses untuk AWS AppSync

Mendukung sesi akses maju (FAS)	Parsial

Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Saat Anda menggunakan beberapa layanan, Anda mungkin melakukan tindakan yang kemudian memulai tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan saat membuat permintaan FAS, lihat Teruskan sesi akses.

Peran layanan untuk AWS AppSync

Mendukung peran layanan	Tidak

Peran layanan adalah sebuah peran IAM yang diambil oleh sebuah layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.

> **Warning**
>
> Mengubah izin untuk peran layanan dapat merusak AWS AppSync fungsionalitas. Edit peran layanan hanya jika AWS AppSync memberikan panduan untuk melakukannya.

Peran terkait layanan untuk AWS AppSync

Mendukung peran terkait layanan	Parsial

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan dapat menggunakan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran tertaut-layanan.

Untuk informasi selengkapnya tentang cara membuat atau mengelola peran yang tertaut dengan layanan, lihat Layanan AWS yang bekerja dengan IAM di Panduan Pengguna IAM. Temukan sebuah layanan dalam tabel yang memiliki `Yes` di kolom Peran terkait layanan. Pilih tautan Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Kebijakan berbasis identitas untuk AWS AppSync

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau memodifikasi AWS AppSync sumber daya. Mereka juga tidak dapat melakukan tugas dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau AWS API. Untuk memberikan izin kepada pengguna untuk melakukan tindakan pada sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat menjalankan peran.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM menggunakan contoh dokumen kebijakan JSON ini, lihat Membuat kebijakan IAM dalam Panduan Pengguna IAM.

Untuk detail tentang tindakan dan jenis sumber daya yang ditentukan oleh AWS AppSync, termasuk format ARN untuk setiap jenis sumber daya, lihat Kunci tindakan, sumber daya, dan kondisi AWS AppSync di Referensi Otorisasi Layanan.

Untuk mempelajari praktik terbaik dalam membuat dan mengonfigurasi kebijakan berbasis identitas IAM, lihat. the section called “Praktik terbaik kebijakan IAM”

Untuk daftar kebijakan berbasis identitas IAM, lihat. AWS AppSync AWS kebijakan terkelola untuk AWS AppSync

Topik

- Menggunakan konsol AWS AppSync
- Izinkan pengguna melihat izin mereka sendiri
- Mengakses satu bucket Amazon S3
- Melihat AWS AppSync widget berdasarkan tag
- AWS kebijakan terkelola untuk AWS AppSync

Menggunakan konsol AWS AppSync

Untuk mengakses AWS AppSync konsol, Anda harus memiliki set izin minimum. Izin ini harus memungkinkan Anda untuk membuat daftar dan melihat detail tentang AWS AppSync sumber daya di Anda Akun AWS. Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk entitas (pengguna atau peran) dengan kebijakan tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau AWS API. Sebaliknya, izinkan akses hanya ke tindakan yang cocok dengan operasi API yang coba dilakukan.

Untuk memastikan bahwa pengguna dan peran IAM masih dapat menggunakan AWS AppSync konsol, lampirkan juga kebijakan AWS AppSync `ConsoleAccess` atau `ReadOnly` AWS terkelola ke entitas. Untuk informasi selengkapnya, lihat Menambahkan izin ke pengguna di Panduan Pengguna IAM.

Izinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan para pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini

mencakup izin untuk menyelesaikan tindakan ini di konsol atau menggunakan API atau secara terprogram. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Mengakses satu bucket Amazon S3

Dalam contoh ini, Anda ingin memberi pengguna IAM di AWS akun Anda akses ke salah satu bucket Amazon S3 Anda. `examplebucket` Anda juga ingin mengizinkan pengguna untuk menambah, memperbarui, dan menghapus objek.

Selain memberikan izin `s3:PutObject`, `s3:GetObject`, dan `s3:DeleteObject` bagi pengguna, kebijakan tersebut juga memberikan izin `s3:ListAllMyBuckets`, `s3:GetBucketLocation`, dan `s3:ListBucket`. Izin-izin tersebut adalah izin tambahan yang diperlukan oleh konsol tersebut. Selain itu, tindakan `s3:PutObjectAcl` dan `s3:GetObjectAcl` diperlukan untuk dapat menyalin, memotong, dan menempel objek di konsol. Untuk contoh panduan yang memberikan izin kepada pengguna dan mengujinya menggunakan konsol, lihat Contoh panduan: Menggunakan kebijakan pengguna untuk mengontrol akses ke bucket Anda.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListBucketsInConsole",
         "Effect":"Allow",
         "Action":[
            "s3:ListAllMyBuckets"
         ],
         "Resource":"arn:aws:s3:::*"
      },
      {
         "Sid":"ViewSpecificBucketInfo",
         "Effect":"Allow",
         "Action":[
            "s3:ListBucket",
            "s3:GetBucketLocation"
         ],
         "Resource":"arn:aws:s3:::examplebucket"
      },
      {
         "Sid":"ManageBucketContents",
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:PutObjectAcl",
            "s3:GetObject",
            "s3:GetObjectAcl",
            "s3:DeleteObject"
         ],
         "Resource":"arn:aws:s3:::examplebucket/*"
      }
   ]
}
```

Melihat AWS AppSync *widget* berdasarkan tag

Anda dapat menggunakan kondisi dalam kebijakan berbasis identitas untuk mengontrol akses ke AWS AppSync sumber daya berdasarkan tag. Contoh ini menunjukkan bagaimana Anda dapat membuat kebijakan yang memungkinkan melihat *widget*. Namun, izin diberikan hanya jika tag *widget* 0wner memiliki nilai nama pengguna pengguna tersebut. Kebijakan ini juga memberi izin yang diperlukan untuk menyelesaikan tindakan ini pada konso tersebutl.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListWidgetsInConsole",
            "Effect": "Allow",
            "Action": "appsync:ListWidgets",
            "Resource": "*"
        },
        {
            "Sid": "ViewWidgetIfOwner",
            "Effect": "Allow",
            "Action": "appsync:GetWidget",
            "Resource": "arn:aws:appsync:*:*:widget/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

Anda dapat melampirkan kebijakan ini ke pengguna IAM di akun Anda. Jika pengguna bernama `richard-roe` mencoba untuk melihat AWS AppSync *widget*, *widget* harus ditandai `0wner=richard-roe` atau`owner=richard-roe`. Jika tidak, aksesnya akan ditolak. Kunci tanda syarat `0wner` cocok dengan `0wner` dan `owner` karena nama kunci syarat tidak terpengaruh huruf besar/kecil. Untuk informasi selengkapnya, lihat Elemen kebijakan IAM JSON: Syarat dalam Panduan Pengguna IAM.

AWS kebijakan terkelola untuk AWS AppSync

Untuk menambahkan izin ke pengguna, grup, dan peran, lebih mudah menggunakan kebijakan AWS terkelola daripada menulis kebijakan sendiri. Dibutuhkan waktu dan keahlian untuk membuat kebijakan terkelola pelanggan IAM yang hanya memberi tim Anda izin yang mereka butuhkan. Untuk memulai dengan cepat, Anda dapat menggunakan kebijakan AWS terkelola kami. Kebijakan ini mencakup kasus penggunaan umum dan tersedia di Akun AWS Anda. Untuk informasi selengkapnya tentang kebijakan AWS AWS terkelola, lihat kebijakan terkelola di Panduan Pengguna IAM.

AWS layanan memelihara dan memperbarui kebijakan AWS terkelola. Anda tidak dapat mengubah izin dalam kebijakan AWS terkelola. Layanan terkadang menambahkan izin tambahan ke kebijakan AWS terkelola untuk mendukung fitur baru. Jenis pembaruan ini akan memengaruhi semua identitas (pengguna, grup, dan peran) di mana kebijakan tersebut dilampirkan. Layanan kemungkinan besar akan memperbarui kebijakan AWS terkelola saat fitur baru diluncurkan atau saat operasi baru tersedia. Layanan tidak menghapus izin dari kebijakan AWS terkelola, sehingga pembaruan kebijakan tidak akan merusak izin yang ada.

Selain itu, AWS mendukung kebijakan terkelola untuk fungsi pekerjaan yang mencakup beberapa layanan. Misalnya, kebijakan ReadOnlyAccess AWS terkelola menyediakan akses hanya-baca ke semua AWS layanan dan sumber daya. Saat layanan meluncurkan fitur baru, AWS tambahkan izin hanya-baca untuk operasi dan sumber daya baru. Untuk melihat daftar dan deskripsi dari kebijakan fungsi tugas, lihat kebijakan yang dikelola AWS untuk fungsi tugas di Panduan Pengguna IAM.

AWS kebijakan terkelola: AWSAppSyncInvokeFullAccess

Gunakan kebijakan `AWSAppSyncInvokeFullAccess` AWS terkelola untuk mengizinkan administrator mengakses AWS AppSync layanan melalui konsol atau secara independen.

Anda dapat melampirkan kebijakan `AWSAppSyncInvokeFullAccess` ke identitas IAM Anda.

Detail izin

Kebijakan ini mencakup izin berikut.

- `AWS AppSync`— Memungkinkan akses administratif penuh ke semua sumber daya di AWS AppSync

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "appsync:GraphQL",
                "appsync:GetGraphqlApi",
                "appsync:ListGraphqlApis",
                "appsync:ListApiKeys"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS kebijakan terkelola: AWSAppSyncSchemaAuthor

Gunakan kebijakan `AWSAppSyncSchemaAuthor` AWS terkelola untuk memungkinkan pengguna IAM mengakses untuk membuat, memperbarui, dan menanyakan skema GraphQL mereka. Untuk informasi tentang apa yang dapat dilakukan pengguna dengan izin ini, lihatMerancang GraphQL API.

Anda dapat melampirkan kebijakan `AWSAppSyncSchemaAuthor` ke identitas IAM Anda.

Detail izin

Kebijakan ini mencakup izin berikut.

- `AWS AppSync`— Memungkinkan tindakan berikut:
  - Membuat skema GraphQL
  - Mengizinkan pembuatan, modifikasi, dan penghapusan tipe, resolver, dan fungsi GraphQL
  - Mengevaluasi logika template permintaan dan respons
  - Mengevaluasi kode dengan runtime dan konteks
  - Mengirim kueri GraphQL ke GraphQL API
  - Mengambil data GraphQL

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "appsync:GraphQL",
                "appsync:CreateResolver",
                "appsync:CreateType",
                "appsync:DeleteResolver",
                "appsync:DeleteType",
                "appsync:GetResolver",
                "appsync:GetType",
                "appsync:GetDataSource",
                "appsync:GetSchemaCreationStatus",
                "appsync:GetIntrospectionSchema",
                "appsync:GetGraphqlApi",
                "appsync:ListTypes",
                "appsync:ListApiKeys",
                "appsync:ListResolvers",
                "appsync:ListDataSources",
                "appsync:ListGraphqlApis",
                "appsync:StartSchemaCreation",
                "appsync:UpdateResolver",
                "appsync:UpdateType",
                "appsync:TagResource",
                "appsync:UntagResource",
                "appsync:ListTagsForResource",
                "appsync:CreateFunction",
                "appsync:UpdateFunction",
                "appsync:GetFunction",
                "appsync:DeleteFunction",
                "appsync:ListFunctions",
                "appsync:ListResolversByFunction",
                "appsync:EvaluateMappingTemplate",
                "appsync:EvaluateCode"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS kebijakan terkelola: AWSAppSyncPushToCloudWatchLogs

AWS AppSync menggunakan Amazon CloudWatch untuk memantau kinerja aplikasi Anda dengan membuat log yang dapat Anda gunakan untuk memecahkan masalah dan mengoptimalkan permintaan GraphQL Anda. Untuk informasi selengkapnya, lihat Pemantauan dan pencatatan.

Gunakan kebijakan `AWSAppSyncPushToCloudWatchLogs` AWS terkelola AWS AppSync untuk memungkinkan mendorong log ke CloudWatch akun pengguna IAM.

Anda dapat melampirkan kebijakan `AWSAppSyncPushToCloudWatchLogs` ke identitas IAM Anda.

Detail izin

Kebijakan ini mencakup izin berikut.

- `CloudWatch Logs`— Memungkinkan AWS AppSync untuk membuat grup log dan aliran dengan nama tertentu. AWS AppSyncmendorong peristiwa log ke aliran log yang ditentukan.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS kebijakan terkelola: AWSAppSyncAdministrator

Gunakan kebijakan `AWSAppSyncAdministrator` AWS terkelola untuk mengizinkan administrator mengakses semua AWS AppSync kecuali AWS konsol.

Anda dapat melampirkan `AWSAppSyncAdministrator` ke entitas IAM Anda. AWS AppSync juga melampirkan kebijakan ini ke peran layanan yang memungkinkannya melakukan tindakan atas nama Anda.

Detail izin

Kebijakan ini mencakup izin berikut.

- `AWS AppSync`— Memungkinkan akses administratif penuh ke semua sumber daya di AWS AppSync
- `IAM`— Memungkinkan tindakan berikut:
  - Membuat peran terkait layanan AWS AppSync untuk memungkinkan menganalisis sumber daya di layanan lain atas nama Anda
  - Menghapus peran terkait layanan
  - Meneruskan peran terkait layanan ke AWS layanan lain untuk mengambil peran nanti dan untuk melakukan tindakan atas nama Anda

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "appsync:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "appsync.amazonaws.com"
```

```
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "appsync.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:DeleteServiceLinkedRole",
                "iam:GetServiceLinkedRoleDeletionStatus"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/appsync.amazonaws.com/
AWSServiceRoleForAppSync*"
        }
    ]
}
```

AWS kebijakan terkelola: AWSAppSyncServiceRolePolicy

Gunakan kebijakan `AWSAppSyncServiceRolePolicy` AWS terkelola untuk mengizinkan akses ke AWS layanan dan sumber daya yang AWS AppSync menggunakan atau mengelola.

Anda tidak dapat melampirkan `AWSAppSyncServiceRolePolicy` ke entitas IAM Anda. Kebijakan ini dilampirkan pada peran terkait layanan yang memungkinkan AWS AppSync untuk melakukan tindakan atas nama Anda. Untuk informasi selengkapnya, lihat Peran terkait layanan untuk AWS AppSync.

Detail izin

Kebijakan ini mencakup izin berikut.

- `X-Ray`— AWS AppSync digunakan AWS X-Ray untuk mengumpulkan data tentang permintaan yang dibuat dalam aplikasi Anda. Untuk informasi selengkapnya, lihat Menelusuri denganAWS X-Ray.

  Kebijakan ini memungkinkan tindakan berikut:

  - Mengambil aturan pengambilan sampel dan hasilnya
  - Mengirim data jejak ke daemon X-Ray

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingTargets",
                "xray:GetSamplingRules",
                "xray:GetSamplingStatisticSummaries"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS AppSync pembaruan kebijakan AWS terkelola

Lihat detail tentang pembaruan kebijakan AWS terkelola AWS AppSync sejak layanan ini mulai melacak perubahan ini. Untuk peringatan otomatis tentang perubahan pada halaman ini, berlangganan umpan RSS di halaman Riwayat AWS AppSync dokumen.

Perubahan	Deskripsi	Tanggal
AWSAppSyncSchemaAuthor - Pembaruan ke kebijakan yang tersedia	Menambahkan tindakan `EvaluateCode` kebijakan	7 Februari 2023

Perubahan	Deskripsi	Tanggal
	untuk memungkinkan pengguna mengevaluasi kode dengan runtime dan konteks.	
AWSAppSyncSchemaAuthor - Pembaruan ke kebijakan yang tersedia	Menambahkan tindakan kebijakan untuk mengizinkan daftar, mendapatkan, membuat, memperbarui, dan menghapus fungsi untuk API. Menambahkan tindakan `EvaluateMappingTemplate` kebijakan untuk memungkinkan pengguna mengevaluasi logika template pemetaan resolver permintaan dan respons. Menambahkan tindakan kebijakan untuk memungkinkan penandaan sumber daya.	Agustus 25, 2022
AWS AppSync mulai melacak perubahan	AWS AppSync mulai melacak perubahan untuk kebijakan yang AWS dikelola.	Agustus 25, 2022

Memecahkan masalah AWS AppSync identitas dan akses

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan AWS AppSync dan IAM.

Saya tidak berwenang untuk melakukan tindakan di AWS AppSync

Jika AWS Management Console memberitahu Anda bahwa Anda tidak berwenang untuk melakukan tindakan, maka Anda harus menghubungi administrator Anda untuk bantuan. Administrator adalah orang yang memberikan nama pengguna dan kata sandi kepada Anda.

Contoh kesalahan berikut terjadi ketika pengguna IAM `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang *`my-example-widget`* sumber daya fiksi, tetapi ia tidak memiliki izin `appsync:`*`GetWidget`* fiksi.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 appsync:GetWidget on resource: my-example-widget
```

Dalam hal ini, Mateo meminta administratornya untuk memperbarui kebijakannya untuk mengizinkan dia mengakses sumber daya *`my-example-widget`* menggunakan tindakan `appsync:`*`GetWidget`*.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan yang tidak diizinkan untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran AWS AppSync.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi ketika pengguna IAM bernama `marymajor` mencoba menggunakan konsol untuk melakukan tindakan di AWS AppSync. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin melihat access key saya

Setelah membuat access key pengguna IAM, Anda dapat melihat access key ID Anda setiap saat. Namun, Anda tidak dapat melihat secret access key Anda lagi. Jika Anda kehilangan secret key, Anda harus membuat pasangan access key baru.

Access key terdiri dari dua bagian: access key ID (misalnya, `AKIAIOSFODNN7EXAMPLE`) dan secret access key (misalnya, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Seperti nama pengguna dan kata sandi, Anda harus menggunakan access key ID dan secret access key sekaligus untuk mengautentikasi permintaan Anda. Kelola access key Anda seaman nama pengguna dan kata sandi Anda.

> **Important**
>
> Jangan memberikan access key Anda kepada pihak ke tiga, bahkan untuk membantu menemukan ID pengguna kanonis Anda. Dengan melakukan ini, Anda mungkin memberi seseorang akses permanen ke Anda Akun AWS.

Saat Anda membuat pasangan access key, Anda diminta menyimpan access key ID dan secret access key di lokasi yang aman. secret access key hanya tersedia saat Anda membuatnya. Jika Anda kehilangan secret access key Anda, Anda harus menambahkan access key baru ke pengguna IAM Anda. Anda dapat memiliki maksimum dua access key. Jika Anda sudah memiliki dua, Anda harus menghapus satu pasangan kunci sebelum membuat pasangan baru. Untuk melihat instruksi, lihat Mengelola access keys di Panduan Pengguna IAM.

Saya seorang administrator dan ingin mengizinkan orang lain mengakses AWS AppSync

Untuk memungkinkan orang lain mengakses AWS AppSync, Anda harus membuat entitas IAM (pengguna atau peran) untuk orang atau aplikasi yang membutuhkan akses. Mereka akan menggunakan kredensial untuk entitas tersebut untuk mengakses AWS. Anda kemudian harus melampirkan kebijakan ke entitas yang memberi mereka izin yang benar. AWS AppSync

Untuk segera mulai, lihat Membuat pengguna dan grup khusus IAM pertama Anda di Panduan Pengguna IAM.

Saya ingin mengizinkan orang di luar AWS akun saya untuk mengakses AWS AppSync sumber daya saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau pengguna di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACL), Anda dapat menggunakan kebijakan tersebut untuk memberi pengguna akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa hal berikut:

- Untuk mempelajari apakah AWS AppSync mendukung fitur-fitur ini, lihatBagaimana AWS AppSync bekerja dengan IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Memberikan akses kepada pengguna eksternal yang sah (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara penggunaan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Perbedaan antara peran IAM dan kebijakan berbasis sumber daya di Panduan Pengguna IAM.

Pencatatan panggilan AWS AppSync API dengan AWS CloudTrail

AWS AppSync terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di AWS AppSync. CloudTrail menangkap panggilan API untuk AWS AppSync sebagai acara. Panggilan yang diambil termasuk panggilan dari AWS AppSync konsol dan panggilan kode ke operasi AWS AppSync API. Jika Anda membuat jejak, Anda dapat mengaktifkan pengiriman CloudTrail acara secara berkelanjutan ke bucket Amazon S3, termasuk acara untuk. AWS AppSync Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di CloudTrail konsol dalam Riwayat acara. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat AWS AppSync, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail tambahan.

Untuk mempelajari selengkapnya CloudTrail, lihat Panduan AWS CloudTrail Pengguna.

AWS AppSync informasi di CloudTrail

CloudTrail diaktifkan di AWS akun Anda saat Anda membuat akun. Ketika aktivitas terjadi di AWS AppSync, aktivitas tersebut dicatat dalam suatu CloudTrail peristiwa bersama dengan peristiwa AWS layanan lainnya dalam riwayat Acara. Anda dapat melihat, mencari, dan mengunduh acara terbaru di

AWS akun Anda. Untuk informasi selengkapnya, lihat Melihat peristiwa dengan Riwayat CloudTrail acara.

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk acara untuk AWS AppSync, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Secara default, saat Anda membuat jejak di konsol, jejak tersebut berlaku untuk semua AWS Wilayah. Jejak mencatat peristiwa dari semua Wilayah di AWS partisi dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat berikut:

- Gambaran umum untuk membuat jejak
- CloudTrail layanan dan integrasi yang didukung
- Mengonfigurasi notifikasi Amazon SNS untuk CloudTrail
- Menerima file CloudTrail log dari beberapa wilayah dan Menerima file CloudTrail log dari beberapa akun

AWS AppSync mendukung pencatatan panggilan yang dilakukan melalui AWS AppSync API. Pada saat ini, panggilan ke API Anda, serta panggilan yang dilakukan ke resolver tidak masuk. AWS AppSync CloudTrail

Setiap peristiwa atau entri log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan berikut:

- Apakah permintaan dibuat dengan root atau AWS Identity and Access Management (IAM) kredensyal pengguna.
- Apakah permintaan dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

Untuk informasi selengkapnya, lihat elemen CloudTrail UserIdentity.

Memahami entri file AWS AppSync log

Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail file log berisi satu atau lebih entri log. Peristiwa mewakili

permintaan tunggal dari sumber manapun dan mencakup informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. CloudTrail file log bukanlah jejak tumpukan yang diurutkan dari panggilan API publik, sehingga file tersebut tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan `GetGraphqlApi` tindakan yang dilakukan melalui AWS AppSync konsol:


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "ABCDEFXAMPLEPRINCIPAL:nikkiwolf",
        "arn": "arn:aws:sts::111122223333:assumed-role/admin/nikkiwolf",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/admin",
                "accountId": "111122223333",
                "userName": "admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-03-12T22:41:48Z"
            }
        }
    },
    "eventTime": "2021-03-12T22:46:18Z",
    "eventSource": "appsync.amazonaws.com",
    "eventName": "GetGraphqlApi",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.69",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.942
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.282-b08
 java/1.8.0_282 vendor/Oracle_Corporation",
    "requestParameters": {
        "apiId": "xhxt3typtfnmidkhcexampleid"
    },
    "responseElements": null,
```

```
    "requestID": "2fc43a35-a552-4b5d-be6e-12553a03dd12",
    "eventID": "b95b0ad9-8c71-4252-a2ec-5dc2fe5f8ae8",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan `CreateApikey` tindakan yang dilakukan melalui: AWS CLI

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "ABCDEFXAMPLEPRINCIPAL",
        "arn": "arn:aws:iam::111122223333:user/nikkiwolf",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "nikkiwolf"
    },
    "eventTime": "2021-03-12T22:49:10Z",
    "eventSource": "appsync.amazonaws.com",
    "eventName": "CreateApiKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.69",
    "userAgent": "aws-cli/2.0.11 Python/3.7.4 Darwin/18.7.0 botocore/2.0.0dev15",
    "requestParameters": {
        "apiId": "xhxt3typtfnmidkhcexampleid"
    },
    "responseElements": {
        "apiKey": {
            "id": "***",
            "expires": 1616191200,
            "deletes": 1621375200
        }
    },
    "requestID": "e152190e-04ba-4d0a-ae7b-6bfc0bcea6af",
    "eventID": "ba3f39e0-9d87-41c5-abbb-2000abcb6013",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
```

```
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```

Praktik terbaik keamanan untuk AWS AppSync

AWS AppSync Mengamankan lebih dari sekadar menyalakan beberapa tuas atau mengatur logging. Bagian berikut membahas praktik terbaik keamanan yang bervariasi tergantung pada cara Anda menggunakan layanan.

Memahami metode otentikasi

AWS AppSync menyediakan beberapa cara untuk mengautentikasi pengguna Anda ke GraphQL API Anda. Setiap metode memiliki trade-off dalam keamanan, auditabilitas, dan kegunaan.

Metode otentikasi umum berikut tersedia:

- Kumpulan pengguna Amazon Cognito memungkinkan GraphQL API Anda menggunakan atribut pengguna untuk kontrol akses dan pemfilteran berbutir halus.
- Token API memiliki masa pakai yang terbatas dan sesuai untuk sistem otomatis, seperti sistem Integrasi Berkelanjutan dan integrasi dengan API eksternal.
- AWS Identity and Access Management (IAM) sesuai untuk aplikasi internal yang dikelola di Anda Akun AWS.
- OpenID Connect memungkinkan Anda untuk mengontrol dan menggabungkan akses dengan protokol OpenID Connect.

Untuk informasi selengkapnya tentang otentikasi dan otorisasi di AWS AppSync, lihat. Otorisasi dan otentikasi

Gunakan TLS untuk resolver HTTP

Saat menggunakan resolver HTTP, pastikan untuk menggunakan koneksi TLS-Secured (HTTPS) sedapat mungkin. Untuk daftar lengkap sertifikat TLS yang AWS AppSync dipercaya, lihat. Otoritas Sertifikat (CA) Diakui olehAWS AppSyncuntuk Titik Akhir HTTPS

Gunakan peran dengan izin sesedikit mungkin

Saat menggunakan resolver seperti DynamoDB resolver, gunakan peran yang memberikan tampilan paling ketat ke sumber daya Anda, seperti tabel Amazon DynamoDB Anda.

Praktik terbaik kebijakan IAM

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus AWS AppSync sumber daya di akun Anda. Tindakan ini dikenai biaya untuk Akun AWS Anda. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulailah dengan kebijakan AWS terkelola dan beralih ke izin hak istimewa paling sedikit — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan AWS terkelola yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Anda Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat kebijakan yang dikelola AWS atau kebijakan yang dikelola AWS untuk fungsi pekerjaan di Panduan Pengguna IAM.
- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukan ini dengan menentukan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, juga dikenal sebagai izin hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk menerapkan izin, lihat Kebijakan dan izin di IAM di Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik Layanan AWS, seperti AWS CloudFormation. Untuk informasi selengkapnya, lihat Elemen kebijakan JSON IAM: Syarat di Panduan Pengguna IAM.
- Menggunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda guna memastikan izin yang aman dan berfungsi – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang

dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk informasi selengkapnya, lihat validasi kebijakan Analizer Akses IAM di Panduan Pengguna IAM.

- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Anda, Akun AWS aktifkan MFA untuk keamanan tambahan. Untuk mewajibkan MFA saat operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat Mengonfigurasi akses API yang dilindungi MFA di Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat Praktik terbaik keamanan di IAM di Panduan Pengguna IAM.

Referensi penyelesai () JavaScript

Bagian berikut menjelaskan APPSYNC_JS runtime dan JavaScript resolver.

Topik

JavaScript ikhtisar penyelesai

AWS AppSync memungkinkan Anda menanggapi permintaan GraphQL dengan melakukan operasi pada sumber data Anda. Untuk setiap bidang GraphQL yang ingin Anda jalankan kueri, mutasi, atau langganan, resolver harus dilampirkan.

Resolver adalah konektor antara GraphQL dan sumber data. Mereka memberi tahu AWS AppSync cara menerjemahkan permintaan GraphQL yang masuk ke dalam instruksi untuk sumber data backend Anda dan bagaimana menerjemahkan respons dari sumber data itu kembali ke respons GraphQL. Dengan AWS AppSync, Anda dapat menulis resolver Anda menggunakan JavaScript dan menjalankannya di lingkungan AWS AppSync (APPSYNC_JS).

AWS AppSync memungkinkan Anda untuk menulis resolver unit atau resolver pipa yang terdiri dari beberapa AWS AppSync fungsi dalam pipa.

Fitur runtime yang didukung

AWS AppSync JavaScript Runtime menyediakan subset JavaScript pustaka, utilitas, dan fitur. Untuk daftar lengkap fitur dan fungsionalitas yang didukung oleh `APPSYNC_JS` runtime, lihat fitur JavaScript runtime untuk resolver dan fungsi.

Penyelesai unit

Unit resolver terdiri dari kode yang mendefinisikan permintaan dan respon handler yang dijalankan terhadap sumber data. Handler permintaan mengambil objek konteks sebagai argumen dan mengembalikan payload permintaan yang digunakan untuk memanggil sumber data Anda. Response handler menerima payload kembali dari sumber data dengan hasil dari permintaan yang dieksekusi. Response handler mengubah payload menjadi respons GraphQL untuk menyelesaikan bidang GraphQL. Dalam contoh di bawah ini, resolver mengambil item dari sumber data DynamoDB:

```
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
  return ddb.get({ key: { id: ctx.args.id } });
}

export const response = (ctx) => ctx.result;
```

Anatomi penyelesai JavaScript pipa

Pipeline resolver terdiri dari kode yang mendefinisikan permintaan dan respon handler dan daftar fungsi. Setiap fungsi memiliki permintaan dan respon handler yang dijalankan terhadap sumber data. Karena delegasi penyelesai pipa berjalan ke daftar fungsi, oleh karena itu tidak ditautkan ke sumber data apa pun. Resolver dan fungsi unit adalah primitif yang menjalankan operasi terhadap sumber data.

Penangan permintaan penyelesai pipa

Handler permintaan dari resolver pipeline (langkah sebelumnya) memungkinkan Anda untuk melakukan beberapa logika persiapan sebelum menjalankan fungsi yang ditentukan.

Daftar fungsi

Daftar fungsi resolver pipeline akan berjalan secara berurutan. Hasil evaluasi penangan permintaan penyelesai pipa tersedia untuk fungsi pertama sebagai. `ctx.prev.result` Setiap hasil evaluasi fungsi tersedia untuk fungsi berikutnya sebagai`ctx.prev.result`.

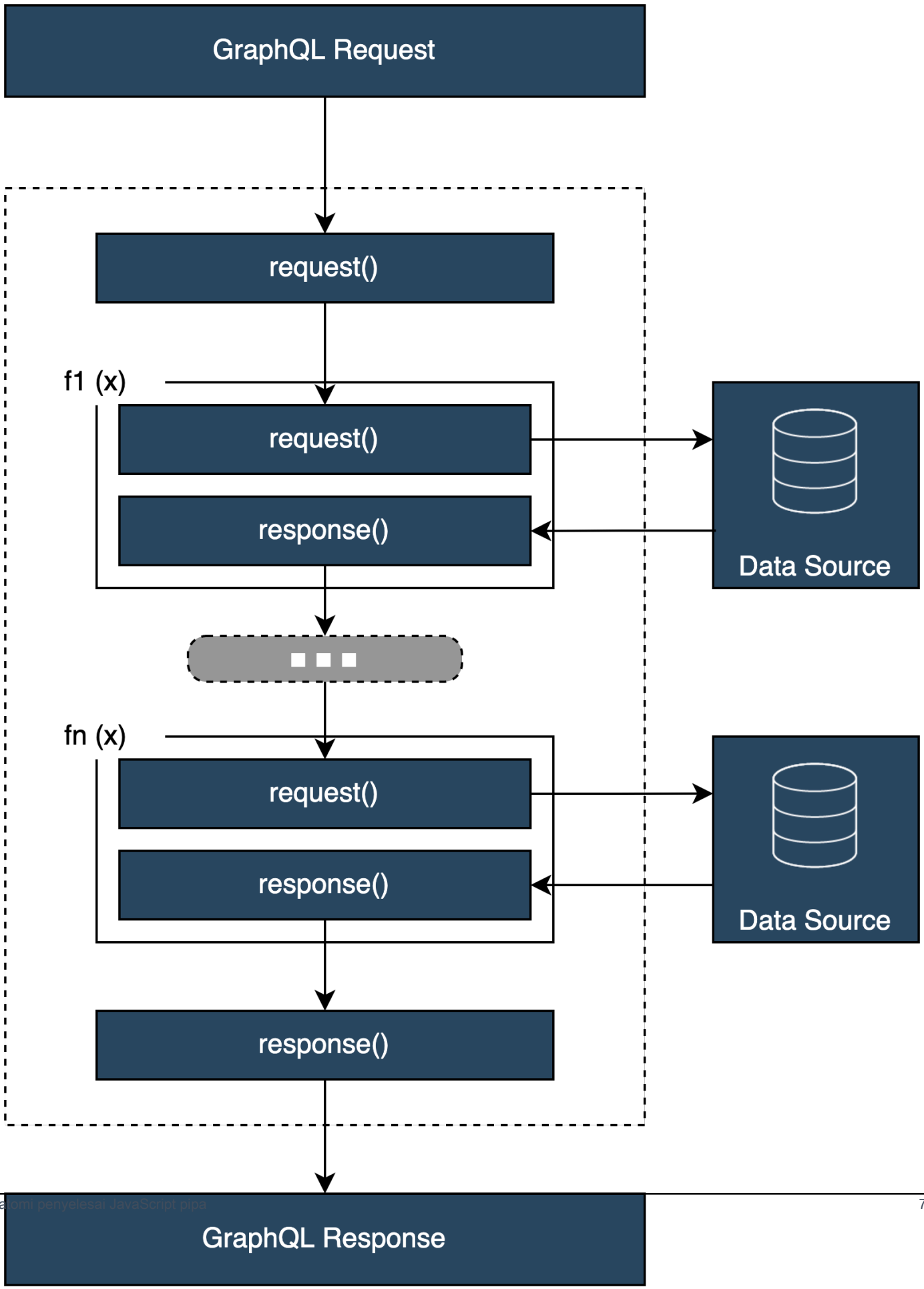
Penangan respons penyelesai pipa

Response handler dari pipeline resolver memungkinkan Anda untuk melakukan beberapa logika akhir dari output fungsi terakhir ke tipe bidang GraphQL yang diharapkan. Output dari fungsi terakhir dalam daftar fungsi tersedia di pengendali respons penyelesai pipa sebagai atau. `ctx.prev.result` `ctx.result`

Aliran eksekusi

Mengingat resolver pipeline yang terdiri dari dua fungsi, daftar di bawah ini mewakili alur eksekusi saat resolver dipanggil:

1. Penangan permintaan penyelesai pipa
2. Fungsi 1: Fungsi permintaan handler
3. Fungsi 1: Pemanggilan sumber data
4. Fungsi 1: Penangan respons fungsi
5. Fungsi 2: Fungsi permintaan handler
6. Fungsi 2: Pemanggilan sumber data
7. Fungsi 2: Fungsi respon handler
8. Penangan respons penyelesai pipa


GraphQL Request
request()
f1 (x)
request()
response()
Data Source
fn (x)
request()
response()
Data Source
response()
GraphQL Response

Utilitas bawaan **APPSYNC_JS** runtime yang berguna

Utilitas berikut dapat membantu Anda ketika Anda bekerja dengan resolver pipa.

ctx.stash

Stash adalah objek yang tersedia di dalam setiap resolver dan permintaan fungsi dan penangan respons. Instans simpanan yang sama hidup melalui satu resolver run. Ini berarti Anda dapat menggunakan stash untuk meneruskan data arbitrer di seluruh penangan permintaan dan respons dan di seluruh fungsi dalam resolver pipeline. Anda dapat menguji simpanan seperti JavaScript objek biasa.

ctx.prev.result

`ctx.prev.result`Ini merupakan hasil dari operasi sebelumnya yang dieksekusi dalam pipa. Jika operasi sebelumnya adalah penangan permintaan resolver pipa, maka `ctx.prev.result` tersedia untuk fungsi pertama dalam rantai. Jika operasi sebelumnya adalah fungsi pertama, maka `ctx.prev.result` mewakili output dari fungsi pertama dan tersedia untuk fungsi kedua dalam pipa. Jika operasi sebelumnya adalah fungsi terakhir, maka `ctx.prev.result` mewakili output dari fungsi terakhir dan tersedia untuk pengendali respon resolver pipeline.

util.error

`util.error`Utilitas ini berguna untuk melempar kesalahan bidang. Menggunakan `util.error` di dalam permintaan fungsi atau penangan respons akan segera menimbulkan kesalahan bidang, yang mencegah fungsi berikutnya dieksekusi. Untuk detail selengkapnya dan `util.error` tanda tangan lainnya, kunjungi fitur JavaScript runtime untuk resolver dan fungsi.

Util.appenError

`util.appendError`mirip dengan`util.error()`, dengan perbedaan utama bahwa itu tidak mengganggu evaluasi handler. Sebaliknya, ini menandakan ada kesalahan dengan bidang, tetapi memungkinkan penangan untuk dievaluasi dan akibatnya mengembalikan data. Menggunakan fungsi `util.appendError` di dalam tidak akan mengganggu aliran eksekusi pipa. Untuk detail selengkapnya dan `util.error` tanda tangan lainnya, kunjungi fitur JavaScript runtime untuk resolver dan fungsi.

Runtime.earlyReturn

`runtime.earlyReturn`Fungsi ini memungkinkan Anda untuk kembali sebelum waktunya dari fungsi permintaan apa pun. Menggunakan `runtime.earlyReturn` bagian dalam penangan

permintaan resolver akan kembali dari resolver. Memanggilnya dari penangan permintaan AWS AppSync fungsi akan kembali dari fungsi dan akan melanjutkan proses ke fungsi berikutnya di pipeline atau penangan respons resolver.

Menulis resolver pipa

Penyelesai pipa juga memiliki permintaan dan penangan respons yang mengelilingi fungsi dalam pipeline: penangan permintaannya dijalankan sebelum permintaan fungsi pertama, dan penangan responsnya dijalankan setelah respons fungsi terakhir. Handler permintaan resolver dapat mengatur data yang akan digunakan oleh fungsi dalam pipeline. Handler respons resolver bertanggung jawab untuk mengembalikan data yang dipetakan ke tipe keluaran bidang GraphQL. Dalam contoh di bawah ini, penangan permintaan resolver, mendefinisikan`allowedGroups`; data yang dikembalikan harus milik salah satu grup ini. Nilai ini dapat digunakan oleh fungsi resolver untuk meminta data. Response handler resolver melakukan pemeriksaan akhir dan memfilter hasilnya untuk memastikan bahwa hanya item milik grup yang diizinkan yang dikembalikan.

```
import { util } from '@aws-appsync/utils';

/**
 * Called before the request function of the first AppSync function in the pipeline.
 *  @param ctx the context object holds contextual information about the function
 invocation.
 */
export function request(ctx) {
  ctx.stash.allowedGroups = ['admin'];
  ctx.stash.startedAt = util.time.nowISO8601();
  return {};
}
/**
 * Called after the response function of the last AppSync function in the pipeline.
 * @param ctx the context object holds contextual information about the function
 invocation.
 */
export function response(ctx) {
  const result = [];
  for (const item of ctx.prev.result) {
    if (ctx.stash.allowedGroups.indexOf(item.group) > -1) result.push(item);
  }
  return result;
}
```

AWS AppSyncFungsi menulis

AWS AppSync fungsi memungkinkan Anda untuk menulis logika umum yang dapat Anda gunakan kembali di beberapa resolver dalam skema Anda. Misalnya, Anda dapat memiliki satu AWS AppSync fungsi `QUERY_ITEMS` yang dipanggil yang bertanggung jawab untuk menanyakan item dari sumber data Amazon DynamoDB. Untuk resolver yang ingin Anda kueri dengan item, cukup tambahkan fungsi ke pipeline resolver dan berikan indeks kueri yang akan digunakan. Logika tidak harus diimplementasikan kembali.

Menulis kode

Misalkan Anda ingin melampirkan resolver pipeline pada bidang bernama `getPost(id:ID!)` yang mengembalikan `Post` tipe dari sumber data Amazon DynamoDB dengan kueri GraphQL berikut:

```
getPost(id:1){
    id
    title
    content
}
```

Pertama, lampirkan resolver sederhana `Query.getPost` dengan kode di bawah ini. Ini adalah contoh kode resolver sederhana. Tidak ada logika yang didefinisikan dalam penangan permintaan, dan penangan respons hanya mengembalikan hasil dari fungsi terakhir.

```
/**
 * Invoked **before** the request handler of the first AppSync function in the
 pipeline.
 * The resolver `request` handler allows to perform some preparation logic
 * before executing the defined functions in your pipeline.
 * @param ctx the context object holds contextual information about the function
 invocation.
 */
export function request(ctx) {
  return {}
}

/**
 * Invoked **after** the response handler of the last AppSync function in the pipeline.
 * The resolver `response` handler allows to perform some final evaluation logic
 * from the output of the last function to the expected GraphQL field type.
 * @param ctx the context object holds contextual information about the function
 invocation.
```

```
 */
export function response(ctx) {
  return ctx.prev.result
}
```

Selanjutnya, tentukan fungsi `GET_ITEM` yang mengambil postitem dari sumber data Anda:

```
import { util } from '@aws-appsync/utils'
import * as ddb from '@aws-appsync/utils/dynamodb'

/**
 * Request a single item from the attached DynamoDB table datasource
 * @param ctx the context object holds contextual information about the function
 invocation.
 */
export function request(ctx) {
 const { id } = ctx.args
 return ddb.get({ key: { id } })
}

/**
 * Returns the result
 * @param ctx the context object holds contextual information about the function
 invocation.
 */
export function response(ctx) {
 const { error, result } = ctx
 if (error) {
  return util.appendError(error.message, error.type, result)
 }
 return ctx.result
}
```

Jika ada kesalahan selama permintaan, penangan respons fungsi menambahkan kesalahan yang akan dikembalikan ke klien pemanggil dalam respons GraphQL. Tambahkan `GET_ITEM` fungsi ke daftar fungsi resolver Anda. Saat Anda menjalankan kueri, penangan permintaan `GET_ITEM` fungsi menggunakan utilitas yang disediakan oleh modul AWS AppSync DynamoDB untuk membuat `DynamoDBGetItem` permintaan menggunakan sebagai kunci. `id` `ddb.get({ key: { id } })`menghasilkan `GetItem` operasi yang sesuai:

```
{
    "operation" : "GetItem",
```

```
    "key" : {
        "id" : { "S" : "1" }
    }
}
```

AWS AppSyncmenggunakan permintaan untuk mengambil data dari Amazon DynamoDB. Setelah data dikembalikan, itu ditangani oleh penangan respons `GET_ITEM` fungsi, yang memeriksa kesalahan dan kemudian mengembalikan hasilnya.

```
{
  "result" : {
    "id": 1,
    "title": "hello world",
    "content": "<long story>"
  }
}
```

Akhirnya, handler respon resolver mengembalikan hasilnya secara langsung.

Bekerja dengan kesalahan

Jika terjadi kesalahan dalam fungsi Anda selama permintaan, kesalahan akan tersedia di pengendali respons fungsi Anda di`ctx.error`. Anda dapat menambahkan kesalahan ke respons GraphQL Anda menggunakan utilitas. `util.appendError` Anda dapat membuat kesalahan tersedia untuk fungsi lain dalam pipeline dengan menggunakan simpanan. Lihat contoh di bawah ini:

```
/**
 * Returns the result
 * @param ctx the context object holds contextual information about the function
 invocation.
 */
export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    if (!ctx.stash.errors) ctx.stash.errors = []
    ctx.stash.errors.push(ctx.error)
    return util.appendError(error.message, error.type, result);
  }
  return ctx.result;
}
```

Utilitas

AWS AppSyncmenyediakan dua pustaka yang membantu dalam pengembangan resolver dengan runtime: `APPSYNC_JS`

- `@aws-appsync/eslint-plugin`- Menangkap dan memperbaiki masalah dengan cepat selama pengembangan.
- `@aws-appsync/utils`- Menyediakan validasi tipe dan pelengkapan otomatis di editor kode.

Mengkonfigurasi plugin eslint

ESLint adalah alat yang menganalisis kode Anda secara statis untuk menemukan masalah dengan cepat. Anda dapat menjalankan ESLint sebagai bagian dari pipeline integrasi berkelanjutan Anda. `@aws-appsync/eslint-plugin`adalah plugin ESLint yang menangkap sintaks yang tidak valid dalam kode Anda saat memanfaatkan runtime. `APPSYNC_JS` Plugin ini memungkinkan Anda untuk dengan cepat mendapatkan umpan balik tentang kode Anda selama pengembangan tanpa harus mendorong perubahan Anda ke cloud.

`@aws-appsync/eslint-plugin`menyediakan dua set aturan yang dapat Anda gunakan selama pengembangan.

“plugin: @aws -appsync/base” mengonfigurasi seperangkat aturan dasar yang dapat Anda manfaatkan dalam proyek Anda:

Rule	Deskripsi
tidak-asinkron	Proses dan janji asinkron tidak didukung.
tidak-menunggu	Proses dan janji asinkron tidak didukung.
tidak ada kelas	Kelas tidak didukung.
tidak-untuk	`for`tidak didukung (kecuali untuk `for-in` dan`for-of`, yang didukung)
tidak lanjutkan	`continue` tidak didukung.
tanpa generator	Generator tidak didukung.

Rule	Deskripsi
tanpa hasil	`yield` tidak didukung.
tanpa label	Label tidak didukung.
tidak-ini	`this`Kata kunci tidak didukung.
tidak mencoba	Struktur coba/tangkap tidak didukung.
tidak-sementara	Sementara loop tidak didukung.
no-disallowed-unary-operators	`++`,`--`, dan operator ~ unary tidak diizinkan.
no-disallowed-binary-operators	`instanceof` Operator tidak diperbolehkan.
tidak ada janji	Proses dan janji asinkron tidak didukung.

“plugin: @aws -appsync/recommended” menyediakan beberapa aturan tambahan tetapi juga mengharuskan Anda untuk menambahkan TypeScript konfigurasi ke proyek Anda.

Rule	Deskripsi
tidak ada rekursi	Panggilan fungsi rekursif tidak diperbolehkan.
no-disallowed-methods	Beberapa metode tidak diperbolehkan. Lihat referensi untuk set lengkap fungsi bawaan yang didukung.
no-function-passing	Melewati fungsi sebagai argumen fungsi ke fungsi tidak diperbolehkan.
no-function-reassign	Fungsi tidak dapat dipindahkan.
no-function-return	Fungsi tidak dapat menjadi nilai kembali fungsi.

Untuk menambahkan plugin ke proyek Anda, ikuti langkah-langkah instalasi dan penggunaan di Memulai dengan ESLint. Kemudian, instal plugin di proyek Anda menggunakan manajer paket proyek Anda (misalnya, npm, yarn, atau pnpm):

```
$ npm install @aws-appsync/eslint-plugin
```

Di `.eslintrc.{js,yml,json}` file Anda, tambahkan “plugin: @aws -appsync/base” atau “plugin: @aws -appsync/recommended” ke properti. `extends` Cuplikan di bawah ini adalah `.eslintrc` konfigurasi sampel dasar untuk: JavaScript

```
{
  "extends": ["plugin:@aws-appsync/base"]
}
```

Untuk menggunakan set aturan “plugin: @aws -appsync/recommended”, instal ketergantungan yang diperlukan:

```
$ npm install -D @typescript-eslint/parser
```

Kemudian, buat `.eslintrc.js` file:

```
{
  "parser": "@typescript-eslint/parser",
  "parserOptions": {
    "ecmaVersion": 2018,
    "project": "./tsconfig.json"
  },
  "extends": ["plugin:@aws-appsync/recommended"]
}
```

Bundling, TypeScript, dan peta sumber

Memanfaatkan pustaka dan menggabungkan kode Anda

Dalam resolver dan kode fungsi Anda, Anda dapat memanfaatkan pustaka kustom dan eksternal selama mereka memenuhi persyaratan. `APPSYNC_JS` Hal ini memungkinkan untuk menggunakan kembali kode yang ada dalam aplikasi Anda. Untuk menggunakan pustaka yang ditentukan oleh beberapa file, Anda harus menggunakan alat bundling, seperti esbuild, untuk menggabungkan kode Anda dalam satu file yang kemudian dapat disimpan ke AWS AppSync resolver atau fungsi Anda.

Saat menggabungkan kode Anda, ingatlah hal berikut:

- `APPSYNC_JS`hanya mendukung modul ECMAScript (ESM).

- `@aws-appsync/*`modul terintegrasi ke dalam `APPSYNC_JS` dan tidak boleh dibundel dengan kode Anda.
- Lingkungan `APPSYNC_JS` runtime mirip dengan NodeJS dalam kode itu tidak berjalan di lingkungan browser.
- Anda dapat menyertakan peta sumber opsional. Namun, jangan sertakan konten sumber.

  Untuk mempelajari lebih lanjut tentang peta sumber, lihat Menggunakan peta sumber.

Misalnya, untuk menggabungkan kode resolver Anda yang terletak di`src/appsync/getPost.resolver.js`, Anda dapat menggunakan perintah esbuild CLI berikut:

```
$ esbuild --bundle \
--sourcemap=inline \
--sources-content=false \
--target=esnext \
--platform=node \
--format=esm \
--external:@aws-appsync/utils \
--outdir=out/appsync \
 src/appsync/getPost.resolver.js
```

Membangun kode Anda dan bekerja dengan TypeScript

TypeScriptadalah bahasa pemrograman yang dikembangkan oleh Microsoft yang menawarkan semua fitur bersama dengan sistem TypeScript pengetikan. JavaScript Anda dapat menggunakan TypeScript untuk menulis kode type-safe dan menangkap kesalahan dan bug pada waktu pembuatan sebelum menyimpan kode Anda. AWS AppSync `@aws-appsync/utils`Paket ini sepenuhnya diketik.

`APPSYNC_JS`Runtime tidak mendukung TypeScript secara langsung. Anda harus terlebih dahulu mentranspile TypeScript kode Anda ke JavaScript kode yang didukung `APPSYNC_JS` runtime sebelum menyimpan kode Anda. AWS AppSync Anda dapat menggunakannya TypeScript untuk menulis kode Anda di lingkungan pengembangan terintegrasi lokal (IDE), tetapi perhatikan bahwa Anda tidak dapat membuat TypeScript kode di AWS AppSync konsol.

Untuk memulai, pastikan Anda telah TypeScriptmenginstal di proyek Anda. Kemudian, konfigurasikan pengaturan TypeScript transkompilasi Anda agar berfungsi dengan `APPSYNC_JS` runtime menggunakan TSConfig. Berikut adalah contoh `tsconfig.json` file dasar yang dapat Anda gunakan:

```
// tsconfig.json
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
   "noEmit": true,
   "moduleResolution": "node",
  }
}
```

Anda kemudian dapat menggunakan alat bundling seperti esbuild untuk mengkompilasi dan menggabungkan kode Anda. Misalnya, mengingat proyek dengan AWS AppSync kode Anda berada di`src/appsync`, Anda dapat menggunakan perintah berikut untuk mengkompilasi dan menggabungkan kode Anda:

```
$ esbuild --bundle \
--sourcemap=inline \
--sources-content=false \
--target=esnext \
--platform=node \
--format=esm \
--external:@aws-appsync/utils \
--outdir=out/appsync \
 src/appsync/**/*.ts
```

Menggunakan Amplify codegen

Anda dapat menggunakan Amplify CLI untuk menghasilkan tipe untuk skema Anda. Dari direktori tempat `schema.graphql` file Anda berada, jalankan perintah berikut dan tinjau petunjuk untuk mengonfigurasi codegen Anda:

```
$  npx @aws-amplify/cli codegen add
```

Untuk membuat ulang codegen Anda dalam keadaan tertentu (misalnya, ketika skema Anda diperbarui), jalankan perintah berikut:

```
$ npx @aws-amplify/cli codegen
```

Anda kemudian dapat menggunakan tipe yang dihasilkan dalam kode resolver Anda. Misalnya, diberikan skema berikut:

```
type Todo {
  id: ID!
  title: String!
  description: String
}

type Mutation {
  createTodo(title: String!, description: String): Todo
}

type Query {
  listTodos: Todo
}
```

Anda dapat menggunakan tipe yang dihasilkan dalam AWS AppSync fungsi contoh berikut:

```
import { Context, util } from '@aws-appsync/utils'
import * as ddb from '@aws-appsync/utils/dynamodb'
import { CreateTodoMutationVariables, Todo } from './API' // codegen

export function request(ctx: Context<CreateTodoMutationVariables>) {
 ctx.args.description = ctx.args.description ?? 'created on ' + util.time.nowISO8601()
 return ddb.put<Todo>({ key: { id: util.autoId() }, item: ctx.args })
}

export function response(ctx) {
 return ctx.result as Todo
}
```

Menggunakan obat generik di TypeScript

Anda dapat menggunakan obat generik dengan beberapa jenis yang disediakan. Misalnya, cuplikan di bawah ini adalah Todo tipe:

```
export type Todo = {
  __typename: "Todo",
  id: string,
  title: string,
  description?: string | null,
};
```

Anda dapat menulis resolver untuk langganan yang memanfaatkan. Todo Di IDE Anda, definisi ketik dan petunjuk pelengkapan otomatis akan memandu Anda menggunakan utilitas `toSubscriptionFilter` transformasi dengan benar:

```
import { util, Context, extensions } from '@aws-appsync/utils'
import { Todo } from './API'

export function request(ctx: Context) {
  return {}
}

export function response(ctx: Context) {
  const filter = util.transform.toSubscriptionFilter<Todo>({
    title: { beginsWith: 'hello' },
    description: { contains: 'created' },
  })
  extensions.setSubscriptionFilter(filter)
  return null
}
```

Linting bundel Anda

Anda dapat secara otomatis lint bundel Anda dengan mengimpor plugin. `esbuild-plugin-eslint` Anda kemudian dapat mengaktifkannya dengan memberikan `plugins` nilai yang memungkinkan kemampuan eslint. Di bawah ini adalah cuplikan yang menggunakan esbuild JavaScript API dalam file bernama: `build.mjs`

```
/* eslint-disable */
import { build } from 'esbuild'
import eslint from 'esbuild-plugin-eslint'
import glob from 'glob'
const files = await glob('src/**/*.ts')

await build({
  format: 'esm',
  target: 'esnext',
  platform: 'node',
  external: ['@aws-appsync/utils'],
  outdir: 'dist/',
  entryPoints: files,
  bundle: true,
  plugins: [eslint({ useEslintrc: true })],
```

```
})
```

Menggunakan peta sumber

Anda dapat memberikan peta sumber inline (`sourcemap`) dengan JavaScript kode Anda. Peta sumber berguna ketika Anda bundel JavaScript atau TypeScript kode dan ingin melihat referensi ke file sumber input Anda di log dan pesan JavaScript kesalahan runtime Anda.

Anda `sourcemap` harus muncul di akhir kode Anda. Ini didefinisikan oleh satu baris komentar yang mengikuti format berikut:

```
//# sourceMappingURL=data:application/json;base64,<base64 encoded string>
```

Inilah contohnya:

```
//# sourceMappingURL=data:application/
json;base64,ewogICJ2ZXJzaW9uIjogMywKICAic291cmNlcyI6IFsibGliLmpzIiwgImNvZGUuanMiXSwKICAibWFwcGl
```

Peta sumber dapat dibuat dengan esbuild. Contoh di bawah ini menunjukkan cara menggunakan esbuild JavaScript API untuk menyertakan peta sumber sebaris saat kode dibuat dan dibundel:

```
/* eslint-disable */
import { build } from 'esbuild'
import eslint from 'esbuild-plugin-eslint'
import glob from 'glob'
const files = await glob('src/**/*.ts')

await build({
  sourcemap: 'inline',
  sourcesContent: false,

  format: 'esm',
  target: 'esnext',
  platform: 'node',
  external: ['@aws-appsync/utils'],
  outdir: 'dist/',
  entryPoints: files,
  bundle: true,
  plugins: [eslint({ useEslintrc: true })],
})
```

Secara khusus, `sourcesContent` opsi `sourcemap` dan menentukan bahwa peta sumber harus ditambahkan sebaris di akhir setiap build tetapi tidak boleh menyertakan konten sumber. Sebagai konvensi, kami sarankan untuk tidak menyertakan konten sumber dalam Anda`sourcemap`. Anda dapat menonaktifkan ini di esbuild dengan menyetel `sources-content` ke`false`.

Untuk mengilustrasikan cara kerja peta sumber, tinjau contoh berikut di mana kode resolver mereferensikan fungsi pembantu dari pustaka pembantu. Kode berisi pernyataan log dalam kode resolver dan di pustaka pembantu:

. /src/default.resolver.ts (resolver Anda)

```
import { Context } from '@aws-appsync/utils'
import { hello, logit } from './helper'

export function request(ctx: Context) {
  console.log('start >')
  logit('hello world', 42, true)
  console.log('< end')
  return 'test'
}

export function response(ctx: Context): boolean {
  hello()
  return ctx.prev.result
}
```

.src/helper.ts (file pembantu)

```
export const logit = (...rest: any[]) => {
  // a special logger
  console.log('[logger]', ...rest.map((r) => `<${r}>`))
}

export const hello = () => {
  // This just returns a simple sentence, but it could do more.
  console.log('i just say hello..')
}
```

Saat Anda membangun dan menggabungkan file resolver, kode resolver Anda akan menyertakan peta sumber inline. Saat resolver Anda berjalan, entri berikut akan muncul di log: CloudWatch

```
INFO - ../src/default.resolver.ts:5:2: "start >"
INFO - ../src/helper.ts:3:2: "[logger]" "<hello world>" "<42>" "<true>"
INFO - ../src/default.resolver.ts:7:2: "< end"
{"logType":"BeforeRequestFunctionEvaluation","path":["logstuff"],"fieldName":"logstuff","resolverArn":"arn:aws:
INFO - ../src/helper.ts:8:2: "i just say hello.."
{"logType":"AfterResponseFunctionEvaluation","path":["logstuff"],"fieldName":"logstuff","resolverArn":"arn:aws:
```

Melihat entri di CloudWatch log, Anda akan melihat bahwa fungsionalitas dari dua file telah dibundel bersama dan berjalan secara bersamaan. Nama file asli dari setiap file juga tercermin dengan jelas dalam log.

Pengujian

Anda dapat menggunakan perintah `EvaluateCode` API untuk menguji resolver dan penangan fungsi Anda dari jarak jauh dengan data tiruan sebelum menyimpan kode Anda ke resolver atau fungsi. Untuk memulai dengan perintah, pastikan Anda telah menambahkan `appsync:evaluatecode` izin ke kebijakan Anda. Misalnya:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "appsync:evaluateCode",
            "Resource": "arn:aws:appsync:<region>:<account>:*"
        }
    ]
}
```

Anda dapat memanfaatkan perintah dengan menggunakan AWSCLI atau AWS SDK. Misalnya, untuk menguji kode Anda menggunakan CLI, cukup arahkan ke file Anda, berikan konteks, dan tentukan handler yang ingin Anda evaluasi:

```
aws appsync evaluate-code \
  --code file://code.js \
  --function request \
  --context file://context.json \
  --runtime name=APPSYNC_JS,runtimeVersion=1.0.0
```

Responsnya `evaluationResult` berisi muatan yang dikembalikan oleh handler Anda. Ini juga berisi `logs` objek yang menyimpan daftar log yang dihasilkan oleh handler Anda selama evaluasi. Hal ini memudahkan untuk men-debug eksekusi kode Anda dan melihat informasi tentang evaluasi Anda untuk membantu memecahkan masalah. Misalnya:

```
{
    "evaluationResult": "{\"operation\":\"PutItem\",\"key\":{\"id\":{\"S\":\"record-id
\"}},\"attributeValues\":{\"owner\":{\"S\":\"John doe\"},\"expectedVersion\":{\"N\":2},
\"authorId\":{\"S\":\"Sammy Davis\"}}}",
    "logs": [
        "INFO - code.js:5:3: \"current id\" \"record-id\"",
        "INFO - code.js:9:3: \"request evaluated\""
    ]
}
```

Hasil evaluasi dapat diuraikan sebagai JSON, yang memberikan:

```
{
  "operation": "PutItem",
  "key": {
    "id": {
      "S": "record-id"
    }
  },
  "attributeValues": {
    "owner": {
      "S": "John doe"
    },
    "expectedVersion": {
      "N": 2
    },
    "authorId": {
      "S": "Sammy Davis"
    }
  }
}
```

Dengan menggunakan SDK, Anda dapat dengan mudah menggabungkan pengujian dari rangkaian pengujian untuk memvalidasi perilaku kode Anda. Contoh kami di sini menggunakan Jest Testing Framework, tetapi rangkaian pengujian apa pun berfungsi. Cuplikan berikut menunjukkan validasi

hipotetis berjalan. Perhatikan bahwa kami mengharapkan respons evaluasi menjadi JSON yang valid, jadi kami gunakan `JSON.parse` untuk mengambil JSON dari respons string:

```
const AWS = require('aws-sdk')
const fs = require('fs')
const client = new AWS.AppSync({ region: 'us-east-2' })
const runtime = {name:'APPSYNC_JS',runtimeVersion:'1.0.0')

test('request correctly calls DynamoDB', async () => {
  const code = fs.readFileSync('./code.js', 'utf8')
  const context = fs.readFileSync('./context.json', 'utf8')
  const contextJSON = JSON.parse(context)

  const response = await client.evaluateCode({ code, context, runtime, function:
 'request' }).promise()
  const result = JSON.parse(response.evaluationResult)

  expect(result.key.id.S).toBeDefined()
  expect(result.attributeValues.firstname.S).toEqual(contextJSON.arguments.firstname)
})
```

Ini menghasilkan hasil sebagai berikut:

```
Ran all test suites.
> jest

PASS ./index.test.js
# request correctly calls DynamoDB (543 ms)
Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 totalTime: 1.511 s, estimated 2 s
```

Migrasi dari VTL ke JavaScript

AWS AppSync memungkinkan Anda untuk menulis logika bisnis Anda untuk resolver dan fungsi Anda menggunakan VTL atau. JavaScript Dengan kedua bahasa, Anda menulis logika yang menginstruksikan AWS AppSync layanan tentang cara berinteraksi dengan sumber data Anda. Dengan VTL, Anda menulis template pemetaan yang harus mengevaluasi ke string yang dikodekan JSON yang valid. Dengan JavaScript, Anda menulis penangan permintaan dan respons yang mengembalikan objek. Anda tidak mengembalikan string yang dikodekan JSON.

Misalnya, ambil template pemetaan VTL berikut untuk mendapatkan item Amazon DynamoDB:

```
{
    "operation": "GetItem",
    "key": {
        "id": $util.dynamodb.toDynamoDBJson($ctx.args.id),
    }
}
```

Utilitas `$util.dynamodb.toDynamoDBJson` mengembalikan string JSON yang dikodekan. Jika `$ctx.args.id` disetel ke`<id>`, template mengevaluasi ke string yang dikodekan JSON yang valid:

```
{
    "operation": "GetItem",
    "key": {
        "id": {"S": "<id>"},
    }
}
```

Saat bekerja dengan JavaScript, Anda tidak perlu mencetak string mentah yang disandikan JSON dalam kode Anda, dan menggunakan utilitas seperti tidak diperlukan. `toDynamoDBJson` Contoh yang setara dari template pemetaan di atas adalah:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'GetItem',
    key: {id: util.dynamodb.toDynamoDB(ctx.args.id)}
  };
}
```

Alternatifnya adalah menggunakan`util.dynamodb.toMapValues`, yang merupakan pendekatan yang direkomendasikan untuk menangani objek nilai:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'GetItem',
    key: util.dynamodb.toMapValues({ id: ctx.args.id }),
  };
}
```

Ini mengevaluasi untuk:

```
{
  "operation": "GetItem",
  "key": {
    "id": {
      "S": "<id>"
    }
  }
}
```

> **Note**
>
> Sebaiknya gunakan modul DynamoDB dengan sumber data DynamoDB:
>
> ```
> import * as ddb from '@aws-appsync/utils/dynamodb'
>
> export function request(ctx) {
>  ddb.get({ key: { id: ctx.args.id } })
> }
> ```

Sebagai contoh lain, ambil template pemetaan berikut untuk menempatkan item di sumber data Amazon DynamoDB:

```
{
    "operation" : "PutItem",
    "key" : {
        "id": $util.dynamodb.toDynamoDBJson($util.autoId()),
    },
    "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
}
```

Saat dievaluasi, string template pemetaan ini harus menghasilkan string yang dikodekan JSON yang valid. Saat menggunakan JavaScript, kode Anda mengembalikan objek permintaan secara langsung:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { id = util.autoId(), ...values } = ctx.args;
  return {
    operation: 'PutItem',
```

```
    key: util.dynamodb.toMapValues({ id }),
    attributeValues: util.dynamodb.toMapValues(values),
  };
}
```

yang mengevaluasi untuk:

```
{
  "operation": "PutItem",
  "key": {
    "id": { "S": "2bff3f05-ff8c-4ed8-92b4-767e29fc4e63" }
  },
  "attributeValues": {
    "firstname": { "S": "Shaggy" },
    "age": { "N": 4 }
  }
}
```

Note

Sebaiknya gunakan modul DynamoDB dengan sumber data DynamoDB:

```
import { util } from '@aws-appsync/utils'
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
 const { id = util.autoId(), ...item } = ctx.args
 return ddb.put({ key: { id }, item })
}
```

Memilih antara akses sumber data langsung dan proxy melalui sumber data Lambda

Dengan AWS AppSync dan `APPSYNC_JS` runtime, Anda dapat menulis kode Anda sendiri yang mengimplementasikan logika bisnis kustom Anda dengan menggunakan AWS AppSync fungsi untuk mengakses sumber data Anda. Ini memudahkan Anda untuk berinteraksi langsung dengan sumber data seperti Amazon DynamoDB, Aurora OpenSearch Tanpa Server, Layanan, API HTTP, AWS dan layanan lainnya tanpa harus menggunakan layanan atau infrastruktur komputasi tambahan. AWS AppSync juga memudahkan untuk berinteraksi dengan AWS Lambda fungsi dengan mengkonfigurasi

sumber data Lambda. Sumber data Lambda memungkinkan Anda menjalankan logika bisnis yang kompleks menggunakan AWS Lambda kemampuan set lengkap untuk menyelesaikan permintaan GraphQL. Dalam kebanyakan kasus, AWS AppSync fungsi yang terhubung langsung ke sumber data targetnya akan menyediakan semua fungsionalitas yang Anda butuhkan. Dalam situasi di mana Anda perlu menerapkan logika bisnis kompleks yang tidak didukung oleh `APPSYNC_JS` runtime, Anda dapat menggunakan sumber data Lambda sebagai proxy untuk berinteraksi dengan sumber data target Anda.

	Integrasi sumber data langsung	Sumber data Lambda sebagai proxy
Kasus penggunaan	AWS AppSync functions interact directly with API data sources.	AWS AppSync functions call Lambdas that interact with API data sources.
Runtime	*`APPSYNC_JS`* (JavaScript)	Setiap runtime Lambda yang didukung
Maximum size of code	32.000 karakter per fungsi AWS AppSync	50 MB (zip, untuk diunggah langsung) per Lambda
External modules	Terbatas - APPSYNC_JS hanya mendukung fitur	Ya
Call any AWS service	Ya - Menggunakan sumber data AWS AppSync HTTP	Ya - Menggunakan AWS SDK
Access to the request header	Ya	Ya
Network access	Tidak	Ya
File system access	Tidak	Ya
Logging and metrics	Ya	Ya
Build and test entirely within AppSync	Ya	Tidak
Cold start	Tidak	Tidak - Dengan konkurensi yang disediakan

Auto-scaling	Ya - secara transparan oleh AWS AppSync	Ya - Seperti yang dikonfigurasi di Lambda
Pricing	Tanpa biaya tambahan	Dibebankan untuk penggunaan Lambda

AWS AppSync fungsi yang terintegrasi langsung dengan sumber data target mereka ideal untuk kasus penggunaan seperti berikut:

- Berinteraksi dengan Amazon DynamoDB, Aurora Tanpa Server, dan Layanan OpenSearch
- Berinteraksi dengan API HTTP dan meneruskan header yang masuk
- Berinteraksi dengan AWS layanan menggunakan sumber data HTTP (dengan AWS AppSync secara otomatis menandatangani permintaan dengan peran sumber data yang disediakan)
- Menerapkan kontrol akses sebelum mengakses sumber data
- Menerapkan penyaringan data yang diambil sebelum memenuhi permintaan
- Menerapkan orkestrasi sederhana dengan eksekusi AWS AppSync fungsi secara berurutan dalam pipa resolver
- Mengontrol koneksi caching dan berlangganan dalam kueri dan mutasi.

AWS AppSync fungsi yang menggunakan sumber data Lambda sebagai proxy sangat ideal untuk kasus penggunaan seperti berikut:

- Menggunakan bahasa selain JavaScript atau Velocity Template Language (VTL)
- Menyesuaikan dan mengendalikan CPU atau memori untuk mengoptimalkan kinerja
- Mengimpor pustaka pihak ketiga atau memerlukan fitur yang tidak didukung di `APPSYNC_JS`
- Membuat beberapa permintaan jaringan dan/atau mendapatkan akses sistem file untuk memenuhi kueri
- Permintaan batching menggunakan konfigurasi batching.

Referensi objek konteks penyelesai

AWS AppSyncmendefinisikan satu set variabel dan fungsi untuk bekerja dengan permintaan dan penangan respon. Ini membuat operasi logis pada data lebih mudah dengan GraphQL. Dokumen ini menjelaskan fungsi-fungsi tersebut dan memberikan contoh.

Mengakses`context`

The`context`argumen dari penangan permintaan dan respons adalah objek yang menyimpan semua informasi kontekstual untuk pemanggilan resolver Anda. Ini memiliki struktur sebagai berikut:

```
type Context = {
  arguments: any;
  args: any;
  identity: Identity;
  source: any;
  error?: {
    message: string;
    type: string;
  };
  stash: any;
  result: any;
  prev: any;
  request: Request;
  info: Info;
};
```

> **Note**
>
> Anda akan sering menemukan bahwa`context`Objek tersebut disebut sebagai`ctx`.

Setiap bidang di`context`objek didefinisikan sebagai berikut:

`context`ladang

`arguments`

Peta yang berisi semua argumen GraphQL untuk bidang ini.

`identity`

Objek yang berisi informasi tentang penelepon. Untuk informasi lebih lanjut tentang struktur bidang ini, lihatIdentitas.

`source`

Peta yang berisi resolusi bidang induk.

`stash`

Stash adalah objek yang tersedia di dalam setiap resolver dan function handler. Objek simpanan yang sama hidup melalui satu resolver run. Ini berarti Anda dapat menggunakan stash untuk meneruskan data arbitrer di seluruh penangan permintaan dan respons dan di seluruh fungsi dalam resolver pipeline.

> **Note**
>
> Anda tidak dapat menghapus atau mengganti seluruh simpanan, tetapi Anda dapat menambahkan, memperbarui, menghapus, dan membaca properti simpanan.

Anda dapat menambahkan item ke simpanan dengan memodifikasi salah satu contoh kode di bawah ini:

```
//Example 1
ctx.stash.newItem = { key: "something" }

//Example 2
Object.assign(ctx.stash, {key1: value1, key2: value})
```

Anda dapat menghapus item dari simpanan dengan memodifikasi kode di bawah ini:

```
delete ctx.stash.key
```

`result`

Wadah untuk hasil resolver ini. Bidang ini hanya tersedia untuk penangan respons.

Misalnya, jika Anda menyelesaikan`author`bidang query berikut:

```
query {
    getPost(id: 1234) {
        postId
        title
        content
        author {
            id
```

```
            name
        }
    }
}
```

Kemudian penuhcontextvariabel tersedia ketika penangan respons dievaluasi:

```
{
  "arguments" : {
    id: "1234"
  },
  "source": {},
  "result" : {
      "postId": "1234",
      "title": "Some title",
      "content": "Some content",
      "author": {
        "id": "5678",
        "name": "Author Name"
      }
  },
  "identity" : {
      "sourceIp" : ["x.x.x.x"],
      "userArn" : "arn:aws:iam::123456789012:user/appsync",
      "accountId" : "666666666666",
      "user" : "AIDAAAAAAAAAAAAAAAAA"
  }
}
```

prev.result

Hasil dari operasi apa pun sebelumnya dieksekusi dalam resolver pipa.

Jika operasi sebelumnya adalah penangan permintaan penyelesai pipa, makactx.prev.resultmewakili hasil evaluasi itu dan tersedia untuk fungsi pertama dalam pipa.

Jika operasi sebelumnya adalah fungsi pertama, makactx.prev.resultmerupakan hasil evaluasi dari penangan respons fungsi pertama dan tersedia untuk fungsi kedua dalam pipa.

Jika operasi sebelumnya adalah fungsi terakhir, makactx.prev.resultmewakili hasil evaluasi dari fungsi terakhir dan tersedia untuk penangan respons penyelesai pipa.

`info`

Objek yang berisi informasi tentang permintaan GraphQL. Untuk struktur bidang ini, lihatInfo.

Identitas

Theidentitybagian berisi informasi tentang penelepon. Bentuk bagian ini tergantung pada jenis otorisasi AndaAWS AppSyncAPI.

Untuk informasi lebih lanjut tentangAWS AppSyncopsi keamanan, lihatOtorisasi dan otentikasi.

`API_KEY`otorisasi

Theidentitybidang tidak dihuni.

`AWS_LAMBDA`otorisasi

Theidentitymemiliki bentuk sebagai berikut:

```
type AppSyncIdentityLambda = {
  resolverContext: any;
};
```

TheidentityberisiresolverContextkunci, berisi yang samaresolverContextkonten yang dikembalikan oleh fungsi Lambda yang mengotorisasi permintaan.

`AWS_IAM`otorisasi

Theidentitymemiliki bentuk sebagai berikut:

```
type AppSyncIdentityIAM = {
  accountId: string;
  cognitoIdentityPoolId: string;
  cognitoIdentityId: string;
  sourceIp: string[];
  username: string;
  userArn: string;
  cognitoIdentityAuthType: string;
  cognitoIdentityAuthProvider: string;
};
```

`AMAZON_COGNITO_USER_POOLS`otorisasi

Theidentitymemiliki bentuk sebagai berikut:

```
type AppSyncIdentityCognito = {
  sourceIp: string[];
  username: string;
  groups: string[] | null;
  sub: string;
  issuer: string;
  claims: any;
  defaultAuthStrategy: string;
};
```

Setiap bidang didefinisikan sebagai berikut:

`accountId`

TheAWSID akun penelepon.

`claims`

Klaim yang dimiliki pengguna.

`cognitoIdentityAuthType`

Entah diautentikasi atau tidak diautentikasi berdasarkan tipe identitas.

`cognitoIdentityAuthProvider`

Daftar informasi penyedia identitas eksternal yang dipisahkan koma yang digunakan dalam memperoleh kredensil yang digunakan untuk menandatangani permintaan.

`cognitoIdentityId`

ID identitas Amazon Cognito dari penelepon.

`cognitoIdentityPoolId`

ID kumpulan identitas Amazon Cognito yang terkait dengan pemanggil.

`defaultAuthStrategy`

Strategi otorisasi default untuk penelepon ini (`ALLOWatauDENY`).

`issuer`

Penerbit token.

sourceIp

Alamat IP sumber penelepon yangAWS AppSyncmenerima. Jika permintaan tidak termasuk`x-forwarded-for`header, nilai IP sumber hanya berisi satu alamat IP dari koneksi TCP. Jika permintaan tersebut mencakup`x-forwarded-for`header, sumber IP adalah daftar alamat IP dari`x-forwarded-for`header, selain alamat IP dari koneksi TCP.

sub

UUID dari pengguna yang diautentikasi.

user

Pengguna IAM.

userArn

Nama Sumber Daya Amazon (ARN) dari pengguna IAM.

username

Nama pengguna pengguna yang diautentikasi. Dalam kasus`AMAZON_COGNITO_USER_POOLS`otorisasi, nilainama penggunaadalah nilai atributcognito:nama pengguna. Dalam kasus`AWS_IAM`otorisasi, nilainama penggunaadalah nilai dariAWSprinsipal pengguna. Jika Anda menggunakan otorisasi IAM dengan kredensi yang dijual dari kumpulan identitas Amazon Cognito, kami sarankan Anda menggunakannya`cognitoIdentityId`.

Akses header permintaan

AWS AppSyncmendukung meneruskan header khusus dari klien dan mengaksesnya di resolver GraphQL Anda dengan menggunakan`ctx.request.headers`. Anda kemudian dapat menggunakan nilai header untuk tindakan seperti memasukkan data ke sumber data atau pemeriksaan otorisasi. Anda dapat menggunakan header permintaan tunggal atau beberapa menggunakan`$curl`dengan kunci API dari baris perintah, seperti yang ditunjukkan pada contoh berikut:

Contoh header tunggal

Misalkan Anda menetapkan header dari`custom`dengan nilai`adia`seperti berikut ini:

```
curl -XPOST -H "Content-Type:application/graphql" -H "custom:nadia" -H "x-api-key:<API-
KEY-VALUE>" -d '{"query":"mutation { createEvent(name: \"demo\", when: \"Next Friday!
\", where: \"Here!\") {id name when where description}}"}' https://<ENDPOINT>/graphql
```

Ini kemudian dapat diakses dengan`ctx.request.headers.custom`. Misalnya, mungkin dalam kode berikut untuk DynamoDB:

```
"custom": util.dynamodb.toDynamoDB(ctx.request.headers.custom)
```

Beberapa contoh header

Anda juga dapat meneruskan beberapa header dalam satu permintaan dan mengaksesnya di penangan resolver. Misalnya, jika`custom`header diatur dengan dua nilai:

```
curl -XPOST -H "Content-Type:application/graphql" -H "custom:bailey" -H "custom:nadia"
 -H "x-api-key:<API-KEY-VALUE>" -d '{"query":"mutation { createEvent(name: \"demo
\", when: \"Next Friday!\", where: \"Here!\") {id name when where description}}"}'
 https://<ENDPOINT>/graphql
```

Anda kemudian dapat mengakses ini sebagai array, seperti`ctx.request.headers.custom[1]`.

> **Note**
>
> AWS AppSynctidak mengekspos header cookie di`ctx.request.headers`.

Akses permintaan nama domain kustom

AWS AppSyncmendukung konfigurasi domain kustom yang dapat Anda gunakan untuk mengakses GraphQL dan titik akhir real-time untuk API Anda. Saat membuat permintaan dengan nama domain khusus, Anda bisa mendapatkan nama domain menggunakan`ctx.request.domainName`.

Saat menggunakan nama domain titik akhir GraphQL default, nilainya adalah`null`.

Info

The`info`bagian berisi informasi tentang permintaan GraphQL. Bagian ini memiliki bentuk berikut:

```
type Info = {
```

```
  fieldName: string;
  parentTypeName: string;
  variables: any;
  selectionSetList: string[];
  selectionSetGraphQL: string;
};
```

Setiap bidang didefinisikan sebagai berikut:

`fieldName`

Nama bidang yang saat ini sedang diselesaikan.

`parentTypeName`

Nama tipe induk untuk bidang yang saat ini sedang diselesaikan.

`variables`

Peta yang menampung semua variabel yang diteruskan ke permintaan GraphQL.

`selectionSetList`

Sebuah daftar representasi bidang dalam set pilihan GraphQL. Bidang yang dialias hanya direferensikan dengan nama alias, bukan nama bidang. Contoh berikut menunjukkan ini secara rinci.

`selectionSetGraphQL`

Representasi string dari set seleksi, diformat sebagai bahasa definisi skema GraphQL (SDL). Meskipun fragmen tidak digabungkan ke dalam kumpulan seleksi, fragmen sebaris dipertahankan, seperti yang ditunjukkan pada contoh berikut.

> **Note**
>
> `JSON.stringify`tidak akan termasuk`selectionSetGraphQL`dan`selectionSetList`dalam serialisasi string. Anda harus merujuk properti ini secara langsung.

Misalnya, jika Anda menyelesaikan`getPost`bidang query berikut:

```
query {
```

```
  getPost(id: $postId) {
    postId
    title
    secondTitle: title
    content
    author(id: $authorId) {
      authorId
      name
    }
    secondAuthor(id: "789") {
      authorId
    }
    ... on Post {
      inlineFrag: comments: {
        id
      }
    }
    ... postFrag
  }
}

fragment postFrag on Post {
  postFrag: comments: {
    id
  }
}
```

Kemudian penuh`ctx.info`variabel yang tersedia saat memproses handler mungkin:

```
{
  "fieldName": "getPost",
  "parentTypeName": "Query",
  "variables": {
    "postId": "123",
    "authorId": "456"
  },
  "selectionSetList": [
    "postId",
    "title",
    "secondTitle"
    "content",
    "author",
    "author/authorId",
```

```
    "author/name",
    "secondAuthor",
    "secondAuthor/authorId",
    "inlineFragComments",
    "inlineFragComments/id",
    "postFragComments",
    "postFragComments/id"
  ],
  "selectionSetGraphQL": "{\n  getPost(id: $postId) {\n    postId\n    title\n
 secondTitle: title\n    content\n    author(id: $authorId) {\n      authorId\n
 name\n    }\n    secondAuthor(id: \"789\") {\n      authorId\n    }\n    ... on Post
 {\n      inlineFrag: comments {\n        id\n      }\n    }\n    ... postFrag\n  }\n}"
}
```

`selectionSetList`mengekspos hanya bidang yang termasuk tipe saat ini. Jika tipe saat ini adalah antarmuka atau gabungan, hanya bidang yang dipilih milik antarmuka yang diekspos. Misalnya, diberikan skema berikut:

```
type Query {
    node(id: ID!): Node
}

interface Node {
    id: ID
}

type Post implements Node {
    id: ID
    title: String
    author: String
}

type Blog implements Node {
    id: ID
    title: String
    category: String
}
```

Dan query berikut:

```
query {
    node(id: "post1") {
```

```
        id
        ... on Post {
            title
        }

        ... on Blog {
            title
        }
    }
}
```

Saat meneleponctx.info.selectionSetListdiQuery.noderesolusi lapangan, hanyaiddterpapar:

```
"selectionSetList": [
    "id"
]
```

JavaScript fitur runtime untuk resolver dan fungsi

Lingkungan APPSYNC_JS runtime menyediakan fungsionalitas yang mirip dengan ECMAScript (ES) versi 6.0. Ini mendukung subset fitur-fiturnya dan menyediakan beberapa metode tambahan (utilitas) yang bukan bagian dari spesifikasi ES. Topik berikut mencantumkan semua fitur bahasa yang didukung.

> **Note**
>
> Saat ini, referensi ini hanya berlaku untuk runtime versi 1.0.0.

Topik

- Fitur runtime yang didukung
- Utilitas bawaan
- Modul bawaan
- Utilitas runtime
- Pembantu waktu di util.time
- Pembantu DynamoDB di util.dynamodb

- Pembantu HTTP di util.http
- Pembantu transformasi di util.transform
- Pembantu string di util.str
- Ekstensi
- Pembantu XMLdi util.xml

Fitur runtime yang didukung

Bagian di bawah ini menjelaskan kumpulan fitur yang didukung dari runtime APPSYNC_JS.

Fitur inti

Fitur inti berikut didukung.

Jenis

Jenis berikut didukung:

- angka
- tali
- boolean
- objek
- larik
- fungsi

Operator

Operator didukung, termasuk:

- operator matematika standar (`+`,`-`,`/`,`%`,`*`, dll.)
- operator penggabungan nullish () `??`
- Rantai opsional () `?.`
- operator bitwise
- `void`dan `typeof` operator

Operator berikut tidak didukung:

- operator unary (`++`,`--`, dan`~`)
- Operator `in`

> **Note**
>
> Gunakan `Object.hasOwn` operator untuk memeriksa apakah properti yang ditentukan ada di objek yang ditentukan.

Pernyataan

Pernyataan berikut didukung:

- `const`
- `let`
- `var`
- `break`
- `else`
- `for-in`
- `for-of`
- `if`
- `return`
- `switch`
- sintaks spread

Berikut ini tidak didukung:

- `catch`
- `continue`
- `do-while`
- `finally`
- `for(initialization; condition; afterthought)`

> **Note**
>
> Pengecualian adalah `for-in` dan `for-of` ekspresi, yang didukung.

- `throw`
- `try`
- `while`
- pernyataan berlabel

Literal

Literal template ES 6 berikut didukung:

- String multi-baris
- Interpolasi ekspresi
- Template bersarang

Fungsi

Sintaks fungsi berikut didukung:

- Deklarasi fungsi didukung.
- Fungsi panah ES 6 didukung.
- ES 6 sisanya sintaks parameter didukung.

Mode ketat

Fungsi beroperasi dalam mode ketat secara default, sehingga Anda tidak perlu menambahkan pernyataan `use_strict` dalam kode fungsi Anda. Ini tidak dapat diubah.

Objek primitif

Objek primitif ES berikut dan fungsinya didukung.

Objek

Objek berikut didukung:

- `Object.assign()`
- `Object.entries()`
- `Object.hasOwn()`
- `Object.keys()`
- `Object.values()`
- `delete`

Tali

String berikut didukung:

- `String.prototype.length()`
- `String.prototype.charAt()`
- `String.prototype.concat()`
- `String.prototype.endsWith()`
- `String.prototype.indexOf()`
- `String.prototype.lastIndexOf()`
- `String.raw()`
- `String.prototype.replace()`

> **Note**
>
> Ekspresi reguler tidak didukung.

- `String.prototype.replaceAll()`

> **Note**
>
> Ekspresi reguler tidak didukung.

- `String.prototype.slice()`
- `String.prototype.split()`
- `String.prototype.startsWith()`
- `String.prototype.toLowerCase()`
- `String.prototype.toUpperCase()`

- `String.prototype.trim()`
- `String.prototype.trimEnd()`
- `String.prototype.trimStart()`

Nomor

Angka-angka berikut didukung:

- `Number.isFinite`
- `Number.isNaN`

Objek dan fungsi bawaan

Fungsi dan objek berikut didukung.

Matematika

Fungsi matematika berikut didukung:

- `Math.random()`
- `Math.min()`
- `Math.max()`
- `Math.round()`
- `Math.floor()`
- `Math.ceil()`

Array

Metode array berikut didukung:

- `Array.prototype.length`
- `Array.prototype.concat()`
- `Array.prototype.fill()`
- `Array.prototype.flat()`
- `Array.prototype.indexOf()`
- `Array.prototype.join()`

- `Array.prototype.lastIndexOf()`
- `Array.prototype.pop()`
- `Array.prototype.push()`
- `Array.prototype.reverse()`
- `Array.prototype.shift()`
- `Array.prototype.slice()`
- `Array.prototype.sort()`

> **Note**
>
> `Array.prototype.sort()`tidak mendukung argumen.

- `Array.prototype.splice()`
- `Array.prototype.unshift()`
- `Array.prototype.forEach()`
- `Array.prototype.map()`
- `Array.prototype.flatMap()`
- `Array.prototype.filter()`
- `Array.prototype.reduce()`
- `Array.prototype.reduceRight()`
- `Array.prototype.find()`
- `Array.prototype.some()`
- `Array.prototype.every()`
- `Array.prototype.findIndex()`
- `Array.prototype.findLast()`
- `Array.prototype.findLastIndex()`
- `delete`

Konsol

Objek konsol tersedia untuk debugging. Selama eksekusi kueri langsung, log konsol /pernyataan kesalahan dikirim ke Amazon CloudWatch Logs (jika logging diaktifkan). Selama evaluasi kode dengan`evaluateCode`, pernyataan log dikembalikan dalam respons perintah.

- `console.error()`
- `console.log()`

JSON

Metode JSON berikut didukung:

- `JSON.parse()`

> **Note**
>
> Mengembalikan string kosong jika string diurai tidak valid JSON.

- `JSON.stringify()`

Fungsi

- `call`Metode `applybind`,, dan tidak didukung.
- Tidak mendukung konstruktor fungsi.
- Melewati fungsi sebagai argumen tidak didukung.
- Panggilan fungsi rekursif tidak didukung.

Janji

Proses asinkron tidak didukung, dan janji tidak didukung.

> **Note**
>
> Akses jaringan dan sistem file tidak didukung dalam `APPSYNC_JS` runtime diAWS AppSync. AWS AppSyncmenangani semua operasi I/O berdasarkan permintaan yang dibuat oleh AWS AppSync resolver atau fungsi. AWS AppSync

Global

Mendukung konstanta global berikut:

- `NaN`

- `Infinity`
- `undefined`
- `util`
- `extensions`
- `runtime`

Jenis kesalahan

Melempar kesalahan dengan `throw` tidak didukung. Anda dapat mengembalikan kesalahan dengan menggunakan `util.error()` fungsi. Anda dapat menyertakan kesalahan dalam respons GraphQL Anda dengan menggunakan fungsi tersebut. `util.appendError`

Untuk informasi selengkapnya, lihat Kesalahan utils.

Utilitas bawaan

`util`Variabel berisi metode utilitas umum untuk membantu Anda bekerja dengan data. Kecuali ditentukan lain, semua utilitas menggunakan set karakter UTF-8.

Pengkodean utilitas

Pengkodean daftar utilitas

`util.urlEncode(String)`

Mengembalikan string masukan sebagai string `application/x-www-form-urlencoded` dikodekan.

`util.urlDecode(String)`

Mendekode string yang `application/x-www-form-urlencoded` dikodekan kembali ke bentuk yang tidak dikodekan.

`util.base64Encode(string) : string`

Mengkodekan input ke dalam string yang dikodekan base64.

`util.base64Decode(string) : string`

Mendekode data dari string yang dikodekan base64.

Utilitas pembuatan ID

Daftar utilitas pembuatan ID

`util.autoId()`

Mengembalikan UUID 128-bit yang dihasilkan secara acak.

`util.autoUlid()`

Mengembalikan ULID 128-bit yang dihasilkan secara acak (Universalally Unique Lexicographically Sortable Identifier).

`util.autoKsuid()`

Mengembalikan 128-bit yang dihasilkan secara acak KSUID (K-Sortable Unique Identifier) base62 dikodekan sebagai String dengan panjang 27.

Kesalahan utils

Daftar utilitas kesalahan

`util.error(String, String?, Object?, Object?)`

Melempar kesalahan khusus. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Selain itu, `errorType` bidang, `data` bidang, dan `errorInfo` bidang dapat ditentukan. `data`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL.

> **Note**
>
> `data`akan disaring berdasarkan set pemilihan kueri. `errorInfo`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL. `errorInfo`tidak akan disaring berdasarkan set pemilihan kueri.

`util.appendError(String, String?, Object?, Object?)`

Menambahkan kesalahan kustom. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil

pemanggilan. Selain itu, `errorType` bidang, `data` bidang, dan `errorInfo` bidang dapat ditentukan. Tidak sepertiutil.error(String, String?, Object?, Object?), evaluasi template tidak akan terganggu, sehingga data dapat dikembalikan ke penelepon. `data`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL.

> **Note**
>
> `data`akan disaring berdasarkan set pemilihan kueri. `errorInfo`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL. `errorInfo`tidak akan disaring berdasarkan set pemilihan kueri.

Utils pencocokan jenis dan pola

Jenis dan pola pencocokan daftar utilitas

`util.matches(String, String) : Boolean`

Mengembalikan nilai true jika pola tertentu dalam argumen pertama cocok dengan data yang disediakan dalam argumen kedua. Pola harus berupa ekspresi reguler sepertiutil.matches("a*b", "aaaaab"). Fungsionalitas ini didasarkan pada Pola, yang dapat Anda referensi untuk dokumentasi lebih lanjut.

`util.authType()`

Mengembalikan String yang menjelaskan jenis multi-auth yang digunakan oleh permintaan, mengembalikan “IAM Authorization”, “User Pool Authorization”, “Open ID Connect Authorization”, atau “API Key Authorization”.

Kembalikan utilitas perilaku nilai

Kembalikan daftar utilitas perilaku nilai

`util.escapeJavaScript(String)`

Mengembalikan string masukan sebagai string JavaScript lolos.

Utils otorisasi penyelesai

Daftar utilitas otorisasi penyelesai

`util.unauthorized()`

Lemparan `Unauthorized` untuk bidang yang sedang diselesaikan. Gunakan ini dalam templat pemetaan permintaan atau respons untuk menentukan apakah akan mengizinkan pemanggil menyelesaikan bidang.

Modul bawaan

Modul adalah bagian dari `APPSYNC_JS` runtime dan menyediakan utilitas untuk membantu menulis JavaScript resolver dan fungsi.

Fungsi modul DynamoDB

Fungsi modul DynamoDB memberikan pengalaman yang ditingkatkan saat berinteraksi dengan sumber data DynamoDB. Anda dapat membuat permintaan terhadap sumber data DynamoDB Anda menggunakan fungsi dan tanpa menambahkan pemetaan tipe.

Modul diimpor menggunakan`@aws-appsync/utils/dynamodb`:

```
// Modules are imported using @aws-appsync/utils/dynamodb
import * as ddb from '@aws-appsync/utils/dynamodb';
```

Fungsi

Daftar fungsi

`get<T>(payload: GetInput): DynamoDBGetItemRequest`

> **Tip**
>
> Lihat the section called “Masukan” untuk informasi tentang GetInput.

Menghasilkan `DynamoDBGetItemRequest` objek untuk membuat GetItempermintaan ke DynamoDB.

```
import { get } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 return get({ key: { id: ctx.args.id } });
}
```

`put<T>(payload): DynamoDBPutItemRequest`

Menghasilkan `DynamoDBPutItemRequest` objek untuk membuat PutItempermintaan ke DynamoDB.

```
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
 return ddb.put({ key: { id: util.autoId() }, item: ctx.args });
}
```

`remove<T>(payload): DynamoDBDeleteItemRequest`

Menghasilkan `DynamoDBDeleteItemRequest` objek untuk membuat DeleteItempermintaan ke DynamoDB.

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 return ddb.remove({ key: { id: ctx.args.id } });
}
```

`scan<T>(payload): DynamoDBScanRequest`

Menghasilkan permintaan `DynamoDBScanRequest` untuk membuat Scan ke DynamoDB.

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const { limit = 10, nextToken } = ctx.args;
 return ddb.scan({ limit, nextToken });
}
```

`sync<T>(payload): DynamoDBSyncRequest`

Menghasilkan `DynamoDBSyncRequest` objek untuk membuat permintaan Sinkronisasi. Permintaan hanya menerima data yang diubah sejak kueri terakhir (pembaruan delta). Permintaan hanya dapat dibuat ke sumber data DynamoDB berversi.

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const { limit = 10, nextToken, lastSync } = ctx.args;
 return ddb.sync({ limit, nextToken, lastSync });
}
```

`update<T>(payload): DynamoDBUpdateItemRequest`

Menghasilkan `DynamoDBUpdateItemRequest` objek untuk membuat UpdateItempermintaan ke DynamoDB.

Operasi

Pembantu operasi memungkinkan Anda untuk mengambil tindakan spesifik pada bagian data Anda selama pembaruan. Untuk memulai, impor `operations` dari`@aws-appsync/utils/dynamodb`:

```
// Modules are imported using operations
import {operations} from '@aws-appsync/utils/dynamodb';
```

Daftar operasi

`add<T>(payload)`

Fungsi pembantu yang menambahkan item atribut baru saat memperbarui DynamoDB.

Contoh

Untuk menambahkan alamat (jalan, kota, dan kode pos) ke item DynamoDB yang ada menggunakan nilai ID:

```
import { update, operations } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const updateObj = {
```

```
  address: operations.add({
   street1: '123 Main St',
   city: 'New York',
   zip: '10001',
  }),
 };
 return update({ key: { id: 1 }, update: updateObj });
}
```

`append <T>(payload)`

Fungsi pembantu yang menambahkan payload ke daftar yang ada di DynamoDB.

Contoh

Untuk menambahkan ID teman (`newFriendIds`) yang baru ditambahkan ke daftar teman yang ada (`friendsIds`) selama pembaruan:

```
import { update, operations } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const newFriendIds = [101, 104, 111];
 const updateObj = {
  friendsIds: operations.append(newFriendIds),
 };
 return update({ key: { id: 1 }, update: updateObj });
}
```

`decrement (by?)`

Fungsi pembantu yang mengurangi nilai atribut yang ada di item saat memperbarui DynamoDB.

Contoh

Untuk mengurangi counter teman (`friendsCount`) sebesar 10:

```
import { update, operations } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const updateObj = {
  friendsCount: operations.decrement(10),
 };
 return update({ key: { id: 1 }, update: updateObj });
```

```
}
```

`increment (by?)`

Fungsi pembantu yang menambah nilai atribut yang ada di item saat memperbarui DynamoDB.

Contoh

Untuk menambah counter teman (`friendsCount`) sebesar 10:

```
import { update, operations } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const updateObj = {
  friendsCount: operations.increment(10),
 };
 return update({ key: { id: 1 }, update: updateObj });
}
```

`prepend <T>(payload)`

Fungsi pembantu yang ditambahkan ke daftar yang ada di DynamoDB.

Contoh

Untuk menambahkan ID teman (`newFriendIds`) yang baru ditambahkan ke daftar teman yang ada (`friendsIds`) selama pembaruan:

```
import { update, operations } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const newFriendIds = [101, 104, 111];
 const updateObj = {
  friendsIds: operations.prepend(newFriendIds),
 };
 return update({ key: { id: 1 }, update: updateObj });
}
```

`replace <T>(payload)`

Fungsi pembantu yang menggantikan atribut yang ada saat memperbarui item di DynamoDB. Ini berguna ketika Anda ingin memperbarui seluruh objek atau subobject di atribut dan bukan hanya kunci di payload.

Contoh

Untuk mengganti alamat (jalan, kota, dan kode pos) dalam suatu `info` objek:

```
import { update, operations } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const updateObj = {
  info: {
   address: operations.replace({
    street1: '123 Main St',
    city: 'New York',
    zip: '10001',
   }),
  },
 };
 return update({ key: { id: 1 }, update: updateObj });
}
```

`updateListItem <T>(payload, index)`

Fungsi pembantu yang menggantikan item dalam daftar.

Contoh

Dalam lingkup update (`newFriendIds`), contoh ini digunakan `updateListItem` untuk memperbarui nilai ID dari item kedua (index:1, new ID:102) dan item ketiga (index:2, new ID:112) dalam list (`friendsIds`).

```
import { update, operations as ops } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 const newFriendIds = [
  ops.updateListItem('102', 1), ops.updateListItem('112', 2)
 ];
 const updateObj = { friendsIds: newFriendIds };
 return update({ key: { id: 1 }, update: updateObj });
}
```

Masukan

Daftar input

`Type GetInput<T>`

```
GetInput<T>: {
    consistentRead?: boolean;
    key: DynamoDBKey<T>;
}
```

Jenis Deklarasi

- `consistentRead?: boolean` (opsional)

  Boolean opsional untuk menentukan apakah Anda ingin melakukan pembacaan yang sangat konsisten dengan DynamoDB.

- `key: DynamoDBKey<T>`(Diperlukan)

  Parameter yang diperlukan yang menentukan kunci item di DynamoDB. Item DynamoDB mungkin memiliki kunci hash tunggal atau kunci hash dan sortir.

`Type PutInput<T>`

```
PutInput<T>: {
    _version?: number;
    condition?: DynamoDBFilterObject<T> | null;
    customPartitionKey?: string;
    item: Partial<T>;
    key: DynamoDBKey<T>;
    populateIndexFields?: boolean;
}
```

Jenis Deklarasi

- `_version?: number` (opsional)
- `condition?: DynamoDBFilterObject<T> | null` (opsional)

  Ketika Anda meletakkan objek dalam tabel DynamoDB, Anda dapat secara opsional menentukan ekspresi kondisional yang mengontrol apakah permintaan harus berhasil atau tidak berdasarkan status objek yang sudah ada di DynamoDB sebelum operasi dilakukan.

- `customPartitionKey?: string` (opsional)

Saat diaktifkan, nilai string ini memodifikasi format `ds_sk` dan `ds_pk` catatan yang digunakan oleh tabel sinkronisasi delta saat pembuatan versi telah diaktifkan. Saat diaktifkan, pemrosesan `populateIndexFields` entri juga diaktifkan.

- `item: Partial<T>`(Diperlukan)

  Sisa atribut item yang akan ditempatkan ke DynamoDB.

- `key: DynamoDBKey<T>`(Diperlukan)

  Parameter yang diperlukan yang menentukan kunci item di DynamoDB di mana put akan dilakukan. Item DynamoDB mungkin memiliki kunci hash tunggal atau kunci hash dan sortir.

- `populateIndexFields?: boolean` (opsional)

  Nilai boolean yang, ketika diaktifkan bersama dengan`customPartitionKey`, membuat entri baru untuk setiap catatan dalam tabel sinkronisasi delta, khususnya di kolom dan`gsi_ds_pk`. `gsi_ds_sk` Untuk informasi selengkapnya, lihat Deteksi dan sinkronisasi konflik di Panduan AWS AppSync Pengembang.

`Type QueryInput<T>`

```
QueryInput<T>: ScanInput<T> & {
    query: DynamoDBKeyCondition<Required<T>>;
}
```

Jenis Deklarasi

- `query: DynamoDBKeyCondition<Required<T>>`(Diperlukan)

  Menentukan kondisi kunci yang menjelaskan item untuk query. Untuk indeks tertentu, kondisi untuk kunci partisi harus berupa persamaan dan kunci pengurutan perbandingan atau a `beginsWith` (ketika itu adalah string). Hanya tipe angka dan string yang didukung untuk kunci partisi dan sortir.

  Contoh

  Ambil `User` tipe di bawah ini:

  ```
  type User = {
    id: string;
    name: string;
    age: number;
  ```

```
  isVerified: boolean;
  friendsIds: string[]
}
```

Kueri hanya dapat menyertakan bidang-bidang berikut:`id`,`name`, dan`age`:

```
const query: QueryInput<User> = {
    name: { eq: 'John' },
    age: { gt: 20 },
}
```

`Type RemoveInput<T>`

```
RemoveInput<T>: {
    _version?: number;
    condition?: DynamoDBFilterObject<T>;
    customPartitionKey?: string;
    key: DynamoDBKey<T>;
    populateIndexFields?: boolean;
}
```

Jenis Deklarasi

- `_version?: number` (opsional)
- `condition?: DynamoDBFilterObject<T>` (opsional)

  Saat Anda menghapus objek di DynamoDB, Anda dapat secara opsional menentukan ekspresi bersyarat yang mengontrol apakah permintaan harus berhasil atau tidak berdasarkan status objek yang sudah ada di DynamoDB sebelum operasi dilakukan.

  Contoh

  Contoh berikut adalah `DeleteItem` ekspresi yang berisi kondisi yang memungkinkan operasi berhasil hanya jika pemilik dokumen cocok dengan pengguna yang membuat permintaan.

```
type Task = {
  id: string;
  title: string;
  description: string;
  owner: string;
  isComplete: boolean;
}
```

```
const condition: DynamoDBFilterObject<Task> = {
  owner: { eq: 'XXXXXXXXXXXXXXXX' },
}

remove<Task>({
   key: {
     id: 'XXXXXXXXXXXXXXXX',
  },
  condition,
});
```

- `customPartitionKey?: string` (opsional)

  Saat diaktifkan, `customPartitionKey` nilai mengubah format `ds_sk` dan `ds_pk` catatan yang digunakan oleh tabel sinkronisasi delta saat pembuatan versi telah diaktifkan. Saat diaktifkan, pemrosesan `populateIndexFields` entri juga diaktifkan.

- `key: DynamoDBKey<T>`(Diperlukan)

  Parameter yang diperlukan yang menentukan kunci item di DynamoDB yang sedang dihapus. Item DynamoDB mungkin memiliki kunci hash tunggal atau kunci hash dan sortir.

  Contoh

  Jika `User` hanya memiliki kunci hash dengan penggunaid, maka kuncinya akan terlihat seperti ini:

  ```
  type User = {
   id: number
   name: string
   age: number
   isVerified: boolean
  }
  const key: DynamoDBKey<User> = {
   id: 1,
  }
  ```

  Jika pengguna tabel memiliki kunci hash (`id`) dan sort key (`name`), maka kuncinya akan terlihat seperti ini:

  ```
  type User = {
   id: number
  ```

```
 name: string
 age: number
 isVerified: boolean
 friendsIds: string[]
}

const key: DynamoDBKey<User> = {
 id: 1,
 name: 'XXXXXXXXXX',
}
```

- `populateIndexFields?: boolean` (opsional)

  Nilai boolean yang, ketika diaktifkan bersama dengan`customPartitionKey`, membuat entri baru untuk setiap catatan dalam tabel sinkronisasi delta, khususnya di kolom dang`si_ds_pk`. `gsi_ds_sk`

`Type ScanInput<T>`

```
ScanInput<T>: {
    consistentRead?: boolean | null;
    filter?: DynamoDBFilterObject<T> | null;
    index?: string | null;
    limit?: number | null;
    nextToken?: string | null;
    scanIndexForward?: boolean | null;
    segment?: number;
    select?: DynamoDBSelectAttributes;
    totalSegments?: number;
}
```

Jenis Deklarasi

- `consistentRead?: boolean | null` (opsional)

  Boolean opsional untuk menunjukkan pembacaan yang konsisten saat menanyakan DynamoDB. Nilai default-nya adalah `false`.

- `filter?: DynamoDBFilterObject<T> | null` (opsional)

  Filter opsional untuk diterapkan pada hasil setelah mengambilnya dari tabel.

- `index?: string | null` (opsional)

  Nama opsional indeks untuk memindai.

- `limit?: number | null` (opsional)

  Jumlah maksimal hasil opsional untuk dikembalikan.

- `nextToken?: string | null` (opsional)

  Token pagination opsional untuk melanjutkan kueri sebelumnya. Ini akan diperoleh dari kueri sebelumnya.

- `scanIndexForward?: boolean | null` (opsional)

  Boolean opsional untuk menunjukkan apakah kueri dilakukan dalam urutan naik atau turun. Secara default, nilai ini diatur ke `true`.

- `segment?: number` (opsional)
- `select?: DynamoDBSelectAttributes` (opsional)

  Atribut untuk kembali dari DynamoDB. Secara default, AWS AppSync DynamoDB resolver hanya mengembalikan atribut yang diproyeksikan ke dalam indeks. Nilai yang didukung adalah:

  - `ALL_ATTRIBUTES`

    Mengembalikan semua atribut item dari tabel tertentu atau indeks. Jika Anda menanyakan indeks sekunder lokal, DynamoDB mengambil seluruh item dari tabel induk untuk setiap item yang cocok dalam indeks. Jika indeks dikonfigurasi untuk memproyeksikan semua atribut item, semua data dapat diperoleh dari indeks sekunder lokal dan tidak diperlukan pengambilan.

  - `ALL_PROJECTED_ATTRIBUTES`

    Mengembalikan semua atribut yang telah diproyeksikan ke dalam indeks. Jika indeks dikonfigurasi untuk memproyeksikan semua atribut, nilai pengembalian ini setara dengan menentukan`ALL_ATTRIBUTES`.

  - `SPECIFIC_ATTRIBUTES`

    Mengembalikan hanya atribut yang tercantum dalam`ProjectionExpression`. Nilai pengembalian ini setara dengan menentukan `ProjectionExpression` tanpa menentukan nilai apa pun untuk. `AttributesToGet`

- `totalSegments?: number` (opsional)

`Type DynamoDBSyncInput<T>`

```
DynamoDBSyncInput<T>: {
```

```
    basePartitionKey?: string;
    deltaIndexName?: string;
    filter?: DynamoDBFilterObject<T> | null;
    lastSync?: number;
    limit?: number | null;
    nextToken?: string | null;
}
```

Jenis Deklarasi

- `basePartitionKey?: string` (opsional)

  Kunci partisi dari tabel dasar yang akan digunakan saat melakukan operasi Sync. Bidang ini memungkinkan operasi Sync dilakukan ketika tabel menggunakan kunci partisi kustom.

- `deltaIndexName?: string` (opsional)

  Indeks yang digunakan untuk operasi Sync. Indeks ini diperlukan untuk mengaktifkan operasi Sinkronisasi di seluruh tabel penyimpanan delta saat tabel menggunakan kunci partisi khusus. Operasi Sinkronisasi akan dilakukan pada GSI (dibuat pada `gsi_ds_pk` dan`gsi_ds_sk`).

- `filter?: DynamoDBFilterObject<T> | null` (opsional)

  Filter opsional untuk diterapkan pada hasil setelah mengambilnya dari tabel.

- `lastSync?: number` (opsional)

  Saat itu, dalam milidetik epoch, di mana operasi Sinkronisasi terakhir yang berhasil dimulai. Jika ditentukan, hanya item yang telah berubah setelah `lastSync` dikembalikan. Bidang ini seharusnya hanya diisi setelah mengambil semua halaman dari operasi Sinkronisasi awal. Jika dihilangkan, hasil dari tabel dasar akan dikembalikan. Jika tidak, hasil dari tabel delta akan dikembalikan.

- `limit?: number | null` (opsional)

  Jumlah maksimum opsional item untuk dievaluasi pada satu waktu. Jika dihilangkan, batas default akan diatur ke `100` item. Nilai maksimum untuk bidang ini adalah `1000` item.

- `nextToken?: string | null` (opsional)

`Type DynamoDBUpdateInput<T>`

```
DynamoDBUpdateInput<T>: {
    _version?: number;
    condition?: DynamoDBFilterObject<T>;
```

```
    customPartitionKey?: string;
    key: DynamoDBKey<T>;
    populateIndexFields?: boolean;
    update: DynamoDBUpdateObject<T>;
}
```

Jenis Deklarasi

- `_version?: number` (opsional)
- `condition?: DynamoDBFilterObject<T>` (opsional)

  Saat Anda memperbarui objek di DynamoDB, Anda dapat secara opsional menentukan ekspresi bersyarat yang mengontrol apakah permintaan harus berhasil atau tidak berdasarkan status objek yang sudah ada di DynamoDB sebelum operasi dilakukan.

- `customPartitionKey?: string` (opsional)

  Saat diaktifkan, `customPartitionKey` nilai mengubah format `ds_sk` dan `ds_pk` catatan yang digunakan oleh tabel sinkronisasi delta saat pembuatan versi telah diaktifkan. Saat diaktifkan, pemrosesan `populateIndexFields` entri juga diaktifkan.

- `key: DynamoDBKey<T>`(Diperlukan)

  Parameter yang diperlukan yang menentukan kunci item di DynamoDB yang sedang diperbarui. Item DynamoDB mungkin memiliki kunci hash tunggal atau kunci hash dan sortir.

- `populateIndexFields?: boolean` (opsional)

  Nilai boolean yang, ketika diaktifkan bersama dengan`customPartitionKey`, membuat entri baru untuk setiap catatan dalam tabel sinkronisasi delta, khususnya di kolom dan`gsi_ds_pk`. `gsi_ds_sk`

- `update: DynamoDBUpdateObject<T>`

  Objek yang menentukan atribut yang akan diperbarui bersama dengan nilai-nilai baru untuk mereka. Objek pembaruan dapat digunakan dengan`add`,`remove`,`replace`,`increment`,`decrement`,`append`,`prepend`,`updateListItem`.

Fungsi modul Amazon RDS

Fungsi modul Amazon RDS memberikan pengalaman yang disempurnakan saat berinteraksi dengan database yang dikonfigurasi dengan Amazon RDS Data API. Modul diimpor menggunakan`@aws-appsync/utils/rds`:

```
import * as rds from '@aws-appsync/utils/rds';
```

Fungsi juga dapat diimpor secara individual. Misalnya, impor di bawah ini menggunakan`sql`:

```
import { sql } from '@aws-appsync/utils/rds';
```

Fungsi

Anda dapat menggunakan pembantu utilitas modul AWS AppSync RDS untuk berinteraksi dengan database Anda.

Pilih

`select`Utilitas membuat `SELECT` pernyataan untuk menanyakan database relasional Anda.

Penggunaan dasar

Dalam bentuk dasarnya, Anda dapat menentukan tabel yang ingin Anda kueri:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

    // Generates statement:
    // "SELECT * FROM "persons"
    return createPgStatement(select({table: 'persons'}));
}
```

Perhatikan bahwa Anda juga dapat menentukan skema dalam pengidentifikasi tabel Anda:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

    // Generates statement:
    // SELECT * FROM "private"."persons"
    return createPgStatement(select({table: 'private.persons'}));
}
```

Menentukan kolom

Anda dapat menentukan kolom dengan `columns` properti. Jika ini tidak disetel ke nilai, defaultnya adalah: *

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name']
    }));
}
```

Anda dapat menentukan tabel kolom juga:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "persons"."name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'persons.name']
    }));
}
```

Batas dan offset

Anda dapat menerapkan `limit` dan `offset` ke kueri:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    // LIMIT :limit
    // OFFSET :offset
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        limit: 10,
        offset: 40
```

```
    }));
}
```

Memesan oleh

Anda dapat mengurutkan hasil Anda dengan `orderBy` properti. Menyediakan array objek yang menentukan kolom dan `dir` properti opsional:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name" FROM "persons"
    // ORDER BY "name", "id" DESC
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        orderBy: [{column: 'name'}, {column: 'id', dir: 'DESC'}]
    }));
}
```

Filter

Anda dapat membangun filter dengan menggunakan objek kondisi khusus:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    // WHERE "name" = :NAME
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        where: {name: {eq: 'Stephane'}}
    }));
}
```

Anda juga dapat menggabungkan filter:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
```

```
    // FROM "persons"
    // WHERE "name" = :NAME and "id" > :ID
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        where: {name: {eq: 'Stephane'}, id: {gt: 10}}
    }));
}
```

Anda juga dapat membuat 0R pernyataan:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    // WHERE "name" = :NAME OR "id" > :ID
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        where: { or: [
            { name: { eq: 'Stephane'} },
            { id: { gt: 10 } }
        ]}
    }));
}
```

Anda juga dapat meniadakan suatu kondisi dengan`not`:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    // WHERE NOT ("name" = :NAME AND "id" > :ID)
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        where: { not: [
            { name: { eq: 'Stephane'} },
            { id: { gt: 10 } }
        ]}
    }));
```

```
}
```

Anda juga dapat menggunakan operator berikut untuk membandingkan nilai:

Operasi	Deskripsi	Jenis nilai yang mungkin
eq	Equal	number, string, boolean
ne	Not equal	number, string, boolean
le	Less than or equal	number, string
lt	Less than	number, string
ge	Greater than or equal	number, string
gt	Greater than	number, string
contains	Like	string
notContains	Not like	string
beginsWith	Starts with prefix	string
between	Between two values	number, string
attributeExists	The attribute is not null	number, string, boolean
size	checks the length of the element	string

Sisipkan

`insert`Utilitas menyediakan cara mudah untuk memasukkan item baris tunggal dalam database Anda dengan operasi. `INSERT`

Penyisipan item tunggal

Untuk menyisipkan item, tentukan tabel dan kemudian masukkan objek nilai Anda. Kunci objek dipetakan ke kolom tabel Anda. Nama kolom secara otomatis lolos, dan nilai dikirim ke database menggunakan peta variabel:

```
import { insert, createMySQLStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const { input: values } = ctx.args;
    const insertStatement = insert({ table: 'persons', values });

    // Generates statement:
    // INSERT INTO `persons`(`name`)
    // VALUES(:NAME)
    return createMySQLStatement(insertStatement)
}
```

Kasus penggunaan MySQL

Anda dapat menggabungkan `insert` diikuti oleh a `select` untuk mengambil baris yang disisipkan:

```
import { insert, select, createMySQLStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const { input: values } = ctx.args;
    const insertStatement = insert({  table: 'persons', values });
    const selectStatement = select({
        table: 'persons',
        columns: '*',
        where: { id: { eq: values.id } },
        limit: 1,
    });

    // Generates statement:
    // INSERT INTO `persons`(`name`)
    // VALUES(:NAME)
    // and
    // SELECT *
    // FROM `persons`
    // WHERE `id` = :ID
    return createMySQLStatement(insertStatement, selectStatement)
}
```

Kasus penggunaan postgres

Dengan Postgres, Anda dapat menggunakan `returning`untuk mendapatkan data dari baris yang Anda masukkan. Ia menerima * atau array nama kolom:

```
import { insert, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const { input: values } = ctx.args;
    const insertStatement = insert({
        table: 'persons',
        values,
        returning: '*'
    });

    // Generates statement:
    // INSERT INTO "persons"("name")
    // VALUES(:NAME)
    // RETURNING *
    return createPgStatement(insertStatement)
}
```

Pembaruan

`updateUtilitas` memungkinkan Anda untuk memperbarui baris yang ada. Anda dapat menggunakan objek kondisi untuk menerapkan perubahan pada kolom yang ditentukan di semua baris yang memenuhi kondisi. Sebagai contoh, katakanlah kita memiliki skema yang memungkinkan kita membuat mutasi ini. Kami ingin memperbarui `name` of `Person` dengan `id` nilai `3` tetapi hanya jika kami mengetahuinya (`known_since`) sejak tahun ini`2000`:

```
mutation Update {
    updatePerson(
        input: {id: 3, name: "Jon"},
        condition: {known_since: {ge: "2000"}}
    ) {
    id
    name
  }
}
```

Penyelesai pembaruan kami terlihat seperti ini:

```
import { update, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const { input: { id, ...values }, condition } = ctx.args;
    const where = {
```

```
        ...condition,
        id: { eq: id },
    };
    const updateStatement = update({
        table: 'persons',
        values,
        where,
        returning: ['id', 'name'],
    });

    // Generates statement:
    // UPDATE "persons"
    // SET "name" = :NAME, "birthday" = :BDAY, "country" = :COUNTRY
    // WHERE "id" = :ID
    // RETURNING "id", "name"
    return createPgStatement(updateStatement)
}
```

Kita dapat menambahkan cek ke kondisi kita untuk memastikan bahwa hanya baris yang memiliki kunci utama `id` sama dengan 3 yang diperbarui. Demikian pula, untuk Postgres`inserts`, Anda dapat menggunakan `returning` untuk mengembalikan data yang dimodifikasi.

Menghapus

`remove`Utilitas memungkinkan Anda untuk menghapus baris yang ada. Anda dapat menggunakan objek kondisi pada semua baris yang memenuhi kondisi. Perhatikan bahwa itu `delete` adalah kata kunci yang dicadangkan di JavaScript. `remove`harus digunakan sebagai gantinya:

```
import { remove, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const { input: { id }, condition } = ctx.args;
    const where = { ...condition, id: { eq: id } };
    const deleteStatement = remove({
        table: 'persons',
        where,
        returning: ['id', 'name'],
    });

    // Generates statement:
    // DELETE "persons"
    // WHERE "id" = :ID
    // RETURNING "id", "name"
```

```
    return createPgStatement(updateStatement)
}
```

Casting

Dalam beberapa kasus, Anda mungkin ingin lebih spesifisitas tentang jenis objek yang benar untuk digunakan dalam pernyataan Anda. Anda dapat menggunakan petunjuk jenis yang disediakan untuk menentukan jenis parameter Anda. AWS AppSyncmendukung petunjuk jenis yang sama dengan Data API. Anda dapat mentransmisikan parameter Anda dengan menggunakan `typeHint` fungsi dari AWS AppSync `rds` modul.

Contoh berikut memungkinkan Anda untuk mengirim array sebagai nilai yang dicor sebagai objek JSON. Kami menggunakan `->` operator untuk mengambil elemen di `index` 2 dalam array JSON:

```
import { sql, createPgStatement, toJsonObject, typeHint } from '@aws-appsync/utils/
rds';

export function request(ctx) {
    const arr = ctx.args.list_of_ids
    const statement = sql`select ${typeHint.JSON(arr)}->2 as value`
    return createPgStatement(statement)
}

export function response(ctx) {
    return toJsonObject(ctx.result)[0][0].value
}
```

Casting juga berguna saat menangani dan membandingkan`DATE`,`TIME`, dan`TIMESTAMP`:

```
import { select, createPgStatement, typeHint } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const when = ctx.args.when
    const statement = select({
        table: 'persons',
        where: { createdAt : { gt: typeHint.DATETIME(when) } }
    })
    return createPgStatement(statement)
}
```

Berikut contoh lain yang menunjukkan bagaimana Anda dapat mengirim tanggal dan waktu saat ini:

```
import { sql, createPgStatement, typeHint } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const now = util.time.nowFormatted('YYYY-MM-dd HH:mm:ss')
    return createPgStatement(sql`select ${typeHint.TIMESTAMP(now)}`)
}
```

Petunjuk tipe yang tersedia

- `typeHint.DATE`- Parameter yang sesuai dikirim sebagai objek dari `DATE` jenis ke database. Format yang diterima adalah `YYYY-MM-DD`.
- `typeHint.DECIMAL`- Parameter yang sesuai dikirim sebagai objek dari `DECIMAL` jenis ke database.
- `typeHint.JSON`- Parameter yang sesuai dikirim sebagai objek dari `JSON` jenis ke database.
- `typeHint.TIME`- Nilai parameter string yang sesuai dikirim sebagai objek dari `TIME` tipe ke database. Format yang diterima adalah `HH:MM:SS[.FFF]`.
- `typeHint.TIMESTAMP`- Nilai parameter string yang sesuai dikirim sebagai objek dari `TIMESTAMP` tipe ke database. Format yang diterima adalah `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `typeHint.UUID`- Nilai parameter string yang sesuai dikirim sebagai objek dari `UUID` tipe ke database.

Utilitas runtime

`runtime`Pustaka menyediakan utilitas untuk mengontrol atau memodifikasi properti runtime dari resolver dan fungsi Anda.

Daftar utilitas runtime

`runtime.earlyReturn(obj?: unknown): never`

Memanggil fungsi ini akan menghentikan eksekusi AWS AppSync fungsi atau resolver saat ini (Unit atau Pipeline Resolver) tergantung pada konteks saat ini. Objek yang ditentukan dikembalikan sebagai hasilnya.

- Saat dipanggil dalam penangan permintaan AWS AppSync fungsi, sumber data dan penangan respons dilewati, dan penangan permintaan fungsi berikutnya (atau pengendali respons penyelesai pipa jika ini adalah fungsi terakhir) dipanggil. AWS AppSync

- Ketika dipanggil dalam penangan permintaan penyelesai AWS AppSync pipa, eksekusi pipeline dilewati, dan pengendali respons penyelesai pipa segera dipanggil.

Contoh

```
import { runtime } from '@aws-appsync/utils'

export function request(ctx) {
  runtime.earlyReturn({ hello: 'world' })
  // code below is not executed
  return ctx.args
}

// never called because request returned early
export function response(ctx) {
  return ctx.result
}
```

Pembantu waktu di util.time

`util.time`Variabel berisi metode datetime untuk membantu menghasilkan stempel waktu, mengonversi antara format datetime, dan mengurai string datetime. Sintaks untuk format datetime didasarkan pada DateTimeFormattermana Anda dapat referensi untuk dokumentasi lebih lanjut. Kami memberikan beberapa contoh di bawah ini, serta daftar metode dan deskripsi yang tersedia.

Penggunaan waktu

Daftar utilitas waktu

`util.time.nowISO8601()`

Mengembalikan representasi String dari UTC dalam format ISO8601.

`util.time.nowEpochSeconds()`

Mengembalikan jumlah detik dari epoch 1970-01-01T 00:00:00 Z ke sekarang.

`util.time.nowEpochMilliSeconds()`

Mengembalikan jumlah milidetik dari epoch 1970-01-01T 00:00:00 Z ke sekarang.

`util.time.nowFormatted(String)`

Mengembalikan string timestamp saat ini di UTC menggunakan format yang ditentukan dari tipe input String.

`util.time.nowFormatted(String, String)`

Mengembalikan string timestamp saat ini untuk zona waktu menggunakan format yang ditentukan dan zona waktu dari jenis input String.

`util.time.parseFormattedToEpochMilliSeconds(String, String)`

Mem-parsing stempel waktu yang diteruskan sebagai String bersama dengan format, lalu mengembalikan stempel waktu sebagai milidetik sejak epoch.

`util.time.parseFormattedToEpochMilliSeconds(String, String, String)`

Mem-parsing stempel waktu yang diteruskan sebagai String bersama dengan format dan zona waktu, lalu mengembalikan stempel waktu sebagai milidetik sejak epoch.

`util.time.parseISO8601ToEpochMilliSeconds(String)`

Mem-parsing stempel waktu ISO8601 yang diteruskan sebagai String, lalu mengembalikan stempel waktu sebagai milidetik sejak epoch.

`util.time.epochMilliSecondsToSeconds(long)`

Mengonversi stempel waktu milidetik epoch menjadi stempel waktu epoch detik.

`util.time.epochMilliSecondsToISO8601(long)`

Mengonversi stempel waktu milidetik epoch menjadi stempel waktu ISO8601.

`util.time.epochMilliSecondsToFormatted(long, String)`

Mengonversi stempel waktu milidetik epoch, diteruskan selama, ke stempel waktu yang diformat sesuai dengan format yang disediakan di UTC.

`util.time.epochMilliSecondsToFormatted(long, String, String)`

Mengonversi stempel waktu milidetik epoch, diteruskan sebagai panjang, ke stempel waktu yang diformat sesuai dengan format yang disediakan di zona waktu yang disediakan.

Pembantu DynamoDB di util.dynamodb

`util.dynamodb`berisi metode pembantu yang memudahkan untuk menulis dan membaca data ke Amazon DynamoDB, seperti pemetaan dan pemformatan tipe otomatis.

ToDynamoDB

Daftar utilitas ToDynamoDB

`util.dynamodb.toDynamoDB(Object)`

Alat konversi objek umum untuk DynamoDB yang mengubah objek masukan ke representasi DynamoDB yang sesuai. Ini berpendirian tentang bagaimana itu mewakili beberapa jenis: misalnya, itu akan menggunakan daftar (“L”) daripada set (“SS”, “NS”, “BS”). Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

Contoh string

```
Input:      util.dynamodb.toDynamoDB("foo")
Output:     { "S" : "foo" }
```

Contoh angka

```
Input:      util.dynamodb.toDynamoDB(12345)
Output:     { "N" : 12345 }
```

Contoh Boolean

```
Input:      util.dynamodb.toDynamoDB(true)
Output:     { "BOOL" : true }
```

Daftar contoh

```
Input:      util.dynamodb.toDynamoDB([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

Contoh peta

```
Input:      util.dynamodb.toDynamoDB({ "foo": "bar", "baz" : 1234, "beep":
 [ "boop"] })
Output:     {
               "M" : {
                   "foo"  : { "S" : "bar" },
                   "baz"  : { "N" : 1234 },
                   "beep" : {
                       "L" : [
                           { "S" : "boop" }
                       ]
                   }
               }
           }
```

ToString utilitas

Daftar utilitas ToString

`util.dynamodb.toString(String)`

Mengkonversi string input ke format string DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toString("foo")
Output:     { "S" : "foo" }
```

`util.dynamodb.toStringSet(List<String>)`

Mengkonversi daftar dengan Strings ke format set string DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toStringSet([ "foo", "bar", "baz" ])
Output:     { "SS" : [ "foo", "bar", "baz" ] }
```

utilitas Tonumber

Daftar utilitas Tonumber

`util.dynamodb.toNumber(Number)`

Mengkonversi angka ke format nomor DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toNumber(12345)
Output:     { "N" : 12345 }
```

`util.dynamodb.toNumberSet(List<Number>)`

Mengkonversi daftar angka ke format set nomor DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toNumberSet([ 1, 23, 4.56 ])
Output:     { "NS" : [ 1, 23, 4.56 ] }
```

Kegunaan toBinary

Daftar utilitas toBinary

`util.dynamodb.toBinary(String)`

Mengkonversi data biner dikodekan sebagai string base64 ke format biner DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toBinary("foo")
Output:     { "B" : "foo" }
```

`util.dynamodb.toBinarySet(List<String>)`

Mengkonversi daftar data biner yang dikodekan sebagai string base64 ke format set biner DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toBinarySet([ "foo", "bar", "baz" ])
Output:     { "BS" : [ "foo", "bar", "baz" ] }
```

utilitas ToBoolean

Daftar utilitas ToBoolean

`util.dynamodb.toBoolean(Boolean)`

Mengkonversi Boolean ke format DynamoDB Boolean yang sesuai. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toBoolean(true)
Output:     { "BOOL" : true }
```

utilitas ToNull

Daftar utilitas ToNull

`util.dynamodb.toNull()`

Mengembalikan null dalam format DynamoDB null. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toNull()
Output:     { "NULL" : null }
```

utilitas ToList

Daftar utilitas ToList

`util.dynamodb.toList(List)`

Mengkonversi daftar objek ke format daftar DynamoDB. Setiap item dalam daftar juga dikonversi ke format DynamoDB yang sesuai. Ini berpendapat tentang bagaimana itu mewakili beberapa objek bersarang: misalnya, itu akan menggunakan daftar (“L”) daripada set (“SS”, “NS”, “BS”). Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toList([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
```

```
                { "N" : 123 },
                {
                    "M" : {
                        "bar" : { "S" : "baz" }
                    }
                }
            ]
        }
```

TomAp utilitas

Daftar utilitas TomAP

`util.dynamodb.toMap(Map)`

Mengkonversi peta ke format peta DynamoDB. Setiap nilai dalam peta juga dikonversi ke format DynamoDB yang sesuai. Ini berpendapat tentang bagaimana itu mewakili beberapa objek bersarang: misalnya, itu akan menggunakan daftar (“L”) daripada set (“SS”, “NS”, “BS”). Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toMap({ "foo": "bar", "baz" : 1234, "beep": [ "boop"] })
Output:     {
               "M" : {
                   "foo"  : { "S" : "bar" },
                   "baz"  : { "N" : 1234 },
                   "beep" : {
                       "L" : [
                           { "S" : "boop" }
                       ]
                   }
               }
           }
```

`util.dynamodb.toMapValues(Map)`

Membuat salinan peta di mana setiap nilai telah dikonversi ke format DynamoDB yang sesuai. Ini berpendapat tentang bagaimana itu mewakili beberapa objek bersarang: misalnya, itu akan menggunakan daftar (“L”) daripada set (“SS”, “NS”, “BS”).

```
Input:      util.dynamodb.toMapValues({ "foo": "bar", "baz" : 1234, "beep":
 [ "boop"] })
```

```
Output:     {
               "foo"  : { "S" : "bar" },
               "baz"  : { "N" : 1234 },
               "beep" : {
                   "L" : [
                       { "S" : "boop" }
                   ]
               }
           }
```

> **Note**
>
> Ini sedikit berbeda `util.dynamodb.toMap(Map)` karena hanya mengembalikan isi dari nilai atribut DynamoDB, tetapi tidak seluruh nilai atribut itu sendiri. Misalnya, pernyataan berikut persis sama:
>
> ```
> util.dynamodb.toMapValues(<map>)
> util.dynamodb.toMap(<map>)("M")
> ```

S3Object utilitas

Daftar utilitas S3Object

`util.dynamodb.toS3Object(String key, String bucket, String region)`

Mengonversi kunci, bucket, dan wilayah menjadi representasi DynamoDB S3 Object. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toS3Object("foo", "bar", region = "baz")
Output:     { "S" : "{ \"s3\" : { \"key\" : \"foo", \"bucket\" : \"bar", \"region
\" : \"baz" } }" }
```

`util.dynamodb.toS3Object(String key, String bucket, String region, String version)`

Mengonversi kunci, bucket, region, dan versi opsional menjadi representasi DynamoDB S3 Object. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      util.dynamodb.toS3Object("foo", "bar", "baz", "beep")
```

```
Output:     { "S" : "{ \"s3\" : { \"key\" : \"foo\", \"bucket\" : \"bar\", \"region
\" : \"baz\", \"version\" = \"beep\" } }" }
```

`util.dynamodb.fromS3ObjectJson(String)`

Menerima nilai string Objek DynamoDB S3 dan mengembalikan peta yang berisi kunci, bucket, wilayah, dan versi opsional.

```
Input:      util.dynamodb.fromS3ObjectJson({ "S" : "{ \"s3\" : { \"key\" : \"foo\",
 \"bucket\" : \"bar\", \"region\" : \"baz\", \"version\" = \"beep\" } }" })
Output:     { "key" : "foo", "bucket" : "bar", "region" : "baz", "version" :
 "beep" }
```

Pembantu HTTP di util.http

`util.http`Utilitas menyediakan metode pembantu yang dapat Anda gunakan untuk mengelola parameter permintaan HTTP dan untuk menambahkan header respons.

daftar util.http utils

`util.http.copyHeaders(headers)`

Menyalin header dari peta tanpa set header HTTP yang dibatasi. Anda dapat menggunakan ini untuk meneruskan header permintaan ke titik akhir HTTP hilir Anda.

`util.http.addResponseHeader(String, Object)`

Menambahkan header kustom tunggal dengan nama (`String`) dan nilai (`Object`) dari respon. Limitasi berikut berlaku pada:

- Nama header tidak dapat cocok dengan header atau AWS AppSync header yang ada AWS atau dibatasi.
- Nama header tidak dapat dimulai dengan awalan terbatas, seperti `x-amzn-` atau. `x-amz-`
- Ukuran header respons khusus tidak boleh melebihi 4 KB. Ini termasuk nama dan nilai header.
- Anda harus menentukan setiap header respons sekali per operasi GraphQL. Namun, jika Anda menentukan header kustom dengan nama yang sama beberapa kali, definisi terbaru akan muncul dalam respons. Semua header dihitung terhadap batas ukuran header terlepas dari penamaan.

`util.http.addResponseHeaders(Map)`

Menambahkan beberapa header respons ke respons dari peta nama (`String`) dan nilai (`Object`) yang ditentukan. Keterbatasan yang sama yang tercantum untuk `addResponseHeader(String, Object)` metode ini juga berlaku untuk metode ini.

Pembantu transformasi di util.transform

`util.transform`berisi metode pembantu yang membuatnya lebih mudah untuk melakukan operasi kompleks terhadap sumber data.

Daftar utilitas pembantu transformasi

`util.transform.toDynamoDBFilterExpression(filterObject: DynamoDBFilterObject) : string`

Mengkonversi string input ke ekspresi filter untuk digunakan dengan DynamoDB. Kami merekomendasikan penggunaan `toDynamoDBFilterExpression` dengan fungsi modul bawaan.

`util.transform.toElasticsearchQueryDSL(object: OpenSearchQueryObject) : string`

Mengkonversi input yang diberikan ke ekspresi OpenSearch Query DSL yang setara, mengembalikannya sebagai string JSON.

Contoh masukan:

```
util.transform.toElasticsearchQueryDSL({
    "upvotes":{
        "ne":15,
        "range":[
            10,
            20
        ]
    },
    "title":{
        "eq":"hihihi",
        "wildcard":"h*i"
    }
  })
```

Contoh keluaran:

```
{
    "bool":{
      "must":[
          {
            "bool":{
              "must":[
                  {
                    "bool":{
                      "must_not":{
                        "term":{
                          "upvotes":15
                        }
                      }
                    }
                  },
                  {
                    "range":{
                      "upvotes":{
                        "gte":10,
                        "lte":20
                      }
                    }
                  }
              ]
            }
          },
          {
            "bool":{
              "must":[
                  {
                    "term":{
                      "title":"hihihi"
                    }
                  },
                  {
                  "wildcard":{
                      "title":"h*i"
                    }
                  }
              ]
            }
          }
```

```
      ]
    }
}
```

> **Note**
>
> Operator default diasumsikan AND.

`util.transform.toSubscriptionFilter(objFilter, ignoredFields?, rules?): SubscriptionFilter`

Mengkonversi objek `Map` masukan ke objek `SubscriptionFilter` ekspresi. `util.transform.toSubscriptionFilter`Metode ini digunakan sebagai masukan ke `extensions.setSubscriptionFilter()` ekstensi. Untuk informasi selengkapnya, lihat Ekstensi.

> **Note**
>
> Parameter dan pernyataan pengembalian tercantum di bawah ini:
> Parameter
>
> - `objFilter`: `SubscriptionFilterObject`
>
>   Objek `Map` masukan yang dikonversi ke objek `SubscriptionFilter` ekspresi.
> - `ignoredFields`: `SubscriptionFilterExcludeKeysType` (opsional)
>
>   Sebuah `List` nama bidang di objek pertama yang akan diabaikan.
> - `rules`: `SubscriptionFilterRuleObject` (opsional)
>
>   Objek `Map` masukan dengan aturan ketat yang disertakan saat Anda membangun objek `SubscriptionFilter` ekspresi. Aturan ketat ini akan dimasukkan dalam objek `SubscriptionFilter` ekspresi sehingga setidaknya salah satu aturan akan dipenuhi untuk melewati filter berlangganan.
>
> Respons
> Mengembalikan `SubscriptionFilter`.

`util.transform.toSubscriptionFilter(Map, List)`

Mengkonversi objek `Map` masukan ke objek `SubscriptionFilter` ekspresi. `util.transform.toSubscriptionFilter`Metode ini digunakan sebagai masukan ke `extensions.setSubscriptionFilter()` ekstensi. Untuk informasi selengkapnya, lihat Ekstensi.

Argumen pertama adalah objek `Map` masukan yang dikonversi ke objek `SubscriptionFilter` ekspresi. Argumen kedua adalah nama `List` bidang yang diabaikan dalam objek `Map` masukan pertama saat membangun objek `SubscriptionFilter` ekspresi.

`util.transform.toSubscriptionFilter(Map, List, Map)`

Mengkonversi objek `Map` masukan ke objek `SubscriptionFilter` ekspresi. `util.transform.toSubscriptionFilter`Metode ini digunakan sebagai masukan ke `extensions.setSubscriptionFilter()` ekstensi. Untuk informasi selengkapnya, lihat Ekstensi.

`util.transform.toDynamoDBConditionExpression(conditionObject)`

Menciptakan ekspresi kondisi DynamoDB.

Argumen filter langganan

Tabel berikut menjelaskan bagaimana argumen dari utilitas berikut didefinisikan:

- `Util.transform.toSubscriptionFilter(objFilter, ignoredFields?, rules?): SubscriptionFilter`

Argument 1: Map

Argumen 1 adalah `Map` objek dengan nilai-nilai kunci berikut:

- nama bidang
- “dan”
- “atau”

Untuk nama bidang sebagai kunci, kondisi pada entri bidang ini adalah dalam bentuk. `"operator" : "value"`

Contoh berikut menunjukkan bagaimana entri dapat ditambahkan ke`Map`:

```
"field_name" : {
                    "operator1" : value
               }

## We can have multiple conditions for the same field_name:

"field_name" : {
                    "operator1" : value
                    "operator2" : value
                    .
                    .
                    .
               }
```

Ketika sebuah bidang memiliki dua atau lebih kondisi di atasnya, semua kondisi ini dianggap menggunakan operasi OR.

Input juga `Map` dapat memiliki “dan” dan “atau” sebagai kunci, menyiratkan bahwa semua entri di dalamnya harus digabungkan menggunakan logika AND atau OR tergantung pada kuncinya. Nilai kunci “dan” dan “atau” mengharapkan berbagai kondisi.

```
"and" : [

            {
                "field_name1" : {
                    "operator1" : value
                }
             },

             {
                "field_name2" : {
                    "operator1" : value
                }
             },
             .
             .
        ].
```

Perhatikan bahwa Anda dapat bersarang “dan” dan “atau”. Artinya, Anda dapat memiliki sarang “dan” /"atau” di dalam blok “dan” /"atau” lainnya. Namun, ini tidak berfungsi untuk bidang sederhana.

```
"and" : [

            {
                "field_name1" : {
                    "operator" : value
                }
             },

             {
                "or" : [
                            {
                                "field_name2" : {
                                    "operator" : value
                                }
                            },

                            {
                                "field_name3" : {
                                    "operator" : value
                                }
                            }

                        ].
```

Contoh berikut menunjukkan masukan argumen 1 menggunakan`util.transform.toSubscriptionFilter(Map) : Map`.

Masukan

Argumen 1: Peta:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
```

```
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 2000
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Keluaran

Hasilnya adalah `Map` objek:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "percentageUp",
          "operator": "lte",
          "value": 50
        },
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
```

```
          "value": 2000
        },
        {
          "fieldName": "author",
          "operator": "eq",
          "value": "Admin"
        }
      ]
    },
    {
      "filters": [
        {
          "fieldName": "percentageUp",
          "operator": "lte",
          "value": 50
        },
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 2000
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    },
    {
      "filters": [
        {
          "fieldName": "percentageUp",
          "operator": "gte",
          "value": 20
        },
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
```

```
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 2000
        },
        {
          "fieldName": "author",
          "operator": "eq",
          "value": "Admin"
        }
      ]
    },
    {
      "filters": [
        {
          "fieldName": "percentageUp",
          "operator": "gte",
          "value": 20
        },
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 2000
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    }
  ]
}
```

Argument 2: List

Argumen 2 berisi nama `List` bidang yang tidak boleh dipertimbangkan dalam input `Map` (argumen 1) saat membangun objek `SubscriptionFilter` ekspresi. `List`Bisa juga kosong.

Contoh berikut menunjukkan masukan argumen 1 dan argumen 2 menggunakan`util.transform.toSubscriptionFilter(Map, List) : Map`.

Masukan

Argumen 1: Peta:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Argumen 2: Daftar:

```
["percentageUp", "author"]
```

Keluaran

Hasilnya adalah `Map` objek:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    }
  ]
}
```

Argument 3: Map

Argumen 3 adalah `Map` objek yang memiliki nama bidang sebagai nilai kunci (tidak dapat memiliki “dan” atau “atau”). Untuk nama bidang sebagai kunci, kondisi pada bidang ini adalah entri dalam bentuk. `"operator" : "value"` Tidak seperti argumen 1, argumen 3 tidak dapat memiliki beberapa kondisi dalam kunci yang sama. Selain itu, argumen 3 tidak memiliki klausa “dan” atau “atau”, jadi tidak ada sarang yang terlibat juga.

Argumen 3 merupakan daftar aturan ketat, yang ditambahkan ke objek `SubscriptionFilter` ekspresi sehingga setidaknya satu dari kondisi ini terpenuhi untuk melewati filter.

```
{
  "fieldname1": {
    "operator": value
  },
  "fieldname2": {
```

```
    "operator": value
  }
}
.
.
.
```

Contoh berikut menunjukkan masukan dari argumen 1, argumen 2, dan argumen 3 menggunakan`util.transform.toSubscriptionFilter(Map, List, Map) : Map`.

Masukan

Argumen 1: Peta:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "lt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
```

```
}
```

Argumen 2: Daftar:

```
["percentageUp", "author"]
```

Argumen 3: Peta:

```
{
  "upvotes": {
    "gte": 250
  },
  "author": {
    "eq": "Person1"
  }
}
```

Keluaran

Hasilnya adalah `Map` objek:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        },
        {
          "fieldName": "upvotes",
```

```
          "operator": "gte",
          "value": 250
        }
      ]
    },
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        },
        {
          "fieldName": "author",
          "operator": "eq",
          "value": "Person1"
        }
      ]
    }
  ]
}
```

Pembantu string di util.str

`util.str`berisi metode untuk membantu operasi String umum.

util.str daftar util.str

`util.str.normalize(String, String)`

Menormalkan string menggunakan salah satu dari empat bentuk normalisasi unicode: NFC, NFD, NFKC, atau NFKD. Argumen pertama adalah string untuk menormalkan. Argumen kedua adalah

"nfc", "nfd", "nfkc", atau "nfkd" yang menentukan jenis normalisasi yang akan digunakan untuk proses normalisasi.

Ekstensi

`extensions`berisi serangkaian metode untuk membuat tindakan tambahan dalam resolver Anda.

Ekstensi caching

`extensions.evictFromApiCache(typeName: string, fieldName: string, keyValuePair: Record<string, string>) : Object`

Mengusir item dari cache sisi AWS AppSync server. Argumen pertama adalah nama tipe. Argumen kedua adalah nama bidang. Argumen ketiga adalah objek yang berisi item pasangan kunci-nilai yang menentukan nilai kunci caching. Anda harus meletakkan item dalam objek dalam urutan yang sama dengan kunci caching di resolver cache. `cachingKey` Untuk informasi selengkapnya tentang caching, lihat Perilaku cache.

Contoh 1:

Contoh ini mengusir item yang di-cache untuk resolver yang dipanggil `Query.allClasses` di mana kunci caching dipanggil digunakan. `context.arguments.semester` Ketika mutasi dipanggil dan resolver berjalan, jika entri berhasil dihapus, maka respons berisi `apiCacheEntriesDeleted` nilai dalam objek ekstensi yang menunjukkan berapa banyak entri yang dihapus.

```
import { util, extensions } from '@aws-appsync/utils';

export const request = (ctx) => ({ payload: null });

export function response(ctx) {
 extensions.evictFromApiCache('Query', 'allClasses', {
  'context.arguments.semester': ctx.args.semester,
 });
 return null;
}
```

> **Note**
>
> Fungsi ini hanya berfungsi untuk mutasi, bukan kueri.

Ekstensi berlangganan

`extensions.setSubscriptionFilter(filterJsonObject)`

Mendefinisikan filter langganan yang disempurnakan. Setiap acara pemberitahuan berlangganan dievaluasi terhadap filter langganan yang disediakan dan mengirimkan pemberitahuan kepada klien jika semua filter mengevaluasi. `true` Argumennya adalah `filterJsonObject` (Informasi lebih lanjut tentang argumen ini dapat ditemukan di bawah di filterJsonObject bagian Argumen:.). Lihat Pemfilteran langganan yang disempurnakan.

> **Note**
>
> Anda dapat menggunakan fungsi ekstensi ini hanya di handler respons dari resolver langganan. Juga, kami sarankan menggunakan `util.transform.toSubscriptionFilter` untuk membuat filter Anda.

`extensions.setSubscriptionInvalidationFilter(filterJsonObject)`

Mendefinisikan filter pembatalan langganan. Filter langganan dievaluasi terhadap muatan pembatalan, lalu membatalkan langganan yang diberikan jika filter mengevaluasi. `true` Argumennya adalah `filterJsonObject` (Informasi lebih lanjut tentang argumen ini dapat ditemukan di bawah di filterJsonObject bagian Argumen:.). Lihat Pemfilteran langganan yang disempurnakan.

> **Note**
>
> Anda dapat menggunakan fungsi ekstensi ini hanya di handler respons dari resolver langganan. Juga, kami sarankan menggunakan `util.transform.toSubscriptionFilter` untuk membuat filter Anda.

`extensions.invalidateSubscriptions(invalidationJsonObject)`

Digunakan untuk memulai pembatalan langganan dari mutasi. Argumennya adalah `invalidationJsonObject` (Informasi lebih lanjut tentang argumen ini dapat ditemukan di bawah di invalidationJsonObject bagian Argumen:.).

> **Note**
>
> Ekstensi ini hanya dapat digunakan dalam template pemetaan respons dari resolver mutasi.
> Anda hanya dapat menggunakan paling banyak lima panggilan `extensions.invalidateSubscriptions()` metode unik dalam satu permintaan. Jika Anda melebihi batas ini, Anda akan menerima kesalahan GraphQL.

Argumen: filterJsonObject

Objek JSON mendefinisikan filter langganan atau pembatalan. Ini adalah array filter dalam `afilterGroup`. Setiap filter adalah kumpulan filter individual.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                }
           ]

        },
        {
           "filters" : [
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
           ]

        }
    ]
}
```

Setiap filter memiliki tiga atribut:

- `fieldName`- Bidang skema GraphQL.
- `operator`— Jenis operator.
- `value`— Nilai untuk dibandingkan dengan `fieldName` nilai notifikasi langganan.

Berikut ini adalah contoh penugasan atribut ini:

```
{
 "fieldName" : "severity",
 "operator" : "le",
 "value" : context.result.severity
}
```

Argumen: invalidationJsonObject

`invalidationJsonObject`Mendefinisikan sebagai berikut:

- `subscriptionField`— Langganan skema GraphQL untuk membatalkan. Langganan tunggal, didefinisikan sebagai string di`subscriptionField`, dianggap untuk pembatalan.
- `payload`— Daftar pasangan kunci-nilai yang digunakan sebagai input untuk membatalkan langganan jika filter pembatalan mengevaluasi terhadap nilainya. `true`

  Contoh berikut membatalkan klien yang berlangganan dan terhubung menggunakan `onUserDelete` langganan saat filter pembatalan yang ditentukan dalam resolver langganan mengevaluasi terhadap nilainya. `true` `payload`

  ```
  export const request = (ctx) => ({ payload: null });

  export function response(ctx) {
   extensions.invalidateSubscriptions({
    subscriptionField: 'onUserDelete',
    payload: { group: 'Developer', type: 'Full-Time' },
   });
   return ctx.result;
  }
  ```

Pembantu XMLdi util.xml

`util.xml`berisi metode untuk membantu dengan konversi string XHTML.

daftar utilitas util.xml

`util.xml.toMap(String) : Object`

Mengkonversi string XHTML ke kamus.

Contoh 1:

```
Input:

<?xml version="1.0" encoding="UTF-8"?>
<posts>
<post>
    <id>1</id>
    <title>Getting started with GraphQL</title>
</post>
</posts>

Output (object):

{
    "posts":{
      "post":{
        "id":1,
        "title":"Getting started with GraphQL"
      }
    }
}
```

Contoh 2:

```
Input:

<?xml version="1.0" encoding="UTF-8"?>
<posts>
<post>
  <id>1</id>
  <title>Getting started with GraphQL</title>
</post>
<post>
  <id>2</id>
  <title>Getting started with AppSync</title>
</post>
```

```
</posts>

Output (JavaScript object):

{
    "posts":{
    "post":[
        {
            "id":1,
            "title":"Getting started with GraphQL"
        },
        {
            "id":2,
            "title":"Getting started with AppSync"
        }
    ]
    }
}
```

`util.xml.toJsonString(String, Boolean?) : String`

Mengkonversi string XHTML ke string JSON. Ini mirip dengan`toMap`, kecuali bahwa outputnya adalah string. Hal ini berguna jika Anda ingin langsung mengkonversi dan mengembalikan respon XMLdari objek HTTP ke JSON. Anda dapat mengatur parameter boolean opsional untuk menentukan apakah Anda ingin string-encode JSON.

JavaScriptreferensi fungsi resolver untuk DynamoDB

TheAWS AppSyncFungsi DynamoDB memungkinkan Anda untuk menggunakanGraphQLuntuk menyimpan dan mengambil data dalam tabel Amazon DynamoDB yang ada di akun Anda. Penyelesai ini bekerja dengan memungkinkan Anda memetakan permintaan GraphQL yang masuk ke dalam panggilan DynamoDB, dan kemudian memetakan respons DynamoDB kembali ke GraphQL. Bagian ini menjelaskan penangan permintaan dan respons untuk operasi DynamoDB yang didukung.

GetItem

TheGetItempermintaan memungkinkan Anda memberi tahuAWS AppSyncDynamoDB berfungsi untuk membuatGetItempermintaan ke DynamoDB, dan memungkinkan Anda untuk menentukan:

- Kunci item di DynamoDB

- Apakah akan menggunakan bacaan yang konsisten atau tidak

TheGetItempermintaan memiliki struktur sebagai berikut:

```
type DynamoDBGetItem = {
  operation: 'GetItem';
  key: { [key: string]: any };
  consistentRead?: ConsistentRead;
  projection?: {
    expression: string;
    expressionNames?: { [key: string]: string };
  };
};
```

Bidang didefinisikan sebagai berikut:

GetItemladang

GetItemdaftar bidang

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`GetItem`Operasi DynamoDB, ini harus diatur ke`GetItem`. Nilai ini diperlukan.

`key`

Kunci item di DynamoDB. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Nilai ini diperlukan.

`consistentRead`

Apakah akan melakukan pembacaan yang sangat konsisten dengan DynamoDB atau tidak. Ini opsional, dan defaultnya`false`.

`projection`

Proyeksi yang digunakan untuk menentukan atribut untuk kembali dari operasi DynamoDB. Untuk informasi lebih lanjut tentang proyeksi, lihatProyeksi. Bidang ini bersifat opsional.

Item yang dikembalikan dari DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON, dan tersedia dalam hasil konteks (`context.result`).

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihatJenis sistem (pemetaan respons).

Untuk informasi lebih lanjut tentangJavaScriptpenyelesai, lihatJavaScriptikhtisar penyelesai.

Contoh

Contoh berikut adalah handler permintaan fungsi untuk query GraphQL`getThing(foo: String!, bar: String!)`:

```
export function request(ctx) {
  const {foo, bar} = ctx.args
  return {
    operation : "GetItem",
    key : util.dynamodb.toMapValues({foo, bar}),
    consistentRead : true
  }
}
```

Untuk informasi lebih lanjut tentang DynamoDB`GetItem`API, lihatDokumentasi API DynamoDB.

PutItem

The`PutItem`meminta dokumen pemetaan memungkinkan Anda memberi tahuAWS AppSyncDynamoDB berfungsi untuk membuat`PutItem`permintaan ke DynamoDB, dan memungkinkan Anda untuk menentukan yang berikut:

- Kunci item di DynamoDB
- Isi lengkap item (terdiri dari`key`dan`attributeValues`)
- Kondisi agar operasi berhasil

The`PutItem`permintaan memiliki struktur sebagai berikut:

```
type DynamoDBPutItemRequest = {
  operation: 'PutItem';
  key: { [key: string]: any };
  attributeValues: { [key: string]: any};
  condition?: ConditionCheckExpression;
  customPartitionKey?: string;
  populateIndexFields?: boolean;
```

```
  _version?: number;
};
```

Bidang didefinisikan sebagai berikut:

PutItemladang

PutItemdaftar bidang

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`PutItem`Operasi DynamoDB, ini harus diatur ke`PutItem`. Nilai ini diperlukan.

`key`

Kunci item di DynamoDB. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Nilai ini diperlukan.

`attributeValues`

Sisa atribut item yang akan dimasukkan ke DynamoDB. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Bidang ini bersifat opsional.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan,`PutItem`request menimpa entri yang ada untuk item tersebut. Untuk informasi lebih lanjut tentang kondisi, lihatEkspresi kondisi. Nilai ini bersifat opsional.

`_version`

Nilai numerik yang mewakili versi item terbaru yang diketahui. Nilai ini bersifat opsional. Bidang ini digunakan untukDeteksi Konflikdan hanya didukung pada sumber data berversi.

`customPartitionKey`

Saat diaktifkan, nilai string ini memodifikasi format`ds_sk`dan`ds_pk`catatan yang digunakan oleh tabel sinkronisasi delta saat pembuatan versi telah diaktifkan (untuk informasi selengkapnya, lihatDeteksi dan sinkronisasi konflikdiAWS AppSyncPanduan Pengembang). Saat diaktifkan, pemrosesan`populateIndexFields`entri juga diaktifkan. Bidang ini bersifat opsional.

`populateIndexFields`

Nilai boolean yang, ketika diaktifkanbersama dengan**`customPartitionKey`**, membuat entri baru untuk setiap catatan di tabel sinkronisasi delta, khususnya di`gsi_ds_pk`dan`gsi_ds_sk`kolom. Untuk informasi lebih lanjut, lihatDeteksi dan sinkronisasi konflikdiAWS AppSyncPanduan Pengembang. Bidang ini bersifat opsional.

Item yang ditulis ke DynamoDB secara otomatis dikonversi ke tipe primitif GraphQL dan JSON dan tersedia dalam hasil konteks (`context.result`).

Item yang ditulis ke DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam hasil konteks (`context.result`).

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihatJenis sistem (pemetaan respons).

Untuk informasi lebih lanjut tentangJavaScriptpenyelesai, lihatJavaScriptikhtisar penyelesai.

Contoh 1

Contoh berikut adalah penangan permintaan fungsi untuk mutasi GraphQL`updateThing(foo: String!, bar: String!, name: String!, version: Int!)`.

Jika tidak ada item dengan kunci yang ditentukan, itu dibuat. Jika item sudah ada dengan kunci yang ditentukan, itu akan ditimpa.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { foo, bar, ...values} = ctx.args
  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({foo, bar}),
    attributeValues: util.dynamodb.toMapValues(values),
  };
}
```

Contoh 2

Contoh berikut adalah penangan permintaan fungsi untuk mutasi GraphQL`updateThing(foo: String!, bar: String!, name: String!, expectedVersion: Int!)`.

Contoh ini memverifikasi bahwa item saat ini di DynamoDB memilikiversionbidang diatur keexpectedVersion.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { foo, bar, name, expectedVersion } = ctx.args;
  const values = { name, version: expectedVersion + 1 };
  let condition = util.transform.toDynamoDBConditionExpression({
    version: { eq: expectedVersion },
  });

  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({ foo, bar }),
    attributeValues: util.dynamodb.toMapValues(values),
    condition,
  };
}
```

Untuk informasi lebih lanjut tentang DynamoDBPutItemAPI, lihatDokumentasi API DynamoDB.

UpdateItem

TheUpdateItempermintaan memungkinkan Anda untuk memberi tahuAWS AppSyncDynamoDB berfungsi untuk membuatUpdateItempermintaan ke DynamoDB dan memungkinkan Anda untuk menentukan yang berikut:

- Kunci item di DynamoDB
- Ekspresi pembaruan yang menjelaskan cara memperbarui item di DynamoDB
- Kondisi agar operasi berhasil

TheUpdateItempermintaan memiliki struktur sebagai berikut:

```
type DynamoDBUpdateItemRequest = {
  operation: 'UpdateItem';
  key: { [key: string]: any };
  update: {
    expression: string;
    expressionNames?: { [key: string]: string };
    expressionValues?: { [key: string]: any };
```

```
  };
  condition?: ConditionCheckExpression;
  customPartitionKey?: string;
  populateIndexFields?: boolean;
  _version?: number;
};
```

Bidang didefinisikan sebagai berikut:

UpdateItemladang

UpdateItemdaftar bidang

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`UpdateItem`Operasi DynamoDB, ini harus diatur ke`UpdateItem`. Nilai ini diperlukan.

`key`

Kunci item di DynamoDB. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Nilai ini diperlukan.

`update`

Theupdatebagian memungkinkan Anda menentukan ekspresi pembaruan yang menjelaskan cara memperbarui item di DynamoDB. Untuk informasi selengkapnya tentang cara menulis ekspresi pembaruan, lihatDynamoDBUpdateExpressionsdokumentasi. Bagian ini diperlukan.

Theupdatebagian memiliki tiga komponen:

`expression`

Ekspresi pembaruan. Nilai ini diperlukan.

`expressionNames`

Substitusi untuk atribut ekspresinamaplaceholder, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nama yang digunakan dalam`expression`, dan nilainya harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalam`expression`.

`expressionValues`

Substitusi untuk atribut ekspresinilaiplaceholder, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nilai yang digunakan dalam`expression`, dan nilainya harus berupa nilai yang diketik. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Ini harus ditentukan. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nilai atribut ekspresi yang digunakan dalam`expression`.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan,`UpdateItem`permintaan memperbarui entri yang ada terlepas dari statusnya saat ini. Untuk informasi lebih lanjut tentang kondisi, lihatEkspresi kondisi. Nilai ini bersifat opsional.

`_version`

Nilai numerik yang mewakili versi item terbaru yang diketahui. Nilai ini bersifat opsional. Bidang ini digunakan untukDeteksi Konflikdan hanya didukung pada sumber data berversi.

`customPartitionKey`

Saat diaktifkan, nilai string ini memodifikasi format`ds_sk`dan`ds_pk`catatan yang digunakan oleh tabel sinkronisasi delta saat pembuatan versi telah diaktifkan (untuk informasi selengkapnya, lihatDeteksi dan sinkronisasi konflikdiAWS AppSyncPanduan Pengembang). Saat diaktifkan, pemrosesan`populateIndexFields`entri juga diaktifkan. Bidang ini bersifat opsional.

`populateIndexFields`

Nilai boolean yang, ketika diaktifkanbersama dengan**`customPartitionKey`**, membuat entri baru untuk setiap catatan di tabel sinkronisasi delta, khususnya dig`si_ds_pk`dang`si_ds_sk`kolom. Untuk informasi lebih lanjut, lihatDeteksi dan sinkronisasi konflikdiAWS AppSyncPanduan Pengembang. Bidang ini bersifat opsional.

Item yang diperbarui di DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam hasil konteks (`context.result`).

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihatJenis sistem (pemetaan respons).

Untuk informasi lebih lanjut tentangJavaScriptpenyelesai, lihatJavaScriptikhtisar penyelesai.

Contoh 1

Contoh berikut adalah penangan permintaan fungsi untuk mutasi GraphQL`upvote(id: ID!)`.

Dalam contoh ini, item di DynamoDB memilikiu`pvotes`dan`version`onbidang bertambah dengan 1.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { id } = ctx.args;
  return {
    operation: 'UpdateItem',
    key: util.dynamodb.toMapValues({ id }),
    update: {
      expression: 'ADD #votefield :plusOne, version :plusOne',
      expressionNames: { '#votefield': 'upvotes' },
      expressionValues: { ':plusOne': { N: 1 } },
    },
  };
}
```

Contoh 2

Contoh berikut adalah penangan permintaan fungsi untuk mutasi GraphQL`updateItem(id: ID!, title: String, author: String, expectedVersion: Int!)`.

Ini adalah contoh kompleks yang memeriksa argumen dan secara dinamis menghasilkan ekspresi pembaruan yang hanya mencakup argumen yang telah disediakan oleh klien. Misalnya, jika`title`dan`author`dihilangkan, mereka tidak diperbarui. Jika argumen ditentukan tetapi nilainya adalah`null`, maka bidang itu dihapus dari objek di DynamoDB. Akhirnya, operasi memiliki kondisi, yang memverifikasi apakah item saat ini di DynamoDB memiliki`version`bidang diatur ke`expectedVersion`:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { args: { input: { id, ...values } } } = ctx;

  const condition = {
    id: { attributeExists: true },
    version: { eq: values.expectedVersion },
  };
  values.expectedVersion += 1;
  return dynamodbUpdateRequest({ keys: { id }, values, condition });
```

```
}


/**
 * Helper function to update an item
 * @returns an UpdateItem request
 */
function dynamodbUpdateRequest(params) {
  const { keys, values, condition: inCondObj } = params;

  const sets = [];
  const removes = [];
  const expressionNames = {};
  const expValues = {};

  // Iterate through the keys of the values
  for (const [key, value] of Object.entries(values)) {
    expressionNames[`#${key}`] = key;
    if (value) {
      sets.push(`#${key} = :${key}`);
      expValues[`:${key}`] = value;
    } else {
      removes.push(`#${key}`);
    }
  }

  let expression = sets.length ? `SET ${sets.join(', ')}` : '';
  expression += removes.length ? ` REMOVE ${removes.join(', ')}` : '';

  const condition = JSON.parse(
    util.transform.toDynamoDBConditionExpression(inCondObj)
  );

  return {
    operation: 'UpdateItem',
    key: util.dynamodb.toMapValues(keys),
    condition,
    update: {
      expression,
      expressionNames,
      expressionValues: util.dynamodb.toMapValues(expValues),
    },
  };
```

```
}
```

Untuk informasi lebih lanjut tentang DynamoDB`UpdateItem`API, lihatDokumentasi API DynamoDB.

DeleteItem

The`DeleteItem`permintaan memungkinkan Anda memberi tahuAWS AppSyncDynamoDB berfungsi untuk membuat`DeleteItem`permintaan ke DynamoDB, dan memungkinkan Anda untuk menentukan yang berikut:

- Kunci item di DynamoDB
- Kondisi agar operasi berhasil

The`DeleteItem`permintaan memiliki struktur sebagai berikut:

```
type DynamoDBDeleteItemRequest = {
  operation: 'DeleteItem';
  key: { [key: string]: any };
  condition?: ConditionCheckExpression;
  customPartitionKey?: string;
  populateIndexFields?: boolean;
  _version?: number;
};
```

Bidang didefinisikan sebagai berikut:

DeleteItemladang

DeleteItemdaftar bidang

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`DeleteItem`Operasi DynamoDB, ini harus diatur ke`DeleteItem`. Nilai ini diperlukan.

`key`

Kunci item di DynamoDB. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Nilai ini diperlukan.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan,`DeleteItem`permintaan menghapus item terlepas dari keadaan saat ini. Untuk informasi lebih lanjut tentang kondisi, lihatEkspresi kondisi. Nilai ini bersifat opsional.

`_version`

Nilai numerik yang mewakili versi item terbaru yang diketahui. Nilai ini bersifat opsional. Bidang ini digunakan untukDeteksi Konflikdan hanya didukung pada sumber data berversi.

`customPartitionKey`

Saat diaktifkan, nilai string ini memodifikasi format`ds_sk`dan`ds_pk`catatan yang digunakan oleh tabel sinkronisasi delta saat pembuatan versi telah diaktifkan (untuk informasi selengkapnya, lihatDeteksi dan sinkronisasi konflikdiAWS AppSyncPanduan Pengembang). Saat diaktifkan, pemrosesan`populateIndexFields`entri juga diaktifkan. Bidang ini bersifat opsional.

`populateIndexFields`

Nilai boolean yang, ketika diaktifkanbersama dengan**`customPartitionKey`**, membuat entri baru untuk setiap catatan di tabel sinkronisasi delta, khususnya dig`si_ds_pk`dang`si_ds_sk`kolom. Untuk informasi lebih lanjut, lihatDeteksi dan sinkronisasi konflikdiAWS AppSyncPanduan Pengembang. Bidang ini bersifat opsional.

Item yang dihapus dari DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam hasil konteks (`context.result`).

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihatJenis sistem (pemetaan respons).

Untuk informasi lebih lanjut tentangJavaScriptpenyelesai, lihatJavaScriptikhtisar penyelesai.

Contoh 1

Contoh berikut adalah penangan permintaan fungsi untuk mutasi GraphQL`deleteItem(id: ID!)`. Jika ada item dengan ID ini, item tersebut akan dihapus.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
```

```
    operation: 'DeleteItem',
    key: util.dynamodb.toMapValues({ id: ctx.args.id }),
  };
}
```

Contoh 2

Contoh berikut adalah penangan permintaan fungsi untuk mutasi GraphQL`deleteItem(id: ID!, expectedVersion: Int!)`. Jika ada item dengan ID ini, itu akan dihapus, tetapi hanya jika`version`bidang diatur ke`expectedVersion`:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const { id, expectedVersion } = ctx.args;
  const condition = {
    id: { attributeExists: true },
    version: { eq: expectedVersion },
  };
  return {
    operation: 'DeleteItem',
    key: util.dynamodb.toMapValues({ id }),
    condition: util.transform.toDynamoDBConditionExpression(condition),
  };
}
```

Untuk informasi lebih lanjut tentang DynamoDB`DeleteItem`API, lihatDokumentasi API DynamoDB.

Kueri

The`Query`objek permintaan memungkinkan Anda memberi tahuAWS AppSyncDynamoDB resolver untuk membuat`Query`permintaan ke DynamoDB, dan memungkinkan Anda untuk menentukan yang berikut:

- Ekspresi kunci
- Indeks mana yang akan digunakan
- Filter tambahan apa pun
- Berapa banyak item yang akan dikembalikan
- Apakah akan menggunakan pembacaan yang konsisten
- arah kueri (maju atau mundur)

- Token pagination

TheQuerypermintaan objek memiliki struktur sebagai berikut:

```
type DynamoDBQueryRequest = {
  operation: 'Query';
  query: {
    expression: string;
    expressionNames?: { [key: string]: string };
    expressionValues?: { [key: string]: any };
  };
  index?: string;
  nextToken?: string;
  limit?: number;
  scanIndexForward?: boolean;
  consistentRead?: boolean;
  select?: 'ALL_ATTRIBUTES' | 'ALL_PROJECTED_ATTRIBUTES' | 'SPECIFIC_ATTRIBUTES';
  filter?: {
    expression: string;
    expressionNames?: { [key: string]: string };
    expressionValues?: { [key: string]: any };
  };
  projection?: {
    expression: string;
    expressionNames?: { [key: string]: string };
  };
};
```

Bidang didefinisikan sebagai berikut:

Bidang kueri

Daftar bidang kueri

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`Query`Operasi DynamoDB, ini harus diatur ke`Query`. Nilai ini diperlukan.

`query`

Thequerybagian memungkinkan Anda menentukan ekspresi kondisi kunci yang menjelaskan item mana yang akan diambil dari DynamoDB. Untuk informasi selengkapnya tentang cara

menulis ekspresi kondisi kunci, lihatDynamoDBKeyConditionsdokumentasi. Bagian ini harus ditentukan.

`expression`

Ekspresi kueri. Bidang ini harus ditentukan.

`expressionNames`

Substitusi untuk atribut ekspresinamaplaceholder, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nama yang digunakan dalam`expression`, dan nilainya harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalam`expression`.

`expressionValues`

Substitusi untuk atribut ekspresinilaiplaceholder, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nilai yang digunakan dalam`expression`, dan nilainya harus berupa nilai yang diketik. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Nilai ini diperlukan. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nilai atribut ekspresi yang digunakan dalam`expression`.

`filter`

Filter tambahan yang dapat digunakan untuk memfilter hasil dari DynamoDB sebelum dikembalikan. Untuk informasi selengkapnya tentang filter, lihatFilter. Bidang ini bersifat opsional.

`index`

Nama indeks untuk query. Operasi kueri DynamoDB memungkinkan Anda untuk memindai Indeks Sekunder Lokal dan Indeks Sekunder Global selain indeks kunci utama untuk kunci hash. Jika ditentukan, ini memberitahu DynamoDB untuk query indeks yang ditentukan. Jika dihilangkan, indeks kunci utama ditanyakan.

`nextToken`

Token pagination untuk melanjutkan kueri sebelumnya. Ini akan diperoleh dari kueri sebelumnya. Bidang ini bersifat opsional.

`limit`

Jumlah maksimum item untuk dievaluasi (belum tentu jumlah item yang cocok). Bidang ini bersifat opsional.

`scanIndexForward`

Boolean yang menunjukkan apakah akan melakukan kueri maju atau mundur. Bidang ini opsional, dan defaultnya`true`.

`consistentRead`

Boolean yang menunjukkan apakah akan menggunakan pembacaan yang konsisten saat menanyakan DynamoDB. Bidang ini opsional, dan defaultnya`false`.

`select`

Secara default,AWS AppSyncDynamoDB resolver hanya mengembalikan atribut yang diproyeksikan ke dalam indeks. Jika lebih banyak atribut diperlukan, Anda dapat mengatur bidang ini. Bidang ini bersifat opsional. Nilai yang didukung adalah:

`ALL_ATTRIBUTES`

Mengembalikan semua atribut item dari tabel tertentu atau indeks. Jika Anda menanyakan indeks sekunder lokal, DynamoDB mengambil seluruh item dari tabel induk untuk setiap item yang cocok dalam indeks. Jika indeks dikonfigurasi untuk memproyeksikan semua atribut item, semua data dapat diperoleh dari indeks sekunder lokal dan tidak diperlukan pengambilan.

`ALL_PROJECTED_ATTRIBUTES`

Diizinkan hanya saat menanyakan indeks. Mengambil semua atribut yang telah diproyeksikan ke dalam indeks. Jika indeks dikonfigurasi untuk memproyeksikan semua atribut, nilai pengembalian ini setara dengan menentukan`ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Mengembalikan hanya atribut yang tercantum dalam`projection`ini`expression`. Nilai pengembalian ini setara dengan menentukan`projection`ini`expression`tanpa menentukan nilai apa pun untuk`Select`.

`projection`

Proyeksi yang digunakan untuk menentukan atribut untuk kembali dari operasi DynamoDB. Untuk informasi lebih lanjut tentang proyeksi, lihatProyeksi. Bidang ini bersifat opsional.

Hasil dari DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam hasil konteks (`context.result`).

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihatJenis sistem (pemetaan respons).

Untuk informasi lebih lanjut tentangJavaScriptpenyelesai, lihatJavaScriptikhtisar penyelesai.

Hasilnya memiliki struktur sebagai berikut:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Bidang didefinisikan sebagai berikut:

`items`

Daftar yang berisi item yang dikembalikan oleh query DynamoDB.

`nextToken`

Jika mungkin ada lebih banyak hasil,`nextToken`berisi token pagination yang dapat Anda gunakan dalam permintaan lain. Perhatikan bahwaAWS AppSyncmengenkripsi dan mengaburkan token pagination yang dikembalikan dari DynamoDB. Ini mencegah data tabel Anda bocor secara tidak sengaja ke penelepon. Perhatikan juga bahwa token pagination ini tidak dapat digunakan di berbagai fungsi atau resolver.

`scannedCount`

Jumlah item yang cocok dengan ekspresi kondisi kueri, sebelum ekspresi filter (jika ada) diterapkan.

Contoh

Contoh berikut adalah handler permintaan fungsi untuk query GraphQL`getPosts(owner: ID!)`.

Dalam contoh ini, indeks sekunder global pada tabel ditanyakan untuk mengembalikan semua posting yang dimiliki oleh ID yang ditentukan.

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const { owner } = ctx.args;
  return {
    operation: 'Query',
```

```
    query: {
      expression: 'ownerId = :ownerId',
      expressionValues: util.dynamodb.toMapValues({ ':ownerId': owner }),
    },
    index: 'owner-index',
  };
}
```

Untuk informasi lebih lanjut tentang DynamoDB`Query`API, lihatDokumentasi API DynamoDB.

Pemindaian

The`Scan`permintaan memungkinkan Anda memberi tahuAWS AppSyncDynamoDB berfungsi untuk membuat`Scan`permintaan ke DynamoDB, dan memungkinkan Anda untuk menentukan yang berikut:

- Filter untuk mengecualikan hasil
- Indeks mana yang akan digunakan
- Berapa banyak item yang akan dikembalikan
- Apakah akan menggunakan pembacaan yang konsisten
- Token pagination
- Pemindaian paralel

The`Scan`permintaan objek memiliki struktur sebagai berikut:

```
type DynamoDBScanRequest = {
  operation: 'Scan';
  index?: string;
  limit?: number;
  consistentRead?: boolean;
  nextToken?: string;
  totalSegments?: number;
  segment?: number;
  filter?: {
    expression: string;
    expressionNames?: { [key: string]: string };
    expressionValues?: { [key: string]: any };
  };
  projection?: {
    expression: string;
    expressionNames?: { [key: string]: string };
```

```
  };
};
```

Bidang didefinisikan sebagai berikut:

Bidang pemindaian

Pindai daftar bidang

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`Scan`Operasi DynamoDB, ini harus diatur ke`Scan`. Nilai ini diperlukan.

`filter`

Filter yang dapat digunakan untuk memfilter hasil dari DynamoDB sebelum dikembalikan. Untuk informasi selengkapnya tentang filter, lihatFilter. Bidang ini bersifat opsional.

`index`

Nama indeks untuk query. Operasi kueri DynamoDB memungkinkan Anda untuk memindai Indeks Sekunder Lokal dan Indeks Sekunder Global selain indeks kunci utama untuk kunci hash. Jika ditentukan, ini memberitahu DynamoDB untuk query indeks yang ditentukan. Jika dihilangkan, indeks kunci utama ditanyakan.

`limit`

Jumlah maksimum item untuk dievaluasi pada satu waktu. Bidang ini bersifat opsional.

`consistentRead`

Boolean yang menunjukkan apakah akan menggunakan pembacaan yang konsisten saat menanyakan DynamoDB. Bidang ini opsional, dan defaultnya`false`.

`nextToken`

Token pagination untuk melanjutkan kueri sebelumnya. Ini akan diperoleh dari kueri sebelumnya. Bidang ini bersifat opsional.

`select`

Secara default,AWS AppSyncFungsi DynamoDB hanya mengembalikan atribut apa pun yang diproyeksikan ke dalam indeks. Jika lebih banyak atribut diperlukan, maka bidang ini dapat diatur. Bidang ini bersifat opsional. Nilai yang didukung adalah:

`ALL_ATTRIBUTES`

Mengembalikan semua atribut item dari tabel tertentu atau indeks. Jika Anda menanyakan indeks sekunder lokal, DynamoDB mengambil seluruh item dari tabel induk untuk setiap item yang cocok dalam indeks. Jika indeks dikonfigurasi untuk memproyeksikan semua atribut item, semua data dapat diperoleh dari indeks sekunder lokal dan tidak diperlukan pengambilan.

`ALL_PROJECTED_ATTRIBUTES`

Diizinkan hanya saat menanyakan indeks. Mengambil semua atribut yang telah diproyeksikan ke dalam indeks. Jika indeks dikonfigurasi untuk memproyeksikan semua atribut, nilai pengembalian ini setara dengan menentukan`ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Mengembalikan hanya atribut yang tercantum dalam`projection`ini`expression`. Nilai pengembalian ini setara dengan menentukan`projection`ini`expression`tanpa menentukan nilai apa pun untuk`Select`.

`totalSegments`

Jumlah segmen untuk mempartisi tabel dengan saat melakukan pemindaian paralel. Bidang ini bersifat opsional, tetapi harus ditentukan jika`segment`ditentukan.

`segment`

Segmen tabel dalam operasi ini saat melakukan pemindaian paralel. Bidang ini bersifat opsional, tetapi harus ditentukan jika`totalSegments`ditentukan.

`projection`

Proyeksi yang digunakan untuk menentukan atribut untuk kembali dari operasi DynamoDB. Untuk informasi lebih lanjut tentang proyeksi, lihatProyeksi. Bidang ini bersifat opsional.

Hasil yang dikembalikan oleh pemindaian DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam hasil konteks (`context.result`).

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihatJenis sistem (pemetaan respons).

Untuk informasi lebih lanjut tentangJavaScriptpenyelesai, lihatJavaScriptikhtisar penyelesai.

Hasilnya memiliki struktur sebagai berikut:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Bidang didefinisikan sebagai berikut:

`items`

Daftar yang berisi item yang dikembalikan oleh pemindaian DynamoDB.

`nextToken`

Jika mungkin ada lebih banyak hasil,`nextToken`berisi token pagination yang dapat Anda gunakan dalam permintaan lain.AWS AppSyncmengenkripsi dan mengaburkan token pagination yang dikembalikan dari DynamoDB. Ini mencegah data tabel Anda bocor secara tidak sengaja ke penelepon. Selain itu, token pagination ini tidak dapat digunakan di berbagai fungsi atau resolver.

`scannedCount`

Jumlah item yang diambil oleh DynamoDB sebelum ekspresi filter (jika ada) diterapkan.

Contoh 1

Contoh berikut adalah handler permintaan fungsi untuk query GraphQL:`allPosts`.

Dalam contoh ini, semua entri dalam tabel dikembalikan.

```
export function request(ctx) {
  return { operation: 'Scan' };
}
```

Contoh 2

Contoh berikut adalah handler permintaan fungsi untuk query GraphQL:`postsMatching(title: String!)`.

Dalam contoh ini, semua entri dalam tabel dikembalikan di mana judul dimulai dengan`title`argumen.

```
export function request(ctx) {
```

```
  const { title } = ctx.args;
  const filter = { filter: { beginsWith: title } };
  return {
    operation: 'Scan',
    filter: JSON.parse(util.transform.toDynamoDBFilterExpression(filter)),
  };
}
```

Untuk informasi lebih lanjut tentang DynamoDB`Scan`API, lihatDokumentasi API DynamoDB.

Sinkronkan

The`Sync`objek permintaan memungkinkan Anda mengambil semua hasil dari tabel DynamoDB dan kemudian hanya menerima data yang diubah sejak kueri terakhir Anda (pembaruan delta).`Sync`permintaan hanya dapat dibuat ke sumber data DynamoDB berversi. Anda dapat menentukan sebagai berikut:

- Filter untuk mengecualikan hasil
- Berapa banyak item yang akan dikembalikan
- Token Paginasi
- Kapan terakhir Anda`Sync`operasi dimulai

The`Sync`permintaan objek memiliki struktur sebagai berikut:

```
type DynamoDBSyncRequest = {
  operation: 'Sync';
  basePartitionKey?: string;
  deltaIndexName?: string;
  limit?: number;
  nextToken?: string;
  lastSync?: number;
  filter?: {
    expression: string;
    expressionNames?: { [key: string]: string };
    expressionValues?: { [key: string]: any };
  };
};
```

Bidang didefinisikan sebagai berikut:

Bidang sinkronisasi

Daftar bidang sinkronisasi

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`Sync`operasi, ini harus diatur ke`Sync`. Nilai ini diperlukan.

`filter`

Filter yang dapat digunakan untuk memfilter hasil dari DynamoDB sebelum dikembalikan. Untuk informasi selengkapnya tentang filter, lihatFilter. Bidang ini bersifat opsional.

`limit`

Jumlah maksimum item untuk dievaluasi pada satu waktu. Bidang ini bersifat opsional. Jika dihilangkan, batas default akan diatur ke`100`barang. Nilai maksimum untuk bidang ini adalah`1000`barang.

`nextToken`

Token pagination untuk melanjutkan kueri sebelumnya. Ini akan diperoleh dari kueri sebelumnya. Bidang ini bersifat opsional.

`lastSync`

Momen, dalam milidetik epoch, ketika yang terakhir berhasil`Sync`operasi dimulai. Jika ditentukan, hanya item yang telah berubah setelahnya`lastSync`dikembalikan. Bidang ini opsional, dan hanya boleh diisi setelah mengambil semua halaman dari inisiasi`Sync`operasi. Jika dihilangkan, hasil dari`Base`tabel akan dikembalikan, jika tidak, hasil dari`ds`meja akan dikembalikan.

`basePartitionKey`

Kunci partisi dari`Base`tabel yang digunakan saat melakukan`Sync`operasi. Bidang ini memungkinkan`Sync`operasi yang akan dilakukan ketika tabel menggunakan kunci partisi kustom. Ini adalah bidang opsional.

`deltaIndexName`

Indeks yang digunakan untuk`Sync`operasi. Indeks ini diperlukan untuk mengaktifkan`Sync`operasi pada seluruh tabel delta store ketika tabel menggunakan kunci partisi kustom. The`Sync`operasi akan dilakukan pada GSI (dibuat pada`gsi_ds_pk`dan`gsi_ds_sk`). Bidang ini bersifat opsional.

Hasil yang dikembalikan oleh sinkronisasi DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam hasil konteks (`context.result`).

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihatJenis sistem (pemetaan respons).

Untuk informasi lebih lanjut tentangJavaScriptpenyelesai, lihatJavaScriptikhtisar penyelesai.

Hasilnya memiliki struktur sebagai berikut:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10,
    startedAt = 1550000000000
}
```

Bidang didefinisikan sebagai berikut:

`items`

Daftar yang berisi item yang dikembalikan oleh sinkronisasi.

`nextToken`

Jika mungkin ada lebih banyak hasil,`nextToken`berisi token pagination yang dapat Anda gunakan dalam permintaan lain.AWS AppSyncmengenkripsi dan mengaburkan token pagination yang dikembalikan dari DynamoDB. Ini mencegah data tabel Anda bocor secara tidak sengaja ke penelepon. Selain itu, token pagination ini tidak dapat digunakan di berbagai fungsi atau resolver.

`scannedCount`

Jumlah item yang diambil oleh DynamoDB sebelum ekspresi filter (jika ada) diterapkan.

`startedAt`

Saat ini, dalam milidetik epoch, ketika operasi sinkronisasi dimulai yang dapat Anda simpan secara lokal dan gunakan dalam permintaan lain sebagai`lastSync`argumen. Jika token pagination disertakan dalam permintaan, nilai ini akan sama dengan yang dikembalikan oleh permintaan untuk halaman pertama hasil.

Contoh 1

Contoh berikut adalah handler permintaan fungsi untuk query GraphQL:`syncPosts(nextToken: String, lastSync: AWSTimestamp)`.

Dalam contoh ini, jika`lastSync`dihilangkan, semua entri di tabel dasar dikembalikan. Jika`lastSync`disediakan, hanya entri dalam tabel sinkronisasi delta yang telah berubah sejak`lastSync`dikembalikan.

```
export function request(ctx) {
  const { nextToken, lastSync } = ctx.args;
  return { operation: 'Sync', limit: 100, nextToken, lastSync };
}
```

BatchGetItem

The`BatchGetItem`objek permintaan memungkinkan Anda memberi tahuAWS AppSyncDynamoDB berfungsi untuk membuat`BatchGetItem`permintaan ke DynamoDB untuk mengambil beberapa item, berpotensi di beberapa tabel. Untuk objek permintaan ini, Anda harus menentukan yang berikut:

- Nama tabel tempat untuk mengambil item dari
- Kunci item untuk mengambil dari setiap tabel

DynamoDB`BatchGetItem`batas berlaku dantidak ada ekspresi kondisidapat disediakan.

The`BatchGetItem`permintaan objek memiliki struktur sebagai berikut:

```
type DynamoDBBatchGetItemRequest = {
  operation: 'BatchGetItem';
  tables: {
    [tableName: string]: {
      keys: { [key: string]: any }[];
      consistentRead?: boolean;
      projection?: {
        expression: string;
        expressionNames?: { [key: string]: string };
      };
    };
  };
};
```

Bidang didefinisikan sebagai berikut:

BatchGetItemladang

BatchGetItemdaftar bidang

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`BatchGetItem`Operasi DynamoDB, ini harus diatur ke`BatchGetItem`. Nilai ini diperlukan.

`tables`

Tabel DynamoDB untuk mengambil item dari. Nilainya adalah peta di mana nama tabel ditentukan sebagai kunci peta. Setidaknya satu meja harus disediakan. Ini`tables`nilai diperlukan.

`keys`

Daftar kunci DynamoDB yang mewakili kunci utama item yang akan diambil. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan).

`consistentRead`

Apakah akan menggunakan pembacaan yang konsisten saat mengeksekusiGetItemoperasi. Nilai ini opsional dan defaultnyapalsu.

`projection`

Proyeksi yang digunakan untuk menentukan atribut untuk kembali dari operasi DynamoDB. Untuk informasi lebih lanjut tentang proyeksi, lihatProyeksi. Bidang ini bersifat opsional.

Hal-hal yang perlu diingat:

- Jika item belum diambil dari tabel,nolelemen muncul di blok data untuk tabel itu.
- Hasil pemanggilan diurutkan per tabel, berdasarkan urutan di mana mereka disediakan di dalam objek permintaan.
- Masing-masing`Get`perintah di dalam`BatchGetItem`bersifat atom, namun, batch dapat diproses sebagian. Jika batch diproses sebagian karena kesalahan, kunci yang belum diproses dikembalikan sebagai bagian dari hasil pemanggilan di dalam`UnprocessedKeys`blok.

- `BatchGetItem`terbatas pada 100 kunci.

Untuk contoh berikut fungsi request handler:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const { authorId, postId } = ctx.args;
  return {
    operation: 'BatchGetItem',
    tables: {
      authors: [util.dynamodb.toMapValues({ authorId })],
      posts: [util.dynamodb.toMapValues({ authorId, postId })],
    },
  };
}
```

Hasil pemanggilan tersedia di`ctx.result`adalah sebagai berikut:

```
{
   "data": {
     "authors": [null],
     "posts": [
        // Was retrieved
        {
          "authorId": "a1",
          "postId": "p2",
          "postTitle": "title",
          "postDescription": "description",
        }
     ]
   },
   "unprocessedKeys": {
     "authors": [
        // This item was not processed due to an error
        {
          "authorId": "a1"
        }
      ],
     "posts": []
   }
}
```

Thectx.errorberisi rincian tentang kesalahan. Kuncinyadata,UnprocessedKeys, dan setiap tombol tabel yang disediakan dalam hasil dalam objek permintaan fungsi dijamin akan hadir dalam hasil pemanggilan. Item yang telah dihapus muncul didatablok. Item yang belum diproses ditandai sebagainoldi dalam blok data dan ditempatkan di dalamUnprocessedKeysblok.

BatchDeleteItem

TheBatchDeleteItemobjek permintaan memungkinkan Anda memberi tahuAWS AppSyncDynamoDB berfungsi untuk membuatBatchWriteItempermintaan ke DynamoDB untuk menghapus beberapa item, berpotensi di beberapa tabel. Untuk objek permintaan ini, Anda harus menentukan yang berikut:

- Nama tabel tempat menghapus item dari
- Kunci item yang akan dihapus dari setiap tabel

DynamoDBBatchWriteItembatas berlaku dantidak ada ekspresi kondisidapat disediakan.

TheBatchDeleteItempermintaan objek memiliki struktur sebagai berikut:

```
type DynamoDBBatchDeleteItemRequest = {
  operation: 'BatchDeleteItem';
  tables: {
    [tableName: string]: { [key: string]: any }[];
  };
};
```

Bidang didefinisikan sebagai berikut:

BatchDeleteItemladang

BatchDeleteItemdaftar bidang

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukanBatchDeleteItemOperasi DynamoDB, ini harus diatur keBatchDeleteItem. Nilai ini diperlukan.

`tables`

Tabel DynamoDB untuk menghapus item dari. Setiap tabel adalah daftar kunci DynamoDB yang mewakili kunci utama item yang akan dihapus. Item DynamoDB mungkin memiliki kunci

hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Setidaknya satu meja harus disediakan. The`tables`nilai diperlukan.

Hal-hal yang perlu diingat:

- Bertentangan dengan`DeleteItem`operasi, item yang dihapus sepenuhnya tidak dikembalikan dalam respons. Hanya kunci yang dilewati yang dikembalikan.
- Jika item belum dihapus dari tabel,nolelemen muncul di blok data untuk tabel itu.
- Hasil pemanggilan diurutkan per tabel, berdasarkan urutan di mana mereka disediakan di dalam objek permintaan.
- Masing-masing`Delete`perintah di dalam`BatchDeleteItem`adalah atom. Namun batch dapat diproses sebagian. Jika batch diproses sebagian karena kesalahan, kunci yang belum diproses dikembalikan sebagai bagian dari hasil pemanggilan di dalamUnprocessedKeysblok.
- `BatchDeleteItem`terbatas pada 25 kunci.

Untuk contoh berikut fungsi request handler:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const { authorId, postId } = ctx.args;
  return {
    operation: 'BatchDeleteItem',
    tables: {
      authors: [util.dynamodb.toMapValues({ authorId })],
      posts: [util.dynamodb.toMapValues({ authorId, postId })],
    },
  };
}
```

Hasil pemanggilan tersedia di`ctx.result`adalah sebagai berikut:

```
{
   "data": {
     "authors": [null],
     "posts": [
        // Was deleted
```

```
        {
          "authorId": "a1",
          "postId": "p2"
        }
      ]
    },
    "unprocessedKeys": {
      "authors": [
         // This key was not processed due to an error
         {
           "authorId": "a1"
         }
       ],
      "posts": []
    }
}
```

The`ctx.error`berisi rincian tentang kesalahan. Kuncinyadata,UnprocessedKeys, dan setiap tombol tabel yang disediakan dalam objek permintaan fungsi dijamin akan hadir dalam hasil pemanggilan. Item yang telah dihapus hadir didatablok. Item yang belum diproses ditandai sebagainoldi dalam blok data dan ditempatkan di dalamUnprocessedKeysblok.

BatchPutItem

The`BatchPutItem`objek permintaan memungkinkan Anda memberi tahuAWS AppSyncDynamoDB berfungsi untuk membuat`BatchWriteItem`permintaan ke DynamoDB untuk menempatkan beberapa item, berpotensi di beberapa tabel. Untuk objek permintaan ini, Anda harus menentukan yang berikut:

- Nama tabel tempat meletakkan item
- Item lengkap untuk dimasukkan ke dalam setiap tabel

DynamoDB`BatchWriteItem`batas berlaku dantidak ada ekspresi kondisidapat disediakan.

The`BatchPutItem`permintaan objek memiliki struktur sebagai berikut:

```
type DynamoDBBatchPutItemRequest = {
  operation: 'BatchPutItem';
  tables: {
    [tableName: string]: { [key: string]: any}[];
  };
```

```
};
```

Bidang didefinisikan sebagai berikut:

BatchPutItemladang

BatchPutItemdaftar bidang

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`BatchPutItem`Operasi DynamoDB, ini harus diatur ke`BatchPutItem`. Nilai ini diperlukan.

`tables`

Tabel DynamoDB untuk menempatkan item di. Setiap entri tabel mewakili daftar item DynamoDB untuk disisipkan untuk tabel khusus ini. Setidaknya satu meja harus disediakan. Nilai ini diperlukan.

Hal-hal yang perlu diingat:

- Item yang dimasukkan sepenuhnya dikembalikan dalam respons, jika berhasil.
- Jika item belum dimasukkan dalam tabel,nolelemen ditampilkan di blok data untuk tabel itu.
- Item yang disisipkan diurutkan per tabel, berdasarkan urutan di mana mereka disediakan di dalam objek permintaan.
- Masing-masing`Put`perintah di dalam`BatchPutItem`bersifat atom, namun, batch dapat diproses sebagian. Jika batch diproses sebagian karena kesalahan, kunci yang belum diproses dikembalikan sebagai bagian dari hasil pemanggilan di dalamUnprocessedKeysblok.
- `BatchPutItem`terbatas pada 25 item.

Untuk contoh berikut fungsi request handler:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const { authorId, postId, name, title } = ctx.args;
  return {
    operation: 'BatchPutItem',
    tables: {
      authors: [util.dynamodb.toMapValues({ authorId, name })],
```

```
      posts: [util.dynamodb.toMapValues({ authorId, postId, title })],
    },
  };
}
```

Hasil pemanggilan tersedia di`ctx.result`adalah sebagai berikut:

```
{
   "data": {
     "authors": [
         null
     ],
     "posts": [
        // Was inserted
        {
          "authorId": "a1",
          "postId": "p2",
          "title": "title"
        }
     ]
   },
   "unprocessedItems": {
     "authors": [
        // This item was not processed due to an error
        {
          "authorId": "a1",
          "name": "a1_name"
        }
      ],
     "posts": []
   }
}
```

The`ctx.error`berisi rincian tentang kesalahan. Kuncinyadata,UnProcessEditems, dan setiap kunci tabel yang disediakan dalam objek permintaan dijamin akan hadir dalam hasil pemanggilan. Item yang telah disisipkan ada didatablok. Item yang belum diproses ditandai sebagainoldi dalam blok data dan ditempatkan di dalamUnProcessEditemsblok.

TransactGetItems

The`TransactGetItems`permintaan objek memungkinkan Anda untuk memberitahuAWS AppSyncDynamoDB berfungsi untuk membuat`TransactGetItems`permintaan ke DynamoDB untuk

mengambil beberapa item, berpotensi di beberapa tabel. Untuk objek permintaan ini, Anda harus menentukan yang berikut:

- Nama tabel dari setiap item permintaan tempat mengambil item dari
- Kunci dari setiap item permintaan untuk diambil dari setiap tabel

DynamoDB`TransactGetItems`batas berlaku dantidak ada ekspresi kondisidapat disediakan.

The`TransactGetItems`permintaan objek memiliki struktur sebagai berikut:

```
type DynamoDBTransactGetItemsRequest = {
  operation: 'TransactGetItems';
  transactItems: { table: string; key: { [key: string]: any }; projection?:
 { expression: string; expressionNames?: { [key: string]: string }; }[];
  };
};
```

Bidang didefinisikan sebagai berikut:

TransactGetItemsladang

TransactGetItemsdaftar bidang

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`TransactGetItems`Operasi DynamoDB, ini harus diatur ke`TransactGetItems`. Nilai ini diperlukan.

`transactItems`

Item permintaan untuk disertakan. Nilainya adalah array item permintaan. Setidaknya satu item permintaan harus disediakan. Ini`transactItems`nilai diperlukan.

`table`

Tabel DynamoDB untuk mengambil item dari. Nilainya adalah string dari nama tabel. Ini`table`nilai diperlukan.

`key`

Kunci DynamoDB mewakili kunci utama item yang akan diambil. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel.

Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan).

`projection`

Proyeksi yang digunakan untuk menentukan atribut untuk kembali dari operasi DynamoDB. Untuk informasi lebih lanjut tentang proyeksi, lihatProyeksi. Bidang ini bersifat opsional.

Hal-hal yang perlu diingat:

- Jika transaksi berhasil, urutan item yang diambil di`items`blok akan sama dengan urutan item permintaan.
- Transaksi dilakukan dalamall-or-nothingcara. Jika ada item permintaan yang menyebabkan kesalahan, seluruh transaksi tidak akan dilakukan dan rincian kesalahan akan dikembalikan.
- Item permintaan yang tidak dapat diambil bukanlah kesalahan. Sebaliknya, anolelemen muncul diitemblok di posisi yang sesuai.
- Jika kesalahan transaksi adalahTransactionCanceledException,`cancellationReasons`blok akan diisi. Urutan alasan pembatalan di`cancellationReasons`blok akan sama dengan urutan item permintaan.
- `TransactGetItems`terbatas pada 25 item permintaan.

Untuk contoh berikut fungsi request handler:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const { authorId, postId } = ctx.args;
  return {
    operation: 'TransactGetItems',
    transactItems: [
      {
        table: 'posts',
        key: util.dynamodb.toMapValues({ postId }),
      },
      {
        table: 'authors',
        key: util.dynamodb.toMapValues({ authorId }),
      },
    ],
```

```
  };
}
```

Jika transaksi berhasil dan hanya item yang diminta pertama yang diambil, hasil pemanggilan tersedia di`ctx.result`adalah sebagai berikut:

```
{
    "items": [
       {
           // Attributes of the first requested item
           "post_id": "p1",
           "post_title": "title",
           "post_description": "description"
       },
       // Could not retrieve the second requested item
       null,
    ],
    "cancellationReasons": null
}
```

Jika transaksi gagal karenaTransactionCanceledExceptiondisebabkan oleh item permintaan pertama, hasil pemanggilan tersedia di`ctx.result`adalah sebagai berikut:

```
{
    "items": null,
    "cancellationReasons": [
       {
           "type":"Sample error type",
           "message":"Sample error message"
       },
       {
           "type":"None",
           "message":"None"
       }
    ]
}
```

The`ctx.error`berisi rincian tentang kesalahan. KuncinyaitemdanPembatalanAlasandijamin akan hadir di`ctx.result`.

TransactWriteItems

The`TransactWriteItems`objek permintaan memungkinkan Anda memberi tahuAWS AppSyncDynamoDB berfungsi untuk membuat`TransactWriteItems`permintaan ke DynamoDB untuk menulis beberapa item, berpotensi ke beberapa tabel. Untuk objek permintaan ini, Anda harus menentukan yang berikut:

- Nama tabel tujuan dari setiap item permintaan
- Pengoperasian setiap item permintaan untuk dilakukan. Ada empat jenis operasi yang didukung:PutItem,UpdateItem,DeleteItem, danConditionCheck
- Kunci dari setiap item permintaan untuk menulis

DynamoDB`TransactWriteItems`batas berlaku.

The`TransactWriteItems`permintaan objek memiliki struktur sebagai berikut:

```
type DynamoDBTransactWriteItemsRequest = {
  operation: 'TransactWriteItems';
  transactItems: TransactItem[];
};
type TransactItem =
  | TransactWritePutItem
  | TransactWriteUpdateItem
  | TransactWriteDeleteItem
  | TransactWriteConditionCheckItem;
type TransactWritePutItem = {
  table: string;
  operation: 'PutItem';
  key: { [key: string]: any };
  attributeValues: { [key: string]: string};
  condition?: TransactConditionCheckExpression;
};
type TransactWriteUpdateItem = {
  table: string;
  operation: 'UpdateItem';
  key: { [key: string]: any };
  update: DynamoDBExpression;
  condition?: TransactConditionCheckExpression;
};
type TransactWriteDeleteItem = {
  table: string;
```

```
  operation: 'DeleteItem';
  key: { [key: string]: any };
  condition?: TransactConditionCheckExpression;
};
type TransactWriteConditionCheckItem = {
  table: string;
  operation: 'ConditionCheck';
  key: { [key: string]: any };
  condition?: TransactConditionCheckExpression;
};
type TransactConditionCheckExpression = {
  expression: string;
  expressionNames?: { [key: string]: string};
  expressionValues?: { [key: string]: any};
  returnValuesOnConditionCheckFailure: boolean;
};
```

TransactWriteItemsladang

TransactWriteItemsdaftar bidang

Bidang didefinisikan sebagai berikut:

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`TransactWriteItems`Operasi DynamoDB, ini harus diatur ke`TransactWriteItems`. Nilai ini diperlukan.

`transactItems`

Item permintaan untuk disertakan. Nilainya adalah array item permintaan. Setidaknya satu item permintaan harus disediakan. Ini`transactItems`nilai diperlukan.

Untuk`PutItem`, bidang didefinisikan sebagai berikut:

`table`

Tabel DynamoDB tujuan. Nilainya adalah string dari nama tabel. Ini`table`nilai diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`PutItem`Operasi DynamoDB, ini harus diatur ke`PutItem`. Nilai ini diperlukan.

`key`

Kunci DynamoDB mewakili kunci utama item yang akan diletakkan. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Nilai ini diperlukan.

`attributeValues`

Sisa atribut item yang akan dimasukkan ke DynamoDB. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Bidang ini bersifat opsional.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan,`PutItem`request menimpa entri yang ada untuk item tersebut. Anda dapat menentukan apakah akan mengambil kembali item yang ada saat pemeriksaan kondisi gagal. Untuk informasi lebih lanjut tentang kondisi transaksional, lihatEkspresi kondisi transaksi. Nilai ini bersifat opsional.

Untuk`UpdateItem`, bidang didefinisikan sebagai berikut:

`table`

Tabel DynamoDB untuk memperbarui. Nilainya adalah string dari nama tabel. Ini`table`nilai diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`UpdateItem`Operasi DynamoDB, ini harus diatur ke`UpdateItem`. Nilai ini diperlukan.

`key`

Kunci DynamoDB mewakili kunci utama item yang akan diperbarui. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Nilai ini diperlukan.

`update`

The`update`bagian memungkinkan Anda menentukan ekspresi pembaruan yang menjelaskan cara memperbarui item di DynamoDB. Untuk informasi selengkapnya tentang

cara menulis ekspresi pembaruan, lihatDynamoDBUpdateExpressionsdokumentasi. Bagian ini diperlukan.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan,`UpdateItem`permintaan memperbarui entri yang ada terlepas dari statusnya saat ini. Anda dapat menentukan apakah akan mengambil kembali item yang ada saat pemeriksaan kondisi gagal. Untuk informasi lebih lanjut tentang kondisi transaksional, lihatEkspresi kondisi transaksi. Nilai ini bersifat opsional.

Untuk`DeleteItem`, bidang didefinisikan sebagai berikut:

`table`

Tabel DynamoDB untuk menghapus item. Nilainya adalah string dari nama tabel. Ini`table`nilai diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`DeleteItem`Operasi DynamoDB, ini harus diatur ke`DeleteItem`. Nilai ini diperlukan.

`key`

Tombol DynamoDB mewakili kunci utama item yang akan dihapus. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Nilai ini diperlukan.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan,`DeleteItem`permintaan menghapus item terlepas dari keadaan saat ini. Anda dapat menentukan apakah akan mengambil kembali item yang ada saat pemeriksaan kondisi gagal. Untuk informasi lebih lanjut tentang kondisi transaksional, lihatEkspresi kondisi transaksi. Nilai ini bersifat opsional.

Untuk`ConditionCheck`, bidang didefinisikan sebagai berikut:

`table`

Tabel DynamoDB untuk memeriksa kondisi. Nilainya adalah string dari nama tabel. Ini`table`nilai diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan`ConditionCheck`Operasi DynamoDB, ini harus diatur ke`ConditionCheck`. Nilai ini diperlukan.

`key`

Kunci DynamoDB mewakili kunci utama item untuk pemeriksaan kondisi. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Nilai ini diperlukan.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Anda dapat menentukan apakah akan mengambil kembali item yang ada saat pemeriksaan kondisi gagal. Untuk informasi lebih lanjut tentang kondisi transaksional, lihatEkspresi kondisi transaksi. Nilai ini diperlukan.

Hal-hal yang perlu diingat:

- Hanya kunci item permintaan yang dikembalikan dalam respons, jika berhasil. Urutan kunci akan sama dengan urutan item permintaan.
- Transaksi dilakukan dalamall-or-nothingcara. Jika ada item permintaan yang menyebabkan kesalahan, seluruh transaksi tidak akan dilakukan dan rincian kesalahan akan dikembalikan.
- Tidak ada dua item permintaan yang dapat menargetkan item yang sama. Jika tidak, mereka akan menyebabkanTransactionCanceledExceptionkesalahan.
- Jika kesalahan transaksi adalahTransactionCanceledException,`cancellationReasons`blok akan diisi. Jika pemeriksaan kondisi item permintaan gagaldanAnda tidak menentukan`returnValuesOnConditionCheckFailure`menjadi`false`, item yang ada di tabel akan diambil dan disimpan di`item`pada posisi yang sesuai`cancellationReasons`blok.
- `TransactWriteItems`terbatas pada 25 item permintaan.

Untuk contoh berikut fungsi request handler:

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const { authorId, postId, title, description, oldTitle, authorName } = ctx.args;
  return {
    operation: 'TransactWriteItems',
    transactItems: [
      {
        table: 'posts',
        operation: 'PutItem',
        key: util.dynamodb.toMapValues({ postId }),
        attributeValues: util.dynamodb.toMapValues({ title, description }),
        condition: util.transform.toDynamoDBConditionExpression({
          title: { eq: oldTitle },
        }),
      },
      {
        table: 'authors',
        operation: 'UpdateItem',
        key: util.dynamodb.toMapValues({ authorId }),
        update: {
          expression: 'SET authorName = :name',
          expressionValues: util.dynamodb.toMapValues({ ':name': authorName }),
        },
      },
    ],
  };
}
```

Jika transaksi berhasil, hasil pemanggilan tersedia di`ctx.result`adalah sebagai berikut:

```
{
    "keys": [
       // Key of the PutItem request
       {
           "post_id": "p1",
       },
       // Key of the UpdateItem request
       {
           "author_id": "a1"
       }
    ],
    "cancellationReasons": null
```

```
}
```

Jika transaksi gagal karena kondisi cek kegagalan`PutItem`permintaan, hasil pemanggilan tersedia di`ctx.result`adalah sebagai berikut:

```
{
    "keys": null,
    "cancellationReasons": [
       {
           "item": {
               "post_id": "p1",
               "post_title": "Actual old title",
               "post_description": "Old description"
           },
           "type": "ConditionCheckFailed",
           "message": "The condition check failed."
       },
       {
           "type": "None",
           "message": "None"
       }
    ]
}
```

The`ctx.error`berisi rincian tentang kesalahan. KuncinyakuncidanPembatalanAlasandijamin akan hadir di`ctx.result`.

Jenis sistem (pemetaan permintaan)

Saat menggunakanAWS AppSyncFungsi DynamoDB untuk memanggil tabel DynamoDB Anda,AWS AppSyncperlu mengetahui jenis setiap nilai yang akan digunakan dalam panggilan itu. Ini karena DynamoDB mendukung lebih banyak tipe primitif daripada GraphQL atau JSON (seperti set dan data biner).AWS AppSyncmembutuhkan beberapa petunjuk saat menerjemahkan antara GraphQL dan DynamoDB, jika tidak maka harus membuat beberapa asumsi tentang bagaimana data disusun dalam tabel Anda.

Untuk informasi selengkapnya tentang tipe data DynamoDB, lihat DynamoDBDeskriptor tipe datadanTipe datadokumentasi.

Nilai DynamoDB diwakili oleh objek JSON yang berisi pasangan kunci-nilai tunggal. Kunci menentukan jenis DynamoDB, dan nilai menentukan nilai itu sendiri. Dalam contoh berikut,

kuncinyaSmenunjukkan bahwa nilainya adalah string, dan nilainya`identifier`adalah nilai string itu sendiri.

```
{ "S" : "identifier" }
```

Perhatikan bahwa objek JSON tidak dapat memiliki lebih dari satu pasangan kunci-nilai. Jika lebih dari satu pasangan kunci-nilai ditentukan, objek permintaan tidak diuraikan.

Nilai DynamoDB digunakan di mana saja dalam objek permintaan di mana Anda perlu menentukan nilai. Beberapa tempat di mana Anda perlu melakukan ini termasuk:`key`dan`attributeValue`bagian, dan`expressionValues`bagian dari bagian ekspresi. Dalam contoh berikut, nilai DynamoDB String`identifier`sedang ditugaskan ke`id`bidang di`key`bagian (mungkin dalam`GetItem`permintaan objek).

```
"key" : {
   "id" : { "S" : "identifier" }
}
```

Tipe yang Didukung

AWS AppSyncmendukung skalar DynamoDB, dokumen, dan jenis set berikut:

Jenis string**S**

Nilai string tunggal. Nilai DynamoDB String dilambangkan dengan:

```
{ "S" : "some string" }
```

Contoh penggunaan adalah:

```
"key" : {
   "id" : { "S" : "some string" }
}
```

Jenis set string**SS**

Satu set nilai string. Nilai DynamoDB String Set dilambangkan dengan:

```
{ "SS" : [ "first value", "second value", ... ] }
```

Contoh penggunaan adalah:

```
"attributeValues" : {
   "phoneNumbers" : { "SS" : [ "+1 555 123 4567", "+1 555 234 5678" ] }
}
```

Jenis nomor**N**

Nilai numerik tunggal. Nilai Nomor DynamoDB dilambangkan dengan:

```
{ "N" : 1234 }
```

Contoh penggunaan adalah:

```
"expressionValues" : {
   ":expectedVersion" : { "N" : 1 }
}
```

Jenis set nomor**NS**

Satu set nilai angka. Nilai DynamoDB Number Set dilambangkan dengan:

```
{ "NS" : [ 1, 2.3, 4 ... ] }
```

Contoh penggunaan adalah:

```
"attributeValues" : {
   "sensorReadings" : { "NS" : [ 67.8, 12.2, 70 ] }
}
```

Tipe biner**B**

Nilai biner. Nilai biner DynamoDB dilambangkan dengan:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

Perhatikan bahwa nilainya sebenarnya adalah string, di mana string adalah representasi yang dikodekan base64 dari data biner.AWS AppSyncmendekode string ini kembali ke nilai binernya sebelum mengirimnya ke DynamoDB.AWS AppSyncmenggunakan skema decoding base64

seperti yang didefinisikan oleh RFC 2045: karakter apa pun yang tidak ada dalam alfabet base64 diabaikan.

Contoh penggunaan adalah:

```
"attributeValues" : {
   "binaryMessage" : { "B" : "SGVsbG8sIFdvcmxkIQo=" }
}
```

Jenis set biner**BS**

Satu set nilai biner. Nilai Set Biner DynamoDB dilambangkan dengan:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

Perhatikan bahwa nilainya sebenarnya adalah string, di mana string adalah representasi yang dikodekan base64 dari data biner.AWS AppSyncmendekode string ini kembali ke nilai binernya sebelum mengirimnya ke DynamoDB.AWS AppSyncmenggunakan skema decoding base64 seperti yang didefinisikan oleh RFC 2045: karakter apa pun yang tidak ada dalam alfabet base64 diabaikan.

Contoh penggunaan adalah:

```
"attributeValues" : {
   "binaryMessages" : { "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ] }
}
```

Jenis Boolean**BOOL**

Nilai Boolean. Nilai DynamoDB Boolean dilambangkan dengan:

```
{ "BOOL" : true }
```

Perhatikan bahwa hanya`true`dan`false`adalah nilai yang valid.

Contoh penggunaan adalah:

```
"attributeValues" : {
   "orderComplete" : { "BOOL" : false }
```

```
}
```

Jenis daftar**L**

Daftar nilai DynamoDB lain yang didukung. Nilai Daftar DynamoDB dilambangkan dengan:

```
{ "L" : [ ... ] }
```

Perhatikan bahwa nilainya adalah nilai gabungan, di mana daftar dapat berisi nol atau lebih dari nilai DynamoDB yang didukung (termasuk daftar lainnya). Daftar ini juga dapat berisi campuran dari berbagai jenis.

Contoh penggunaan adalah:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

Jenis peta**M**

Mewakili kumpulan pasangan kunci-nilai yang tidak berurutan dari nilai DynamoDB lain yang didukung. Nilai DynamoDB Map dilambangkan dengan:

```
{ "M" : { ... } }
```

Perhatikan bahwa peta dapat berisi nol atau lebih pasangan nilai kunci. Kuncinya harus berupa string, dan nilainya dapat berupa nilai DynamoDB yang didukung (termasuk peta lain). Peta juga dapat berisi campuran dari berbagai jenis.

Contoh penggunaan adalah:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

Tipe nol**`NULL`**

Sebuah nilai nol. Nilai DynamoDB Null dilambangkan dengan:

```
{ "NULL" : null }
```

Contoh penggunaan adalah:

```
"attributeValues" : {
   "phoneNumbers" : { "NULL" : null }
}
```

Untuk informasi selengkapnya tentang setiap jenis, lihatDokumentasi DynamoDB.

Jenis sistem (pemetaan respons)

Saat menerima respons dari DynamoDB,AWS AppSyncsecara otomatis mengubahnya menjadi tipe primitif GraphQL dan JSON. Setiap atribut di DynamoDB diterjemahkan dan dikembalikan dalam konteks penangan respons.

Misalnya, jika DynamoDB mengembalikan yang berikut:

```
{
    "id" : { "S" : "1234" },
    "name" : { "S" : "Nadia" },
    "age" : { "N" : 25 }
}
```

Ketika hasilnya dikembalikan dari resolver pipeline Anda,AWS AppSyncmengubahnya menjadi tipe GraphQL dan JSON sebagai:

```
{
    "id" : "1234",
    "name" : "Nadia",
    "age" : 25
}
```

Bagian ini menjelaskan bagaimanaAWS AppSyncmengkonversi skalar DynamoDB berikut, dokumen, dan jenis set:

Jenis string**S**

Nilai string tunggal. Nilai DynamoDB String dikembalikan sebagai string.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB String berikut:

```
{ "S" : "some string" }
```

AWS AppSyncmengubahnya menjadi string:

```
"some string"
```

Jenis set string**SS**

Satu set nilai string. Nilai DynamoDB String Set dikembalikan sebagai daftar string.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB String Set berikut:

```
{ "SS" : [ "first value", "second value", ... ] }
```

AWS AppSyncmengubahnya menjadi daftar string:

```
[ "+1 555 123 4567", "+1 555 234 5678" ]
```

Jenis nomor**N**

Nilai numerik tunggal. Nilai DynamoDB Number dikembalikan sebagai angka.

Misalnya, jika DynamoDB mengembalikan nilai Nomor DynamoDB berikut:

```
{ "N" : 1234 }
```

AWS AppSyncmengubahnya menjadi angka:

```
1234
```

Jenis set nomor**NS**

Satu set nilai angka. Nilai DynamoDB Number Set dikembalikan sebagai daftar angka.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Number Set berikut:

```
{ "NS" : [ 67.8, 12.2, 70 ] }
```

AWS AppSyncmengubahnya menjadi daftar angka:

```
[ 67.8, 12.2, 70 ]
```

Tipe biner**B**

Nilai biner. Nilai biner DynamoDB dikembalikan sebagai string yang berisi representasi base64 dari nilai tersebut.

Misalnya, jika DynamoDB mengembalikan nilai biner DynamoDB berikut:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

AWS AppSyncmengubahnya menjadi string yang berisi representasi base64 dari nilai:

```
"SGVsbG8sIFdvcmxkIQo="
```

Perhatikan bahwa data biner dikodekan dalam skema pengkodean base64 seperti yang ditentukan dalamRFC 4648danRFC 2045.

Jenis set biner**BS**

Satu set nilai biner. Nilai DynamoDB Binary Set dikembalikan sebagai daftar string yang berisi representasi base64 dari nilai-nilai.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Binary Set berikut:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

AWS AppSyncmengubahnya menjadi daftar string yang berisi representasi nilai base64:

```
[ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ]
```

Perhatikan bahwa data biner dikodekan dalam skema pengkodean base64 seperti yang ditentukan dalamRFC 4648danRFC 2045.

Jenis Boolean**BOOL**

Nilai Boolean. Nilai DynamoDB Boolean dikembalikan sebagai Boolean.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Boolean berikut:

```
{ "BOOL" : true }
```

AWS AppSyncmengubahnya menjadi Boolean:

```
true
```

Jenis daftar**L**

Daftar nilai DynamoDB lain yang didukung. Nilai Daftar DynamoDB dikembalikan sebagai daftar nilai, di mana setiap nilai dalam juga dikonversi.

Misalnya, jika DynamoDB mengembalikan nilai Daftar DynamoDB berikut:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

AWS AppSyncmengubahnya menjadi daftar nilai yang dikonversi:

```
[ "A string value", 1, [ "Another string value", "Even more string values!" ] ]
```

Jenis peta**M**

Kumpulan kunci/nilai dari nilai DynamoDB lain yang didukung. Nilai DynamoDB Map dikembalikan sebagai objek JSON, di mana setiap kunci/nilai juga dikonversi.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Map berikut:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

AWS AppSyncmengubahnya menjadi objek JSON:

```
{
   "someString" : "A string value",
   "someNumber" : 1,
   "stringSet"  : [ "Another string value", "Even more string values!" ]
}
```

Tipe nol**`NULL`**

Sebuah nilai nol.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Null berikut:

```
{ "NULL" : null }
```

AWS AppSyncmengubahnya menjadi nol:

```
null
```

Filter

Saat menanyakan objek di DynamoDB menggunakan`Query`dan`Scan`operasi, Anda dapat secara opsional menentukan`filter`yang mengevaluasi hasil dan hanya mengembalikan nilai yang diinginkan.

Properti filter dari`Query`atau`Scan`permintaan memiliki struktur sebagai berikut:

```
type DynamoDBExpression = {
  expression: string;
  expressionNames?: { [key: string]: string};
  expressionValues?: { [key: string]: any};
};
```

Bidang didefinisikan sebagai berikut:

`expression`

Ekspresi kueri. Untuk informasi selengkapnya tentang cara menulis ekspresi filter, lihatDynamoDBQueryFilterdanDynamoDBScanFilterdokumentasi. Bidang ini harus ditentukan.

`expressionNames`

Substitusi untuk atribut ekspresinamaplaceholder, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nama yang digunakan dalam`expression`. Nilai harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalam`expression`.

`expressionValues`

Substitusi untuk atribut ekspresinilaiplaceholder, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nilai yang digunakan dalam`expression`, dan nilainya harus berupa nilai yang diketik. Untuk informasi selengkapnya tentang cara menentukan "nilai yang diketik", lihatJenis sistem (pemetaan permintaan). Ini harus ditentukan. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nilai atribut ekspresi yang digunakan dalam`expression`.

Contoh

Contoh berikut adalah bagian filter untuk permintaan, di mana entri diambil dari DynamoDB hanya dikembalikan jika judul dimulai dengan`title`argumen.

Di sini kita menggunakan`util.transform.toDynamoDBFilterExpression`untuk secara otomatis membuat filter dari objek:

```
const filter = util.transform.toDynamoDBFilterExpression({
  title: { beginsWith: 'far away' },
});

const request = {};
request.filter = JSON.parse(filter);
```

Ini menghasilkan filter berikut:

```
{
  "filter": {
    "expression": "(begins_with(#title,:title_beginsWith))",
    "expressionNames": { "#title": "title" },
    "expressionValues": {
      ":title_beginsWith": { "S": "far away" }
```

```
    }
  }
}
```

Ekspresi kondisi

Saat Anda mengubah objek di DynamoDB dengan menggunakan`PutItem`,`UpdateItem`, dan`DeleteItem`Operasi DynamoDB, Anda dapat secara opsional menentukan ekspresi kondisi yang mengontrol apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB sebelum operasi dilakukan.

TheAWS AppSyncFungsi DynamoDB memungkinkan ekspresi kondisi ditentukan dalam`PutItem`,`UpdateItem`, dan`DeleteItem`meminta objek, dan juga strategi untuk diikuti jika kondisi gagal dan objek tidak diperbarui.

Contoh 1

Berikut ini`PutItem`objek permintaan tidak memiliki ekspresi kondisi. Akibatnya, ia menempatkan item di DynamoDB bahkan jika item dengan kunci yang sama sudah ada, sehingga menimpa item yang ada.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { foo, bar, ...values} = ctx.args
  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({foo, bar}),
    attributeValues: util.dynamodb.toMapValues(values),
  };
}
```

Contoh 2

Berikut ini`PutItem`objek memang memiliki ekspresi kondisi yang memungkinkan operasi berhasil hanya jika item dengan kunci yang sama melakukannyatidakada di DynamoDB.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { foo, bar, ...values} = ctx.args
  return {
```

```
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({foo, bar}),
    attributeValues: util.dynamodb.toMapValues(values),
    condition: { expression: "attribute_not_exists(id)" }
  };
}
```

Secara default, jika pemeriksaan kondisi gagal,AWS AppSyncFungsi DynamoDB memberikan kesalahan di`ctx.error`. Anda dapat mengembalikan kesalahan untuk mutasi dan nilai objek saat ini di DynamoDB di`data`bidang di`error`bagian dari respons GraphQL.

Namun,AWS AppSyncFungsi DynamoDB menawarkan beberapa fitur tambahan untuk membantu pengembang menangani beberapa kasus tepi umum:

- JikaAWS AppSyncFungsi DynamoDB dapat menentukan bahwa nilai saat ini di DynamoDB cocok dengan hasil yang diinginkan, ia memperlakukan operasi seolah-olah berhasil.
- Alih-alih mengembalikan kesalahan, Anda dapat mengonfigurasi fungsi untuk memanggil fungsi Lambda khusus untuk memutuskan bagaimanaAWS AppSyncFungsi DynamoDB harus menangani kegagalan.

Ini dijelaskan secara lebih rinci dalamMenangani kegagalan pemeriksaan kondisibagian.

Untuk informasi selengkapnya tentang ekspresi kondisi DynamoDB, lihatDynamoDBConditionExpressionsdokumentasi.

Menentukan suatu kondisi

ThePutItem,UpdateItem, danDeleteItempermintaan objek semua memungkinkan opsionalconditionbagian yang akan ditentukan. Jika dihilangkan, tidak ada pemeriksaan kondisi yang dilakukan. Jika ditentukan, kondisi harus benar agar operasi berhasil.

SEBUAHconditionbagian memiliki struktur sebagai berikut:

```
type ConditionCheckExpression = {
  expression: string;
  expressionNames?: { [key: string]: string};
  expressionValues?: { [key: string]: any};
  equalsIgnore?: string[];
  consistentRead?: boolean;
  conditionalCheckFailedHandler?: {
```

```
    strategy: 'Custom' | 'Reject';
    lambdaArn?: string;
  };
};
```

Bidang berikut menentukan kondisi:

`expression`

Ekspresi pembaruan itu sendiri. Untuk informasi selengkapnya tentang cara menulis ekspresi kondisi, lihatDynamoDBConditionExpressionsdokumentasi. Bidang ini harus ditentukan.

`expressionNames`

Substitusi untuk placeholder nama atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nama yang digunakan dalamekspresi, dan nilainya harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalamekspresi.

`expressionValues`

Substitusi untuk placeholder nilai atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nilai yang digunakan dalam ekspresi, dan nilainya harus berupa nilai yang diketik. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Ini harus ditentukan. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nilai atribut ekspresi yang digunakan dalam ekspresi.

Bidang yang tersisa memberi tahuAWS AppSyncFungsi DynamoDB bagaimana menangani kegagalan pemeriksaan kondisi:

`equalsIgnore`

Ketika pemeriksaan kondisi gagal saat menggunakan`PutItem`operasi,AWS AppSyncFungsi DynamoDB membandingkan item yang saat ini di DynamoDB terhadap item yang coba ditulisnya. Jika mereka sama, itu memperlakukan operasi seolah-olah berhasil. Anda dapat menggunakan`equalsIgnore`bidang untuk menentukan daftar atribut yangAWS AppSyncharus mengabaikan saat melakukan perbandingan itu. Misalnya, jika satu-satunya perbedaan adalah`version`atribut, ia memperlakukan operasi seolah-olah berhasil. Bidang ini bersifat opsional.

`consistentRead`

Ketika pemeriksaan kondisi gagal,AWS AppSyncmendapatkan nilai item saat ini dari DynamoDB menggunakan pembacaan yang sangat konsisten. Anda dapat menggunakan bidang ini untuk memberi tahuAWS AppSyncDynamoDB berfungsi untuk menggunakan pembacaan yang akhirnya konsisten sebagai gantinya. Bidang ini opsional, dan defaultnya`true`.

`conditionalCheckFailedHandler`

Bagian ini memungkinkan Anda untuk menentukan bagaimanaAWS AppSyncFungsi DynamoDB memperlakukan kegagalan pemeriksaan kondisi setelah membandingkan nilai saat ini di DynamoDB dengan hasil yang diharapkan. Bagian ini opsional. Jika dihilangkan, defaultnya adalah strategi`Reject`.

`strategy`

StrateginyaAWS AppSyncFungsi DynamoDB mengambil setelah membandingkan nilai saat ini di DynamoDB terhadap hasil yang diharapkan. Bidang ini diperlukan dan memiliki nilai yang mungkin berikut:

`Reject`

Mutasi gagal, dan kesalahan untuk mutasi dan nilai objek saat ini di DynamoDB dalam a`data`bidang di`error`bagian dari respons GraphQL.

`Custom`

TheAWS AppSyncFungsi DynamoDB memanggil fungsi Lambda khusus untuk memutuskan bagaimana menangani kegagalan pemeriksaan kondisi. Ketika`strategy`diatur ke`Custom`,`lambdaArn`field harus berisi ARN dari fungsi Lambda untuk dipanggil.

`lambdaArn`

ARN dari fungsi Lambda untuk memanggil yang menentukan bagaimanaAWS AppSyncFungsi DynamoDB harus menangani kegagalan pemeriksaan kondisi. Bidang ini hanya harus ditentukan ketika`strategy`diatur ke`Custom`. Untuk informasi selengkapnya tentang cara menggunakan fitur ini, lihatMenangani kegagalan pemeriksaan kondisi.

Menangani kegagalan pemeriksaan kondisi

Ketika pemeriksaan kondisi gagal,AWS AppSyncFungsi DynamoDB dapat meneruskan kesalahan untuk mutasi dan nilai objek saat ini dengan menggunakan`util.appendError`utilitas. Ini

menambahkan`data`bidang di`error`bagian dari respons GraphQL. Namun,AWS AppSyncFungsi DynamoDB menawarkan beberapa fitur tambahan untuk membantu pengembang menangani beberapa kasus tepi umum:

- JikaAWS AppSyncFungsi DynamoDB dapat menentukan bahwa nilai saat ini di DynamoDB cocok dengan hasil yang diinginkan, ia memperlakukan operasi seolah-olah berhasil.
- Alih-alih mengembalikan kesalahan, Anda dapat mengonfigurasi fungsi untuk memanggil fungsi Lambda khusus untuk memutuskan bagaimanaAWS AppSyncFungsi DynamoDB harus menangani kegagalan.

Diagram alur untuk proses ini adalah:

Memeriksa hasil yang diinginkan

Ketika pemeriksaan kondisi gagal,AWS AppSyncFungsi DynamoDB melakukan`GetItem`Permintaan DynamoDB untuk mendapatkan nilai item saat ini dari DynamoDB. Secara default, ini menggunakan pembacaan yang sangat konsisten, namun ini dapat dikonfigurasi menggunakan`consistentRead`bidang di`condition`memblokir dan membandingkannya dengan hasil yang diharapkan:

- Untuk`PutItem`operasi,AWS AppSyncFungsi DynamoDB membandingkan nilai saat ini dengan nilai yang coba ditulisnya, tidak termasuk atribut apa pun yang tercantum`equalsIgnore`dari perbandingan. Jika itemnya sama, ia memperlakukan operasi sebagai berhasil dan mengembalikan item yang diambil dari DynamoDB. Jika tidak, ia mengikuti strategi yang dikonfigurasi.

  Misalnya, jika`PutItem`objek permintaan tampak seperti berikut:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { id, name, version} = ctx.args
  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({foo, bar}),
    attributeValues: util.dynamodb.toMapValues({ name, version: version+1 }),
    condition: {
      expression: "version = :expectedVersion",
      expressionValues: util.dynamodb.toMapValues({':expectedVersion': version}),
      equalsIgnore: ['version']
```

```
    }
  };
}
```

Dan item yang saat ini di DynamoDB tampak seperti berikut:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

TheAWS AppSyncFungsi DynamoDB akan membandingkan item yang dicoba tulis dengan nilai saat ini, lihat bahwa satu-satunya perbedaan adalah`version`bidang, tetapi karena dikonfigurasi untuk mengabaikan`version`bidang, ia memperlakukan operasi sebagai berhasil dan mengembalikan item yang diambil dari DynamoDB.

- Untuk`DeleteItem`operasi,AWS AppSyncFungsi DynamoDB memeriksa untuk memverifikasi bahwa item dikembalikan dari DynamoDB. Jika tidak ada barang yang dikembalikan, itu memperlakukan operasi sebagai berhasil. Jika tidak, ia mengikuti strategi yang dikonfigurasi.
- Untuk`UpdateItem`operasi,AWS AppSyncFungsi DynamoDB tidak memiliki informasi yang cukup untuk menentukan apakah item yang saat ini di DynamoDB cocok dengan hasil yang diharapkan, dan oleh karena itu mengikuti strategi yang dikonfigurasi.

Jika keadaan objek saat ini di DynamoDB berbeda dari hasil yang diharapkan,AWS AppSyncFungsi DynamoDB mengikuti strategi yang dikonfigurasi, untuk menolak mutasi atau memanggil fungsi Lambda untuk menentukan apa yang harus dilakukan selanjutnya.

Mengikuti strategi “tolak”

Saat mengikuti`Reject`strategi,AWS AppSyncFungsi DynamoDB mengembalikan kesalahan untuk mutasi, dan nilai saat ini dari objek di DynamoDB juga dikembalikan`data`bidang die`error`bagian dari respons GraphQL. Item yang dikembalikan dari DynamoDB dimasukkan melalui penangan respons fungsi untuk menerjemahkannya ke dalam format yang diharapkan klien, dan difilter oleh set pilihan.

Misalnya, diberikan permintaan mutasi berikut:

```
mutation {
    updatePerson(id: 1, name: "Steve", expectedVersion: 1) {
        Name
```

```
        theVersion
    }
}
```

Jika item yang dikembalikan dari DynamoDB terlihat seperti berikut:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

Dan fungsi respon handler terlihat seperti berikut:

```
import { util } from '@aws-appsync/utils';
export function response(ctx) {
  const { version, ...values } = ctx.result;
  const result = { ...values, theVersion: version };
  if (ctx.error) {
    if (error) {
      return util.appendError(error.message, error.type, result, null);
    }
  }
  return result
}
```

Respons GraphQL terlihat seperti berikut:

```
{
  "data": null,
  "errors": [
    {
      "message": "The conditional request failed (Service: AmazonDynamoDBv2;
 Status Code: 400; Error Code: ConditionalCheckFailedException; Request ID:
 ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ)"
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      "data": {
        "Name": "Steve",
        "theVersion": 8
      },
      ...
    }
```

```
  ]
}
```

Juga, jika ada bidang dalam objek yang dikembalikan diisi oleh resolver lain dan mutasi telah berhasil, mereka tidak akan diselesaikan ketika objek dikembalikan dalam`error`bagian.

Mengikuti strategi “kustom”

Saat mengikuti`Custom`strategi,AWS AppSyncFungsi DynamoDB memanggil fungsi Lambda untuk memutuskan apa yang harus dilakukan selanjutnya. Fungsi Lambda memilih salah satu opsi berikut:

- `reject`mutasi. Ini memberitahuAWS AppSyncFungsi DynamoDB untuk berperilaku seolah-olah strategi yang dikonfigurasi`Reject`, mengembalikan kesalahan untuk mutasi dan nilai objek saat ini di DynamoDB seperti yang dijelaskan di bagian sebelumnya.
- `discard`mutasi. Ini memberitahuAWS AppSyncDynamoDB berfungsi untuk diam-diam mengabaikan kegagalan pemeriksaan kondisi dan mengembalikan nilai dalam DynamoDB.
- `retry`mutasi. Ini memberitahuAWS AppSyncDynamoDB berfungsi untuk mencoba kembali mutasi dengan objek permintaan baru.

Permintaan doa Lambda

TheAWS AppSyncFungsi DynamoDB memanggil fungsi Lambda yang ditentukan dalam`lambdaArn`. Ia menggunakan yang sama`service-role-arn`dikonfigurasi pada sumber data. Muatan doa memiliki struktur berikut:

```
{
    "arguments": { ... },
    "requestMapping": {... },
    "currentValue": { ... },
    "resolver": { ... },
    "identity": { ... }
}
```

Bidang didefinisikan sebagai berikut:

`arguments`

Argumen dari mutasi GraphQL. Ini sama dengan argumen yang tersedia untuk objek permintaan di`context.arguments`.

`requestMapping`

Objek permintaan untuk operasi ini.

`currentValue`

Nilai objek saat ini di DynamoDB.

`resolver`

Informasi tentangAWS AppSyncresolver atau fungsi.

`identity`

Informasi tentang penelepon. Ini sama dengan informasi identitas yang tersedia untuk objek permintaan di`context.identity`.

Contoh lengkap dari payload:

```
{
    "arguments": {
        "id": "1",
        "name": "Steve",
        "expectedVersion": 1
    },
    "requestMapping": {
        "version" : "2017-02-28",
        "operation" : "PutItem",
        "key" : {
           "id" : { "S" : "1" }
        },
        "attributeValues" : {
           "name" : { "S" : "Steve" },
           "version" : { "N" : 2 }
        },
        "condition" : {
           "expression" : "version = :expectedVersion",
           "expressionValues" : {
               ":expectedVersion" : { "N" : 1 }
           },
           "equalsIgnore": [ "version" ]
        }
    },
    "currentValue": {
        "id" : { "S" : "1" },
```

```
        "name" : { "S" : "Steve" },
        "version" : { "N" : 8 }
    },
    "resolver": {
        "tableName": "People",
        "awsRegion": "us-west-2",
        "parentType": "Mutation",
        "field": "updatePerson",
        "outputType": "Person"
    },
    "identity": {
        "accountId": "123456789012",
        "sourceIp": "x.x.x.x",
        "user": "AIDAAAAAAAAAAAAAAAAA",
        "userArn": "arn:aws:iam::123456789012:user/appsync"
    }
}
```

Tanggapan Doa Lambda

Fungsi Lambda dapat memeriksa payload pemanggilan dan menerapkan logika bisnis apa pun untuk memutuskan bagaimanaAWS AppSyncFungsi DynamoDB harus menangani kegagalan. Ada tiga opsi untuk menangani kegagalan pemeriksaan kondisi:

- `reject`mutasi. Payload respons untuk opsi ini harus memiliki struktur ini:

  ```
  {
      "action": "reject"
  }
  ```

  Ini memberitahuAWS AppSyncFungsi DynamoDB untuk berperilaku seolah-olah strategi yang dikonfigurasi`Reject`, mengembalikan kesalahan untuk mutasi dan nilai objek saat ini di DynamoDB, seperti yang dijelaskan pada bagian di atas.
- `discard`mutasi. Payload respons untuk opsi ini harus memiliki struktur ini:

  ```
  {
      "action": "discard"
  }
  ```

  Ini memberitahuAWS AppSyncDynamoDB berfungsi untuk diam-diam mengabaikan kegagalan pemeriksaan kondisi dan mengembalikan nilai dalam DynamoDB.

- `retry`mutasi. Payload respons untuk opsi ini harus memiliki struktur ini:

```
{
    "action": "retry",
    "retryMapping": { ... }
}
```

Ini memberitahuAWS AppSyncDynamoDB berfungsi untuk mencoba kembali mutasi dengan objek permintaan baru. Struktur dari`retryMapping`bagian tergantung pada operasi DynamoDB, dan merupakan bagian dari objek permintaan penuh untuk operasi itu.

Untuk`PutItem`,`retryMapping`bagian memiliki struktur sebagai berikut. Untuk deskripsi`attributeValues`lapangan, lihatPutItem.

```
{
    "attributeValues": { ... },
    "condition": {
        "equalsIgnore" = [ ... ],
        "consistentRead" = true
    }
}
```

Untuk`UpdateItem`,`retryMapping`bagian memiliki struktur sebagai berikut. Untuk deskripsi`update`bagian, lihatUpdateItem.

```
{
    "update" : {
        "expression" : "someExpression"
        "expressionNames" : {
            "#foo" : "foo"
        },
        "expressionValues" : {
            ":bar" : ... typed value
        }
    },
    "condition": {
        "consistentRead" = true
    }
}
```

UntukDeleteItem,retryMappingbagian memiliki struktur sebagai berikut.

```
{
    "condition": {
        "consistentRead" = true
    }
}
```

Tidak ada cara untuk menentukan operasi atau kunci yang berbeda untuk dikerjakan. TheAWS AppSyncFungsi DynamoDB hanya memungkinkan percobaan ulang dari operasi yang sama pada objek yang sama. Juga,conditionbagian tidak mengizinkan aconditionalCheckFailedHandleruntuk ditentukan. Jika percobaan ulang gagal,AWS AppSyncFungsi DynamoDB mengikutiRejectstrategi.

Berikut adalah contoh fungsi Lambda untuk menangani kegagalanPutItempermintaan. Logika bisnis melihat siapa yang membuat panggilan. Jika itu dibuat olehjeffTheAdmin, ia mencoba ulang permintaan, memperbaruiversiondanexpectedVersiondari item yang saat ini ada di DynamoDB. Jika tidak, ia menolak mutasi.

```
exports.handler = (event, context, callback) => {
    console.log("Event: "+ JSON.stringify(event));

    // Business logic goes here.

    var response;
    if ( event.identity.user == "jeffTheAdmin" ) {
        response = {
            "action" : "retry",
            "retryMapping" : {
                "attributeValues" : event.requestMapping.attributeValues,
                "condition" : {
                    "expression" : event.requestMapping.condition.expression,
                    "expressionValues" :
 event.requestMapping.condition.expressionValues
                }
            }
        }
        response.retryMapping.attributeValues.version = { "N" :
 event.currentValue.version.N + 1 }
```

```
        response.retryMapping.condition.expressionValues[':expectedVersion'] =
 event.currentValue.version

    } else {
        response = { "action" : "reject" }
    }

    console.log("Response: "+ JSON.stringify(response))
    callback(null, response)
};
```

Ekspresi kondisi transaksi

Ekspresi kondisi transaksi tersedia dalam permintaan keempat jenis operasi di`TransactWriteItems`, yaitu,`PutItem`,`DeleteItem`,`UpdateItem`, dan`ConditionCheck`.

Untuk`PutItem`,`DeleteItem`, dan`UpdateItem`, ekspresi kondisi transaksi adalah opsional. Untuk`ConditionCheck`, ekspresi kondisi transaksi diperlukan.

Contoh 1

Transaksional berikut`DeleteItem`fungsi permintaan handler tidak memiliki ekspresi kondisi. Akibatnya, ia menghapus item di DynamoDB.

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const { postId } = ctx.args;
  return {
    operation: 'TransactWriteItems',
    transactItems: [
      {
        table: 'posts',
        operation: 'DeleteItem',
        key: util.dynamodb.toMapValues({ postId }),
      }
    ],
  };
}
```

Contoh 2

Transaksional berikut`DeleteItem`function request handler memang memiliki ekspresi kondisi transaksi yang memungkinkan operasi berhasil hanya jika penulis posting itu sama dengan nama tertentu.

```
import { util } from '@aws-appsync/utils';

export function request(ctx) {
  const { postId, authorName} = ctx.args;
  return {
    operation: 'TransactWriteItems',
    transactItems: [
      {
        table: 'posts',
        operation: 'DeleteItem',
        key: util.dynamodb.toMapValues({ postId }),
        condition: util.transform.toDynamoDBConditionExpression({
          authorName: { eq: authorName },
        }),
      }
    ],
  };
}
```

Jika pemeriksaan kondisi gagal, itu akan menyebabkan`TransactionCanceledException`dan detail kesalahan akan dikembalikan`ctx.result.cancellationReasons`. Perhatikan bahwa secara default, item lama di DynamoDB yang membuat pemeriksaan kondisi gagal akan dikembalikan`ctx.result.cancellationReasons`.

Menentukan suatu kondisi

The`PutItem`,`UpdateItem`, dan`DeleteItem`permintaan objek semua memungkinkan opsional`condition`bagian yang akan ditentukan. Jika dihilangkan, tidak ada pemeriksaan kondisi yang dilakukan. Jika ditentukan, kondisi harus benar agar operasi berhasil. The`ConditionCheck`harus memiliki`condition`bagian yang akan ditentukan. Syaratnya harus benar agar seluruh transaksi berhasil.

SEBUAH`condition`bagian memiliki struktur sebagai berikut:

```
type TransactConditionCheckExpression = {
```

```
  expression: string;
  expressionNames?: { [key: string]: string };
  expressionValues?: { [key: string]: string };
  returnValuesOnConditionCheckFailure: boolean;
};
```

Bidang berikut menentukan kondisi:

`expression`

Ekspresi pembaruan itu sendiri. Untuk informasi selengkapnya tentang cara menulis ekspresi kondisi, lihatDynamoDBConditionExpressionsdokumentasi. Bidang ini harus ditentukan.

`expressionNames`

Substitusi untuk placeholder nama atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nama yang digunakan dalamekspresi, dan nilainya harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalamekspresi.

`expressionValues`

Substitusi untuk placeholder nilai atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nilai yang digunakan dalam ekspresi, dan nilainya harus berupa nilai yang diketik. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihatJenis sistem (pemetaan permintaan). Ini harus ditentukan. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nilai atribut ekspresi yang digunakan dalam ekspresi.

`returnValuesOnConditionCheckFailure`

Tentukan apakah akan mengambil item di DynamoDB kembali ketika pemeriksaan kondisi gagal. Item yang diambil akan berada di`ctx.result.cancellationReasons[<index>].item`, dimana`<index>`adalah indeks item permintaan yang gagal dalam pemeriksaan kondisi. Nilai ini default ke true.

Proyeksi

Saat membaca objek di DynamoDB menggunakan`GetItem`,`Scan`,`Query`,`BatchGetItem`, dan`TransactGetItems`operasi, Anda dapat secara opsional menentukan proyeksi yang

mengidentifikasi atribut yang Anda inginkan. Properti proyeksi memiliki struktur berikut, yang mirip dengan filter:

```
type DynamoDBExpression = {
  expression: string;
  expressionNames?: { [key: string]: string}
};
```

Bidang didefinisikan sebagai berikut:

`expression`

Ekspresi proyeksi, yang merupakan string. Untuk mengambil atribut tunggal, tentukan namanya. Untuk beberapa atribut, nama harus berupa nilai yang dipisahkan koma. Untuk informasi lebih lanjut tentang menulis ekspresi proyeksi, lihatEkspresi proyeksi DynamoDBdokumentasi. Bidang ini wajib diisi.

`expressionNames`

Substitusi untuk atribut ekspresinamaplaceholder dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nama yang digunakan dalam`expression`. Nilai harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional dan hanya boleh diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalam`expression`. Untuk informasi lebih lanjut tentang`expressionNames`, lihatDokumentasi DynamoDB.

Contoh 1

Contoh berikut adalah bagian proyeksi untukJavaScriptfungsi di mana hanya atribut`author`dan`id`dikembalikan dari DynamoDB:

```
projection : {
    expression : "#author, id",
    expressionNames : {
        "#author" : "author"
    }
}
```

> **Tip**
>
> Anda dapat mengakses set pemilihan permintaan GraphQL Anda menggunakanselectionSetList. Bidang ini memungkinkan Anda untuk membingkai ekspresi proyeksi Anda secara dinamis sesuai dengan kebutuhan Anda.

> **Note**
>
> Saat menggunakan ekspresi proyeksi dengan`Query`dan`Scan`operasi, nilai untuk`select`harus`SPECIFIC_ATTRIBUTES`. Untuk informasi lebih lanjut, lihatDokumentasi DynamoDB.

JavaScript referensi fungsi resolver untuk OpenSearch

AWS AppSync Resolver untuk Amazon OpenSearch Service memungkinkan Anda menggunakan GraphQL untuk menyimpan dan mengambil data dalam domain OpenSearch Layanan yang ada di akun Anda. Resolver ini bekerja dengan memungkinkan Anda memetakan permintaan GraphQL yang masuk ke dalam permintaan OpenSearch Layanan, dan kemudian memetakan respons OpenSearch Layanan kembali ke GraphQL. Bagian ini menjelaskan permintaan fungsi dan penangan respons untuk operasi OpenSearch Layanan yang didukung.

Permintaan

Sebagian besar objek permintaan OpenSearch Layanan memiliki struktur umum di mana hanya beberapa potong berubah. Contoh berikut menjalankan pencarian terhadap domain OpenSearch Layanan, di mana dokumen adalah jenis`post` dan diindeks di bawah`id`. Parameter pencarian didefinisikan di`body` bagian, dengan banyak klausa kueri umum didefinisikan di`query` lapangan. Contoh ini akan mencari dokumen yang berisi`"Nadia"`,`"Bailey"`, atau, atau keduanya, di`author` bidang dokumen:

```
export function request(ctx) {
  return {
    operation: 'GET',
    path: '/id/post/_search',
    params: {
      headers: {},
```

```
      queryString: {},
      body: {
        from: 0,
        size: 50,
        query: {
          bool: {
            should: [
              { match: { author: 'Nadia' } },
              { match: { author: 'Bailey' } },
            ],
          },
        },
      },
    },
  };
}
```

Response

Seperti sumber data lainnya, OpenSearch Service mengirimkan responsAWS AppSync yang perlu dikonversi ke GraphQL.

Sebagian besar kueri GraphQL mencari`_source` bidang dari respons OpenSearch Layanan. Karena Anda dapat melakukan pencarian untuk mengembalikan dokumen individual atau daftar dokumen, ada dua pola respons umum yang digunakan dalam OpenSearch Service:

Daftar Hasil

```
export function response(ctx) {
  const entries = [];
  for (const entry of ctx.result.hits.hits) {
    entries.push(entry['_source']);
  }
  return entries;
}
```

Item Individu

```
export function response(ctx) {
  return ctx.result['_source']
}
```

operationbidang

(Permintaan handler saja)

Metode HTTP atau kata kerja (GET, POST, PUT, HEAD atau DELETE) yangAWS AppSync mengirimkan ke domain OpenSearch Service. Baik kunci dan nilainya harus berupa string.

```
"operation" : "PUT"
```

pathbidang

(Permintaan handler saja)

Jalur pencarian untuk permintaan OpenSearch Layanan dariAWS AppSync. Ini membentuk URL untuk kata kerja HTTP operasi. Baik kunci dan nilainya harus berupa string.

```
"path" : "/indexname/type"

"path" : "/indexname/type/_search"
```

Saat penangan permintaan dievaluasi, jalur ini dikirim sebagai bagian dari permintaan HTTP, termasuk domain OpenSearch Layanan. Misalnya, contoh sebelumnya mungkin diterjemahkan ke:

```
GET https://opensearch-domain-name.REGION.es.amazonaws.com/indexname/type/_search
```

paramsbidang

(Permintaan handler saja)

Digunakan untuk menentukan tindakan apa yang dilakukan pencarian Anda, paling umum dengan menetapkan nilai kueri di dalam tubuh. Namun, ada beberapa kemampuan lain yang dapat dikonfigurasi, seperti pemformatan tanggapan.

- header

  Informasi header, sebagai pasangan nilai kunci. Baik kunci dan nilainya harus berupa string. Misalnya:

  ```
  "headers" : {
  ```

```
    "Content-Type" : "application/json"
}
```

> **Note**
>
> AWS AppSync saat ini hanya mendukung JSON sebagai`Content-Type`.

- QueryString

  Pasangan nilai kunci yang menentukan opsi umum, seperti pemformatan kode untuk respons JSON. Baik kunci dan nilainya harus berupa string. Misalnya, jika Anda ingin mendapatkan JSON yang diformat dengan cukup, Anda akan menggunakan:

```
"queryString" : {
    "pretty" : "true"
}
```

- tubuh

  Ini adalah bagian utama dari permintaan Anda, memungkinkanAWS AppSync untuk menyusun permintaan pencarian yang terbentuk dengan baik ke domain OpenSearch Layanan Anda. Kuncinya harus berupa string yang terdiri dari sebuah objek. Beberapa demonstrasi ditunjukkan di bawah ini.

Contoh 1

Kembalikan semua dokumen dengan pencocokan kota “seattle”:

```
export function request(ctx) {
  return {
    operation: 'GET',
    path: '/id/post/_search',
    params: {
      headers: {},
      queryString: {},
      body: { from: 0, size: 50, query: { match: { city: 'seattle' } } },
    },
  };
}
```

Contoh 2

Kembalikan semua dokumen yang cocok dengan “washington” sebagai kota atau negara bagian:

```
export function request(ctx) {
  return {
    operation: 'GET',
    path: '/id/post/_search',
    params: {
      headers: {},
      queryString: {},
      body: {
        from: 0,
        size: 50,
        query: {
          multi_match: { query: 'washington', fields: ['city', 'state'] },
        },
      },
    },
  };
}
```

Melewati variabel

(Permintaan handler saja)

Anda juga dapat melewati variabel sebagai bagian dari evaluasi di handler permintaan Anda. Misalnya, anggap Anda memiliki kueri GraphQL seperti berikut ini:

```
query {
    searchForState(state: "washington"){
        ...
    }
}
```

Fungsi permintaan handler bisa menjadi berikut:

```
export function request(ctx) {
  return {
    operation: 'GET',
    path: '/id/post/_search',
```

```
    params: {
      headers: {},
      queryString: {},
      body: {
        from: 0,
        size: 50,
        query: {
          multi_match: { query: ctx.args.state, fields: ['city', 'state'] },
        },
      },
    },
  };
}
```

JavaScript referensi fungsi resolver untuk Lambda

Anda dapat menggunakanAWS AppSync fungsiAWS Lambda untuk membentuk permintaan dariAWS AppSync fungsi Lambda yang terletak di akun Anda, dan respons dari fungsi Lambda Anda kembali keAWS AppSync. Anda juga dapat menentukan jenis operasi yang akan dilakukan di objek permintaan Anda. Bagian ini menjelaskan permintaan untuk operasi Lambda yang didukung.

Permintaan objek

Objek permintaan Lambda cukup sederhana dan memungkinkan informasi konteks sebanyak mungkin untuk diteruskan ke fungsi Lambda Anda.

```
type LambdaRequest = {
  operation: 'Invoke' | 'BatchInvoke';
  payload: any;
};
```

Berikut adalah contoh di mana kita melewati`field` nilai dan argumen bidang GraphQL dari konteks.

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    payload: { field: 'getPost', arguments: ctx.args },
  };
}
```

Seluruh dokumen pemetaan dilewatkan sebagai input ke fungsi Lambda Anda, sehingga contoh sebelumnya sekarang akan terlihat seperti berikut:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments": {
            "id": "postId1"
        }
    }
}
```

Operasi

Sumber data Lambda memungkinkan Anda menentukan dua operasi:`Invoke` dan`BatchInvoke`. `Invoke`Operasi memungkinkanAWS AppSync tahu untuk memanggil fungsi Lambda Anda untuk setiap resolver bidang GraphQL. `BatchInvoke`menginstruksikanAWS AppSync untuk permintaan batch untuk bidang GraphQL saat ini.

`operation`diperlukan.

Sebab`Invoke`, permintaan yang diselesaikan sama persis dengan payload input fungsi Lambda. Jadi contoh berikut permintaan handler:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    payload: { field: 'getPost', arguments: ctx.args },
  };
}
```

diselesaikan dan diteruskan ke fungsi Lambda, sebagai berikut:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "arguments": {
            "id": "postId1"
```

```
        }
    }
}
```

Untuk`BatchInvoke`, permintaan diterapkan untuk setiap resolver bidang dalam batch. Untuk keringkasan,AWS AppSync menggabungkan semua`payload` nilai permintaan ke dalam daftar di bawah satu objek yang cocok dengan objek permintaan.

Contoh permintaan handler berikut menunjukkan penggabungan:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    payload: ctx,
  };
}
```

Permintaan ini dievaluasi dan diselesaikan ke dalam dokumen pemetaan berikut:

```
{
    "version": "2018-05-29",
    "operation": "BatchInvoke",
    "payload": [
        {...}, // context for batch item 1
        {...}, // context for batch item 2
        {...}  // context for batch item 3
    ]
}
```

di mana setiap elemen dari`payload` daftar sesuai dengan item batch tunggal. Fungsi Lambda juga diharapkan untuk mengembalikan respons berbentuk daftar, cocok dengan urutan item yang dikirim dalam permintaan, sebagai berikut:

```
[
    { "data": {...}, "errorMessage": null, "errorType": null }, // result for batch
 item 1
    { "data": {...}, "errorMessage": null, "errorType": null }, // result for batch
 item 2
    { "data": {...}, "errorMessage": null, "errorType": null }  // result for batch
 item 3
]
```

`operation`diperlukan.

Muatan

`payload`Bidang adalah wadah yang dapat Anda gunakan untuk meneruskan data apa pun ke fungsi Lambda.

Jika`operation` bidang diatur ke`BatchInvoke`,AWS AppSync membungkus`payload` nilai-nilai yang ada ke dalam daftar.

`payload`adalah opsional.

Objek respons

Seperti sumber data lainnya, fungsi Lambda Anda mengirimkan responsAWS AppSync yang harus dikonversi ke tipe GraphQL.

Hasil dari fungsi Lambda diatur pada properti`context` hasil (`context.result`).

Jika bentuk respons fungsi Lambda Anda sama persis dengan bentuk tipe GraphQL, Anda dapat meneruskan respons menggunakan handler respons fungsi berikut:

```
export function response(ctx) {
  return ctx.result
}
```

Tidak ada batasan bidang atau bentuk yang diperlukan yang berlaku untuk objek respons. Namun, karena GraphQL diketik dengan kuat, respons yang diselesaikan harus sesuai dengan tipe GraphQL yang diharapkan.

Respons batch fungsi Lambda

Jika`operation` bidang diatur ke`BatchInvoke`,AWS AppSync mengharapkan daftar item kembali dari fungsi Lambda. Agar dapatAWS AppSync memetakan setiap hasil kembali ke item permintaan asli, daftar respons harus sesuai dengan ukuran dan urutan. Hal ini OK untuk memiliki`null` item dalam daftar respon;`ctx.result` diatur ke null sesuai.

JavaScript referensi fungsi resolver untuk sumber EventBridge data

Permintaan dan respons fungsiAWS AppSync resolver yang digunakan dengan sumber EventBridge data memungkinkan Anda mengirim peristiwa khusus ke EventBridge bus Amazon.

Permintaan

Penangan permintaan memungkinkan Anda mengirim beberapa acara khusus ke bus EventBridge acara:

```
export function request(ctx) {
  return {
    "operation" : "PutEvents",
    "events" : [{}]
  }
}
```

EventBridge `PutEvents`Permintaan memiliki definisi tipe berikut:

```
type PutEventsRequest = {
  operation: 'PutEvents'
  events: {
    source: string
    detail: { [key: string]: any }
    detailType: string
    resources?: string[]
    time?: string // RFC3339 Timestamp format
  }[]
}
```

Response

Jika`PutEvents` operasi berhasil, respon dari EventBridge termasuk dalam`ctx.result`:

```
export function response(ctx) {
  if(ctx.error)
    util.error(ctx.error.message, ctx.error.type, ctx.result)
  else
    return ctx.result
}
```

Kesalahan yang terjadi saat melakukan`PutEvents` operasi seperti`InternalExceptions` atau`Timeouts` akan muncul di`ctx.error`. Untuk daftar EventBridge kesalahan umum, lihat referensi kesalahanEventBridge umum.

`result`Akan memiliki definisi tipe berikut:

```
type PutEventsResult = {
  Entries: {
    ErrorCode: string
    ErrorMessage: string
    EventId: string
  }[]
  FailedEntry: number
}
```

- Entri

  Hasil acara yang tertelan, baik sukses maupun tidak berhasil. Jika konsumsi berhasil, entri memilikiEventID di dalamnya. Jika tidak, Anda dapat menggunakanErrorCode danErrorMessage untuk mengidentifikasi masalah dengan entri.

  Untuk setiap record, indeks dari elemen respon adalah sama dengan indeks dalam array permintaan.

- FailedEntryCount

  Jumlah entri yang gagal. Nilai ini direpresentasikan sebagai integer.

Untuk informasi lebih lanjut tentang responPutEvents, lihat PutEvents.

Contoh respon sampel 1

Contoh berikut adalahPutEvents operasi dengan dua peristiwa sukses:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        }
    ],
    "FailedEntryCount" : 0
}
```

Contoh respon sampel 2

Contoh berikut adalah`PutEvents` operasi dengan tiga peristiwa, dua keberhasilan dan satu gagal:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        },
        {
            "ErrorCode" : "SampleErrorCode",
            "ErrorMessage" : "Sample Error Message"
        }
    ],
    "FailedEntryCount" : 1
}
```

PutEventsbidang

- Versi

  Umum untuk semua template pemetaan permintaan,`version` bidang mendefinisikan versi yang digunakan template. Bidang ini wajib diisi. Nilai`2018-05-29` adalah satu-satunya versi yang didukung untuk template EventBridge pemetaan.

- Operasi

  Satu-satunya operasi yang didukung adalah`PutEvents`. Operasi ini memungkinkan Anda untuk menambahkan acara khusus ke bus acara Anda.

- Event

  Array peristiwa yang akan ditambahkan ke bus acara. Array ini harus memiliki alokasi 1 - 10 item.

  `Event`Objek berisi bidang-bidang berikut:

  - `"source"`: String yang mendefinisikan sumber acara.
  - `"detail"`Objek JSON yang dapat Anda gunakan untuk melampirkan informasi tentang peristiwa. Bidang ini dapat berupa peta kosong (`{ }`).
  - `"detailType`: String yang mengidentifikasi tipe peristiwa.

- `"resources"`: Array JSON yang mengidentifikasi sumber daya yang terlibat dalam peristiwa tersebut. Bidang ini bisa menjadi array kosong.
- `"time"`: Acara timestamp disediakan sebagai string. Ini harus mengikuti format timestamp RFC3339.

Cuplikan di bawah ini adalah beberapa contoh`Event` objek yang valid:

Contoh 1

```
{
    "source" : "source1",
    "detail" : {
        "key1" : [1,2,3,4],
        "key2" : "strval"
    },
    "detailType" : "sampleDetailType",
    "resources" : ["Resouce1", "Resource2"],
    "time" : "2022-01-10T05:00:10Z"
}
```

Contoh 2

```
{
    "source" : "source1",
    "detail" : {},
    "detailType" : "sampleDetailType"
}
```

Contoh 3

```
{
    "source" : "source1",
    "detail" : {
        "key1" : 1200
    },
    "detailType" : "sampleDetailType",
    "resources" : []
}
```

JavaScript Referensi fungsi resolver untuk sumber data None

Permintaan dan respons fungsiAWS AppSync resolver dengan sumber data tipe None memungkinkan Anda membentuk permintaan untuk operasiAWS AppSync lokal.

Permintaan

Permintaan handler dapat sederhana dan memungkinkan Anda untuk lulus informasi kontekstual sebanyak mungkin melaluipayload lapangan.

```
type NONERequest = {
  payload: any;
};
```

Berikut adalah contoh di mana argumen lapangan diteruskan ke payload:

```
export function request(ctx) {
  return {
    payload: context.args
  };
}
```

Nilaipayload lapangan akan diteruskan ke handler respon fungsi dan tersedia dicontext.result.

Muatan

payloadBidang adalah wadah yang dapat digunakan untuk lulus data yang kemudian dibuat tersedia untuk handler fungsi respon.

payloadBidang bersifat opsional.

Response

Karena tidak ada sumber data, nilaipayload lapangan akan diteruskan ke handler respon fungsi dan diatur padacontext.result properti.

Jika bentuk nilaipayload bidang sama persis dengan bentuk tipe GraphQL, Anda dapat meneruskan respons menggunakan handler respons berikut:

```
export function request(ctx) {
  return ctx.result;
}
```

Tidak ada batasan kolom atau bentuk yang diperlukan yang berlaku untuk respons pengembalian. Namun, karena GraphQL diketik dengan kuat, respons yang diselesaikan harus sesuai dengan tipe GraphQL yang diharapkan.

JavaScript referensi fungsi resolver untuk HTTP

Fungsi resolver AWS AppSync HTTP memungkinkan Anda mengirim permintaan dari titik akhir HTTP apa pun, dan tanggapan dari titik akhir HTTP Anda kembali AWS AppSync ke. AWS AppSync Dengan penangan permintaan Anda, Anda dapat memberikan petunjuk AWS AppSync tentang sifat operasi yang akan dipanggil. Bagian ini menjelaskan konfigurasi yang berbeda untuk resolver HTTP yang didukung.

Permintaan

```
type HTTPRequest = {
  method: 'PUT' | 'POST' | 'GET' | 'DELETE' | 'PATCH';
  params?: {
    query?: { [key: string]: any };
    headers?: { [key: string]: string };
    body?: any;
  };
  resourcePath: string;
};
```

Cuplikan berikut adalah contoh permintaan HTTP POST, dengan `text/plain` badan:

```
export function request(ctx) {
  return {
    method: 'POST',
    params: {
      headers: { 'Content-Type': 'text/plain' },
      body: 'this is an example of text body',
    },
    resourcePath: '/',
  };
```

```
}
```

Metode

Minta handler saja

Metode HTTP atau kata kerja (GET, POST, PUT, PATCH, atau DELETE) yang AWS AppSync mengirim ke titik akhir HTTP.

```
"method": "PUT"
```

ResourcePath

Minta handler saja

Jalur sumber daya yang ingin Anda akses. Seiring dengan titik akhir di sumber data HTTP, jalur sumber daya membentuk URL tempat AWS AppSync layanan membuat permintaan.

```
"resourcePath": "/v1/users"
```

Ketika permintaan dievaluasi, jalur ini dikirim sebagai bagian dari permintaan HTTP, termasuk titik akhir HTTP. Misalnya, contoh sebelumnya mungkin menerjemahkan ke yang berikut:

```
PUT <endpoint>/v1/users
```

Bidang Params

Minta handler saja

Digunakan untuk menentukan tindakan apa yang dilakukan penelusuran Anda, paling sering dengan menetapkan nilai kueri di dalam badan. Namun, ada beberapa kemampuan lain yang dapat dikonfigurasi, seperti pemformatan respons.

header

Informasi header, sebagai pasangan kunci-nilai. Baik kunci dan nilainya harus berupa string.

Sebagai contoh:

```
"headers" : {
    "Content-Type" : "application/json"
}
```

`Content-Type`Header yang didukung saat ini adalah:

```
text/*
application/xml
application/json
application/soap+xml
application/x-amz-json-1.0
application/x-amz-json-1.1
application/vnd.api+json
application/x-ndjson
```

Anda tidak dapat mengatur header HTTP berikut:

```
HOST
CONNECTION
USER-AGENT
EXPECTATION
TRANSFER_ENCODING
CONTENT_LENGTH
```

kueri

Pasangan nilai kunci yang menentukan opsi umum, seperti pemformatan kode untuk respons JSON. Baik kunci dan nilainya harus berupa string. Contoh berikut menunjukkan bagaimana Anda dapat mengirim string query sebagai`?type=json`:

```
"query" : {
    "type" : "json"
}
```

tubuh

Tubuh berisi badan permintaan HTTP yang Anda pilih untuk disetel. Badan permintaan selalu berupa string yang dikodekan UTF-8 kecuali jenis konten menentukan charset.

```
"body":"body string"
```

Respons

Lihat contoh di sini.

JavaScript referensi fungsi resolver untuk Amazon RDS

Fungsi dan resolver AWS AppSync RDS memungkinkan pengembang untuk mengirim SQL kueri ke database cluster Amazon Aurora menggunakan API Data RDS dan mendapatkan kembali hasil kueri ini. Anda dapat menulis SQL pernyataan yang dikirim ke API Data dengan menggunakan AWS AppSync template `sql` yang diberi tag `rds` modul atau dengan menggunakan fungsi `rds` modul`select`,, `insert`, `update`, dan `remove` pembantu. AWS AppSyncmenggunakan `ExecuteStatement`tindakan RDS Data Service untuk menjalankan pernyataan SQL terhadap database.

Topik

Templat dengan tag SQL

AWS AppSynctemplate yang `sql` diberi tag memungkinkan Anda membuat pernyataan statis yang dapat menerima nilai dinamis saat runtime dengan menggunakan ekspresi template. AWS AppSyncmembangun peta variabel dari nilai ekspresi untuk membuat `SqlParameterized`kueri yang dikirim ke API Data Tanpa Server Amazon Aurora. Dengan metode ini, nilai dinamis tidak mungkin diteruskan pada waktu berjalan untuk memodifikasi pernyataan asli, yang dapat menyebabkan eksekusi yang tidak disengaja. Semua nilai dinamis dilewatkan sebagai parameter, tidak dapat memodifikasi pernyataan asli, dan tidak dieksekusi oleh database. Ini membuat kueri Anda kurang rentan terhadap serangan SQL injeksi.

> **Note**
>
> Dalam semua kasus, saat menulis SQL pernyataan, Anda harus mengikuti pedoman keamanan untuk menangani data yang Anda terima sebagai input dengan benar.

> **Note**
>
> Template `sql` yang ditandai hanya mendukung melewati nilai variabel. Anda tidak dapat menggunakan ekspresi untuk menentukan nama kolom atau tabel secara dinamis. Namun, Anda dapat menggunakan fungsi utilitas untuk membangun pernyataan dinamis.

Dalam contoh berikut, kami membuat kueri yang memfilter berdasarkan nilai `col` argumen yang diatur secara dinamis dalam kueri GraphQL saat dijalankan. Nilai hanya dapat ditambahkan ke pernyataan menggunakan ekspresi tag:

```
import { sql, createMySQLStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
  const query = sql
SELECT * FROM table
WHERE column = ${ctx.args.col}
  ;
  return createMySQLStatement(query);
}
```

Dengan meneruskan semua nilai dinamis melalui peta variabel, kami mengandalkan mesin database untuk menangani dan membersihkan nilai dengan aman.

Membuat pernyataan

Fungsi dan resolver dapat berinteraksi dengan database MySQL dan PostgreSQL. Gunakan `createMySQLStatement` dan `createPgStatement` masing-masing untuk membangun pernyataan. Misalnya, `createMySQLStatement` dapat membuat query MySQL. Fungsi-fungsi ini menerima hingga dua pernyataan, berguna ketika permintaan harus segera mengambil hasil. Dengan MySQL, Anda dapat melakukan:

```
import { sql, createMySQLStatement } from '@aws-appsync/utils/rds';
```

```
export function request(ctx) {
    const { id, text } = ctx.args;
    const s1 = sql`insert into Post(id, text) values(${id}, ${text})`;
    const s2 = sql`select * from Post where id = ${id}`;
    return createMySQLStatement(s1, s2);
}
```

> **Note**
>
> `createPgStatement`dan `createMySQLStatement` tidak luput atau mengutip pernyataan yang dibangun dengan template yang `sql` diberi tag.

Mengambil data

Hasil dari pernyataan SQL Anda yang dieksekusi tersedia di handler respons Anda di objek. `context.result` Hasilnya adalah string JSON dengan elemen respons dari `ExecuteStatement` tindakan. Saat diurai, hasilnya memiliki bentuk sebagai berikut:

```
type SQLStatementResults = {
    sqlStatementResults: {
        records: any[];
        columnMetadata: any[];
        numberOfRecordsUpdated: number;
        generatedFields?: any[]
    }[]
}
```

Anda dapat menggunakan `toJsonObject` utilitas untuk mengubah hasilnya menjadi daftar objek JSON yang mewakili baris yang dikembalikan. Sebagai contoh:

```
import { toJsonObject } from '@aws-appsync/utils/rds';

export function response(ctx) {
    const { error, result } = ctx;
    if (error) {
        return util.appendError(
            error.message,
            error.type,
```

```
            result
        )
    }
    return toJsonObject(result)[1][0]
}
```

Perhatikan bahwa `toJsonObject` mengembalikan array hasil pernyataan. Jika Anda memberikan satu pernyataan, panjang array adalah`1`. Jika Anda memberikan dua pernyataan, panjang array adalah`2`. Setiap hasil dalam array berisi `0` atau lebih baris. `toJsonObject`kembali `null` jika nilai hasil tidak valid atau tidak terduga.

Fungsi utilitas

Anda dapat menggunakan pembantu utilitas modul AWS AppSync RDS untuk berinteraksi dengan database Anda.

SQL Pilih

`select`Utilitas membuat `SELECT` pernyataan untuk menanyakan database relasional Anda.

Penggunaan dasar

Dalam bentuk dasarnya, Anda dapat menentukan tabel yang ingin Anda kueri:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

    // Generates statement:
    // "SELECT * FROM "persons"
    return createPgStatement(select({table: 'persons'}));
}
```

Perhatikan bahwa Anda juga dapat menentukan skema dalam pengidentifikasi tabel Anda:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

    // Generates statement:
    // SELECT * FROM "private"."persons"
```

```
    return createPgStatement(select({table: 'private.persons'}));
}
```

Menentukan kolom

Anda dapat menentukan kolom dengan `columns` properti. Jika ini tidak disetel ke nilai, defaultnya adalah: *

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name']
    }));
}
```

Anda dapat menentukan tabel kolom juga:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "persons"."name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'persons.name']
    }));
}
```

Batas dan offset

Anda dapat menerapkan `limit` dan `offset` ke kueri:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    // LIMIT :limit
```

```
    // OFFSET :offset
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        limit: 10,
        offset: 40
    }));
}
```

Memesan oleh

Anda dapat mengurutkan hasil Anda dengan `orderBy` properti. Menyediakan array objek yang menentukan kolom dan `dir` properti opsional:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name" FROM "persons"
    // ORDER BY "name", "id" DESC
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        orderBy: [{column: 'name'}, {column: 'id', dir: 'DESC'}]
    }));
}
```

Filter

Anda dapat membangun filter dengan menggunakan objek kondisi khusus:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    // WHERE "name" = :NAME
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        where: {name: {eq: 'Stephane'}}
    }));
}
```

Anda juga dapat menggabungkan filter:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    // WHERE "name" = :NAME and "id" > :ID
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        where: {name: {eq: 'Stephane'}, id: {gt: 10}}
    }));
}
```

Anda juga dapat membuat OR pernyataan:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    // WHERE "name" = :NAME OR "id" > :ID
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        where: { or: [
            { name: { eq: 'Stephane'} },
            { id: { gt: 10 } }
        ]}
    }));
}
```

Anda juga dapat meniadakan suatu kondisi dengan`not`:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    // WHERE NOT ("name" = :NAME AND "id" > :ID)
    return createPgStatement(select({
        table: 'persons',
```

```
        columns: ['id', 'name'],
        where: { not: [
            { name: { eq: 'Stephane'} },
            { id: { gt: 10 } }
        ]}
    }));
}
```

Anda juga dapat menggunakan operator berikut untuk membandingkan nilai:

Operasi	Deskripsi	Jenis nilai yang mungkin
eq	Equal	number, string, boolean
ne	Not equal	number, string, boolean
le	Less than or equal	number, string
lt	Less than	number, string
ge	Greater than or equal	number, string
gt	Greater than	number, string
contains	Like	string
notContains	Not like	string
beginsWith	Starts with prefix	string
between	Between two values	number, string
attributeExists	The attribute is not null	number, string, boolean
size	checks the length of the element	string

Sisipkan SQL

`insert`Utilitas menyediakan cara mudah untuk memasukkan item baris tunggal dalam database Anda dengan operasi. `INSERT`

Penyisipan item tunggal

Untuk menyisipkan item, tentukan tabel dan kemudian masukkan objek nilai Anda. Kunci objek dipetakan ke kolom tabel Anda. Nama kolom secara otomatis lolos, dan nilai dikirim ke database menggunakan peta variabel:

```
import { insert, createMySQLStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const { input: values } = ctx.args;
    const insertStatement = insert({ table: 'persons', values });

    // Generates statement:
    // INSERT INTO `persons`(`name`)
    // VALUES(:NAME)
    return createMySQLStatement(insertStatement)
}
```

Kasus penggunaan MySQL

Anda dapat menggabungkan `insert` diikuti oleh a `select` untuk mengambil baris yang disisipkan:

```
import { insert, select, createMySQLStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const { input: values } = ctx.args;
    const insertStatement = insert({  table: 'persons', values });
    const selectStatement = select({
        table: 'persons',
        columns: '*',
        where: { id: { eq: values.id } },
        limit: 1,
    });

    // Generates statement:
    // INSERT INTO `persons`(`name`)
    // VALUES(:NAME)
    // and
    // SELECT *
    // FROM `persons`
    // WHERE `id` = :ID
    return createMySQLStatement(insertStatement, selectStatement)
```

```
}
```

Kasus penggunaan postgres

Dengan Postgres, Anda dapat menggunakan `returning`untuk mendapatkan data dari baris yang Anda masukkan. Ia menerima * atau array nama kolom:

```
import { insert, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const { input: values } = ctx.args;
    const insertStatement = insert({
        table: 'persons',
        values,
        returning: '*'
    });

    // Generates statement:
    // INSERT INTO "persons"("name")
    // VALUES(:NAME)
    // RETURNING *
    return createPgStatement(insertStatement)
}
```

Pembaruan SQL

`update`Utilitas memungkinkan Anda untuk memperbarui baris yang ada. Anda dapat menggunakan objek kondisi untuk menerapkan perubahan pada kolom yang ditentukan di semua baris yang memenuhi kondisi. Sebagai contoh, katakanlah kita memiliki skema yang memungkinkan kita membuat mutasi ini. Kami ingin memperbarui `name` of `Person` dengan `id` nilai `3` tetapi hanya jika kami mengetahuinya (`known_since`) sejak tahun ini`2000`:

```
mutation Update {
    updatePerson(
        input: {id: 3, name: "Jon"},
        condition: {known_since: {ge: "2000"}}
    ) {
    id
    name
  }
}
```

Penyelesai pembaruan kami terlihat seperti ini:

```
import { update, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const { input: { id, ...values }, condition } = ctx.args;
    const where = {
        ...condition,
        id: { eq: id },
    };
    const updateStatement = update({
        table: 'persons',
        values,
        where,
        returning: ['id', 'name'],
    });

    // Generates statement:
    // UPDATE "persons"
    // SET "name" = :NAME, "birthday" = :BDAY, "country" = :COUNTRY
    // WHERE "id" = :ID
    // RETURNING "id", "name"
    return createPgStatement(updateStatement)
}
```

Kita dapat menambahkan cek ke kondisi kita untuk memastikan bahwa hanya baris yang memiliki kunci utama `id` sama dengan 3 yang diperbarui. Demikian pula, untuk Postgres`inserts`, Anda dapat menggunakan `returning` untuk mengembalikan data yang dimodifikasi.

SQL Hapus

`remove`Utilitas memungkinkan Anda untuk menghapus baris yang ada. Anda dapat menggunakan objek kondisi pada semua baris yang memenuhi kondisi. Perhatikan bahwa itu `delete` adalah kata kunci yang dicadangkan di JavaScript. `remove`harus digunakan sebagai gantinya:

```
import { remove, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const { input: { id }, condition } = ctx.args;
    const where = { ...condition, id: { eq: id } };
    const deleteStatement = remove({
        table: 'persons',
```

```
        where,
        returning: ['id', 'name'],
    });

    // Generates statement:
    // DELETE "persons"
    // WHERE "id" = :ID
    // RETURNING "id", "name"
    return createPgStatement(updateStatement)
}
```

Casting

Dalam beberapa kasus, Anda mungkin ingin lebih spesifisitas tentang jenis objek yang benar untuk digunakan dalam pernyataan Anda. Anda dapat menggunakan petunjuk jenis yang disediakan untuk menentukan jenis parameter Anda. AWS AppSyncmendukung petunjuk jenis yang sama dengan Data API. Anda dapat mentransmisikan parameter Anda dengan menggunakan `typeHint` fungsi dari AWS AppSync `rds` modul.

Contoh berikut memungkinkan Anda untuk mengirim array sebagai nilai yang dicor sebagai objek JSON. Kami menggunakan `->` operator untuk mengambil elemen di `index 2` dalam array JSON:

```
import { sql, createPgStatement, toJsonObject, typeHint } from '@aws-appsync/utils/
rds';

export function request(ctx) {
    const arr = ctx.args.list_of_ids
    const statement = sql`select ${typeHint.JSON(arr)}->2 as value`
    return createPgStatement(statement)
}

export function response(ctx) {
    return toJsonObject(ctx.result)[0][0].value
}
```

Casting juga berguna saat menangani dan membandingkan`DATE`,`TIME`, dan`TIMESTAMP`:

```
import { select, createPgStatement, typeHint } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const when = ctx.args.when
```

```
    const statement = select({
        table: 'persons',
        where: { createdAt : { gt: typeHint.DATETIME(when) } }
    })
    return createPgStatement(statement)
}
```

Berikut contoh lain yang menunjukkan bagaimana Anda dapat mengirim tanggal dan waktu saat ini:

```
import { sql, createPgStatement, typeHint } from '@aws-appsync/utils/rds';

export function request(ctx) {
    const now = util.time.nowFormatted('YYYY-MM-dd HH:mm:ss')
    return createPgStatement(sql`select ${typeHint.TIMESTAMP(now)}`)
}
```

Petunjuk tipe yang tersedia

- `typeHint.DATE`- Parameter yang sesuai dikirim sebagai objek dari `DATE` jenis ke database. Format yang diterima adalah `YYYY-MM-DD`.
- `typeHint.DECIMAL`- Parameter yang sesuai dikirim sebagai objek dari `DECIMAL` jenis ke database.
- `typeHint.JSON`- Parameter yang sesuai dikirim sebagai objek dari `JSON` jenis ke database.
- `typeHint.TIME`- Nilai parameter string yang sesuai dikirim sebagai objek dari `TIME` tipe ke database. Format yang diterima adalah `HH:MM:SS[.FFF]`.
- `typeHint.TIMESTAMP`- Nilai parameter string yang sesuai dikirim sebagai objek dari `TIMESTAMP` tipe ke database. Format yang diterima adalah `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `typeHint.UUID`- Nilai parameter string yang sesuai dikirim sebagai objek dari `UUID` tipe ke database.

Referensi template pemetaan resolver (VTL)

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Bagian berikut akan menjelaskan bagaimana operasi utilitas dapat digunakan dalam template pemetaan.

Topik

Ikhtisar template pemetaan resolver

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync memungkinkan Anda menanggapi permintaan GraphQL dengan melakukan operasi pada sumber daya Anda. Untuk setiap bidang GraphQL yang ingin Anda jalankan kueri atau mutasi, resolver harus dilampirkan untuk berkomunikasi dengan sumber data. Komunikasi biasanya melalui parameter atau operasi yang unik untuk sumber data.

Resolver adalah konektor antara GraphQL dan sumber data. Mereka memberi tahu AWS AppSync cara menerjemahkan permintaan GraphQL yang masuk ke dalam instruksi untuk sumber data backend Anda, dan bagaimana menerjemahkan respons dari sumber data itu kembali ke respons GraphQL. Mereka ditulis dalam Apache Velocity Template Language (VTL), yang mengambil permintaan Anda sebagai input dan output dokumen JSON yang berisi instruksi untuk resolver. Anda dapat menggunakan template pemetaan untuk instruksi sederhana, seperti meneruskan argumen dari bidang GraphQL, atau untuk instruksi yang lebih kompleks, seperti perulangan argumen untuk membangun item sebelum memasukkan item ke DynamoDB.

Ada dua jenis resolver yang memanfaatkan template pemetaan dengan cara AWS AppSync yang sedikit berbeda:

- Penyelesai unit
- Penyelesai pipa

Penyelesai unit

Resolver unit adalah entitas mandiri yang hanya menyertakan template permintaan dan respons. Gunakan ini untuk operasi tunggal sederhana seperti mencantumkan item dari satu sumber data.

- Templat permintaan: Ambil permintaan yang masuk setelah operasi GraphQL diuraikan dan ubah menjadi konfigurasi permintaan untuk operasi sumber data yang dipilih.
- Template respons: Menafsirkan respons dari sumber data Anda dan memetakannya ke bentuk tipe keluaran bidang GraphQL.

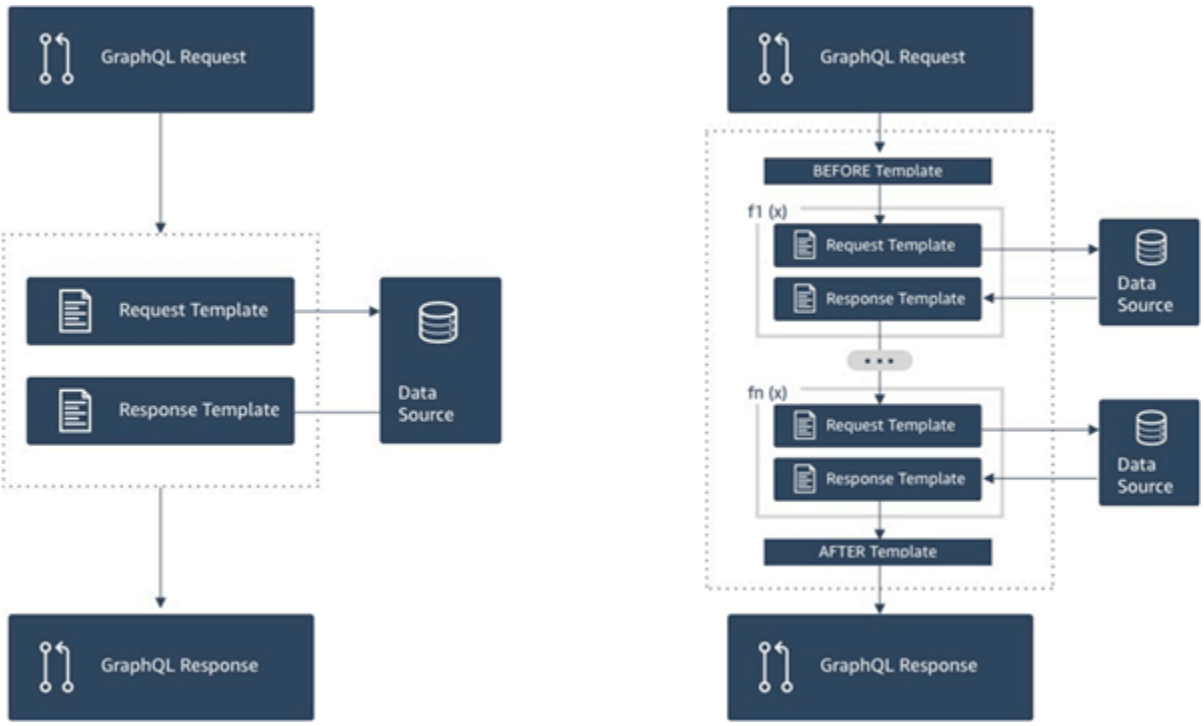
Penyelesai pipa

Pipeline resolver berisi satu atau lebih fungsi yang dilakukan secara berurutan. Setiap fungsi menyertakan template permintaan dan template respons. Pipeline resolver juga memiliki template before dan after template yang mengelilingi urutan fungsi yang berisi template. Template after memetakan ke tipe output bidang GraphQL. Pipeline resolver berbeda dari resolver unit dalam cara

template respon memetakan output. Pipeline resolver dapat memetakan ke output apa pun yang Anda inginkan, termasuk input untuk fungsi lain atau template setelah resolver pipeline.

Fungsi Pipeline resolver memungkinkan Anda untuk menulis logika umum yang dapat Anda gunakan kembali di beberapa resolver dalam skema Anda. Fungsi dilampirkan langsung ke sumber data, dan seperti resolver unit, berisi format template pemetaan permintaan dan respons yang sama.

Diagram berikut menunjukkan aliran proses resolver unit di sebelah kiri dan resolver pipa di sebelah kanan.



Pipeline resolver berisi superset fungsionalitas yang didukung oleh unit resolver, dan banyak lagi, dengan biaya yang sedikit lebih rumit.

Anatomi penyelesai pipa

Pipeline resolver terdiri dari template Sebelum pemetaan, template After mapping, dan daftar fungsi. Setiap fungsi memiliki template pemetaan permintaan dan respons yang dijalankan terhadap sumber data. Karena penyelesai pipa mendelegasikan eksekusi ke daftar fungsi, oleh karena itu tidak ditautkan ke sumber data apa pun. Resolver dan fungsi unit adalah primitif yang menjalankan operasi terhadap sumber data. Lihat ikhtisar template pemetaan Resolver untuk informasi selengkapnya.

Sebelum memetakan template

Template pemetaan permintaan dari resolver pipeline, atau langkah Sebelum, memungkinkan Anda untuk melakukan beberapa logika persiapan sebelum menjalankan fungsi yang ditentukan.

Daftar fungsi

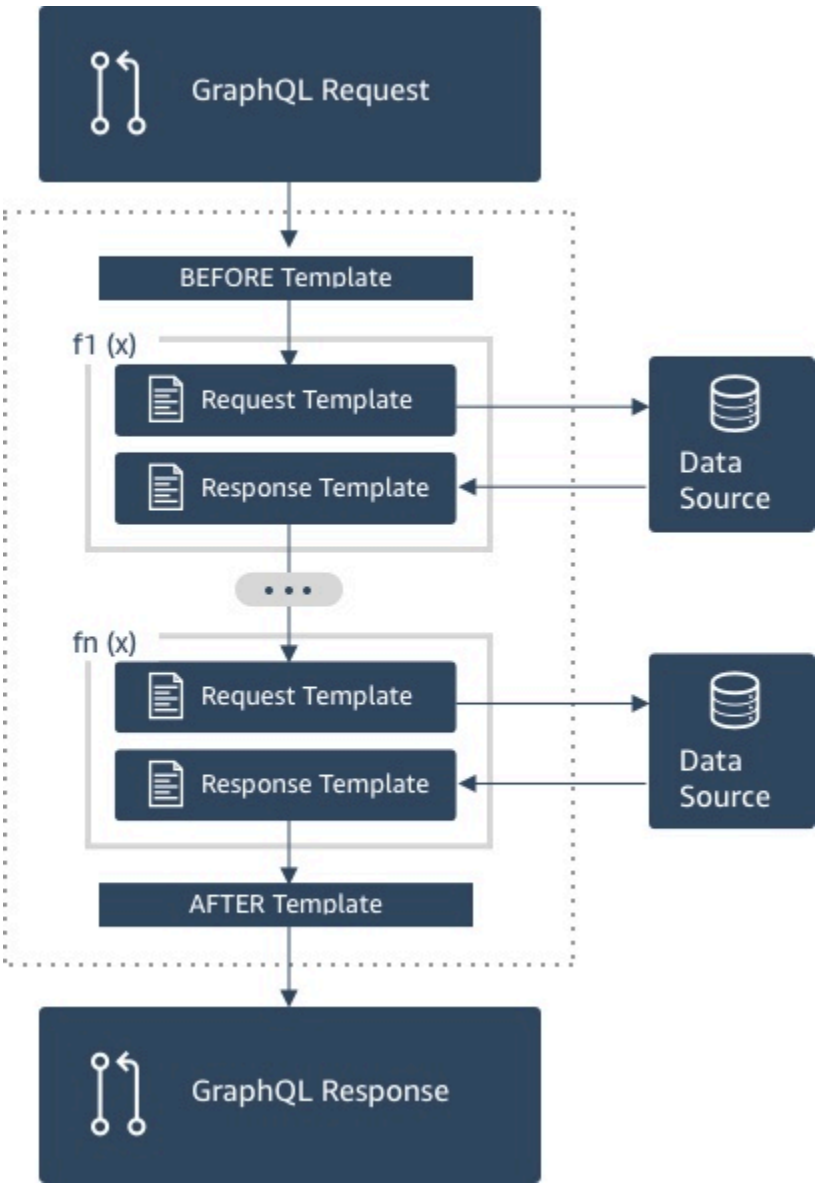
Daftar fungsi resolver pipeline akan berjalan secara berurutan. Hasil evaluasi template pemetaan permintaan penyelesai pipa dibuat tersedia untuk fungsi pertama sebagai. `$ctx.prev.result` Setiap output fungsi tersedia untuk fungsi berikutnya sebagai`$ctx.prev.result`.

Setelah template pemetaan

Template pemetaan respons dari resolver pipa, atau langkah Setelah, memungkinkan Anda untuk melakukan beberapa logika pemetaan akhir dari output fungsi terakhir ke jenis bidang GraphQL yang diharapkan. Output dari fungsi terakhir dalam daftar fungsi tersedia di template pemetaan pipeline resolver sebagai atau. `$ctx.prev.result` `$ctx.result`

Aliran eksekusi

Mengingat resolver pipeline yang terdiri dari dua fungsi, daftar di bawah ini mewakili alur eksekusi saat resolver dipanggil:



1. Pipeline resolver Sebelum memetakan template
2. Fungsi 1: Templat pemetaan permintaan fungsi
3. Fungsi 1: Pemanggilan sumber data
4. Fungsi 1: Templat pemetaan respons fungsi
5. Fungsi 2: Templat pemetaan permintaan fungsi
6. Fungsi 2: Pemanggilan sumber data
7. Fungsi 2: Templat pemetaan respons fungsi
8. Pemecah pipa Setelah template pemetaan

> **Note**
>
> Aliran eksekusi resolver pipa searah dan didefinisikan secara statis pada resolver.

Utilitas Bahasa Template Kecepatan Apache (VTL) yang Berguna

Ketika kompleksitas aplikasi meningkat, utilitas dan arahan VTL ada di sini untuk memfasilitasi produktivitas pengembangan. Utilitas berikut dapat membantu Anda ketika Anda bekerja dengan resolver pipa.

$ ctx.simpanan

Stash adalah `Map` yang tersedia di dalam setiap resolver dan template pemetaan fungsi. Instans simpanan yang sama hidup melalui eksekusi resolver tunggal. Artinya, Anda dapat menggunakan simpanan untuk meneruskan data arbitrer di seluruh templat pemetaan permintaan dan respons, dan di seluruh fungsi dalam penyelesai pipa. Stash mengekspos metode yang sama seperti struktur data peta Java.

$ctx.prev.result

`$ctx.prev.result`Ini merupakan hasil dari operasi sebelumnya yang dijalankan di resolver pipa.

Jika operasi sebelumnya adalah template Sebelum pemetaan pipeline resolver, maka `$ctx.prev.result` mewakili output dari evaluasi template dan tersedia untuk fungsi pertama dalam pipeline. Jika operasi sebelumnya adalah fungsi pertama, maka `$ctx.prev.result` mewakili output dari fungsi pertama dan tersedia untuk fungsi kedua dalam pipa. Jika operasi sebelumnya adalah fungsi terakhir, maka `$ctx.prev.result` mewakili output dari fungsi terakhir dan tersedia untuk template After mapping resolver pipeline.

#return (data: Objek)

`#return(data: Object)`Arahan ini berguna jika Anda perlu kembali sebelum waktunya dari template pemetaan apa pun. `#return(data: Object)`analog dengan kata kunci return dalam bahasa pemrograman karena kembali dari blok logika cakupan terdekat. Apa artinya ini adalah bahwa menggunakan `#return` di dalam template pemetaan resolver kembali dari resolver. Menggunakan `#return(data: Object)` dalam set template pemetaan resolver pada bidang `data` GraphQL. Selain itu, menggunakan `#return(data: Object)` dari template pemetaan fungsi kembali dari fungsi dan melanjutkan eksekusi ke fungsi berikutnya dalam pipeline atau template pemetaan respons resolver.

#return

Ini sama dengan`#return(data: Object)`, tetapi `null` akan dikembalikan sebagai gantinya.

$ util.error

`$util.error`Utilitas ini berguna untuk melempar kesalahan bidang. Menggunakan `$util.error` di dalam template pemetaan fungsi akan segera menimbulkan kesalahan bidang, yang mencegah fungsi selanjutnya dieksekusi. Untuk detail selengkapnya dan `$util.error` tanda tangan lainnya, kunjungi referensi utilitas template pemetaan Resolver.

$util.AppPenError

`$util.appendError`Ini mirip dengan`$util.error()`, dengan perbedaan utama bahwa itu tidak mengganggu evaluasi template pemetaan. Sebaliknya, itu menandakan ada kesalahan dengan bidang, tetapi memungkinkan template untuk dievaluasi dan akibatnya mengembalikan data. Menggunakan fungsi `$util.appendError` di dalam tidak akan mengganggu aliran eksekusi pipa. Untuk detail selengkapnya dan `$util.error` tanda tangan lainnya, kunjungi referensi utilitas template pemetaan Resolver.

Contoh Templat

Misalkan Anda memiliki sumber data DynamoDB dan resolver Unit pada bidang **`getPost(id:ID!)`** bernama yang mengembalikan **`Post`** tipe dengan query GraphQL berikut:

```
getPost(id:1){
    id
    title
    content
}
```

Template resolver Anda mungkin terlihat seperti berikut:

```
     {
    "version" : "2018-05-29",
    "operation" : "GetItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    }
}
```

Ini akan menggantikan nilai parameter `id` input 1 for `${ctx.args.id}` dan menghasilkan JSON berikut:

```
     {
    "version" : "2018-05-29",
    "operation" : "GetItem",
    "key" : {
        "id" : { "S" : "1" }
    }
}
```

AWS AppSync menggunakan template ini untuk menghasilkan instruksi untuk berkomunikasi dengan DynamoDB dan mendapatkan data (atau melakukan operasi lain yang sesuai). Setelah data kembali, AWS AppSync jalankan melalui template pemetaan respons opsional, yang dapat Anda gunakan untuk melakukan pembentukan data atau logika. Misalnya, ketika kita mendapatkan hasil kembali dari DynamoDB, mereka mungkin terlihat seperti ini:

```
{
        "id" : 1,
        "theTitle" : "AWS AppSync works offline!",
        "theContent-part1" : "It also has realtime functionality",
        "theContent-part2" : "using GraphQL"
}
```

Anda dapat memilih untuk menggabungkan dua bidang ke dalam satu bidang dengan template pemetaan respons berikut:

```
{
        "id" : $util.toJson($context.data.id),
        "title" : $util.toJson($context.data.theTitle),
        "content" : $util.toJson("${context.data.theContent-part1}
 ${context.data.theContent-part2}")
}
```

Berikut adalah bagaimana data dibentuk setelah template diterapkan ke data:

```
{
        "id" : 1,
        "title" : "AWS AppSync works offline!",
        "content" : "It also has realtime functionality using GraphQL"
```

```
}
```

Data ini diberikan kembali sebagai respons terhadap klien sebagai berikut:

```
{
        "data": {
                "getPost":      {
                        "id" : 1,
                        "title" : "AWS AppSync works offline!",
                        "content" : "It also has realtime functionality using GraphQL"
                }
        }
}
```

Perhatikan bahwa dalam sebagian besar keadaan, template pemetaan respons adalah passthrough data yang sederhana, sebagian besar berbeda jika Anda mengembalikan item individual atau daftar item. Untuk item individual, passthrough adalah:

```
$util.toJson($context.result)
```

Untuk daftar passthrough biasanya:

```
$util.toJson($context.result.items)
```

Untuk melihat lebih banyak contoh resolver unit dan pipeline, lihat tutorial Resolver.

Aturan deserialisasi template pemetaan yang dievaluasi

Template pemetaan mengevaluasi ke string. Dalam AWS AppSync, string output harus mengikuti struktur JSON agar valid.

Selain itu, aturan deserialisasi berikut diberlakukan.

Kunci duplikat tidak diperbolehkan dalam objek JSON

Jika string template pemetaan yang dievaluasi mewakili objek JSON atau berisi objek yang memiliki kunci duplikat, template pemetaan mengembalikan pesan kesalahan berikut:

```
Duplicate field 'aField' detected on Object. Duplicate JSON keys are not allowed.
```

Contoh kunci duplikat dalam templat pemetaan permintaan yang dievaluasi:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "postId": "1",
        "field": "getPost" ## key 'field' has been redefined
    }
}
```

Untuk memperbaiki kesalahan ini, jangan mendefinisikan ulang kunci di objek JSON.

Karakter tertinggal tidak diizinkan di objek JSON

Jika string template pemetaan yang dievaluasi mewakili objek JSON dan berisi karakter asing yang mengikuti, template pemetaan mengembalikan pesan kesalahan berikut:

`Trailing characters at the end of the JSON string are not allowed.`

Contoh karakter tertinggal dalam templat pemetaan permintaan yang dievaluasi:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "postId": "1",
    }
}extraneouschars
```

Untuk memperbaiki kesalahan ini, pastikan bahwa template yang dievaluasi secara ketat mengevaluasi ke JSON.

Panduan pemrograman template pemetaan Resolver

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Ini adalah tutorial pemrograman gaya buku masak dengan Apache Velocity Template Language (VTL) di. AWS AppSync Jika Anda terbiasa dengan bahasa pemrograman lain seperti JavaScript, C, atau Java, itu harus cukup mudah.

AWS AppSync menggunakan VTL untuk menerjemahkan permintaan GraphQL dari klien menjadi permintaan ke sumber data Anda. Kemudian membalikkan proses untuk menerjemahkan respons sumber data kembali ke respons GraphQL. VTL adalah bahasa template logis yang memberi Anda kekuatan untuk memanipulasi permintaan dan respons dalam aliran permintaan/respons standar aplikasi web, menggunakan teknik seperti:

- Nilai default untuk item baru
- Validasi masukan dan pemformatan
- Mengubah dan membentuk data
- Mengulangi daftar, peta, dan array untuk mencabut atau mengubah nilai
- Filter/ubah tanggapan berdasarkan identitas pengguna
- Pemeriksaan otorisasi yang kompleks

Misalnya, Anda mungkin ingin melakukan validasi nomor telepon dalam layanan pada argumen GraphQL, atau mengonversi parameter input ke huruf besar sebelum menyimpannya di DynamoDB. Atau mungkin Anda ingin sistem klien memberikan kode, sebagai bagian dari argumen GraphQL, klaim token JWT, atau header HTTP, dan hanya merespons dengan data jika kode tersebut cocok dengan string tertentu dalam daftar. Ini semua adalah pemeriksaan logis yang dapat Anda lakukan dengan VTL di. AWS AppSync

VTL memungkinkan Anda menerapkan logika menggunakan teknik pemrograman yang mungkin sudah tidak asing lagi. Namun, itu dibatasi untuk berjalan dalam alur permintaan/respons standar untuk memastikan bahwa GraphQL API Anda dapat diskalakan seiring dengan pertumbuhan basis pengguna Anda. Karena AWS AppSync juga mendukung AWS Lambda sebagai resolver, Anda dapat menulis fungsi Lambda dalam bahasa pemrograman pilihan Anda (Node.js, Python, Go, Java, dll.) Jika Anda membutuhkan lebih banyak fleksibilitas.

Pengaturan

Teknik umum ketika belajar bahasa adalah mencetak hasil (misalnya, `console.log(variable)` dalam JavaScript) untuk melihat apa yang terjadi. Dalam tutorial ini, kami mendemonstrasikan ini dengan membuat skema GraphQL sederhana dan meneruskan peta nilai ke fungsi Lambda. Fungsi

Lambda mencetak nilai dan kemudian merespons dengan mereka. Ini akan memungkinkan Anda untuk memahami aliran permintaan/respons dan melihat teknik pemrograman yang berbeda.

Mulailah dengan membuat skema GraphQL berikut:

```
type Query {
    get(id: ID, meta: String): Thing
}

type Thing {
    id: ID!
    title: String!
    meta: String
}

schema {
    query: Query
}
```

Sekarang buat AWS Lambda fungsi berikut, menggunakan Node.js sebagai bahasa:

```
exports.handler = (event, context, callback) => {
    console.log('VTL details: ', event);
    callback(null, event);
};
```

Di panel Sumber Data AWS AppSync konsol, tambahkan fungsi Lambda ini sebagai sumber data baru. Arahkan kembali ke halaman Skema konsol dan klik tombol ATTACH di sebelah kanan, di sebelah `get(...):Thing` kueri. Untuk template permintaan, pilih template yang ada dari menu Invoke dan teruskan argumen. Untuk template respons, pilih Kembalikan hasil Lambda.

Buka Amazon CloudWatch Logs untuk fungsi Lambda Anda di satu lokasi, dan dari tab Kueri AWS AppSync konsol, jalankan kueri GraphQL berikut:

```
query test {
  get(id:123 meta:"testing"){
    id
    meta
  }
}
```

Respons GraphQL harus `id:123` berisi `meta:testing` dan, karena fungsi Lambda menggemakannya kembali. Setelah beberapa detik, Anda akan melihat catatan di CloudWatch Log dengan detail ini juga.

Variabel

VTL menggunakan referensi, yang dapat Anda gunakan untuk menyimpan atau memanipulasi data. Ada tiga jenis referensi dalam VTL: variabel, properti, dan metode. Variabel memiliki $ tanda di depannya dan dibuat dengan `#set` arahan:

```
#set($var = "a string")
```

Variabel menyimpan jenis serupa yang Anda kenal dari bahasa lain, seperti angka, string, array, daftar, dan peta. Anda mungkin telah memperhatikan payload JSON dikirim dalam template permintaan default untuk resolver Lambda:

```
"payload": $util.toJson($context.arguments)
```

Beberapa hal yang perlu diperhatikan di sini - pertama, AWS AppSync menyediakan beberapa fungsi kenyamanan untuk operasi umum. Dalam contoh ini, `$util.toJson` mengkonversi variabel ke JSON. Kedua, variabel secara otomatis `$context.arguments` diisi dari permintaan GraphQL sebagai objek peta. Anda dapat membuat peta baru sebagai berikut:

```
#set( $myMap = {
  "id": $context.arguments.id,
  "meta": "stuff",
  "upperMeta" : $context.arguments.meta.toUpperCase()
} )
```

Anda sekarang telah membuat variabel bernama`$myMap`, yang memiliki kunci`id`,`meta`, dan`upperMeta`. Ini juga menunjukkan beberapa hal:

- `id`diisi dengan kunci dari argumen GraphQL. Ini umum di VTL untuk mengambil argumen dari klien.
- `meta`di-hardcode dengan nilai, menampilkan nilai default.
- `upperMeta`mengubah `meta` argumen menggunakan metode`.toUpperCase()`.

Letakkan kode sebelumnya di bagian atas template permintaan Anda dan ubah `payload` untuk menggunakan `$myMap` variabel baru:

```
"payload": $util.toJson($myMap)
```

Jalankan fungsi Lambda Anda, dan Anda dapat melihat perubahan respons serta data ini di CloudWatch log. Saat Anda berjalan melalui sisa tutorial ini, kami akan terus mengisi `$myMap` sehingga Anda dapat menjalankan tes serupa.

Anda juga dapat mengatur properties_ pada variabel Anda. Ini bisa berupa string sederhana, array, atau JSON:

```
#set($myMap.myProperty = "ABC")
#set($myMap.arrProperty = ["Write", "Some", "GraphQL"])
#set($myMap.jsonProperty = {
    "AppSync" : "Offline and Realtime",
    "Cognito" : "AuthN and AuthZ"
})
```

Referensi Tenang

Karena VTL adalah bahasa templating, secara default, setiap referensi yang Anda berikan akan melakukan. `.toString()` Jika referensi tidak terdefinisi, ia mencetak representasi referensi yang sebenarnya, sebagai string. Misalnya:

```
#set($myValue = 5)
##Prints '5'
$myValue

##Prints '$somethingelse'
$somethingelse
```

Untuk mengatasi hal ini, VTL memiliki referensi senyap atau sintaks referensi diam, yang memberi tahu mesin template untuk menekan perilaku ini. Sintaks untuk ini adalah`$!{}`. Misalnya, jika kita mengubah kode sebelumnya sedikit untuk digunakan`$!{somethingelse}`, pencetakan ditekan:

```
#set($myValue = 5)
##Prints '5'
$myValue
```

```
##Nothing prints out
$!{somethingelse}
```

Metode Panggilan

Dalam contoh sebelumnya, kami menunjukkan kepada Anda cara membuat variabel dan secara bersamaan menetapkan nilai. Anda juga dapat melakukan ini dalam dua langkah dengan menambahkan data ke peta Anda seperti yang ditunjukkan berikut:

```
#set ($myMap = {})
#set ($myList = [])

##Nothing prints out
$!{myMap.put("id", "first value")}
##Prints "first value"
$!{myMap.put("id", "another value")}
##Prints true
$!{myList.add("something")}
```

Namun ada sesuatu yang perlu diketahui tentang perilaku ini. Meskipun notasi referensi senyap `$!{}` memungkinkan Anda untuk memanggil metode, seperti di atas, itu tidak akan menekan nilai yang dikembalikan dari metode yang dieksekusi. Inilah sebabnya kami mencatat `##Prints "first value"` dan `##Prints true` di atas. Hal ini dapat menyebabkan kesalahan saat Anda mengulangi peta atau daftar, seperti memasukkan nilai di mana kunci sudah ada, karena output menambahkan string tak terduga ke template setelah evaluasi.

Solusi untuk ini terkadang memanggil metode menggunakan `#set` arahan dan mengabaikan variabel. Misalnya:

```
#set ($myMap = {})
#set($discard = $myMap.put("id", "first value"))
```

Anda dapat menggunakan teknik ini dalam template Anda, karena mencegah string yang tidak terduga dicetak dalam template. AWS AppSync menyediakan fungsi kenyamanan alternatif yang menawarkan perilaku yang sama dalam notasi yang lebih ringkas. Ini memungkinkan Anda untuk tidak perlu memikirkan spesifik implementasi ini. Anda dapat mengakses fungsi ini di bawah `$util.quiet()` atau `$util.qr()` aliasnya. Misalnya:

```
#set ($myMap = {})
#set ($myList = [])
```

```
##Nothing prints out
$util.quiet($myMap.put("id", "first value"))
##Nothing prints out
$util.qr($myList.add("something"))
```

String

Seperti banyak bahasa pemrograman, string bisa sulit untuk ditangani, terutama ketika Anda ingin membangunnya dari variabel. Ada beberapa hal umum yang muncul dengan VTL.

Misalkan Anda memasukkan data sebagai string ke sumber data seperti DynamoDB, tetapi diisi dari variabel, seperti argumen GraphQL. Sebuah string akan memiliki tanda kutip ganda, dan untuk mereferensikan variabel dalam string Anda hanya perlu `"${}"` (jadi tidak `!` seperti dalam notasi referensi yang tenang). Ini mirip dengan template literal di JavaScript: https://developer.mozilla.org/en-US/docs/Web/ JavaScript /reference/Template_literals

```
#set($firstname = "Jeff")
$!{myMap.put("Firstname", "${firstname}")}
```

Anda dapat melihat ini di template permintaan DynamoDB, `"author": { "S" : "${context.arguments.author}"}` seperti saat menggunakan argumen dari klien GraphQL, atau untuk pembuatan ID otomatis seperti. `"id" : { "S" : "$util.autoId()"}` Ini berarti Anda dapat mereferensikan variabel atau hasil dari metode di dalam string untuk mengisi data.

Anda juga dapat menggunakan metode publik dari kelas Java String, seperti menarik substring:

```
#set($bigstring = "This is a long string, I want to pull out everything after the
 comma")
#set ($comma = $bigstring.indexOf(','))
#set ($comma = $comma +2)
#set ($substring = $bigstring.substring($comma))

$util.qr($myMap.put("substring", "${substring}"))
```

Penggabungan string juga merupakan tugas yang sangat umum. Anda dapat melakukan ini dengan referensi variabel sendiri atau dengan nilai statis:

```
#set($s1 = "Hello")
#set($s2 = " World")
```

```
$util.qr($myMap.put("concat","$s1$s2"))
$util.qr($myMap.put("concat2","Second $s1 World"))
```

Loop

Sekarang Anda telah membuat variabel dan disebut metode, Anda dapat menambahkan beberapa logika ke kode Anda. Tidak seperti bahasa lain, VTL hanya mengizinkan loop, di mana jumlah iterasi telah ditentukan sebelumnya. Tidak ada `do..while` dalam Velocity. Desain ini memastikan bahwa proses evaluasi selalu berakhir, dan memberikan batas untuk skalabilitas saat operasi GraphQL Anda dijalankan.

Loop dibuat dengan `#foreach` dan mengharuskan Anda untuk menyediakan variabel loop dan objek iterable seperti array, daftar, peta, atau koleksi. Contoh pemrograman klasik dengan `#foreach` loop adalah mengulang item dalam koleksi dan mencetaknya, jadi dalam kasus kami, kami mencabutnya dan menambahkannya ke peta:

```
#set($start = 0)
#set($end = 5)
#set($range = [$start..$end])

#foreach($i in $range)
   ##$util.qr($myMap.put($i, "abc"))
   ##$util.qr($myMap.put($i, $i.toString()+"foo")) ##Concat variable with string
   $util.qr($myMap.put($i, "${i}foo"))     ##Reference a variable in a string with
 "${varname}"
#end
```

Contoh ini menunjukkan beberapa hal. Yang pertama adalah menggunakan variabel dengan `[..]` operator rentang untuk membuat objek iterable. Kemudian setiap item direferensikan oleh variabel `$i` yang dapat Anda operasikan. Pada contoh sebelumnya, Anda juga melihat Komentar yang dilambangkan dengan pound ganda. ## Ini juga menampilkan menggunakan variabel loop di kedua kunci atau nilai, serta metode penggabungan yang berbeda menggunakan string.

Perhatikan bahwa `$i` adalah integer, sehingga Anda dapat memanggil `.toString()` metode. Untuk tipe GraphQL INT, ini bisa berguna.

Anda juga dapat menggunakan operator jangkauan secara langsung, misalnya:

```
#foreach($item in [1..5])
```

```
    ...
#end
```

Array

Anda telah memanipulasi peta hingga saat ini, tetapi array juga umum di VTL. Dengan array Anda juga memiliki akses ke beberapa metode dasar seperti`.isEmpty()`,,,, dan `.size()` `.set()` `.get()`.`add()`, seperti yang ditunjukkan di bawah ini:

```
#set($array = [])
#set($idx = 0)

##adding elements
$util.qr($array.add("element in array"))
$util.qr($myMap.put("array", $array[$idx]))

##initialize array vals on create
#set($arr2 = [42, "a string", 21, "test"])

$util.qr($myMap.put("arr2", $arr2[$idx]))
$util.qr($myMap.put("isEmpty", $array.isEmpty()))  ##isEmpty == false
$util.qr($myMap.put("size", $array.size()))

##Get and set items in an array
$util.qr($myMap.put("set", $array.set(0, 'changing array value')))
$util.qr($myMap.put("get", $array.get(0)))
```

Contoh sebelumnya menggunakan notasi indeks array untuk mengambil elemen dengan. `arr2[$idx]` Anda dapat mencari berdasarkan nama dari peta/kamus dengan cara yang sama:

```
#set($result = {
    "Author" : "Nadia",
    "Topic" : "GraphQL"
})

$util.qr($myMap.put("Author", $result["Author"]))
```

Ini sangat umum ketika memfilter hasil yang kembali dari sumber data di Response Template saat menggunakan kondisional.

Cek Bersyarat

Bagian sebelumnya dengan `#foreach` menampilkan beberapa contoh penggunaan logika untuk mengubah data dengan VTL. Anda juga dapat menerapkan pemeriksaan bersyarat untuk mengevaluasi data saat runtime:

```
#if(!$array.isEmpty())
    $util.qr($myMap.put("ifCheck", "Array not empty"))
#else
    $util.qr($myMap.put("ifCheck", "Your array is empty"))
#end
```

`#if()`Pemeriksaan ekspresi Boolean di atas bagus, tetapi Anda juga dapat menggunakan operator dan `#elseif()` untuk percabangan:

```
#if ($arr2.size() == 0)
    $util.qr($myMap.put("elseIfCheck", "You forgot to put anything into this array!"))
#elseif ($arr2.size() == 1)
    $util.qr($myMap.put("elseIfCheck", "Good start but please add more stuff"))
#else
    $util.qr($myMap.put("elseIfCheck", "Good job!"))
#end
```

Kedua contoh ini menunjukkan negasi (!) dan kesetaraan (==). Kita juga dapat menggunakan ||, &&, **>**, **<**, **>=**, **<=**, dan! =.

```
#set($T = true)
#set($F = false)

#if ($T || $F)
  $util.qr($myMap.put("OR", "TRUE"))
#end

#if ($T && $F)
  $util.qr($myMap.put("AND", "TRUE"))
#end
```

Catatan: Hanya `Boolean.FALSE` dan `null` dianggap salah dalam kondisional. Nol (0) dan string kosong (“”) tidak setara dengan false.

Operator

Tidak ada bahasa pemrograman yang lengkap tanpa beberapa operator untuk melakukan beberapa tindakan matematika. Berikut adalah beberapa contoh untuk Anda mulai:

```
#set($x = 5)
#set($y = 7)
#set($z = $x + $y)
#set($x-y = $x - $y)
#set($xy = $x * $y)
#set($xDIVy = $x / $y)
#set($xMODy = $x % $y)

$util.qr($myMap.put("z", $z))
$util.qr($myMap.put("x-y", $x-y))
$util.qr($myMap.put("x*y", $xy))
$util.qr($myMap.put("x/y", $xDIVy))
$util.qr($myMap.put("x|y", $xMODy))
```

Loop dan Kondisional Bersama

Hal ini sangat umum ketika mengubah data dalam VTL, seperti sebelum menulis atau membaca dari sumber data, untuk mengulang objek dan kemudian melakukan pemeriksaan sebelum melakukan tindakan. Menggabungkan beberapa alat dari bagian sebelumnya memberi Anda banyak fungsionalitas. Salah satu alat praktis adalah mengetahui bahwa `#foreach` secara otomatis memberi Anda a `.count` pada setiap item:

```
#foreach ($item in $arr2)
  #set($idx = "item" + $foreach.count)
  $util.qr($myMap.put($idx, $item))
#end
```

Misalnya, mungkin Anda hanya ingin mencabut nilai dari peta jika berada di bawah ukuran tertentu. Menggunakan hitungan bersama dengan kondisional dan `#break` pernyataan memungkinkan Anda untuk melakukan ini:

```
#set($hashmap = {
  "DynamoDB" : "https://aws.amazon.com/dynamodb/",
  "Amplify" : "https://github.com/aws/aws-amplify",
  "DynamoDB2" : "https://aws.amazon.com/dynamodb/",
  "Amplify2" : "https://github.com/aws/aws-amplify"
```

```
})

#foreach ($key in $hashmap.keySet())
    #if($foreach.count > 2)
    #break
  #end
    $util.qr($myMap.put($key, $hashmap.get($key)))
#end
```

Yang sebelumnya `#foreach` diulang dengan `.keySet()`, yang dapat Anda gunakan di peta. Ini memberi Anda akses untuk mendapatkan `$key` dan mereferensikan nilai dengan a `.get($key)`. Argumen GraphQL dari klien AWS AppSync disimpan sebagai peta. Mereka juga dapat diulang dengan `.entrySet()`, yang kemudian Anda dapat mengakses kunci dan nilai sebagai Set, dan mengisi variabel lain atau melakukan pemeriksaan kondisional yang kompleks, seperti validasi atau transformasi input:

```
#foreach( $entry in $context.arguments.entrySet() )
#if ($entry.key == "XYZ" && $entry.value == "BAD")
    #set($myvar = "...")
  #else
    #break
  #end
#end
```

Contoh umum lainnya secara otomatis mengisi informasi default, seperti versi objek awal saat menyinkronkan data (sangat penting dalam resolusi konflik) atau pemilik default objek untuk pemeriksaan otorisasi - Mary membuat posting blog ini, jadi:

```
#set($myMap.owner ="Mary")
#set($myMap.defaultOwners = ["Admins", "Editors"])
```

Konteks

Sekarang setelah Anda lebih terbiasa melakukan pemeriksaan logis di AWS AppSync resolver dengan VTL, lihat objek konteksnya:

```
$util.qr($myMap.put("context", $context))
```

Ini berisi semua informasi yang dapat Anda akses dalam permintaan GraphQL Anda. Untuk penjelasan rinci, lihat referensi konteks.

Penyaringan

Sejauh ini dalam tutorial ini semua informasi dari fungsi Lambda Anda telah dikembalikan ke kueri GraphQL dengan transformasi JSON yang sangat sederhana:

```
$util.toJson($context.result)
```

Logika VTL sama kuatnya ketika Anda mendapatkan tanggapan dari sumber data, terutama saat melakukan pemeriksaan otorisasi pada sumber daya. Mari kita telusuri beberapa contoh. Pertama coba ubah template respons Anda seperti ini:

```
#set($data = {
    "id" : "456",
    "meta" : "Valid Response"
})

$util.toJson($data)
```

Apa pun yang terjadi dengan operasi GraphQL Anda, nilai hardcode dikembalikan kembali ke klien. Ubah ini sedikit sehingga `meta` bidang diisi dari respons Lambda, atur sebelumnya dalam tutorial dalam nilai saat mempelajari tentang `elseIfCheck` kondisional:

```
#set($data = {
    "id" : "456"
})

#foreach($item in $context.result.entrySet())
    #if($item.key == "elseIfCheck")
        $util.qr($data.put("meta", $item.value))
    #end
#end

$util.toJson($data)
```

`$context.result`adalah peta, sehingga Anda dapat menggunakan `entrySet()` untuk melakukan logika pada kunci atau nilai yang dikembalikan. Karena `$context.identity` berisi informasi tentang pengguna yang melakukan operasi GraphQL, jika Anda mengembalikan informasi otorisasi dari sumber data, maka Anda dapat memutuskan untuk mengembalikan semua, sebagian, atau tidak ada data ke pengguna berdasarkan logika Anda. Ubah template respons Anda agar terlihat seperti berikut:

```
#if($context.result["id"] == 123)
    $util.toJson($context.result)
  #else
    $util.unauthorized()
#end
```

Jika Anda menjalankan kueri GraphQL Anda, data akan dikembalikan seperti biasa. Namun, jika Anda mengubah argumen id menjadi sesuatu selain 123 (`query test { get(id:456 meta:"badrequest"){} }`), Anda akan mendapatkan pesan kegagalan otorisasi.

Anda dapat menemukan lebih banyak contoh skenario otorisasi di bagian kasus penggunaan otorisasi.

Lampiran - Contoh Template

Jika Anda mengikuti tutorial, Anda mungkin telah membangun template ini langkah demi langkah. Jika Anda belum melakukannya, kami menyertakannya di bawah ini untuk disalin untuk pengujian.

Templat Permintaan

```
#set( $myMap = {
  "id": $context.arguments.id,
  "meta": "stuff",
  "upperMeta" : "$context.arguments.meta.toUpperCase()"
} )

##This is how you would do it in two steps with a "quiet reference" and you can use it
 for invoking methods, such as .put() to add items to a Map
#set ($myMap2 = {})
$util.qr($myMap2.put("id", "first value"))

## Properties are created with a dot notation
#set($myMap.myProperty = "ABC")
#set($myMap.arrProperty = ["Write", "Some", "GraphQL"])
#set($myMap.jsonProperty = {
    "AppSync" : "Offline and Realtime",
    "Cognito" : "AuthN and AuthZ"
})

##When you are inside a string and just have ${} without ! it means stuff inside curly
 braces are a reference
#set($firstname = "Jeff")
```

```
$util.qr($myMap.put("Firstname", "${firstname}"))

#set($bigstring = "This is a long string, I want to pull out everything after the
 comma")
#set ($comma = $bigstring.indexOf(','))
#set ($comma = $comma +2)
#set ($substring = $bigstring.substring($comma))
$util.qr($myMap.put("substring", "${substring}"))

##Classic for-each loop over N items:
#set($start = 0)
#set($end = 5)
#set($range = [$start..$end])
#foreach($i in $range)          ##Can also use range operator directly like
 #foreach($item in [1...5])
   ##$util.qr($myMap.put($i, "abc"))
   ##$util.qr($myMap.put($i, $i.toString()+"foo")) ##Concat variable with string
   $util.qr($myMap.put($i, "${i}foo"))     ##Reference a variable in a string with
 "${varname)"
#end

##Operators don't work
#set($x = 5)
#set($y = 7)
#set($z = $x + $y)
#set($x-y = $x - $y)
#set($xy = $x * $y)
#set($xDIVy = $x / $y)
#set($xMODy = $x % $y)
$util.qr($myMap.put("z", $z))
$util.qr($myMap.put("x-y", $x-y))
$util.qr($myMap.put("x*y", $xy))
$util.qr($myMap.put("x/y", $xDIVy))
$util.qr($myMap.put("x|y", $xMODy))

##arrays
#set($array = ["first"])
#set($idx = 0)
$util.qr($myMap.put("array", $array[$idx]))
##initialize array vals on create
#set($arr2 = [42, "a string", 21, "test"])
$util.qr($myMap.put("arr2", $arr2[$idx]))
$util.qr($myMap.put("isEmpty", $array.isEmpty()))  ##Returns false
$util.qr($myMap.put("size", $array.size()))
```

```
##Get and set items in an array
$util.qr($myMap.put("set", $array.set(0, 'changing array value')))
$util.qr($myMap.put("get", $array.get(0)))

##Lookup by name from a Map/dictionary in a similar way:
#set($result = {
    "Author" : "Nadia",
    "Topic" : "GraphQL"
})
$util.qr($myMap.put("Author", $result["Author"]))


##Conditional examples
#if(!$array.isEmpty())
$util.qr($myMap.put("ifCheck", "Array not empty"))
#else
$util.qr($myMap.put("ifCheck", "Your array is empty"))
#end

#if ($arr2.size() == 0)
$util.qr($myMap.put("elseIfCheck", "You forgot to put anything into this array!"))
#elseif ($arr2.size() == 1)
$util.qr($myMap.put("elseIfCheck", "Good start but please add more stuff"))
#else
$util.qr($myMap.put("elseIfCheck", "Good job!"))
#end

##Above showed negation(!) and equality (==), we can also use OR, AND, >, <, >=, <=,
 and !=
#set($T = true)
#set($F = false)
#if ($T || $F)
  $util.qr($myMap.put("OR", "TRUE"))
#end

#if ($T && $F)
  $util.qr($myMap.put("AND", "TRUE"))
#end

##Using the foreach loop counter - $foreach.count
#foreach ($item in $arr2)
  #set($idx = "item" + $foreach.count)
  $util.qr($myMap.put($idx, $item))
#end
```

```
##Using a Map and plucking out keys/vals
#set($hashmap = {
    "DynamoDB" : "https://aws.amazon.com/dynamodb/",
    "Amplify" : "https://github.com/aws/aws-amplify",
    "DynamoDB2" : "https://aws.amazon.com/dynamodb/",
    "Amplify2" : "https://github.com/aws/aws-amplify"
})

#foreach ($key in $hashmap.keySet())
    #if($foreach.count > 2)
        #break
    #end
    $util.qr($myMap.put($key, $hashmap.get($key)))
#end

##concatenate strings
#set($s1 = "Hello")
#set($s2 = " World")
$util.qr($myMap.put("concat","$s1$s2"))
$util.qr($myMap.put("concat2","Second $s1 World"))

$util.qr($myMap.put("context", $context))

{
    "version" : "2017-02-28",
    "operation": "Invoke",
    "payload": $util.toJson($myMap)
}
```

Template Respon

```
#set($data = {
"id" : "456"
})
#foreach($item in $context.result.entrySet())   ##$context.result is a MAP so we use
 entrySet()
    #if($item.key == "ifCheck")
        $util.qr($data.put("meta", "$item.value"))
    #end
#end

##Uncomment this out if you want to test and remove the below #if check
```

```
##$util.toJson($data)

#if($context.result["id"] == 123)
    $util.toJson($context.result)
  #else
    $util.unauthorized()
#end
```

Referensi konteks template pemetaan penyelesai

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSyncmendefinisikan satu set variabel dan fungsi untuk bekerja dengan template pemetaan resolver. Ini membuat operasi logis pada data lebih mudah dengan GraphQL. Dokumen ini menjelaskan fungsi-fungsi tersebut dan memberikan contoh untuk bekerja dengan template.

Mengakses **`$context`**

`$context`Variabel adalah peta yang menyimpan semua informasi kontekstual untuk pemanggilan resolver Anda. Ini memiliki struktur sebagai berikut:

```
{
   "arguments" : { ... },
   "source" : { ... },
   "result" : { ... },
   "identity" : { ... },
   "request" : { ... },
   "info": { ... }
}
```

> **Note**
>
> Jika Anda mencoba mengakses entri kamus/peta (seperti entri di`context`) dengan kuncinya untuk mengambil nilai, Velocity Template Language (VTL) memungkinkan Anda langsung menggunakan notasi. `<dictionary-element>.<key-name>` Namun, ini mungkin

tidak berfungsi untuk semua kasus, seperti ketika nama kunci memiliki karakter khusus (misalnya, garis bawah "_"). Kami menyarankan Anda selalu menggunakan `<dictionary-element>.get("<key-name>")` notasi.

Setiap bidang dalam `$context` peta didefinisikan sebagai berikut:

`$context`bidang

`arguments`

Peta yang berisi semua argumen GraphQL untuk bidang ini.

`identity`

Objek yang berisi informasi tentang penelepon. Untuk informasi selengkapnya tentang struktur bidang ini, lihat Identitas.

`source`

Peta yang berisi resolusi bidang induk.

`stash`

Stash adalah peta yang tersedia di dalam setiap resolver dan template pemetaan fungsi. Instans simpanan yang sama hidup melalui eksekusi resolver tunggal. Ini berarti Anda dapat menggunakan stash untuk meneruskan data arbitrer di seluruh template pemetaan permintaan dan respons, dan di seluruh fungsi dalam penyelesai pipeline. Stash mengekspos metode yang sama dengan struktur data Peta Java.

`result`

Wadah untuk hasil resolver ini. Bidang ini hanya tersedia untuk template pemetaan respons.

Misalnya, jika Anda menyelesaikan `author` bidang kueri berikut:

```
query {
    getPost(id: 1234) {
        postId
        title
        content
        author {
            id
```

```
            name
        }
    }
}
```

Kemudian `$context` variabel lengkap yang tersedia saat memproses template pemetaan respons mungkin:

```
{
  "arguments" : {
    id: "1234"
  },
  "source": {},
  "result" : {
      "postId": "1234",
      "title": "Some title",
      "content": "Some content",
      "author": {
        "id": "5678",
        "name": "Author Name"
      }
  },
  "identity" : {
      "sourceIp" : ["x.x.x.x"],
      "userArn" : "arn:aws:iam::123456789012:user/appsync",
      "accountId" : "666666666666",
      "user" : "AIDAAAAAAAAAAAAAAAAA"
  }
}
```

`prev.result`

Hasil dari operasi apa pun sebelumnya dijalankan dalam resolver pipa.

Jika operasi sebelumnya adalah template Sebelum pemetaan pipeline resolver, maka `$ctx.prev.result` mewakili output dari evaluasi template dan tersedia untuk fungsi pertama dalam pipeline.

Jika operasi sebelumnya adalah fungsi pertama, maka `$ctx.prev.result` mewakili output dari fungsi pertama dan tersedia untuk fungsi kedua dalam pipa.

Jika operasi sebelumnya adalah fungsi terakhir, maka `$ctx.prev.result` mewakili output dari fungsi terakhir dan tersedia untuk template After mapping resolver pipeline.

`info`

Objek yang berisi informasi tentang permintaan GraphQL. Untuk struktur bidang ini, lihat Info.

Identitas

`identity`Bagian ini berisi informasi tentang penelepon. Bentuk bagian ini tergantung pada jenis otorisasi AWS AppSync API Anda.

Untuk informasi selengkapnya tentang opsi AWS AppSync keamanan, lihat Otorisasi dan otentikasi.

`API_KEY`otorisasi

`identity`Bidang tidak dihuni.

`AWS_LAMBDA`otorisasi

Berisi `resolverContext` kunci, `identity` berisi `resolverContext` konten yang sama yang dikembalikan oleh fungsi Lambda yang mengotorisasi permintaan.

`AWS_IAM`otorisasi

Ini `identity` memiliki bentuk sebagai berikut:

```
{
    "accountId" : "string",
    "cognitoIdentityPoolId" : "string",
    "cognitoIdentityId" : "string",
    "sourceIp" : ["string"],
    "username" : "string", // IAM user principal
    "userArn" : "string",
    "cognitoIdentityAuthType" : "string", // authenticated/unauthenticated based on
 the identity type
    "cognitoIdentityAuthProvider" : "string" // the auth provider that was used to
 obtain the credentials
}
```

`AMAZON_COGNITO_USER_POOLS`otorisasi

Ini `identity` memiliki bentuk sebagai berikut:

```
{
    "sub" : "uuid",
```

```
    "issuer" : "string",
    "username" : "string"
    "claims" : { ... },
    "sourceIp" : ["x.x.x.x"],
    "defaultAuthStrategy" : "string"
}
```

Setiap bidang didefinisikan sebagai berikut:

`accountId`

ID AWS akun penelepon.

`claims`

Klaim yang dimiliki pengguna.

`cognitoIdentityAuthType`

Entah diautentikasi atau tidak diautentikasi berdasarkan tipe identitas.

`cognitoIdentityAuthProvider`

Daftar informasi penyedia identitas eksternal yang dipisahkan koma yang digunakan dalam memperoleh kredensil yang digunakan untuk menandatangani permintaan.

`cognitoIdentityId`

ID identitas Amazon Cognito dari penelepon.

`cognitoIdentityPoolId`

ID kumpulan identitas Amazon Cognito yang terkait dengan pemanggil.

`defaultAuthStrategy`

Strategi otorisasi default untuk penelepon ini (`ALLOW`atau`DENY`).

`issuer`

Penerbit token.

`sourceIp`

Alamat IP sumber penelepon yang AWS AppSync menerima. Jika permintaan tidak menyertakan `x-forwarded-for` header, nilai IP sumber hanya berisi satu alamat IP dari koneksi TCP. Jika

permintaan menyertakan `x-forwarded-for` header, IP sumber adalah daftar alamat IP dari `x-forwarded-for` header, selain alamat IP dari koneksi TCP.

`sub`

UUID dari pengguna yang diautentikasi.

`user`

Pengguna IAM.

`userArn`

Nama Sumber Daya Amazon (ARN) dari pengguna IAM.

`username`

Nama pengguna pengguna yang diautentikasi. Dalam hal `AMAZON_COGNITO_USER_POOLS` otorisasi, nilai nama pengguna adalah nilai atribut cognito:username. Dalam hal `AWS_IAM` otorisasi, nilai nama pengguna adalah nilai prinsipal AWS pengguna. Jika Anda menggunakan otorisasi IAM dengan kredensil yang dijual dari kumpulan identitas Amazon Cognito, kami sarankan Anda menggunakannya. `cognitoIdentityId`

Akses header permintaan

AWS AppSync mendukung meneruskan header khusus dari klien dan mengaksesnya di resolver GraphQL Anda dengan menggunakan. `$context.request.headers` Anda kemudian dapat menggunakan nilai header untuk tindakan seperti memasukkan data ke sumber data atau pemeriksaan otorisasi. Anda dapat menggunakan header permintaan tunggal atau beberapa menggunakan `$curl` kunci API dari baris perintah, seperti yang ditunjukkan pada contoh berikut:

Contoh header tunggal

Misalkan Anda menetapkan header `custom` dengan nilai `nadia` seperti berikut:

```
curl -XPOST -H "Content-Type:application/graphql" -H "custom:nadia" -H "x-api-key:<API-
KEY-VALUE>" -d '{"query":"mutation { createEvent(name: \"demo\", when: \"Next Friday!
\", where: \"Here!\") {id name when where description}}"}' https://<ENDPOINT>/graphql
```

Ini kemudian dapat diakses dengan`$context.request.headers.custom`. Misalnya, mungkin dalam VTL berikut untuk DynamoDB:

```
"custom": $util.dynamodb.toDynamoDBJson($context.request.headers.custom)
```

Beberapa contoh header

Anda juga dapat meneruskan beberapa header dalam satu permintaan dan mengaksesnya di template pemetaan resolver. Misalnya, jika `custom` header diatur dengan dua nilai:

```
curl -XPOST -H "Content-Type:application/graphql" -H "custom:bailey" -H "custom:nadia"
 -H "x-api-key:<API-KEY-VALUE>" -d '{"query":"mutation { createEvent(name: \"demo
\", when: \"Next Friday!\", where: \"Here!\") {id name when where description}}"}'
 https://<ENDPOINT>/graphql
```

Anda kemudian dapat mengakses ini sebagai array, seperti`$context.request.headers.custom[1]`.

> **Note**
>
> AWS AppSynctidak mengekspos header cookie di`$context.request.headers`.

Akses permintaan nama domain kustom

AWS AppSyncmendukung konfigurasi domain kustom yang dapat Anda gunakan untuk mengakses GraphQL dan titik akhir real-time untuk API Anda. Saat membuat permintaan dengan nama domain khusus, Anda bisa mendapatkan nama domain menggunakan`$context.request.domainName`.

Saat menggunakan nama domain titik akhir GraphQL default, nilainya adalah. `null`

Info

`info`Bagian ini berisi informasi tentang permintaan GraphQL. Bagian ini memiliki bentuk sebagai berikut:

```
{
    "fieldName": "string",
    "parentTypeName": "string",
    "variables": { ... },
    "selectionSetList": ["string"],
    "selectionSetGraphQL": "string"
}
```

Setiap bidang didefinisikan sebagai berikut:

`fieldName`

Nama bidang yang saat ini sedang diselesaikan.

`parentTypeName`

Nama tipe induk untuk bidang yang saat ini sedang diselesaikan.

`variables`

Peta yang menampung semua variabel yang diteruskan ke permintaan GraphQL.

`selectionSetList`

Sebuah daftar representasi bidang dalam set pilihan GraphQL. Bidang yang dialias hanya direferensikan dengan nama alias, bukan nama bidang. Contoh berikut menunjukkan ini secara rinci.

`selectionSetGraphQL`

Sebuah representasi string dari set seleksi, diformat sebagai GraphQL skema definisi bahasa (SDL). Meskipun fragmen tidak digabungkan ke dalam kumpulan seleksi, fragmen sebaris dipertahankan, seperti yang ditunjukkan pada contoh berikut.

> **Note**
>
> Saat menggunakan `$utils.toJson()` on`context.info`, nilai yang `selectionSetGraphQL` dan `selectionSetList` kembali tidak diserialisasikan secara default.

Misalnya, jika Anda menyelesaikan `getPost` bidang kueri berikut:

```
query {
  getPost(id: $postId) {
    postId
    title
    secondTitle: title
    content
    author(id: $authorId) {
      authorId
```

```
      name
    }
    secondAuthor(id: "789") {
      authorId
    }
    ... on Post {
      inlineFrag: comments: {
        id
      }
    }
    ... postFrag
  }
}

fragment postFrag on Post {
  postFrag: comments: {
    id
  }
}
```

Kemudian `$context.info` variabel lengkap yang tersedia saat memproses template pemetaan mungkin:

```
{
  "fieldName": "getPost",
  "parentTypeName": "Query",
  "variables": {
    "postId": "123",
    "authorId": "456"
  },
  "selectionSetList": [
    "postId",
    "title",
    "secondTitle"
    "content",
    "author",
    "author/authorId",
    "author/name",
    "secondAuthor",
    "secondAuthor/authorId",
    "inlineFragComments",
    "inlineFragComments/id",
    "postFragComments",
```

```
    "postFragComments/id"
  ],
  "selectionSetGraphQL": "{\n  getPost(id: $postId) {\n    postId\n    title\n
 secondTitle: title\n    content\n    author(id: $authorId) {\n      authorId\n
 name\n    }\n    secondAuthor(id: \"789\") {\n      authorId\n    }\n    ... on Post
 {\n      inlineFrag: comments {\n        id\n      }\n    }\n    ... postFrag\n  }\n}"
}
```

`selectionSetList`mengekspos hanya bidang yang termasuk tipe saat ini. Jika tipe saat ini adalah antarmuka atau gabungan, hanya bidang yang dipilih milik antarmuka yang diekspos. Misalnya, diberikan skema berikut:

```
type Query {
    node(id: ID!): Node
}

interface Node {
    id: ID
}

type Post implements Node {
    id: ID
    title: String
    author: String
}

type Blog implements Node {
    id: ID
    title: String
    category: String
}
```

Dan query berikut:

```
query {
    node(id: "post1") {
        id
        ... on Post {
            title
        }

        ... on Blog {
```

```
            title
        }
    }
}
```

Saat memanggil `$ctx.info.selectionSetList` pada resolusi `Query.node` bidang, hanya yang `id` diekspos:

```
"selectionSetList": [
    "id"
]
```

Masukan sanitasi

Aplikasi harus membersihkan input yang tidak tepercaya untuk mencegah pihak eksternal menggunakan aplikasi di luar penggunaan yang dimaksudkan. Karena `$context` berisi input pengguna di properti seperti`$context.arguments`,,,,`$context.identity`, dan `$context.result` `$context.info.variables``$context.request.headers`, kehati-hatian harus dilakukan untuk membersihkan nilainya dalam templat pemetaan.

Karena template pemetaan mewakili JSON, sanitasi input mengambil bentuk melarikan diri dari karakter yang dicadangkan JSON dari string yang mewakili input pengguna. Praktik terbaik adalah menggunakan `$util.toJson()` utilitas untuk melarikan diri dari karakter yang dicadangkan JSON dari nilai string sensitif saat menempatkannya ke dalam templat pemetaan.

Misalnya, dalam template pemetaan permintaan Lambda berikut, karena kami mengakses string input pelanggan yang tidak aman (`$context.arguments.id`), kami membungkusnya dengan `$util.toJson()` untuk mencegah karakter JSON yang tidak lolos dari melanggar template JSON.

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "postId": $util.toJson($context.arguments.id)
    }
}
```

Berbeda dengan template pemetaan di bawah ini, di mana kita langsung menyisipkan `$context.arguments.id` tanpa sanitasi. Ini tidak berfungsi untuk string yang berisi tanda kutip

yang tidak di-escaped atau karakter cadangan JSON lainnya, dan dapat membiarkan template Anda terbuka untuk kegagalan.

```
## DO NOT DO THIS
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "postId": "$context.arguments.id" ## Unsafe! Do not insert $context string
 values without escaping JSON characters.
    }
}
```

Referensi utilitas template pemetaan penyelesai

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSyncmendefinisikan satu set utilitas yang dapat Anda gunakan dalam resolver GraphQL untuk menyederhanakan interaksi dengan sumber data. Beberapa utilitas ini untuk penggunaan umum dengan sumber data apa pun, seperti menghasilkan ID atau stempel waktu. Lainnya khusus untuk jenis sumber data.

Topik

- Pembantu utilitas di $ util
- AWS AppSync arahan
- Pembantu waktu di $ util.time
- Daftar pembantu di $util.list
- Pembantu peta di $ util.map
- Pembantu DynamoDB di $util.dynamodb
- Pembantu Amazon RDS di $util.rds
- Pembantu HTTP di $ util.http
- Pembantu XMLdi $ util.xml.

Pembantu utilitas di $ util

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util`Variabel berisi metode utilitas umum untuk membantu Anda bekerja dengan data. Kecuali ditentukan lain, semua utilitas menggunakan set karakter UTF-8.

utilitas penguraian JSON

JSON menguraikan daftar utilitas

`$util.parseJson(String) : Object`

Mengambil “stringified” JSON dan mengembalikan representasi objek dari hasilnya.

`$util.toJson(Object) : String`

Mengambil objek dan mengembalikan representasi JSON “stringified” dari objek itu.

Pengkodean utilitas

Pengkodean daftar utilitas

`$util.urlEncode(String) : String`

Mengembalikan string masukan sebagai string `application/x-www-form-urlencoded` dikodekan.

`$util.urlDecode(String) : String`

Mendekode string yang `application/x-www-form-urlencoded` dikodekan kembali ke bentuk yang tidak dikodekan.

`$util.base64Encode( byte[] ) : String`

Mengkodekan input ke dalam string yang dikodekan base64.

`$util.base64Decode(String) : byte[]`

Mendekode data dari string yang dikodekan base64.

Utilitas pembuatan ID

Daftar utilitas pembuatan ID

`$util.autoId() : String`

Mengembalikan UUID 128-bit yang dihasilkan secara acak.

`$util.autoUlid() : String`

Mengembalikan ULID 128-bit yang dihasilkan secara acak (Universalally Unique Lexicographically Sortable Identifier).

`$util.autoKsuid() : String`

Mengembalikan 128-bit yang dihasilkan secara acak KSUID (K-Sortable Unique Identifier) base62 dikodekan sebagai String dengan panjang 27.

Kesalahan utils

Daftar utilitas kesalahan

`$util.error(String)`

Melempar kesalahan khusus. Gunakan ini dalam templat pemetaan permintaan atau respons untuk mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan.

`$util.error(String, String)`

Melempar kesalahan khusus. Gunakan ini dalam templat pemetaan permintaan atau respons untuk mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Anda juga dapat menentukan file`errorType`.

`$util.error(String, String, Object)`

Melempar kesalahan khusus. Gunakan ini dalam templat pemetaan permintaan atau respons untuk mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Anda juga

dapat menentukan `errorType` dan `data` bidang. `data`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL.

> **Note**
>
> `data`akan disaring berdasarkan set pemilihan kueri.

`$util.error(String, String, Object, Object)`

Melempar kesalahan khusus. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Selain itu, `errorType` bidang, `data` bidang, dan `errorInfo` bidang dapat ditentukan. `data`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL.

> **Note**
>
> `data`akan disaring berdasarkan set pemilihan kueri. `errorInfo`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL.
> `errorInfo`TIDAK akan difilter berdasarkan set pemilihan kueri.

`$util.appendError(String)`

Menambahkan kesalahan kustom. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Tidak seperti`$util.error(String)`, evaluasi template tidak akan terganggu, sehingga data dapat dikembalikan ke penelepon.

`$util.appendError(String, String)`

Menambahkan kesalahan kustom. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Selain itu, `errorType` dapat ditentukan. Tidak seperti`$util.error(String, String)`, evaluasi template tidak akan terganggu, sehingga data dapat dikembalikan ke penelepon.

`$util.appendError(String, String, Object)`

Menambahkan kesalahan kustom. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil

pemanggilan. Selain itu, bidang `errorType` dan `data` bidang dapat ditentukan. Tidak seperti`$util.error(String, String, Object)`, evaluasi template tidak akan terganggu, sehingga data dapat dikembalikan ke penelepon. `data`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL.

> **Note**
>
> `data`akan disaring berdasarkan set pemilihan kueri.

`$util.appendError(String, String, Object, Object)`

Menambahkan kesalahan kustom. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Selain itu, `errorType` bidang, `data` bidang, dan `errorInfo` bidang dapat ditentukan. Tidak seperti`$util.error(String, String, Object, Object)`, evaluasi template tidak akan terganggu, sehingga data dapat dikembalikan ke penelepon. `data`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL.

> **Note**
>
> `data`akan disaring berdasarkan set pemilihan kueri. `errorInfo`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL. `errorInfo`TIDAK akan difilter berdasarkan set pemilihan kueri.

Utils validasi kondisi

Daftar utilitas validasi kondisi

`$util.validate(Boolean, String) : void`

Jika kondisinya salah, lempar a CustomTemplateException dengan pesan yang ditentukan.

`$util.validate(Boolean, String, String) : void`

Jika kondisinya salah, lempar a CustomTemplateException dengan pesan dan jenis kesalahan yang ditentukan.

```
$util.validate(Boolean, String, String, Object) : void
```

Jika kondisinya salah, lempar a CustomTemplateException dengan pesan dan jenis kesalahan yang ditentukan, serta data untuk kembali dalam respons.

Utils perilaku nol

Daftar utilitas perilaku nol

```
$util.isNull(Object) : Boolean
```

Mengembalikan nilai true jika objek yang disediakan adalah null.

```
$util.isNullOrEmpty(String) : Boolean
```

Mengembalikan nilai true jika data yang disediakan adalah null atau string kosong. Jika tidak, mengembalikan false.

```
$util.isNullOrBlank(String) : Boolean
```

Mengembalikan nilai true jika data yang disediakan adalah null atau string kosong. Jika tidak, mengembalikan false.

```
$util.defaultIfNull(Object, Object) : Object
```

Mengembalikan Object pertama jika tidak null. Jika tidak, mengembalikan objek kedua sebagai “Objek default”.

```
$util.defaultIfNullOrEmpty(String, String) : String
```

Mengembalikan String pertama jika tidak null atau kosong. Jika tidak, mengembalikan String kedua sebagai “String default”.

```
$util.defaultIfNullOrBlank(String, String) : String
```

Mengembalikan String pertama jika tidak null atau kosong. Jika tidak, mengembalikan String kedua sebagai “String default”.

Utils pencocokan pola

Jenis dan pola pencocokan daftar utilitas

`$util.typeOf(Object) : String`

Mengembalikan String menggambarkan jenis Object. Identifikasi tipe yang didukung adalah: “Null”, “Number”, “String”, “Map”, “List”, “Boolean”. Jika suatu tipe tidak dapat diidentifikasi, tipe yang dikembalikan adalah “Objek”.

`$util.matches(String, String) : Boolean`

Mengembalikan nilai true jika pola yang ditentukan dalam argumen pertama cocok dengan data yang disediakan dalam argumen kedua. Pola harus berupa ekspresi reguler seperti`$util.matches("a*b", "aaaaab")`. Fungsionalitas ini didasarkan pada Pola, yang dapat Anda referensikan untuk dokumentasi lebih lanjut.

`$util.authType() : String`

Mengembalikan String yang menjelaskan jenis multi-auth yang digunakan oleh permintaan, mengembalikan “IAM Authorization”, “User Pool Authorization”, “Open ID Connect Authorization”, atau “API Key Authorization”.

Utils validasi objek

Daftar utilitas validasi objek

`$util.isString(Object) : Boolean`

Mengembalikan nilai true jika Object adalah String.

`$util.isNumber(Object) : Boolean`

Mengembalikan nilai true jika Object adalah Number.

`$util.isBoolean(Object) : Boolean`

Mengembalikan nilai true jika Object adalah Boolean.

`$util.isList(Object) : Boolean`

Mengembalikan nilai true jika Object adalah List.

`$util.isMap(Object) : Boolean`

Mengembalikan nilai true jika Object adalah Peta.

CloudWatch utilitas logging

CloudWatch daftar utilitas log

`$util.log.info(Object) : Void`

Mencatat representasi String dari Objek yang disediakan ke aliran log yang diminta saat pencatatan tingkat permintaan dan tingkat bidang diaktifkan dengan tingkat CloudWatch log pada API. `ALL`

`$util.log.info(String, Object...) : Void`

Mencatat representasi String dari Objek yang disediakan ke aliran log yang diminta saat pencatatan tingkat permintaan dan tingkat bidang diaktifkan dengan tingkat CloudWatch log pada API. `ALL` Utilitas ini akan menggantikan semua variabel yang ditunjukkan oleh “{}” dalam format input pertama String dengan representasi String dari Objek yang disediakan secara berurutan.

`$util.log.error(Object) : Void`

Mencatat representasi String dari Objek yang disediakan ke aliran log yang diminta saat CloudWatch logging tingkat bidang diaktifkan dengan level log `ERROR` atau level log `ALL` pada API.

`$util.log.error(String, Object...) : Void`

Mencatat representasi String dari Objek yang disediakan ke aliran log yang diminta saat CloudWatch logging tingkat bidang diaktifkan dengan level log `ERROR` atau level log `ALL` pada API. Utilitas ini akan menggantikan semua variabel yang ditunjukkan oleh “{}” dalam format input pertama String dengan representasi String dari Objek yang disediakan secara berurutan.

Kembalikan utilitas perilaku nilai

Kembalikan daftar utilitas perilaku nilai

`$util.qr()` dan `$util.quiet()`

Menjalankan pernyataan VTL sambil menekan nilai yang dikembalikan. Ini berguna untuk menjalankan metode tanpa menggunakan placeholder sementara, seperti menambahkan item ke peta. Sebagai contoh:

```
#set ($myMap = {})
```

```
#set($discard = $myMap.put("id", "first value"))
```

Menjadi:

```
#set ($myMap = {})
$util.qr($myMap.put("id", "first value"))
```

`$util.escapeJavaScript(String) : String`

Mengembalikan string masukan sebagai string JavaScript lolos.

`$util.urlEncode(String) : String`

Mengembalikan string masukan sebagai string `application/x-www-form-urlencoded` dikodekan.

`$util.urlDecode(String) : String`

Mendekode string yang `application/x-www-form-urlencoded` dikodekan kembali ke bentuk yang tidak dikodekan.

`$util.base64Encode( byte[] ) : String`

Mengkodekan input ke dalam string yang dikodekan base64.

`$util.base64Decode(String) : byte[]`

Mendekode data dari string yang dikodekan base64.

`$util.parseJson(String) : Object`

Mengambil “stringified” JSON dan mengembalikan representasi objek dari hasilnya.

`$util.toJson(Object) : String`

Mengambil objek dan mengembalikan representasi JSON “stringified” dari objek itu.

`$util.autoId() : String`

Mengembalikan UUID 128-bit yang dihasilkan secara acak.

`$util.autoUlid() : String`

Mengembalikan ULID 128-bit yang dihasilkan secara acak (Universalally Unique Lexicographically Sortable Identifier).

`$util.autoKsuid() : String`

Mengembalikan 128-bit yang dihasilkan secara acak KSUID (K-Sortable Unique Identifier) base62 dikodekan sebagai String dengan panjang 27.

`$util.unauthorized()`

Lemparan `Unauthorized` untuk bidang yang sedang diselesaikan. Gunakan ini dalam templat pemetaan permintaan atau respons untuk menentukan apakah akan mengizinkan pemanggil menyelesaikan bidang.

`$util.error(String)`

Melempar kesalahan khusus. Gunakan ini dalam templat pemetaan permintaan atau respons untuk mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan.

`$util.error(String, String)`

Melempar kesalahan khusus. Gunakan ini dalam templat pemetaan permintaan atau respons untuk mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Anda juga dapat menentukan file`errorType`.

`$util.error(String, String, Object)`

Melempar kesalahan khusus. Gunakan ini dalam templat pemetaan permintaan atau respons untuk mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Anda juga dapat menentukan `errorType` dan `data` bidang. `data`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL. Catatan: `data` akan difilter berdasarkan set pemilihan kueri.

`$util.error(String, String, Object, Object)`

Melempar kesalahan khusus. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Selain itu, `errorType` bidang, `data` bidang, dan `errorInfo` bidang dapat ditentukan. `data`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL. Catatan: `data` akan difilter berdasarkan set pemilihan kueri. `errorInfo`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL. Catatan: TIDAK **`errorInfo`** akan difilter berdasarkan set pemilihan kueri.

`$util.appendError(String)`

Menambahkan kesalahan kustom. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil

pemanggilan. Tidak seperti`$util.error(String)`, evaluasi template tidak akan terganggu, sehingga data dapat dikembalikan ke penelepon.

`$util.appendError(String, String)`

Menambahkan kesalahan kustom. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Selain itu, `errorType` dapat ditentukan. Tidak seperti`$util.error(String, String)`, evaluasi template tidak akan terganggu, sehingga data dapat dikembalikan ke penelepon.

`$util.appendError(String, String, Object)`

Menambahkan kesalahan kustom. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Selain itu, bidang `errorType` dan `data` bidang dapat ditentukan. Tidak seperti`$util.error(String, String, Object)`, evaluasi template tidak akan terganggu, sehingga data dapat dikembalikan ke penelepon. `data`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL. Catatan: `data` akan difilter berdasarkan set pemilihan kueri.

`$util.appendError(String, String, Object, Object)`

Menambahkan kesalahan kustom. Ini dapat digunakan dalam template pemetaan permintaan atau respons jika template mendeteksi kesalahan dengan permintaan atau dengan hasil pemanggilan. Selain itu, `errorType` bidang, `data` bidang, dan `errorInfo` bidang dapat ditentukan. Tidak seperti`$util.error(String, String, Object, Object)`, evaluasi template tidak akan terganggu, sehingga data dapat dikembalikan ke penelepon. `data`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL. Catatan: `data` akan difilter berdasarkan set pemilihan kueri. `errorInfo`Nilai akan ditambahkan ke `error` blok yang sesuai di `errors` dalam respons GraphQL. Catatan: TIDAK **`errorInfo`** akan difilter berdasarkan set pemilihan kueri.

`$util.validate(Boolean, String) : void`

Jika kondisinya salah, lempar a CustomTemplateException dengan pesan yang ditentukan.

`$util.validate(Boolean, String, String) : void`

Jika kondisinya salah, lempar a CustomTemplateException dengan pesan dan jenis kesalahan yang ditentukan.

$util.validate(Boolean, String, String, Object) : void

Jika kondisinya salah, lempar a CustomTemplateException dengan pesan dan jenis kesalahan yang ditentukan, serta data untuk kembali dalam respons.

$util.isNull(Object) : Boolean

Mengembalikan nilai true jika objek yang disediakan adalah null.

$util.isNullOrEmpty(String) : Boolean

Mengembalikan nilai true jika data yang disediakan adalah null atau string kosong. Jika tidak, mengembalikan false.

$util.isNullOrBlank(String) : Boolean

Mengembalikan nilai true jika data yang disediakan adalah null atau string kosong. Jika tidak, mengembalikan false.

$util.defaultIfNull(Object, Object) : Object

Mengembalikan Object pertama jika tidak null. Jika tidak, mengembalikan objek kedua sebagai “Objek default”.

$util.defaultIfNullOrEmpty(String, String) : String

Mengembalikan String pertama jika tidak null atau kosong. Jika tidak, mengembalikan String kedua sebagai “String default”.

$util.defaultIfNullOrBlank(String, String) : String

Mengembalikan String pertama jika tidak null atau kosong. Jika tidak, mengembalikan String kedua sebagai “String default”.

$util.isString(Object) : Boolean

Mengembalikan nilai true jika Object adalah String.

$util.isNumber(Object) : Boolean

Mengembalikan nilai true jika Object adalah Number.

$util.isBoolean(Object) : Boolean

Mengembalikan nilai true jika Object adalah Boolean.

`$util.isList(Object) : Boolean`

Mengembalikan nilai true jika Object adalah Daftar.

`$util.isMap(Object) : Boolean`

Mengembalikan nilai true jika Object adalah Peta.

`$util.typeOf(Object) : String`

Mengembalikan String menggambarkan jenis Object. Identifikasi tipe yang didukung adalah: “Null”, “Number”, “String”, “Map”, “List”, “Boolean”. Jika suatu tipe tidak dapat diidentifikasi, tipe yang dikembalikan adalah “Objek”.

`$util.matches(String, String) : Boolean`

Mengembalikan nilai true jika pola yang ditentukan dalam argumen pertama cocok dengan data yang disediakan dalam argumen kedua. Pola harus berupa ekspresi reguler seperti`$util.matches("a*b", "aaaaab")`. Fungsionalitas ini didasarkan pada Pola, yang dapat Anda referensikan untuk dokumentasi lebih lanjut.

`$util.authType() : String`

Mengembalikan String yang menjelaskan jenis multi-auth yang digunakan oleh permintaan, mengembalikan “IAM Authorization”, “User Pool Authorization”, “Open ID Connect Authorization”, atau “API Key Authorization”.

`$util.log.info(Object) : Void`

Mencatat representasi String dari Objek yang disediakan ke aliran log yang diminta saat pencatatan tingkat permintaan dan tingkat bidang diaktifkan dengan tingkat CloudWatch log pada API. `ALL`

`$util.log.info(String, Object...) : Void`

Mencatat representasi String dari Objek yang disediakan ke aliran log yang diminta saat pencatatan tingkat permintaan dan tingkat bidang diaktifkan dengan tingkat CloudWatch log pada API. `ALL` Utilitas ini akan menggantikan semua variabel yang ditunjukkan oleh “{}” dalam format input pertama String dengan representasi String dari Objek yang disediakan secara berurutan.

`$util.log.error(Object) : Void`

Mencatat representasi String dari Objek yang disediakan ke aliran log yang diminta saat CloudWatch logging tingkat bidang diaktifkan dengan level log `ERROR` atau level log `ALL` pada API.

`$util.log.error(String, Object...) : Void`

Mencatat representasi String dari Objek yang disediakan ke aliran log yang diminta saat CloudWatch logging tingkat bidang diaktifkan dengan level log `ERROR` atau level log `ALL` pada API. Utilitas ini akan menggantikan semua variabel yang ditunjukkan oleh "{}" dalam format input pertama String dengan representasi String dari Objek yang disediakan secara berurutan.

`$util.escapeJavaScript(String) : String`

Mengembalikan string masukan sebagai string JavaScript lolos.

Otorisasi penyelesai

Daftar otorisasi penyelesai

`$util.unauthorized()`

Lemparan `Unauthorized` untuk bidang yang sedang diselesaikan. Gunakan ini dalam templat pemetaan permintaan atau respons untuk menentukan apakah akan mengizinkan pemanggil menyelesaikan bidang.

AWS AppSync arahan

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync mengekspos arahan untuk memfasilitasi produktivitas pengembang saat menulis di VTL.

Directive utils

`#return(Object)`

`#return(Object)`Ini memungkinkan Anda untuk kembali sebelum waktunya dari template pemetaan apa pun. `#return(Object)`analog dengan kata kunci return dalam bahasa pemrograman, karena akan kembali dari blok logika cakupan terdekat. Menggunakan

`#return(Object)` bagian dalam template pemetaan resolver akan kembali dari resolver. Selain itu, menggunakan `#return(Object)` dari template pemetaan fungsi akan kembali dari fungsi dan akan melanjutkan proses ke fungsi berikutnya di pipeline atau template pemetaan respons resolver.

`#return`

`#return`Arahan menunjukkan perilaku yang sama seperti`#return(Object)`, tetapi `null` akan dikembalikan sebagai gantinya.

Pembantu waktu di $ util.time

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util.time`Variabel berisi metode datetime untuk membantu menghasilkan stempel waktu, mengonversi antara format datetime, dan mengurai string datetime. Sintaks untuk format datetime didasarkan pada DateTimeFormattermana Anda dapat referensi untuk dokumentasi lebih lanjut. Kami memberikan beberapa contoh di bawah ini, serta daftar metode dan deskripsi yang tersedia.

Penggunaan waktu

Daftar utilitas waktu

`$util.time.nowISO8601() : String`

Mengembalikan representasi String dari UTC dalam format ISO8601.

`$util.time.nowEpochSeconds() : long`

Mengembalikan jumlah detik dari epoch 1970-01-01T 00:00:00 Z ke sekarang.

`$util.time.nowEpochMilliSeconds() : long`

Mengembalikan jumlah milidetik dari epoch 1970-01-01T 00:00:00 Z ke sekarang.

`$util.time.nowFormatted(String) : String`

Mengembalikan string timestamp saat ini di UTC menggunakan format yang ditentukan dari tipe input String.

`$util.time.nowFormatted(String, String) : String`

Mengembalikan string timestamp saat ini untuk zona waktu menggunakan format yang ditentukan dan zona waktu dari jenis input String.

`$util.time.parseFormattedToEpochMilliSeconds(String, String) : Long`

Mem-parsing stempel waktu yang diteruskan sebagai String bersama dengan format, lalu mengembalikan stempel waktu sebagai milidetik sejak epoch.

`$util.time.parseFormattedToEpochMilliSeconds(String, String, String) : Long`

Mem-parsing stempel waktu yang diteruskan sebagai String bersama dengan format dan zona waktu, lalu mengembalikan stempel waktu sebagai milidetik sejak epoch.

`$util.time.parseISO8601ToEpochMilliSeconds(String) : Long`

Mem-parsing stempel waktu ISO8601 yang diteruskan sebagai String, lalu mengembalikan stempel waktu sebagai milidetik sejak epoch.

`$util.time.epochMilliSecondsToSeconds(long) : long`

Mengonversi stempel waktu milidetik epoch menjadi stempel waktu epoch detik.

`$util.time.epochMilliSecondsToISO8601(long) : String`

Mengonversi stempel waktu milidetik epoch menjadi stempel waktu ISO8601.

`$util.time.epochMilliSecondsToFormatted(long, String) : String`

Mengonversi stempel waktu milidetik epoch, diteruskan selama, ke stempel waktu yang diformat sesuai dengan format yang disediakan di UTC.

`$util.time.epochMilliSecondsToFormatted(long, String, String) : String`

Mengonversi stempel waktu milidetik epoch, diteruskan sebagai panjang, ke stempel waktu yang diformat sesuai dengan format yang disediakan di zona waktu yang disediakan.

Contoh fungsi mandiri

```
$util.time.nowISO8601()                                          :
 2018-02-06T19:01:35.749Z
$util.time.nowEpochSeconds()                                     : 1517943695
$util.time.nowEpochMilliSeconds()                                : 1517943695750
```

```
$util.time.nowFormatted("yyyy-MM-dd HH:mm:ssZ")                    : 2018-02-06
 19:01:35+0000
$util.time.nowFormatted("yyyy-MM-dd HH:mm:ssZ", "+08:00")          : 2018-02-07
 03:01:35+0800
$util.time.nowFormatted("yyyy-MM-dd HH:mm:ssZ", "Australia/Perth") : 2018-02-07
 03:01:35+0800
```

Contoh konversi

```
#set( $nowEpochMillis = 1517943695758 )
$util.time.epochMilliSecondsToSeconds($nowEpochMillis)
    : 1517943695
$util.time.epochMilliSecondsToISO8601($nowEpochMillis)
    : 2018-02-06T19:01:35.758Z
$util.time.epochMilliSecondsToFormatted($nowEpochMillis, "yyyy-MM-dd HH:mm:ssZ")
    : 2018-02-06 19:01:35+0000
$util.time.epochMilliSecondsToFormatted($nowEpochMillis, "yyyy-MM-dd HH:mm:ssZ",
 "+08:00") : 2018-02-07 03:01:35+0800
```

Contoh penguraian

```
$util.time.parseISO8601ToEpochMilliSeconds("2018-02-01T17:21:05.180+08:00")
              : 1517476865180
$util.time.parseFormattedToEpochMilliSeconds("2018-02-02 01:19:22+0800", "yyyy-MM-dd
 HH:mm:ssZ")     : 1517505562000
$util.time.parseFormattedToEpochMilliSeconds("2018-02-02 01:19:22", "yyyy-MM-dd
 HH:mm:ss", "+08:00") : 1517505562000
```

Penggunaan dengan skalar AWS AppSync yang ditentukan

Format berikut kompatibel dengan`AWSDate`,`AWSDateTime`, dan`AWSTime`.

```
$util.time.nowFormatted("yyyy-MM-dd[XXX]", "-07:00:30")              :
 2018-07-11-07:00
$util.time.nowFormatted("yyyy-MM-dd'T'HH:mm:ss[XXXXX]", "-07:00:30")  :
 2018-07-11T15:14:15-07:00:30
```

Daftar pembantu di $util.list

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util.list`berisi metode untuk membantu operasi Daftar umum seperti menghapus atau mempertahankan item dari daftar untuk memfilter kasus penggunaan.

Daftar utilitas

`$util.list.copyAndRetainAll(List, List) : List`

Membuat salinan dangkal dari daftar yang disediakan dalam argumen pertama sambil mempertahankan hanya item yang ditentukan dalam argumen kedua, jika ada. Semua item lainnya akan dihapus dari salinan.

`$util.list.copyAndRemoveAll(List, List) : List`

Membuat salinan dangkal dari daftar yang disediakan dalam argumen pertama sambil menghapus item apa pun di mana item ditentukan dalam argumen kedua, jika ada. Semua item lainnya akan disimpan dalam salinan.

`$util.list.sortList(List, Boolean, String) : List`

Mengurutkan daftar objek, yang disediakan dalam argumen pertama. Jika argumen kedua benar, daftar diurutkan dengan cara menurun; jika argumen kedua salah, daftar diurutkan dengan cara menaik. Argumen ketiga adalah nama string dari properti yang digunakan untuk mengurutkan daftar objek kustom. Jika itu adalah daftar hanya Strings, Integers, Floats, atau Doubles, argumen ketiga dapat berupa string acak. Jika semua objek tidak dari kelas yang sama, daftar asli dikembalikan. Hanya daftar yang berisi maksimal 1000 objek yang didukung. Berikut ini adalah contoh penggunaan utilitas ini:

```
 INPUT:      $util.list.sortList([{"description":"youngest", "age":5},
{"description":"middle", "age":45}, {"description":"oldest", "age":85}], false,
 "description")
 OUTPUT:     [{"description":"middle", "age":45}, {"description":"oldest",
 "age":85}, {"description":"youngest", "age":5}]
```

Pembantu peta di $ util.map

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util.map`berisi metode untuk membantu operasi Peta umum seperti menghapus atau mempertahankan item dari Peta untuk memfilter kasus penggunaan.

Peta utils

`$util.map.copyAndRetainAllKeys(Map, List) : Map`

Membuat salinan dangkal dari peta pertama sambil mempertahankan hanya kunci yang ditentukan dalam daftar, jika ada. Semua kunci lainnya akan dihapus dari salinan.

`$util.map.copyAndRemoveAllKeys(Map, List) : Map`

Membuat salinan dangkal dari peta pertama sambil menghapus entri di mana kunci ditentukan dalam daftar, jika ada. Semua kunci lainnya akan disimpan dalam salinan.

Pembantu DynamoDB di $util.dynamodb

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util.dynamodb`berisi metode pembantu yang memudahkan untuk menulis dan membaca data ke Amazon DynamoDB, seperti pemetaan dan pemformatan tipe otomatis. Metode ini dirancang untuk membuat pemetaan tipe primitif dan Daftar ke format input DynamoDB yang tepat secara otomatis, yang merupakan format. `Map { "TYPE" : VALUE }`

Misalnya, sebelumnya, template pemetaan permintaan untuk membuat item baru di DynamoDB mungkin terlihat seperti ini:

```
{
```

```
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key": {
        "id" : { "S" : "$util.autoId()" }
    },
    "attributeValues" : {
         "title" : { "S" : $util.toJson($ctx.args.title) },
         "author" : { "S" : $util.toJson($ctx.args.author) },
         "version" : { "N", $util.toJson($ctx.args.version) }
    }
}
```

Jika kita ingin menambahkan bidang ke objek, kita harus memperbarui kueri GraphQL dalam skema, serta template pemetaan permintaan. Namun, kami sekarang dapat merestrukturisasi template pemetaan permintaan kami sehingga secara otomatis mengambil bidang baru yang ditambahkan dalam skema kami dan menambahkannya ke DynamoDB dengan tipe yang benar:

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key": {
        "id" : $util.dynamodb.toDynamoDBJson($util.autoId())
    },
    "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
}
```

Pada contoh sebelumnya, kita menggunakan `$util.dynamodb.toDynamoDBJson(...)` helper untuk secara otomatis mengambil id yang dihasilkan dan mengubahnya menjadi representasi DynamoDB dari atribut string. Kami kemudian mengambil semua argumen dan mengonversinya ke representasi DynamoDB mereka dan mengeluarkannya ke bidang `attributeValues` di template.

Setiap helper memiliki dua versi: versi yang mengembalikan objek (misalnya,`$util.dynamodb.toString(...)`), dan versi yang mengembalikan objek sebagai string JSON (misalnya,`$util.dynamodb.toStringJson(...)`). Pada contoh sebelumnya, kita menggunakan versi yang mengembalikan data sebagai string JSON. Jika Anda ingin memanipulasi objek sebelum digunakan dalam template, Anda dapat memilih untuk mengembalikan objek sebagai gantinya, seperti yang ditunjukkan berikut:

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
```

```
    "key": {
        "id" : $util.dynamodb.toDynamoDBJson($util.autoId())
    },

    #set( $myFoo = $util.dynamodb.toMapValues($ctx.args) )
    #set( $myFoo.version = $util.dynamodb.toNumber(1) )
    #set( $myFoo.timestamp = $util.dynamodb.toString($util.time.nowISO8601()))

    "attributeValues" : $util.toJson($myFoo)
}
```

Pada contoh sebelumnya, kita mengembalikan argumen yang dikonversi sebagai peta alih-alih string JSON, dan kemudian menambahkan `timestamp` bidang `version` and sebelum akhirnya mengeluarkannya ke `attributeValues` bidang dalam template menggunakan. `$util.toJson(...)`

Versi JSON dari masing-masing pembantu setara dengan membungkus versi non-JSON. `$util.toJson(...)` Misalnya, pernyataan berikut persis sama:

```
$util.toStringJson("Hello, World!")
$util.toJson($util.toString("Hello, World!"))
```

ToDynamoDB

Daftar utilitas ToDynamoDB

`$util.dynamodb.toDynamoDB(Object) : Map`

Alat konversi objek umum untuk DynamoDB yang mengubah objek masukan ke representasi DynamoDB yang sesuai. Ini berpendirian tentang bagaimana itu mewakili beberapa jenis: misalnya, itu akan menggunakan daftar (“L”) daripada set (“SS”, “NS”, “BS”). Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

Contoh string

```
Input:      $util.dynamodb.toDynamoDB("foo")
Output:     { "S" : "foo" }
```

Contoh angka

```
Input:      $util.dynamodb.toDynamoDB(12345)
```

```
Output:     { "N" : 12345 }
```

Contoh Boolean

```
Input:      $util.dynamodb.toDynamoDB(true)
Output:     { "BOOL" : true }
```

Daftar contoh

```
Input:      $util.dynamodb.toDynamoDB([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

Contoh peta

```
Input:      $util.dynamodb.toDynamoDB({ "foo": "bar", "baz" : 1234, "beep":
 [ "boop"] })
Output:     {
               "M" : {
                   "foo"  : { "S" : "bar" },
                   "baz"  : { "N" : 1234 },
                   "beep" : {
                       "L" : [
                           { "S" : "boop" }
                       ]
                   }
               }
           }
```

`$util.dynamodb.toDynamoDBJson(Object) : String`

Sama seperti`$util.dynamodb.toDynamoDB(Object) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

ToString utilitas

Daftar utilitas ToString

`$util.dynamodb.toString(String) : String`

Mengkonversi string input ke format string DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toString("foo")
Output:     { "S" : "foo" }
```

`$util.dynamodb.toStringJson(String) : Map`

Sama seperti`$util.dynamodb.toString(String) : String`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

`$util.dynamodb.toStringSet(List<String>) : Map`

Mengkonversi daftar dengan Strings ke format set string DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toStringSet([ "foo", "bar", "baz" ])
Output:     { "SS" : [ "foo", "bar", "baz" ] }
```

`$util.dynamodb.toStringSetJson(List<String>) : String`

Sama seperti`$util.dynamodb.toStringSet(List<String>) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

utilitas Tonumber

Daftar utilitas Tonumber

`$util.dynamodb.toNumber(Number) : Map`

Mengkonversi angka ke format nomor DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toNumber(12345)
Output:     { "N" : 12345 }
```

`$util.dynamodb.toNumberJson(Number) : String`

Sama seperti`$util.dynamodb.toNumber(Number) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

`$util.dynamodb.toNumberSet(List<Number>) : Map`

Mengkonversi daftar angka ke format set nomor DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toNumberSet([ 1, 23, 4.56 ])
Output:     { "NS" : [ 1, 23, 4.56 ] }
```

`$util.dynamodb.toNumberSetJson(List<Number>) : String`

Sama seperti`$util.dynamodb.toNumberSet(List<Number>) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

Kegunaan toBinary

Daftar utilitas toBinary

`$util.dynamodb.toBinary(String) : Map`

Mengkonversi data biner dikodekan sebagai string base64 ke format biner DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toBinary("foo")
Output:     { "B" : "foo" }
```

`$util.dynamodb.toBinaryJson(String) : String`

Sama seperti`$util.dynamodb.toBinary(String) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

`$util.dynamodb.toBinarySet(List<String>) : Map`

Mengkonversi daftar data biner yang dikodekan sebagai string base64 ke format set biner DynamoDB. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toBinarySet([ "foo", "bar", "baz" ])
```

```
Output:     { "BS" : [ "foo", "bar", "baz" ] }
```

`$util.dynamodb.toBinarySetJson(List<String>) : String`

Sama seperti`$util.dynamodb.toBinarySet(List<String>) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

utilitas ToBoolean

Daftar utilitas ToBoolean

`$util.dynamodb.toBoolean(Boolean) : Map`

Mengkonversi Boolean ke format DynamoDB Boolean yang sesuai. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toBoolean(true)
Output:     { "BOOL" : true }
```

`$util.dynamodb.toBooleanJson(Boolean) : String`

Sama seperti`$util.dynamodb.toBoolean(Boolean) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

utilitas ToNull

Daftar utilitas ToNull

`$util.dynamodb.toNull() : Map`

Mengembalikan null dalam format DynamoDB null. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toNull()
Output:     { "NULL" : null }
```

`$util.dynamodb.toNullJson() : String`

Sama seperti`$util.dynamodb.toNull() : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

utilitas ToList

Daftar utilitas ToList

`$util.dynamodb.toList(List) : Map`

Mengkonversi daftar objek ke format daftar DynamoDB. Setiap item dalam daftar juga dikonversi ke format DynamoDB yang sesuai. Ini berpendapat tentang bagaimana itu mewakili beberapa objek bersarang: misalnya, itu akan menggunakan daftar (“L”) daripada set (“SS”, “NS”, “BS”). Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toList([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

`$util.dynamodb.toListJson(List) : String`

Sama seperti`$util.dynamodb.toList(List) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

TomAp utilitas

Daftar utilitas TomAP

`$util.dynamodb.toMap(Map) : Map`

Mengkonversi peta ke format peta DynamoDB. Setiap nilai dalam peta juga dikonversi ke format DynamoDB yang sesuai. Ini berpendapat tentang bagaimana itu mewakili beberapa objek bersarang: misalnya, itu akan menggunakan daftar (“L”) daripada set (“SS”, “NS”, “BS”). Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toMap({ "foo": "bar", "baz" : 1234, "beep": [ "boop"] })
```

```
Output:     {
               "M" : {
                   "foo"  : { "S" : "bar" },
                   "baz"  : { "N" : 1234 },
                   "beep" : {
                       "L" : [
                           { "S" : "boop" }
                       ]
                   }
               }
           }
```

`$util.dynamodb.toMapJson(Map) : String`

Sama seperti`$util.dynamodb.toMap(Map) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

`$util.dynamodb.toMapValues(Map) : Map`

Membuat salinan peta di mana setiap nilai telah dikonversi ke format DynamoDB yang sesuai. Ini berpendapat tentang bagaimana itu mewakili beberapa objek bersarang: misalnya, itu akan menggunakan daftar (“L”) daripada set (“SS”, “NS”, “BS”).

```
Input:      $util.dynamodb.toMapValues({ "foo": "bar", "baz" : 1234, "beep":
 [ "boop"] })
Output:     {
               "foo"  : { "S" : "bar" },
               "baz"  : { "N" : 1234 },
               "beep" : {
                   "L" : [
                       { "S" : "boop" }
                   ]
               }
           }
```

> **Note**
>
> Ini sedikit berbeda `$util.dynamodb.toMap(Map) : Map` karena hanya mengembalikan isi dari nilai atribut DynamoDB, tetapi tidak seluruh nilai atribut itu sendiri. Misalnya, pernyataan berikut persis sama:
>
> ```
> $util.dynamodb.toMapValues($map)
> ```

```
$util.dynamodb.toMap($map).get("M")
```

`$util.dynamodb.toMapValuesJson(Map) : String`

Sama seperti`$util.dynamodb.toMapValues(Map) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

S3Object utilitas

Daftar utilitas S3Object

`$util.dynamodb.toS3object(String key, String bucket, String region) : Map`

Mengonversi kunci, bucket, dan wilayah menjadi representasi DynamoDB S3 Object. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toS3object("foo", "bar", region = "baz")
Output:     { "S" : "{ \"s3\" : { \"key\" : \"foo", \"bucket\" : \"bar", \"region
\" : \"baz" } }" }
```

`$util.dynamodb.toS3objectJson(String key, String bucket, String region) : String`

Sama seperti`$util.dynamodb.toS3object(String key, String bucket, String region) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

`$util.dynamodb.toS3object(String key, String bucket, String region, String version) : Map`

Mengonversi kunci, bucket, region, dan versi opsional menjadi representasi DynamoDB S3 Object. Ini mengembalikan sebuah objek yang menggambarkan nilai atribut DynamoDB.

```
Input:      $util.dynamodb.toS3object("foo", "bar", "baz", "beep")
Output:     { "S" : "{ \"s3\" : { \"key\" : \"foo\", \"bucket\" : \"bar\", \"region
\" : \"baz\", \"version\" = \"beep\" } }" }
```

`$util.dynamodb.toS3ObjectJson(String key, String bucket, String region, String version) : String`

Sama seperti`$util.dynamodb.toS3Object(String key, String bucket, String region, String version) : Map`, tetapi mengembalikan nilai atribut DynamoDB sebagai string JSON dikodekan.

`$util.dynamodb.fromS3ObjectJson(String) : Map`

Menerima nilai string Objek DynamoDB S3 dan mengembalikan peta yang berisi kunci, bucket, wilayah, dan versi opsional.

```
Input:      $util.dynamodb.fromS3ObjectJson({ "S" : "{ \"s3\" : { \"key\" : \"foo\",
 \"bucket\" : \"bar\", \"region\" : \"baz\", \"version\" = \"beep\" } }" })
Output:     { "key" : "foo", "bucket" : "bar", "region" : "baz", "version" :
 "beep" }
```

Pembantu Amazon RDS di $util.rds

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util.rds`berisi metode pembantu yang memformat operasi Amazon RDS dengan menyingkirkan data asing dalam output hasil

$util.rds daftar utils

`$util.rds.toJsonString(String serializedSQLResult): String`

Mengembalikan `String` dengan mengubah format hasil operasi API Data Amazon Relational Database Service (Amazon RDS) mentah stringified ke string yang lebih ringkas. String yang dikembalikan adalah daftar serial catatan SQL dari set hasil. Setiap catatan direpresentasikan sebagai kumpulan pasangan kunci-nilai. Kunci adalah nama kolom yang sesuai.

Jika pernyataan yang sesuai dalam input adalah kueri SQL yang menyebabkan mutasi (misalnya INSERT, UPDATE, DELETE), maka daftar kosong dikembalikan. Misalnya, kueri `select * from Books limit 2` memberikan hasil mentah dari operasi Data Amazon RDS:

```
{
    "sqlStatementResults": [
        {
            "numberOfRecordsUpdated": 0,
            "records": [
                [
                    {
                        "stringValue": "Mark Twain"
                    },
                    {
                        "stringValue": "Adventures of Huckleberry Finn"
                    },
                    {
                        "stringValue": "978-1948132817"
                    }
                ],
                [
                    {
                        "stringValue": "Jack London"
                    },
                    {
                        "stringValue": "The Call of the Wild"
                    },
                    {
                        "stringValue": "978-1948132275"
                    }
                  ]
            ],
            "columnMetadata": [
                {
                    "isSigned": false,
                    "isCurrency": false,
                    "label": "author",
                    "precision": 200,
                    "typeName": "VARCHAR",
                    "scale": 0,
                    "isAutoIncrement": false,
                    "isCaseSensitive": false,
                    "schemaName": "",
                    "tableName": "Books",
                    "type": 12,
                    "nullable": 0,
                    "arrayBaseColumnType": 0,
```

```
                    "name": "author"
                },
                {
                    "isSigned": false,
                    "isCurrency": false,
                    "label": "title",
                    "precision": 200,
                    "typeName": "VARCHAR",
                    "scale": 0,
                    "isAutoIncrement": false,
                    "isCaseSensitive": false,
                    "schemaName": "",
                    "tableName": "Books",
                    "type": 12,
                    "nullable": 0,
                    "arrayBaseColumnType": 0,
                    "name": "title"
                },
                {
                    "isSigned": false,
                    "isCurrency": false,
                    "label": "ISBN-13",
                    "precision": 15,
                    "typeName": "VARCHAR",
                    "scale": 0,
                    "isAutoIncrement": false,
                    "isCaseSensitive": false,
                    "schemaName": "",
                    "tableName": "Books",
                    "type": 12,
                    "nullable": 0,
                    "arrayBaseColumnType": 0,
                    "name": "ISBN-13"
                }
            ]
        }
    ]
}
```

`util.rds.toJsonString`adalah:

```
[
  {
```

```
    "author": "Mark Twain",
    "title": "Adventures of Huckleberry Finn",
    "ISBN-13": "978-1948132817"
  },
  {
    "author": "Jack London",
    "title": "The Call of the Wild",
    "ISBN-13": "978-1948132275"
  },
]
```

`$util.rds.toJsonObject(String serializedSQLResult): Object`

Ini sama dengan`util.rds.toJsonString`, tetapi dengan hasilnya menjadi JSON`Object`.

Pembantu HTTP di $ util.http

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util.http`Utilitas menyediakan metode pembantu yang dapat Anda gunakan untuk mengelola parameter permintaan HTTP dan untuk menambahkan header respons.

$ util.http daftar utilitas

`$util.http.copyHeaders(Map) : Map`

Menyalin header dari peta tanpa set header HTTP yang dibatasi. Anda dapat menggunakan ini untuk meneruskan header permintaan ke titik akhir HTTP hilir Anda.

```
{
    ...
    "params": {
        ...
        "headers": $util.http.copyHeaders($ctx.request.headers),
        ...
    },
```

```
    ...
}
```

`$util.http.addResponseHeader(String, Object)`

Menambahkan header kustom tunggal dengan nama (`String`) dan nilai (`Object`) dari respon. Limitasi berikut berlaku pada:

- Nama header tidak dapat cocok dengan header atau AWS AppSync header yang ada AWS atau dibatasi.
- Nama header tidak dapat dimulai dengan awalan terbatas, seperti `x-amzn-` atau. `x-amz-`
- Ukuran header respons kustom tidak boleh melebihi 4 KB. Ini termasuk nama dan nilai header.
- Anda harus menentukan setiap header respons sekali per operasi GraphQL. Namun, jika Anda menentukan header kustom dengan nama yang sama beberapa kali, definisi terbaru akan muncul dalam respons. Semua header dihitung terhadap batas ukuran header terlepas dari penamaan.

```
...
$util.http.addResponseHeader("itemsCount", 7)
$util.http.addResponseHeader("render", $ctx.args.render)
...
```

`$util.http.addResponseHeaders(Map)`

Menambahkan beberapa header respons ke respons dari peta nama (`String`) dan nilai (`Object`) yang ditentukan. Keterbatasan yang sama yang tercantum untuk `addResponseHeader(String, Object)` metode ini juga berlaku untuk metode ini.

```
...
#set($headersMap = {})
$util.qr($headersMap.put("headerInt", 12))
$util.qr($headersMap.put("headerString", "stringValue"))
$util.qr($headersMap.put("headerObject", {"field1": 7, "field2": "string"}))
$util.http.addResponseHeaders($headersMap)
...
```

Pembantu XMLdi $ util.xml.

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util.xml`berisi metode pembantu yang dapat membuatnya lebih mudah untuk menerjemahkan respons XHTML ke JSON atau Kamus.

$ util.xmldaftar utils

`$util.xml.toMap(String) : Map`

Mengkonversi string XHTML ke Kamus.

```
Input:

<?xml version="1.0" encoding="UTF-8"?>
<posts>
<post>
  <id>1</id>
  <title>Getting started with GraphQL</title>
</post>
</posts>

Output (JSON representation):

{
  "posts":{
    "post":{
      "id":1,
      "title":"Getting started with GraphQL"
    }
  }
}


Input:

<?xml version="1.0" encoding="UTF-8"?>
```

```
<posts>
<post>
  <id>1</id>
  <title>Getting started with GraphQL</title>
</post>
<post>
  <id>2</id>
  <title>Getting started with AWS AppSync</title>
</post>
</posts>

Output (JSON representation):

{
  "posts":{
    "post":[
        {
          "id":1,
          "title":"Getting started with GraphQL"
        },
        {
          "id":2,
          "title":"Getting started with AWS AppSync"
        }
    ]
  }
}
```

`$util.xml.toJsonString(String) : String`

Mengkonversi string XHTML ke string JSON. Ini mirip dengan TomAP, kecuali bahwa outputnya adalah string. Hal ini berguna jika Anda ingin langsung mengkonversi dan mengembalikan respon XMLdari objek HTTP ke JSON.

`$util.xml.toJsonString(String, Boolean) : String`

Mengkonversi string XHTML ke string JSON dengan parameter Boolean opsional untuk menentukan apakah Anda ingin string-encode JSON.

Pembantu transformasi di $util.transform

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util.transform`berisi metode pembantu yang memudahkan untuk melakukan operasi kompleks terhadap sumber data, seperti operasi filter Amazon DynamoDB.

Pembantu transformasi

Daftar utilitas pembantu transformasi

`$util.transform.toDynamoDBFilterExpression(Map) : Map`

Mengkonversi string input ke ekspresi filter untuk digunakan dengan DynamoDB.

```
Input:

$util.transform.toDynamoDBFilterExpression({
    "title":{
      "contains":"Hello World"
    }
  })

Output:

{
    "expression" : "contains(#title, :title_contains)"
    "expressionNames" : {
        "#title" : "title",
    },
    "expressionValues" : {
        ":title_contains" : { "S" : "Hello World" }
    },
}
```

`$util.transform.toElasticsearchQueryDSL(Map) : Map`

Mengkonversi input yang diberikan ke ekspresi OpenSearch Query DSL yang setara, mengembalikannya sebagai string JSON.

```
Input:

$util.transform.toElasticsearchQueryDSL({
    "upvotes":{
        "ne":15,
        "range":[
            10,
            20
        ]
    },
    "title":{
        "eq":"hihihi",
        "wildcard":"h*i"
    }
  })

Output:
{
    "bool":{
      "must":[
          {
            "bool":{
              "must":[
                  {
                    "bool":{
                      "must_not":{
                        "term":{
                          "upvotes":15
                        }
                      }
                    }
                  },
                  {
                    "range":{
                      "upvotes":{
                        "gte":10,
                        "lte":20
                      }
                    }
```

```
                    }
                ]
              }
            },
            {
              "bool":{
                "must":[
                    {
                      "term":{
                        "title":"hihihi"
                      }
                    },
                    {
                    "wildcard":{
                        "title":"h*i"
                      }
                    }
                ]
              }
            }
        ]
    }
}
```

Operator default diasumsikan AND.

Filter berlangganan pembantu transformasi

Pembantu transformasi berlangganan menyaring daftar utils

`$util.transform.toSubscriptionFilter(Map) : Map`

Mengkonversi objek `Map` input ke objek `SubscriptionFilter` ekspresi. `$util.transform.toSubscriptionFilter`Metode ini digunakan sebagai masukan ke `$extensions.setSubscriptionFilter()` ekstensi. Untuk informasi selengkapnya, lihat Ekstensi.

`$util.transform.toSubscriptionFilter(Map, List) : Map`

Mengkonversi objek `Map` input ke objek `SubscriptionFilter` ekspresi. `$util.transform.toSubscriptionFilter`Metode ini digunakan sebagai masukan ke

`$extensions.setSubscriptionFilter()` ekstensi. Untuk informasi selengkapnya, lihat Ekstensi.

Argumen pertama adalah objek `Map` masukan yang dikonversi ke objek `SubscriptionFilter` ekspresi. Argumen kedua adalah nama `List` bidang yang diabaikan dalam objek `Map` masukan pertama saat membangun objek `SubscriptionFilter` ekspresi.

```
$util.transform.toSubscriptionFilter(Map, List, Map) : Map
```

Mengkonversi objek `Map` input ke objek `SubscriptionFilter` ekspresi. `$util.transform.toSubscriptionFilter`Metode ini digunakan sebagai masukan ke `$extensions.setSubscriptionFilter()` ekstensi. Untuk informasi selengkapnya, lihat Ekstensi.

Argumen pertama adalah objek `Map` masukan yang dikonversi ke objek `SubscriptionFilter` ekspresi, argumen kedua adalah nama `List` bidang yang akan diabaikan dalam objek `Map` masukan pertama, dan argumen ketiga adalah objek `Map` masukan dari aturan ketat yang disertakan saat membangun objek `SubscriptionFilter` ekspresi. Aturan ketat ini disertakan dalam objek `SubscriptionFilter` ekspresi sedemikian rupa sehingga setidaknya salah satu aturan akan dipenuhi untuk melewati filter berlangganan.

Argumen filter langganan

Tabel berikut menjelaskan bagaimana argumen dari utilitas berikut didefinisikan:

- `$util.transform.toSubscriptionFilter(Map) : Map`
- `$util.transform.toSubscriptionFilter(Map, List) : Map`
- `$util.transform.toSubscriptionFilter(Map, List, Map) : Map`

Argument 1: Map

Argumen 1 adalah `Map` objek dengan nilai-nilai kunci berikut:

- nama bidang
- “dan”
- “atau”

Untuk nama bidang sebagai kunci, kondisi pada entri bidang ini adalah dalam bentuk. `"operator" : "value"`

Contoh berikut menunjukkan bagaimana entri dapat ditambahkan ke`Map`:

```
"field_name" : {
                    "operator1" : value
               }

## We can have multiple conditions for the same field_name:

"field_name" : {
                    "operator1" : value
                    "operator2" : value
                    .
                    .
                    .
               }
```

Ketika sebuah bidang memiliki dua atau lebih kondisi di atasnya, semua kondisi ini dianggap menggunakan operasi OR.

Input juga `Map` dapat memiliki “dan” dan “atau” sebagai kunci, menyiratkan bahwa semua entri di dalamnya harus digabungkan menggunakan logika AND atau OR tergantung pada kuncinya. Nilai kunci “dan” dan “atau” mengharapkan berbagai kondisi.

```
"and" : [

            {
                "field_name1" : {
                    "operator1" : value
                }
             },

             {
                "field_name2" : {
                    "operator1" : value
                }
             },
             .
             .
        ].
```

Perhatikan bahwa Anda dapat bersarang “dan” dan “atau”. Artinya, Anda dapat memiliki sarang “dan” /"atau” di dalam blok “dan” /"atau” lainnya. Namun, ini tidak berfungsi untuk bidang sederhana.

```
"and" : [

            {
                "field_name1" : {
                    "operator" : value
                }
             },

             {
                "or" : [
                            {
                                "field_name2" : {
                                    "operator" : value
                                }
                            },

                            {
                                "field_name3" : {
                                    "operator" : value
                                }
                            }

                        ].
```

Contoh berikut menunjukkan masukan argumen 1 menggunakan`$util.transform.toSubscriptionFilter(Map) : Map`.

Masukan

Argumen 1: Peta:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
```

```
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 2000
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Keluaran

Hasilnya adalah `Map` objek:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "percentageUp",
          "operator": "lte",
          "value": 50
        },
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
```

```
          "value": 2000
        },
        {
          "fieldName": "author",
          "operator": "eq",
          "value": "Admin"
        }
      ]
    },
    {
      "filters": [
        {
          "fieldName": "percentageUp",
          "operator": "lte",
          "value": 50
        },
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 2000
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    },
    {
      "filters": [
        {
          "fieldName": "percentageUp",
          "operator": "gte",
          "value": 20
        },
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
```

```
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 2000
        },
        {
          "fieldName": "author",
          "operator": "eq",
          "value": "Admin"
        }
      ]
    },
    {
      "filters": [
        {
          "fieldName": "percentageUp",
          "operator": "gte",
          "value": 20
        },
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 2000
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    }
  ]
}
```

Argument 2: List

Argumen 2 berisi nama `List` bidang yang tidak boleh dipertimbangkan dalam input `Map` (argumen 1) saat membangun objek `SubscriptionFilter` ekspresi. `List`Bisa juga kosong.

Contoh berikut menunjukkan masukan argumen 1 dan argumen 2 menggunakan`$util.transform.toSubscriptionFilter(Map, List) : Map`.

Masukan

Argumen 1: Peta:

```
{

  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Argumen 2: Daftar:

```
["percentageUp", "author"]
```

Keluaran

Hasilnya adalah `Map` objek:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    }
  ]
}
```

Argument 3: Map

Argumen 3 adalah `Map` objek yang memiliki nama bidang sebagai nilai kunci (tidak dapat memiliki “dan” atau “atau”). Untuk nama bidang sebagai kunci, kondisi pada bidang ini adalah entri dalam bentuk. `"operator" : "value"` Tidak seperti argumen 1, argumen 3 tidak dapat memiliki beberapa kondisi dalam kunci yang sama. Selain itu, argumen 3 tidak memiliki klausa “dan” atau “atau”, jadi tidak ada sarang yang terlibat juga.

Argumen 3 merupakan daftar aturan ketat, yang ditambahkan ke objek `SubscriptionFilter` ekspresi sehingga setidaknya satu dari kondisi ini terpenuhi untuk melewati filter.

```
{
  "fieldname1": {
    "operator": value
  },
```

```
  "fieldname2": {
    "operator": value
  }
}
.
.
.
```

Contoh berikut menunjukkan masukan dari argumen 1, argumen 2, dan argumen 3 menggunakan`$util.transform.toSubscriptionFilter(Map, List, Map) : Map`.

Masukan

Argumen 1: Peta:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "lt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
```

```
  ]
}
```

Argumen 2: Daftar:

```
["percentageUp", "author"]
```

Argumen 3: Peta:

```
{
  "upvotes": {
    "gte": 250
  },
  "author": {
    "eq": "Person1"
  }
}
```

Keluaran

Hasilnya adalah `Map` objek:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        },
        {
          "fieldName": "upvotes",
```

```
          "operator": "gte",
          "value": 250
        }
      ]
    },
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        },
        {
          "fieldName": "author",
          "operator": "eq",
          "value": "Person1"
        }
      ]
    }
  ]
}
```

Pembantu matematika di $util.math

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util.math`berisi metode untuk membantu operasi Matematika umum.

$util.math daftar utilitas

`$util.math.roundNum(Double) : Integer`

Mengambil ganda dan membulatkannya ke bilangan bulat terdekat.

`$util.math.minVal(Double, Double) : Double`

Mengambil dua ganda dan mengembalikan nilai minimum antara dua ganda.

`$util.math.maxVal(Double, Double) : Double`

Mengambil dua ganda dan mengembalikan nilai maksimum antara dua ganda.

`$util.math.randomDouble() : Double`

Mengembalikan ganda acak antara 0 dan 1.

> **Important**
>
> Fungsi ini tidak boleh digunakan untuk apa pun yang membutuhkan keacakan entropi tinggi (misalnya, kriptografi).

`$util.math.randomWithinRange(Integer, Integer) : Integer`

Mengembalikan nilai integer acak dalam rentang yang ditentukan, dengan argumen pertama menentukan nilai yang lebih rendah dari rentang dan argumen kedua menentukan nilai atas rentang.

> **Important**
>
> Fungsi ini tidak boleh digunakan untuk apa pun yang membutuhkan keacakan entropi tinggi (misalnya, kriptografi).

Pembantu string di $util.str

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$util.str`berisi metode untuk membantu operasi String umum.

$util.str daftar utils

`$util.str.toUpper(String) : String`

Mengambil string dan mengubahnya menjadi huruf besar sepenuhnya.

`$util.str.toLower(String) : String`

Mengambil string dan mengubahnya menjadi huruf kecil sepenuhnya.

`$util.str.toReplace(String, String, String) : String`

Mengganti substring dalam string dengan string lain. Argumen pertama menentukan string untuk melakukan operasi penggantian. Argumen kedua menentukan substring untuk menggantikan. Argumen ketiga menentukan string untuk menggantikan argumen kedua dengan. Berikut ini adalah contoh penggunaan utilitas ini:

```
INPUT:      $util.str.toReplace("hello world", "hello", "mellow")
OUTPUT:     "mellow world"
```

`$util.str.normalize(String, String) : String`

Menormalkan string menggunakan salah satu dari empat bentuk normalisasi unicode: NFC, NFD, NFKC, atau NFKD. Argumen pertama adalah string untuk menormalkan. Argumen kedua adalah “nfc”, “nfd”, “nfkc”, atau “nfkd” yang menentukan jenis normalisasi yang akan digunakan untuk proses normalisasi.

Ekstensi

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

`$extensions`berisi serangkaian metode untuk membuat tindakan tambahan dalam resolver Anda.

$ ekstensi. evictFromApiCache (String, String, Objek): Objek

Mengusir item dari cache sisi AWS AppSync server. Argumen pertama adalah nama tipe. Argumen kedua adalah nama bidang. Argumen ketiga adalah objek yang berisi item pasangan kunci-nilai yang menentukan nilai kunci caching. Anda harus meletakkan item dalam objek dalam urutan yang sama dengan kunci caching di resolver cache. `cachingKey`

> **Note**
>
> Utilitas ini hanya berfungsi untuk mutasi, bukan kueri.

$ ekstensi. setSubscriptionFilter(filterJsonObject)

Mendefinisikan filter langganan yang disempurnakan. Setiap acara pemberitahuan berlangganan dievaluasi terhadap filter langganan yang disediakan dan mengirimkan pemberitahuan kepada klien jika semua filter mengevaluasi. `true` Argumennya `filterJsonObject` seperti yang dijelaskan di bawah ini.

> **Note**
>
> Anda dapat menggunakan metode ekstensi ini hanya dalam template pemetaan respons dari resolver langganan.

$ ekstensi. setSubscriptionInvalidationFilter (filterJsonObject)

Mendefinisikan filter pembatalan langganan. Filter langganan dievaluasi terhadap muatan pembatalan, lalu membatalkan langganan yang diberikan jika filter mengevaluasi. `true` Argumennya `filterJsonObject` seperti yang dijelaskan di bawah ini.

> **Note**
>
> Anda dapat menggunakan metode ekstensi ini hanya dalam template pemetaan respons dari resolver langganan.

Argumen: filterJsonObject

Objek JSON mendefinisikan filter langganan atau pembatalan. Ini adalah array filter dalam `afilterGroup`. Setiap filter adalah kumpulan filter individual.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                }
           ]

        },
        {
           "filters" : [
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
           ]

        }
    ]
}
```

Setiap filter memiliki tiga atribut:

- `fieldName`- Bidang skema GraphQL.
- `operator`— Jenis operator.
- `value`— Nilai untuk dibandingkan dengan `fieldName` nilai notifikasi langganan.

Berikut ini adalah contoh penugasan atribut ini:

```
{
 "fieldName" : "severity",
 "operator" : "le",
```

```
 "value" : $context.result.severity
}
```

Bidang: fieldName

Jenis string `fieldName` mengacu pada bidang yang ditentukan dalam skema GraphQL yang cocok dengan payload notifikasi `fieldName` langganan. Ketika kecocokan ditemukan, bidang skema GraphQL dibandingkan dengan filter notifikasi `value` langganan. `value` Dalam contoh berikut, `fieldName` filter cocok dengan `service` bidang yang ditentukan dalam tipe GraphQL tertentu. Jika payload notifikasi berisi `service` bidang yang `value` setara dengan`AWS AppSync`, filter akan mengevaluasi: `true`

```
{
 "fieldName" : "service",
 "operator" : "eq",
 "value" : "AWS AppSync"
}
```

Bidang: nilai

Nilai dapat berupa tipe yang berbeda berdasarkan operator:

- Nomor tunggal atau Boolean
  - Contoh string:`"test"`, `"service"`
  - Contoh nomor:`1`,`2`, `45.75`
  - Contoh Boolean:`true`, `false`
- Pasangan angka atau string
  - Contoh pasangan string:`["test1","test2"]`, `["start","end"]`
  - Contoh pasangan angka:`[1,4]`,`[67,89]`, `[12.45, 95.45]`
- Array angka atau string
  - Contoh array string: `["test1","test2","test3","test4","test5"]`
  - Contoh array angka:`[1,2,3,4,5]`, `[12.11,46.13,45.09,12.54,13.89]`

Bidang: operator

String peka huruf besar/kecil dengan nilai yang mungkin berikut:

Operasi	Deskripsi	Jenis nilai yang mungkin
eq	Equal	integer, float, string, Boolean
ne	Not equal	integer, float, string, Boolean
le	Less than or equal	integer, float, string
lt	Less than	integer, float, string
ge	Greater than or equal	integer, float, string
gt	Greater than	integer, float, string
contains	Checks for a subsequence or value in the set.	integer, float, string
notContains	Checks for the absence of a subsequence or absence of a value in the set.	integer, float, string
beginsWith	Checks for a prefix.	string
in	Checks for matching elements that are in the list.	Array of integer, float, or string
notIn	Checks for matching elements that aren't in the list.	Array of integer, float, or string
between	Between two values	integer, float, string
containsAny	Contains common elements	integer, float, string

Tabel berikut menjelaskan bagaimana setiap operator digunakan dalam pemberitahuan berlangganan.

eq (equal)

`eq`Operator mengevaluasi `true` apakah nilai bidang notifikasi langganan cocok dan benar-benar sama dengan nilai filter. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `service` bidang dengan nilai yang setara `AWS AppSync` dengan.

Jenis nilai yang mungkin: integer, float, string, Boolean

```
{
 "fieldName" : "service",
 "operator" : "eq",
 "value" : "AWS AppSync"
}
```

ne (not equal)

`ne`Operator mengevaluasi `true` apakah nilai bidang notifikasi langganan berbeda dari nilai filter. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `service` bidang dengan nilai yang berbeda dari`AWS AppSync`.

Jenis nilai yang mungkin: integer, float, string, Boolean

```
{
 "fieldName" : "service",
 "operator" : "ne",
 "value" : "AWS AppSync"
}
```

le (less or equal)

`le`Operator mengevaluasi `true` apakah nilai bidang notifikasi langganan kurang dari atau sama dengan nilai filter. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `size` bidang dengan nilai kurang dari atau sama dengan`5`.

Jenis nilai yang mungkin: integer, float, string

```
{
 "fieldName" : "size",
 "operator" : "le",
 "value" : 5
```

```
}
```

lt (less than)

`lt`Operator mengevaluasi `true` apakah nilai bidang notifikasi langganan lebih rendah dari nilai filter. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `size` bidang dengan nilai lebih rendah dari`5`.

Jenis nilai yang mungkin: integer, float, string

```
{
 "fieldName" : "size",
 "operator" : "lt",
 "value" : 5
}
```

ge (greater or equal)

`ge`Operator mengevaluasi `true` apakah nilai bidang notifikasi langganan lebih besar dari atau sama dengan nilai filter. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `size` bidang dengan nilai lebih besar dari atau sama dengan`5`.

Jenis nilai yang mungkin: integer, float, string

```
{
 "fieldName" : "size",
 "operator" : "ge",
 "value" : 5
}
```

gt (greater than)

`gt`Operator mengevaluasi `true` apakah nilai bidang notifikasi langganan lebih besar dari nilai filter. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `size` bidang dengan nilai lebih besar dari`5`.

Jenis nilai yang mungkin: integer, float, string

```
{
 "fieldName" : "size",
 "operator" : "gt",
```

```
 "value" : 5
}
```

contains

`contains`Operator memeriksa substring, urutan, atau nilai dalam satu set atau item tunggal. Filter dengan `contains` operator mengevaluasi `true` apakah nilai bidang notifikasi langganan berisi nilai filter. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `seats` bidang dengan nilai array yang berisi nilai`10`.

Jenis nilai yang mungkin: integer, float, string

```
{
 "fieldName" : "seats",
 "operator" : "contains",
 "value" : 10
}
```

Dalam contoh lain, filter mengevaluasi `true` apakah notifikasi langganan memiliki `event` bidang dengan `launch` substring.

```
{
 "fieldName" : "event",
 "operator" : "contains",
 "value" : "launch"
}
```

notContains

`notContains`Operator memeriksa tidak adanya substring, urutan, atau nilai dalam satu set atau item tunggal. Filter dengan `notContains` operator mengevaluasi `true` jika nilai bidang notifikasi langganan tidak berisi nilai filter. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `seats` bidang dengan nilai array yang tidak mengandung nilai`10`.

Jenis nilai yang mungkin: integer, float, string

```
{
 "fieldName" : "seats",
 "operator" : "notContains",
```

```
 "value" : 10
}
```

Dalam contoh lain, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki nilai `event` bidang tanpa `launch` sebagai urutannya.

```
{
 "fieldName" : "event",
 "operator" : "notContains",
 "value" : "launch"
}
```

beginsWith

`beginsWith`Operator memeriksa awalan dalam string. Filter yang berisi `beginsWith` operator mengevaluasi `true` apakah nilai bidang notifikasi langganan dimulai dengan nilai filter. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `service` bidang dengan nilai yang dimulai dengan`AWS`.

Jenis nilai yang mungkin: string

```
{
 "fieldName" : "service",
 "operator" : "beginsWith",
 "value" : "AWS"
}
```

in

`in`Operator memeriksa elemen yang cocok dalam array. Filter yang berisi `in` operator mengevaluasi `true` apakah nilai bidang notifikasi langganan ada dalam larik. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `severity` bidang dengan salah satu nilai yang ada dalam array:`[1,2,3]`.

Jenis nilai yang mungkin: Array integer, float, atau string

```
{
 "fieldName" : "severity",
 "operator" : "in",
 "value" : [1,2,3]
```

```
}
```

notIn

`notIn`Operator memeriksa elemen yang hilang dalam array. Filter yang berisi `notIn` operator mengevaluasi `true` jika nilai bidang notifikasi langganan tidak ada dalam larik. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `severity` bidang dengan salah satu nilai yang tidak ada dalam array:`[1,2,3]`.

Jenis nilai yang mungkin: Array integer, float, atau string

```
{
 "fieldName" : "severity",
 "operator" : "notIn",
 "value" : [1,2,3]
}
```

between

`between`Operator memeriksa nilai antara dua angka atau string. Filter yang berisi `between` operator mengevaluasi `true` apakah nilai bidang notifikasi langganan berada di antara pasangan nilai filter. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `severity` bidang dengan nilai`2`,`3`,`4`.

Jenis nilai yang mungkin: Sepasang integer, float, atau string

```
{
 "fieldName" : "severity",
 "operator" : "between",
 "value" : [1,5]
}
```

containsAny

`containsAny`Operator memeriksa elemen umum dalam array. Filter dengan `containsAny` operator mengevaluasi `true` apakah persimpangan bidang notifikasi langganan menetapkan nilai dan nilai set filter tidak kosong. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `seats` bidang dengan nilai array yang berisi salah satu `10` atau`15`. Ini berarti bahwa filter akan mengevaluasi `true` apakah pemberitahuan langganan memiliki nilai `seats` bidang `[10,11]` atau`[15,20,30]`.

Jenis nilai yang mungkin: integer, float, atau string

```
{
 "fieldName" : "seats",
 "operator" : "contains",
 "value" : [10, 15]
}
```

Logika DAN

Anda dapat menggabungkan beberapa filter menggunakan logika AND dengan mendefinisikan beberapa entri dalam `filters` objek dalam array. `filterGroup` Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `userId` bidang dengan nilai yang setara dengan 1 DAN nilai `group` bidang salah satu `Admin` atau`Developer`.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                },
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
           ]

        }
    ]
}
```

Logika ATAU

Anda dapat menggabungkan beberapa filter menggunakan logika OR dengan mendefinisikan beberapa objek filter dalam `filterGroup` array. Dalam contoh berikut, filter mengevaluasi `true` apakah pemberitahuan langganan memiliki `userId` bidang dengan nilai yang setara dengan 1 ATAU nilai `group` bidang dari salah satu `Admin` atau`Developer`.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                }
           ]

        },
        {
           "filters" : [
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
           ]

        }
    ]
}
```

Pengecualian

Perhatikan bahwa ada beberapa batasan untuk menggunakan filter:

- Dalam `filters` objek, bisa ada maksimal lima `fieldName` item unik per filter. Ini berarti Anda dapat menggabungkan maksimal lima `fieldName` objek individu menggunakan logika AND.
- Bisa ada maksimum dua puluh nilai untuk `containsAny` operator.
- Bisa ada maksimal lima nilai untuk `in` dan `notIn` operator.
- Setiap string dapat maksimal 256 karakter.
- Setiap perbandingan string peka huruf besar/kecil.
- Pemfilteran objek bersarang memungkinkan hingga lima tingkat penyaringan bersarang.
- Masing-masing `filterGroup` dapat memiliki maksimal 10`filters`. Ini berarti Anda dapat menggabungkan maksimal 10 `filters` menggunakan logika OR.
  - `in`Operator adalah kasus khusus logika OR. Dalam contoh berikut, ada dua`filters`:

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                },
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
           ]
        }
    ]
}
```

Grup filter sebelumnya dievaluasi sebagai berikut dan dihitung menuju batas filter maksimum:

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                },
                {
                    "fieldName" : "group",
                    "operator" : "eq",
                    "value" : "Admin"
                }
           ]
        },
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
```

```
                    "value" : 1
                },
                {
                    "fieldName" : "group",
                    "operator" : "eq",
                    "value" : "Developer"
                }
            ]
        }
    ]
}
```

$extensions.InvalidateSubscriptions () invalidationJsonObject

Digunakan untuk memulai pembatalan langganan dari mutasi. Argumennya `invalidationJsonObject` seperti yang dijelaskan di bawah ini.

> **Note**
>
> Ekstensi ini hanya dapat digunakan dalam template pemetaan respons dari resolver mutasi. Anda hanya dapat menggunakan paling banyak lima panggilan `$extensions.invalidateSubscriptions()` metode unik dalam satu permintaan. Jika Anda melebihi batas ini, Anda akan menerima kesalahan GraphQL.

Argumen: invalidationJsonObject

`invalidationJsonObject`Mendefinisikan sebagai berikut:

- `subscriptionField`— Langganan skema GraphQL untuk membatalkan. Langganan tunggal, didefinisikan sebagai string di`subscriptionField`, dianggap untuk pembatalan.
- `payload`— Daftar pasangan kunci-nilai yang digunakan sebagai input untuk membatalkan langganan jika filter pembatalan mengevaluasi terhadap nilainya. `true`

  Contoh berikut membatalkan klien yang berlangganan dan terhubung menggunakan `onUserDelete` langganan saat filter pembatalan yang ditentukan dalam resolver langganan mengevaluasi terhadap nilainya. `true` `payload`

```
$extensions.invalidateSubscriptions({
```

```
        "subscriptionField": "onUserDelete",
        "payload": {
                "group": "Developer"
                "type" : "Full-Time"
      }
    })
```

Referensi template pemetaan Resolver untuk DynamoDB

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync DynamoDB resolver memungkinkan Anda menggunakan GraphQL untuk menyimpan dan mengambil data dalam tabel Amazon DynamoDB yang ada di akun Anda. Penyelesai ini bekerja dengan memungkinkan Anda memetakan permintaan GraphQL yang masuk ke dalam panggilan DynamoDB, dan kemudian memetakan respons DynamoDB kembali ke GraphQL. Bagian ini menjelaskan template pemetaan untuk operasi DynamoDB yang didukung.

GetItem

Dokumen pemetaan `GetItem` permintaan memungkinkan Anda memberi tahu AWS AppSync DynamoDB resolver untuk membuat `GetItem` permintaan ke DynamoDB, dan memungkinkan Anda untuk menentukan:

- Kunci item di DynamoDB
- Apakah akan menggunakan bacaan yang konsisten atau tidak

Dokumen `GetItem` pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "foo" : ... typed value,
        "bar" : ... typed value
```

```
    },
    "consistentRead" : true,
    "projection" : {
        ...
    }
}
```

Bidang didefinisikan sebagai berikut:

GetItembidang

GetItem daftar bidang

`version`

Versi definisi template. `2017-02-28`dan `2018-05-29` saat ini didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `GetItem` DynamoDB, ini harus diatur ke. `GetItem` Nilai ini diperlukan.

`key`

Kunci item di DynamoDB. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Nilai ini diperlukan.

`consistentRead`

Apakah akan melakukan pembacaan yang sangat konsisten dengan DynamoDB atau tidak. Ini opsional, dan defaultnya. `false`

`projection`

Proyeksi yang digunakan untuk menentukan atribut yang akan dikembalikan dari operasi DynamoDB. Untuk informasi selengkapnya tentang proyeksi, lihat Proyeksi. Bidang ini bersifat opsional.

Item yang dikembalikan dari DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON, dan tersedia dalam konteks pemetaan (). `$context.result`

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihat Mengetik sistem (pemetaan respons).

Untuk informasi selengkapnya tentang template pemetaan respons, lihat Ringkasan template pemetaan Resolver.

Contoh

Contoh berikut adalah template pemetaan untuk query GraphQL: `getThing(foo: String!, bar: String!)`

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "foo" : $util.dynamodb.toDynamoDBJson($ctx.args.foo),
        "bar" : $util.dynamodb.toDynamoDBJson($ctx.args.bar)
    },
    "consistentRead" : true
}
```

Untuk informasi selengkapnya tentang DynamoDB API, lihat `GetItem` dokumentasi DynamoDB API.

PutItem

Dokumen pemetaan `PutItem` permintaan memungkinkan Anda memberi tahu AWS AppSync DynamoDB resolver untuk membuat `PutItem` permintaan ke DynamoDB, dan memungkinkan Anda untuk menentukan hal berikut:

- Kunci item di DynamoDB
- Isi lengkap item (terdiri dari `key` dan`attributeValues`)
- Kondisi agar operasi berhasil

Dokumen `PutItem` pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2018-05-29",
    "operation" : "PutItem",
    "customPartitionKey" : "foo",
    "populateIndexFields" : boolean value,
    "key": {
        "foo" : ... typed value,
        "bar" : ... typed value
```

```
    },
    "attributeValues" : {
        "baz" : ... typed value
    },
    "condition" : {
       ...
    },
    "_version" : 1
}
```

Bidang didefinisikan sebagai berikut:

PutItem bidang

PutItem daftar bidang

`version`

Versi definisi template. `2017-02-28`dan `2018-05-29` saat ini didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `PutItem` DynamoDB, ini harus diatur ke. `PutItem` Nilai ini diperlukan.

`key`

Kunci item di DynamoDB. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Nilai ini diperlukan.

`attributeValues`

Sisa atribut item yang akan dimasukkan ke DynamoDB. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Bidang ini bersifat opsional.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan, `PutItem` permintaan akan menimpa entri yang ada untuk item tersebut. Untuk informasi selengkapnya tentang kondisi, lihat Ekspresi kondisi. Nilai ini bersifat opsional.

`_version`

Nilai numerik yang mewakili versi item terbaru yang diketahui. Nilai ini bersifat opsional. Bidang ini digunakan untuk Deteksi Konflik dan hanya didukung pada sumber data berversi.

`customPartitionKey`

Saat diaktifkan, nilai string ini mengubah format *ds_sk* dan *ds_pk* catatan yang digunakan oleh tabel sinkronisasi delta saat pembuatan versi telah diaktifkan (untuk informasi selengkapnya, lihat Deteksi konflik dan sinkronisasi di Panduan Pengembang AWS AppSync). Saat diaktifkan, pemrosesan `populateIndexFields` entri juga diaktifkan. Bidang ini bersifat opsional.

`populateIndexFields`

Nilai boolean yang, ketika diaktifkan bersama dengan **`customPartitionKey`**, membuat entri baru untuk setiap catatan dalam tabel sinkronisasi delta, khususnya di kolom dang`si_ds_pk`. `gsi_ds_sk` Untuk informasi selengkapnya, lihat Deteksi dan sinkronisasi konflik di Panduan AWS AppSync Pengembang. Bidang ini bersifat opsional.

Item yang ditulis ke DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam konteks pemetaan (). `$context.result`

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihat Mengetik sistem (pemetaan respons).

Untuk informasi selengkapnya tentang template pemetaan respons, lihat Ringkasan template pemetaan Resolver.

Contoh 1

Contoh berikut adalah template pemetaan untuk mutasi GraphQL. `updateThing(foo: String!, bar: String!, name: String!, version: Int!)`

Jika tidak ada item dengan kunci yang ditentukan, itu dibuat. Jika item sudah ada dengan kunci yang ditentukan, itu akan ditimpa.

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key": {
        "foo" : $util.dynamodb.toDynamoDBJson($ctx.args.foo),
        "bar" : $util.dynamodb.toDynamoDBJson($ctx.args.bar)
```

```
    },
    "attributeValues" : {
        "name"    : $util.dynamodb.toDynamoDBJson($ctx.args.name),
        "version" : $util.dynamodb.toDynamoDBJson($ctx.args.version)
    }
}
```

Contoh 2

Contoh berikut adalah template pemetaan untuk mutasi GraphQL. `updateThing(foo: String!, bar: String!, name: String!, expectedVersion: Int!)`

Contoh ini memeriksa untuk memastikan item yang saat ini di DynamoDB memiliki bidang yang disetel `version` ke. `expectedVersion`

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key": {
        "foo" : $util.dynamodb.toDynamoDBJson($ctx.args.foo),
        "bar" : $util.dynamodb.toDynamoDBJson($ctx.args.bar)
    },
    "attributeValues" : {
        "name"    : $util.dynamodb.toDynamoDBJson($ctx.args.name),
        #set( $newVersion = $context.arguments.expectedVersion + 1 )
        "version" : $util.dynamodb.toDynamoDBJson($newVersion)
    },
    "condition" : {
        "expression" : "version = :expectedVersion",
        "expressionValues" : {
            ":expectedVersion" : $util.dynamodb.toDynamoDBJson($expectedVersion)
        }
    }
}
```

Untuk informasi selengkapnya tentang DynamoDB API, lihat `PutItem` dokumentasi DynamoDB API.

UpdateItem

Dokumen pemetaan `UpdateItem` permintaan memungkinkan Anda untuk memberitahu AWS AppSync DynamoDB resolver untuk membuat `UpdateItem` permintaan ke DynamoDB dan memungkinkan Anda untuk menentukan yang berikut:

- Kunci item di DynamoDB
- Ekspresi pembaruan yang menjelaskan cara memperbarui item di DynamoDB
- Kondisi agar operasi berhasil

Dokumen `UpdateItem` pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2018-05-29",
    "operation" : "UpdateItem",
    "customPartitionKey" : "foo",
    "populateIndexFields" : boolean value,
    "key": {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "update" : {
        "expression" : "someExpression",
        "expressionNames" : {
           "#foo" : "foo"
       },
       "expressionValues" : {
           ":bar" : ... typed value
       }
    },
    "condition" : {
        ...
    },
    "_version" : 1
}
```

Bidang didefinisikan sebagai berikut:

UpdateItem bidang

UpdateItem daftar bidang

`version`

Versi definisi template. `2017-02-28`dan `2018-05-29` saat ini didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `UpdateItem` DynamoDB, ini harus diatur ke. `UpdateItem` Nilai ini diperlukan.

`key`

Kunci item di DynamoDB. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Nilai ini diperlukan.

`update`

`update`Bagian ini memungkinkan Anda menentukan ekspresi pembaruan yang menjelaskan cara memperbarui item di DynamoDB. Untuk informasi selengkapnya tentang cara menulis ekspresi pembaruan, lihat dokumentasi DynamoDB UpdateExpressions . Bagian ini diperlukan.

`update`Bagian ini memiliki tiga komponen:

`expression`

Ekspresi pembaruan. Nilai ini diperlukan.

`expressionNames`

Substitusi untuk placeholder nama atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nama yang digunakan dalam`expression`, dan nilainya harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalam. `expression`

`expressionValues`

Substitusi untuk placeholder nilai atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nilai yang digunakan dalam`expression`, dan nilainya harus berupa nilai yang diketik. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Ini harus ditentukan. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nilai atribut ekspresi yang digunakan dalam. `expression`

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan, `UpdateItem`

permintaan akan memperbarui entri yang ada terlepas dari statusnya saat ini. Untuk informasi selengkapnya tentang kondisi, lihat Ekspresi kondisi. Nilai ini bersifat opsional.

`_version`

Nilai numerik yang mewakili versi item terbaru yang diketahui. Nilai ini bersifat opsional. Bidang ini digunakan untuk Deteksi Konflik dan hanya didukung pada sumber data berversi.

`customPartitionKey`

Saat diaktifkan, nilai string ini mengubah format *ds_sk* dan *ds_pk* catatan yang digunakan oleh tabel sinkronisasi delta saat pembuatan versi telah diaktifkan (untuk informasi selengkapnya, lihat Deteksi konflik dan sinkronisasi di Panduan Pengembang AWS AppSync). Saat diaktifkan, pemrosesan `populateIndexFields` entri juga diaktifkan. Bidang ini bersifat opsional.

`populateIndexFields`

Nilai boolean yang, ketika diaktifkan bersama dengan **`customPartitionKey`**, membuat entri baru untuk setiap catatan dalam tabel sinkronisasi delta, khususnya di kolom dan`gsi_ds_pk`. `gsi_ds_sk` Untuk informasi selengkapnya, lihat Deteksi dan sinkronisasi konflik di Panduan AWS AppSync Pengembang. Bidang ini bersifat opsional.

Item yang diperbarui di DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam konteks pemetaan (). `$context.result`

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihat Mengetik sistem (pemetaan respons).

Untuk informasi selengkapnya tentang template pemetaan respons, lihat Ringkasan template pemetaan Resolver.

Contoh 1

Contoh berikut adalah template pemetaan untuk mutasi GraphQL. `upvote(id: ID!)`

Dalam contoh ini, item di DynamoDB memiliki `version` dan `upvotes` bidangnya bertambah 1.

```
{
    "version" : "2017-02-28",
    "operation" : "UpdateItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },
```

```
    "update" : {
        "expression" : "ADD #votefield :plusOne, version :plusOne",
        "expressionNames" : {
            "#votefield" : "upvotes"
        },
        "expressionValues" : {
            ":plusOne" : { "N" : 1 }
        }
    }
}
```

Contoh 2

Contoh berikut adalah template pemetaan untuk mutasi GraphQL. `updateItem(id: ID!, title: String, author: String, expectedVersion: Int!)`

Ini adalah contoh kompleks yang memeriksa argumen dan secara dinamis menghasilkan ekspresi pembaruan yang hanya mencakup argumen yang telah disediakan oleh klien. Misalnya, jika `title` dan `author` dihilangkan, mereka tidak diperbarui. Jika argumen ditentukan tetapi nilainya`null`, maka bidang itu dihapus dari objek di DynamoDB. Akhirnya, operasi memiliki kondisi, yang memverifikasi apakah item saat ini di DynamoDB memiliki `version` bidang yang disetel ke: `expectedVersion`

```
{
    "version" : "2017-02-28",

    "operation" : "UpdateItem",

    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },

    ## Set up some space to keep track of things we're updating **
    #set( $expNames  = {} )
    #set( $expValues = {} )
    #set( $expSet = {} )
    #set( $expAdd = {} )
    #set( $expRemove = [] )

    ## Increment "version" by 1 **
    $!{expAdd.put("version", ":newVersion")}
    $!{expValues.put(":newVersion", { "N" : 1 })}
```

```
    ## Iterate through each argument, skipping "id" and "expectedVersion" **
    #foreach( $entry in $context.arguments.entrySet() )
        #if( $entry.key != "id" && $entry.key != "expectedVersion" )
            #if( (!$entry.value) && ("$!{entry.value}" == "") )
                ## If the argument is set to "null", then remove that attribute from
the item in DynamoDB **

                #set( $discard = ${expRemove.add("#${entry.key}")} )
                $!{expNames.put("#${entry.key}", "$entry.key")}
            #else
                ## Otherwise set (or update) the attribute on the item in DynamoDB **

                $!{expSet.put("#${entry.key}", ":${entry.key}")}
                $!{expNames.put("#${entry.key}", "$entry.key")}

                #if( $entry.key == "ups" || $entry.key == "downs" )
                    $!{expValues.put(":${entry.key}", { "N" : $entry.value })}
                #else
                    $!{expValues.put(":${entry.key}", { "S" : "${entry.value}" })}
                #end
            #end
        #end
    #end

    ## Start building the update expression, starting with attributes we're going to
SET **
    #set( $expression = "" )
    #if( !${expSet.isEmpty()} )
        #set( $expression = "SET" )
        #foreach( $entry in $expSet.entrySet() )
            #set( $expression = "${expression} ${entry.key} = ${entry.value}" )
            #if ( $foreach.hasNext )
                #set( $expression = "${expression}," )
            #end
        #end
    #end

    ## Continue building the update expression, adding attributes we're going to ADD **
    #if( !${expAdd.isEmpty()} )
        #set( $expression = "${expression} ADD" )
        #foreach( $entry in $expAdd.entrySet() )
            #set( $expression = "${expression} ${entry.key} ${entry.value}" )
            #if ( $foreach.hasNext )
```

```
                #set( $expression = "${expression}," )
            #end
        #end
    #end

    ## Continue building the update expression, adding attributes we're going to REMOVE
 **
    #if( !${expRemove.isEmpty()} )
        #set( $expression = "${expression} REMOVE" )

        #foreach( $entry in $expRemove )
            #set( $expression = "${expression} ${entry}" )
            #if ( $foreach.hasNext )
                #set( $expression = "${expression}," )
            #end
        #end
    #end

    ## Finally, write the update expression into the document, along with any
 expressionNames and expressionValues **
    "update" : {
        "expression" : "${expression}"
        #if( !${expNames.isEmpty()} )
            ,"expressionNames" : $utils.toJson($expNames)
        #end
        #if( !${expValues.isEmpty()} )
            ,"expressionValues" : $utils.toJson($expValues)
        #end
    },

    "condition" : {
        "expression"       : "version = :expectedVersion",
        "expressionValues" : {
            ":expectedVersion" :
 $util.dynamodb.toDynamoDBJson($ctx.args.expectedVersion)
        }
    }
}
```

Untuk informasi selengkapnya tentang DynamoDB API, lihat `UpdateItem` dokumentasi DynamoDB API.

DeleteItem

Dokumen pemetaan `DeleteItem` permintaan memungkinkan Anda memberi tahu AWS AppSync DynamoDB resolver untuk membuat `DeleteItem` permintaan ke DynamoDB, dan memungkinkan Anda untuk menentukan hal berikut:

- Kunci item di DynamoDB
- Kondisi agar operasi berhasil

Dokumen `DeleteItem` pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2018-05-29",
    "operation" : "DeleteItem",
    "customPartitionKey" : "foo",
    "populateIndexFields" : boolean value,
    "key": {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "condition" : {
        ...
    },
    "_version" : 1
}
```

Bidang didefinisikan sebagai berikut:

DeleteItembidang

DeleteItemdaftar bidang

`version`

Versi definisi template. `2017-02-28`dan `2018-05-29` saat ini didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `DeleteItem` DynamoDB, ini harus diatur ke. `DeleteItem` Nilai ini diperlukan.

`key`

Kunci item di DynamoDB. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Nilai ini diperlukan.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan, `DeleteItem` permintaan menghapus item terlepas dari keadaan saat ini. Untuk informasi selengkapnya tentang kondisi, lihat Ekspresi kondisi. Nilai ini bersifat opsional.

`_version`

Nilai numerik yang mewakili versi item terbaru yang diketahui. Nilai ini bersifat opsional. Bidang ini digunakan untuk Deteksi Konflik dan hanya didukung pada sumber data berversi.

`customPartitionKey`

Saat diaktifkan, nilai string ini mengubah format *`ds_sk`* dan *`ds_pk`* catatan yang digunakan oleh tabel sinkronisasi delta saat pembuatan versi telah diaktifkan (untuk informasi selengkapnya, lihat Deteksi konflik dan sinkronisasi di Panduan Pengembang AWS AppSync). Saat diaktifkan, pemrosesan `populateIndexFields` entri juga diaktifkan. Bidang ini bersifat opsional.

`populateIndexFields`

Nilai boolean yang, ketika diaktifkan bersama dengan **`customPartitionKey`**, membuat entri baru untuk setiap catatan dalam tabel sinkronisasi delta, khususnya di kolom dang`si_ds_pk`. `gsi_ds_sk` Untuk informasi selengkapnya, lihat Deteksi dan sinkronisasi konflik di Panduan AWS AppSync Pengembang. Bidang ini bersifat opsional.

Item yang dihapus dari DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam konteks pemetaan (). `$context.result`

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihat Mengetik sistem (pemetaan respons).

Untuk informasi selengkapnya tentang template pemetaan respons, lihat Ringkasan template pemetaan Resolver.

Contoh 1

Contoh berikut adalah template pemetaan untuk mutasi GraphQL. `deleteItem(id: ID!)` Jika ada item dengan ID ini, item tersebut akan dihapus.

```
{
    "version" : "2017-02-28",
    "operation" : "DeleteItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    }
}
```

Contoh 2

Contoh berikut adalah template pemetaan untuk mutasi GraphQL. `deleteItem(id: ID!, expectedVersion: Int!)` Jika item ada dengan ID ini, itu akan dihapus, tetapi hanya jika `version` bidangnya disetel ke`expectedVersion`:

```
{
    "version" : "2017-02-28",
    "operation" : "DeleteItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },
    "condition" : {
        "expression"       : "attribute_not_exists(id) OR version = :expectedVersion",
        "expressionValues" : {
            ":expectedVersion" : $util.dynamodb.toDynamoDBJson($expectedVersion)
        }
    }
}
```

Untuk informasi selengkapnya tentang DynamoDB API, lihat `DeleteItem` dokumentasi DynamoDB API.

Kueri

Dokumen pemetaan `Query` permintaan memungkinkan Anda memberi tahu AWS AppSync DynamoDB resolver untuk membuat `Query` permintaan ke DynamoDB, dan memungkinkan Anda untuk menentukan hal berikut:

- Ekspresi kunci
- Indeks mana yang akan digunakan
- Filter tambahan apa pun
- Berapa banyak item untuk dikembalikan
- Apakah akan menggunakan pembacaan yang konsisten
- arah kueri (maju atau mundur)
- Token pagination

Dokumen `Query` pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2017-02-28",
    "operation" : "Query",
    "query" : {
        "expression" : "some expression",
        "expressionNames" : {
            "#foo" : "foo"
        },
        "expressionValues" : {
            ":bar" : ... typed value
        }
    },
    "index" : "fooIndex",
    "nextToken" : "a pagination token",
    "limit" : 10,
    "scanIndexForward" : true,
    "consistentRead" : false,
    "select" : "ALL_ATTRIBUTES" | "ALL_PROJECTED_ATTRIBUTES" | "SPECIFIC_ATTRIBUTES",
    "filter" : {
        ...
    },
    "projection" : {
        ...
    }
}
```

Bidang didefinisikan sebagai berikut:

Bidang kueri

Daftar bidang kueri

`version`

Versi definisi template. `2017-02-28`dan `2018-05-29` saat ini didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `Query` DynamoDB, ini harus diatur ke. `Query` Nilai ini diperlukan.

`query`

`query`Bagian ini memungkinkan Anda menentukan ekspresi kondisi kunci yang menjelaskan item mana yang akan diambil dari DynamoDB. Untuk informasi selengkapnya tentang cara menulis ekspresi kondisi kunci, lihat dokumentasi DynamoDB KeyConditions . Bagian ini harus ditentukan.

`expression`

Ekspresi kueri. Bidang ini harus ditentukan.

`expressionNames`

Substitusi untuk placeholder nama atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nama yang digunakan dalam`expression`, dan nilainya harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalam. `expression`

`expressionValues`

Substitusi untuk placeholder nilai atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nilai yang digunakan dalam`expression`, dan nilainya harus berupa nilai yang diketik. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Nilai ini diperlukan. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nilai atribut ekspresi yang digunakan dalam. `expression`

`filter`

Filter tambahan yang dapat digunakan untuk memfilter hasil dari DynamoDB sebelum dikembalikan. Untuk informasi selengkapnya tentang filter, lihat Menyaring. Bidang ini bersifat opsional.

`index`

Nama indeks untuk query. Operasi query DynamoDB memungkinkan Anda untuk memindai Indeks Sekunder Lokal dan Indeks Sekunder Global selain indeks kunci utama untuk kunci hash. Jika ditentukan, ini memberitahu DynamoDB untuk query indeks tertentu. Jika dihilangkan, indeks kunci utama ditanyakan.

`nextToken`

Token pagination untuk melanjutkan kueri sebelumnya. Ini akan diperoleh dari kueri sebelumnya. Bidang ini bersifat opsional.

`limit`

Jumlah maksimum item untuk dievaluasi (belum tentu jumlah item yang cocok). Bidang ini bersifat opsional.

`scanIndexForward`

Boolean yang menunjukkan apakah akan melakukan kueri maju atau mundur. Bidang ini opsional, dan defaultnya. `true`

`consistentRead`

Boolean yang menunjukkan apakah akan menggunakan pembacaan yang konsisten saat menanyakan DynamoDB. Bidang ini opsional, dan defaultnya. `false`

`select`

Secara default, AWS AppSync DynamoDB resolver hanya mengembalikan atribut yang diproyeksikan ke dalam indeks. Jika lebih banyak atribut diperlukan, Anda dapat mengatur bidang ini. Bidang ini bersifat opsional. Nilai yang didukung adalah:

`ALL_ATTRIBUTES`

Mengembalikan semua atribut item dari tabel tertentu atau indeks. Jika Anda menanyakan indeks sekunder lokal, DynamoDB mengambil seluruh item dari tabel induk untuk setiap item yang cocok dalam indeks. Jika indeks dikonfigurasi untuk memproyeksikan semua atribut item, semua data dapat diperoleh dari indeks sekunder lokal dan tidak diperlukan pengambilan.

`ALL_PROJECTED_ATTRIBUTES`

Diizinkan hanya saat menanyakan indeks. Mengambil semua atribut yang telah diproyeksikan ke dalam indeks. Jika indeks dikonfigurasi untuk memproyeksikan semua atribut, nilai pengembalian ini setara dengan menentukan`ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Mengembalikan hanya atribut yang tercantum dalam `projection`'s`expression`. Nilai pengembalian ini setara dengan menentukan `projection`'s `expression` tanpa menentukan nilai apa pun untuk. `Select`

`projection`

Proyeksi yang digunakan untuk menentukan atribut yang akan dikembalikan dari operasi DynamoDB. Untuk informasi selengkapnya tentang proyeksi, lihat Proyeksi. Bidang ini bersifat opsional.

Hasil dari DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam konteks pemetaan (). `$context.result`

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihat Mengetik sistem (pemetaan respons).

Untuk informasi selengkapnya tentang template pemetaan respons, lihat Ringkasan template pemetaan Resolver.

Hasilnya memiliki struktur sebagai berikut:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Bidang didefinisikan sebagai berikut:

`items`

Daftar yang berisi item yang dikembalikan oleh query DynamoDB.

`nextToken`

Jika mungkin ada lebih banyak hasil, `nextToken` berisi token pagination yang dapat Anda gunakan dalam permintaan lain. Perhatikan bahwa AWS AppSync mengenkripsi dan mengaburkan token pagination yang dikembalikan dari DynamoDB. Ini mencegah data tabel Anda bocor secara tidak sengaja ke penelepon. Perhatikan juga bahwa token pagination ini tidak dapat digunakan di berbagai resolver.

`scannedCount`

Jumlah item yang cocok dengan ekspresi kondisi kueri, sebelum ekspresi filter (jika ada) diterapkan.

Contoh

Contoh berikut adalah template pemetaan untuk query GraphQL. `getPosts(owner: ID!)`

Dalam contoh ini, indeks sekunder global pada tabel ditanyakan untuk mengembalikan semua posting yang dimiliki oleh ID yang ditentukan.

```
{
    "version" : "2017-02-28",
    "operation" : "Query",
    "query" : {
        "expression" : "ownerId = :ownerId",
        "expressionValues" : {
            ":ownerId" : $util.dynamodb.toDynamoDBJson($context.arguments.owner)
        }
    },
    "index" : "owner-index"
}
```

Untuk informasi selengkapnya tentang DynamoDB API, lihat `Query` dokumentasi DynamoDB API.

Pemindaian

Dokumen pemetaan `Scan` permintaan memungkinkan Anda memberi tahu AWS AppSync DynamoDB resolver untuk membuat `Scan` permintaan ke DynamoDB, dan memungkinkan Anda untuk menentukan hal berikut:

- Filter untuk mengecualikan hasil
- Indeks mana yang akan digunakan
- Berapa banyak item untuk dikembalikan
- Apakah akan menggunakan pembacaan yang konsisten
- Token pagination
- Pemindaian paralel

Dokumen `Scan` pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "index" : "fooIndex",
    "limit" : 10,
    "consistentRead" : false,
    "nextToken" : "aPaginationToken",
    "totalSegments" : 10,
    "segment" : 1,
    "filter" : {
        ...
    },
    "projection" : {
        ...
    }
}
```

Bidang didefinisikan sebagai berikut:

Bidang pemindaian

Pindai daftar bidang

`version`

Versi definisi template. `2017-02-28`dan `2018-05-29` saat ini didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `Scan` DynamoDB, ini harus diatur ke. `Scan` Nilai ini diperlukan.

`filter`

Filter yang dapat digunakan untuk memfilter hasil dari DynamoDB sebelum dikembalikan. Untuk informasi selengkapnya tentang filter, lihat Menyaring. Bidang ini bersifat opsional.

`index`

Nama indeks untuk query. Operasi query DynamoDB memungkinkan Anda untuk memindai Indeks Sekunder Lokal dan Indeks Sekunder Global selain indeks kunci utama untuk kunci hash.

Jika ditentukan, ini memberitahu DynamoDB untuk query indeks tertentu. Jika dihilangkan, indeks kunci utama ditanyakan.

`limit`

Jumlah maksimum item untuk dievaluasi pada satu waktu. Bidang ini bersifat opsional.

`consistentRead`

Boolean yang menunjukkan apakah akan menggunakan pembacaan yang konsisten saat menanyakan DynamoDB. Bidang ini opsional, dan defaultnya. `false`

`nextToken`

Token pagination untuk melanjutkan kueri sebelumnya. Ini akan diperoleh dari kueri sebelumnya. Bidang ini bersifat opsional.

`select`

Secara default, AWS AppSync DynamoDB resolver hanya mengembalikan atribut apa pun yang diproyeksikan ke dalam indeks. Jika lebih banyak atribut diperlukan, maka bidang ini dapat diatur. Bidang ini bersifat opsional. Nilai yang didukung adalah:

`ALL_ATTRIBUTES`

Mengembalikan semua atribut item dari tabel tertentu atau indeks. Jika Anda menanyakan indeks sekunder lokal, DynamoDB mengambil seluruh item dari tabel induk untuk setiap item yang cocok dalam indeks. Jika indeks dikonfigurasi untuk memproyeksikan semua atribut item, semua data dapat diperoleh dari indeks sekunder lokal dan tidak diperlukan pengambilan.

`ALL_PROJECTED_ATTRIBUTES`

Diizinkan hanya saat menanyakan indeks. Mengambil semua atribut yang telah diproyeksikan ke dalam indeks. Jika indeks dikonfigurasi untuk memproyeksikan semua atribut, nilai pengembalian ini setara dengan menentukan`ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Mengembalikan hanya atribut yang tercantum dalam `projection`'s`expression`. Nilai pengembalian ini setara dengan menentukan `projection`'s `expression` tanpa menentukan nilai apa pun untuk. `Select`

`totalSegments`

Jumlah segmen untuk mempartisi tabel dengan saat melakukan pemindaian paralel. Bidang ini opsional, tetapi harus ditentukan jika `segment` ditentukan.

`segment`

Segmen tabel dalam operasi ini saat melakukan pemindaian paralel. Bidang ini opsional, tetapi harus ditentukan jika `totalSegments` ditentukan.

`projection`

Proyeksi yang digunakan untuk menentukan atribut yang akan dikembalikan dari operasi DynamoDB. Untuk informasi selengkapnya tentang proyeksi, lihat Proyeksi. Bidang ini bersifat opsional.

Hasil yang dikembalikan oleh pemindaian DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam konteks pemetaan (). `$context.result`

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihat Mengetik sistem (pemetaan respons).

Untuk informasi selengkapnya tentang template pemetaan respons, lihat Ringkasan template pemetaan Resolver.

Hasilnya memiliki struktur sebagai berikut:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Bidang didefinisikan sebagai berikut:

`items`

Daftar yang berisi item yang dikembalikan oleh pemindaian DynamoDB.

`nextToken`

Jika mungkin ada lebih banyak hasil, `nextToken` berisi token pagination yang dapat Anda gunakan dalam permintaan lain. AWS AppSync mengenkripsi dan mengaburkan token pagination yang dikembalikan dari DynamoDB. Ini mencegah data tabel Anda bocor secara tidak sengaja ke penelepon. Selain itu, token pagination ini tidak dapat digunakan di berbagai resolver.

`scannedCount`

Jumlah item yang diambil oleh DynamoDB sebelum ekspresi filter (jika ada) diterapkan.

Contoh 1

Contoh berikut adalah template pemetaan untuk query GraphQL:. `allPosts`

Dalam contoh ini, semua entri dalam tabel dikembalikan.

```
{
    "version" : "2017-02-28",
    "operation" : "Scan"
}
```

Contoh 2

Contoh berikut adalah template pemetaan untuk query GraphQL:. `postsMatching(title: String!)`

Dalam contoh ini, semua entri dalam tabel dikembalikan di mana judul dimulai dengan `title` argumen.

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "filter" : {
        "expression" : "begins_with(title, :title)",
        "expressionValues" : {
            ":title" : $util.dynamodb.toDynamoDBJson($context.arguments.title)
        },
    }
}
```

Untuk informasi selengkapnya tentang DynamoDB API, lihat `Scan` dokumentasi DynamoDB API.

Sinkronkan

Dokumen pemetaan `Sync` permintaan memungkinkan Anda mengambil semua hasil dari tabel DynamoDB dan kemudian hanya menerima data yang diubah sejak kueri terakhir Anda (pembaruan

delta). `Sync`permintaan hanya dapat dibuat ke sumber data DynamoDB berversi. Anda dapat menentukan sebagai berikut:

- Filter untuk mengecualikan hasil
- Berapa banyak item untuk dikembalikan
- Token Paginasi
- Ketika `Sync` operasi terakhir Anda dimulai

Dokumen `Sync` pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2018-05-29",
    "operation" : "Sync",
    "basePartitionKey": "Base Tables PartitionKey",
    "deltaIndexName": "delta-index-name",
    "limit" : 10,
    "nextToken" : "aPaginationToken",
    "lastSync" :  1550000000000,
    "filter" : {
        ...
    }
}
```

Bidang didefinisikan sebagai berikut:

Bidang sinkronisasi

Daftar bidang sinkronisasi

`version`

Versi definisi template. Hanya saat `2018-05-29` ini didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan `Sync` operasi, ini harus diatur ke`Sync`. Nilai ini diperlukan.

`filter`

Filter yang dapat digunakan untuk memfilter hasil dari DynamoDB sebelum dikembalikan. Untuk informasi selengkapnya tentang filter, lihat Menyaring. Bidang ini bersifat opsional.

`limit`

Jumlah maksimum item untuk dievaluasi pada satu waktu. Bidang ini bersifat opsional. Jika dihilangkan, batas default akan diatur ke `100` item. Nilai maksimum untuk bidang ini adalah `1000` item.

`nextToken`

Token pagination untuk melanjutkan kueri sebelumnya. Ini akan diperoleh dari kueri sebelumnya. Bidang ini bersifat opsional.

`lastSync`

Momen, dalam milidetik zaman, ketika `Sync` operasi sukses terakhir dimulai. Jika ditentukan, hanya item yang telah berubah setelah `lastSync` dikembalikan. Bidang ini opsional, dan hanya boleh diisi setelah mengambil semua halaman dari operasi awal`Sync`. Jika dihilangkan, hasil dari tabel Base akan dikembalikan, jika tidak, hasil dari tabel Delta akan dikembalikan.

`basePartitionKey`

Kunci partisi dari tabel Basis yang digunakan saat melakukan `Sync` operasi. Bidang ini memungkinkan `Sync` operasi yang akan dilakukan ketika tabel menggunakan kunci partisi kustom. Ini adalah bidang opsional.

`deltaIndexName`

Indeks yang digunakan untuk `Sync` operasi. Indeks ini diperlukan untuk mengaktifkan `Sync` operasi di seluruh tabel penyimpanan delta saat tabel menggunakan kunci partisi khusus. `Sync`Operasi akan dilakukan pada GSI (dibuat pada `gsi_ds_pk` dan`gsi_ds_sk`). Bidang ini bersifat opsional.

Hasil yang dikembalikan oleh sinkronisasi DynamoDB secara otomatis diubah menjadi tipe primitif GraphQL dan JSON dan tersedia dalam konteks pemetaan (). `$context.result`

Untuk informasi selengkapnya tentang konversi tipe DynamoDB, lihat Mengetik sistem (pemetaan respons).

Untuk informasi selengkapnya tentang template pemetaan respons, lihat Ringkasan template pemetaan Resolver.

Hasilnya memiliki struktur sebagai berikut:

```
{
```

```
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10,
    startedAt = 1550000000000
}
```

Bidang didefinisikan sebagai berikut:

`items`

Daftar yang berisi item yang dikembalikan oleh sinkronisasi.

`nextToken`

Jika mungkin ada lebih banyak hasil, `nextToken` berisi token pagination yang dapat Anda gunakan dalam permintaan lain. AWS AppSync mengenkripsi dan mengaburkan token pagination yang dikembalikan dari DynamoDB. Ini mencegah data tabel Anda bocor secara tidak sengaja ke penelepon. Selain itu, token pagination ini tidak dapat digunakan di berbagai resolver.

`scannedCount`

Jumlah item yang diambil oleh DynamoDB sebelum ekspresi filter (jika ada) diterapkan.

`startedAt`

Saat ini, dalam milidetik epoch, ketika operasi sinkronisasi dimulai yang dapat Anda simpan secara lokal dan gunakan dalam permintaan lain sebagai argumen Anda. `lastSync` Jika token pagination disertakan dalam permintaan, nilai ini akan sama dengan yang dikembalikan oleh permintaan untuk halaman pertama hasil.

Contoh 1

Contoh berikut adalah template pemetaan untuk query GraphQL:. `syncPosts(nextToken: String, lastSync: AWSTimestamp)`

Dalam contoh ini, jika `lastSync` dihilangkan, semua entri dalam tabel dasar dikembalikan. Jika `lastSync` disediakan, hanya entri dalam tabel sinkronisasi delta yang telah berubah sejak itu `lastSync` dikembalikan.

```
{
    "version" : "2018-05-29",
    "operation" : "Sync",
```

```
    "limit": 100,
    "nextToken": $util.toJson($util.defaultIfNull($ctx.args.nextToken, null)),
    "lastSync": $util.toJson($util.defaultIfNull($ctx.args.lastSync, null))
}
```

BatchGetItem

Dokumen pemetaan `BatchGetItem` permintaan memungkinkan Anda memberi tahu AWS AppSync DynamoDB resolver untuk membuat `BatchGetItem` permintaan ke DynamoDB untuk mengambil beberapa item, berpotensi di beberapa tabel. Untuk template permintaan ini, Anda harus menentukan yang berikut:

- Nama tabel tempat untuk mengambil item dari
- Kunci item untuk mengambil dari setiap tabel

Batas `BatchGetItem` DynamoDB berlaku dan tidak ada ekspresi kondisi yang dapat diberikan.

Dokumen `BatchGetItem` pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2018-05-29",
    "operation" : "BatchGetItem",
    "tables" : {
        "table1": {
           "keys": [
              ## Item to retrieve Key
              {
                   "foo" : ... typed value,
                   "bar" : ... typed value
              },
              ## Item2 to retrieve Key
              {
                   "foo" : ... typed value,
                   "bar" : ... typed value
              }
            ],
            "consistentRead": true|false,
            "projection" : {
                ...
            }
        },
```

```
        "table2": {
           "keys": [
              ## Item3 to retrieve Key
              {
                   "foo" : ... typed value,
                   "bar" : ... typed value
              },
              ## Item4 to retrieve Key
              {
                   "foo" : ... typed value,
                   "bar" : ... typed value
              }
            ],
            "consistentRead": true|false,
            "projection" : {
                 ...
            }
        }
    }
}
```

Bidang didefinisikan sebagai berikut:

BatchGetItembidang

BatchGetItemdaftar bidang

`version`

Versi definisi template. Hanya `2018-05-29` didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `BatchGetItem` DynamoDB, ini harus diatur ke. `BatchGetItem` Nilai ini diperlukan.

`tables`

Tabel DynamoDB untuk mengambil item dari. Nilainya adalah peta di mana nama tabel ditentukan sebagai kunci peta. Setidaknya satu meja harus disediakan. `tables`Nilai ini diperlukan.

`keys`

Daftar kunci DynamoDB yang mewakili kunci utama item yang akan diambil. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada

struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan).

`consistentRead`

Apakah akan menggunakan pembacaan yang konsisten saat menjalankan GetItemoperasi. Nilai ini opsional dan default ke false.

`projection`

Proyeksi yang digunakan untuk menentukan atribut yang akan dikembalikan dari operasi DynamoDB. Untuk informasi selengkapnya tentang proyeksi, lihat Proyeksi. Bidang ini bersifat opsional.

Hal-hal yang perlu diingat:

- Jika item belum diambil dari tabel, elemen null muncul di blok data untuk tabel itu.
- Hasil pemanggilan diurutkan per tabel, berdasarkan urutan penyediaannya di dalam template pemetaan permintaan.
- Setiap `Get` perintah di dalam a `BatchGetItem` adalah atom, namun, batch dapat diproses sebagian. Jika batch diproses sebagian karena kesalahan, kunci yang belum diproses dikembalikan sebagai bagian dari hasil pemanggilan di dalam blok UnprocessedKeys.
- `BatchGetItem`terbatas pada 100 kunci.

Untuk contoh template pemetaan permintaan berikut:

```
{
  "version": "2018-05-29",
  "operation": "BatchGetItem",
  "tables": {
    "authors": [
        {
          "author_id": {
            "S": "a1"
          }
        },
    ],
    "posts": [
        {
          "author_id": {
            "S": "a1"
```

```
          },
          "post_id": {
            "S": "p2"
          }
        }
    ],
  }
}
```

Hasil pemanggilan yang tersedia `$ctx.result` adalah sebagai berikut:

```
{
   "data": {
     "authors": [null],
     "posts": [
        # Was retrieved
        {
          "author_id": "a1",
          "post_id": "p2",
          "post_title": "title",
          "post_description": "description",
        }
     ]
   },
   "unprocessedKeys": {
     "authors": [
        # This item was not processed due to an error
        {
          "author_id": "a1"
        }
      ],
     "posts": []
   }
}
```

`$ctx.error`Berisi rincian tentang kesalahan. Data kunci, UnprocessedKeys, dan setiap kunci tabel yang disediakan dalam template pemetaan permintaan dijamin akan hadir dalam hasil pemanggilan. Item yang telah dihapus muncul di blok data. Item yang belum diproses ditandai sebagai null di dalam blok data dan ditempatkan di dalam blok UnprocessedKeys.

Untuk contoh yang lebih lengkap, ikuti tutorial DynamoDB Batch dengan di sini AppSync Tutorial: DynamoDB batch resolvers.

BatchDeleteItem

Dokumen pemetaan `BatchDeleteItem` permintaan memungkinkan Anda memberi tahu AWS AppSync DynamoDB resolver untuk membuat `BatchWriteItem` permintaan ke DynamoDB untuk menghapus beberapa item, berpotensi di beberapa tabel. Untuk template permintaan ini, Anda harus menentukan yang berikut:

- Nama tabel tempat menghapus item
- Kunci item yang akan dihapus dari setiap tabel

Batas `BatchWriteItem` DynamoDB berlaku dan tidak ada ekspresi kondisi yang dapat diberikan.

Dokumen `BatchDeleteItem` pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2018-05-29",
    "operation" : "BatchDeleteItem",
    "tables" : {
        "table1": [
        ## Item to delete Key
        {
             "foo" : ... typed value,
             "bar" : ... typed value
        },
        ## Item2 to delete Key
        {
             "foo" : ... typed value,
             "bar" : ... typed value
        }],
        "table2": [
        ## Item3 to delete Key
        {
             "foo" : ... typed value,
             "bar" : ... typed value
        },
        ## Item4 to delete Key
        {
             "foo" : ... typed value,
             "bar" : ... typed value
        }],
    }
```

```
}
```

Bidang didefinisikan sebagai berikut:

BatchDeleteItembidang

BatchDeleteItemdaftar bidang

`version`

Versi definisi template. Hanya `2018-05-29` didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `BatchDeleteItem` DynamoDB, ini harus diatur ke. `BatchDeleteItem` Nilai ini diperlukan.

`tables`

Tabel DynamoDB untuk menghapus item dari. Setiap tabel adalah daftar kunci DynamoDB yang mewakili kunci utama item yang akan dihapus. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan "nilai yang diketik", lihat Mengetik sistem (pemetaan permintaan). Setidaknya satu meja harus disediakan. `tables`Nilai diperlukan.

Hal-hal yang perlu diingat:

- Berlawanan dengan `DeleteItem` operasi, item yang dihapus sepenuhnya tidak dikembalikan dalam respons. Hanya kunci yang dilewati yang dikembalikan.
- Jika item belum dihapus dari tabel, elemen null muncul di blok data untuk tabel itu.
- Hasil pemanggilan diurutkan per tabel, berdasarkan urutan penyediaannya di dalam template pemetaan permintaan.
- Setiap `Delete` perintah di dalam a `BatchDeleteItem` adalah atom. Namun batch dapat diproses sebagian. Jika batch diproses sebagian karena kesalahan, kunci yang belum diproses dikembalikan sebagai bagian dari hasil pemanggilan di dalam blok UnprocessedKeys.
- `BatchDeleteItem`terbatas pada 25 kunci.

Untuk contoh template pemetaan permintaan berikut:

```
{
  "version": "2018-05-29",
  "operation": "BatchDeleteItem",
  "tables": {
    "authors": [
        {
          "author_id": {
            "S": "a1"
          }
        },
    ],
    "posts": [
        {
          "author_id": {
            "S": "a1"
          },
          "post_id": {
            "S": "p2"
          }
        }
    ],
  }
}
```

Hasil pemanggilan yang tersedia `$ctx.result` adalah sebagai berikut:

```
{
   "data": {
     "authors": [null],
     "posts": [
        # Was deleted
        {
          "author_id": "a1",
          "post_id": "p2"
        }
     ]
   },
   "unprocessedKeys": {
     "authors": [
        # This key was not processed due to an error
        {
          "author_id": "a1"
        }
```

```
      ],
     "posts": []
   }
}
```

`$ctx.error`Berisi rincian tentang kesalahan. Data kunci, UnprocessedKeys, dan setiap kunci tabel yang disediakan dalam template pemetaan permintaan dijamin akan hadir dalam hasil pemanggilan. Item yang telah dihapus ada di blok data. Item yang belum diproses ditandai sebagai null di dalam blok data dan ditempatkan di dalam blok UnprocessedKeys.

Untuk contoh yang lebih lengkap, ikuti tutorial DynamoDB Batch dengan di sini AppSync Tutorial: DynamoDB batch resolvers.

BatchPutItem

Dokumen pemetaan `BatchPutItem` permintaan memungkinkan Anda memberi tahu AWS AppSync DynamoDB resolver untuk membuat `BatchWriteItem` permintaan ke DynamoDB untuk menempatkan beberapa item, berpotensi di beberapa tabel. Untuk template permintaan ini, Anda harus menentukan yang berikut:

- Nama tabel tempat meletakkan item
- Item lengkap untuk dimasukkan ke dalam setiap tabel

Batas `BatchWriteItem` DynamoDB berlaku dan tidak ada ekspresi kondisi yang dapat diberikan.

Dokumen `BatchPutItem` pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2018-05-29",
    "operation" : "BatchPutItem",
    "tables" : {
        "table1": [
        ## Item to put
        {
             "foo" : ... typed value,
             "bar" : ... typed value
        },
        ## Item2 to put
        {
             "foo" : ... typed value,
             "bar" : ... typed value
```

```
        }],
        "table2": [
        ## Item3 to put
        {
             "foo" : ... typed value,
             "bar" : ... typed value
        },
        ## Item4 to put
        {
             "foo" : ... typed value,
             "bar" : ... typed value
        }],
    }
}
```

Bidang didefinisikan sebagai berikut:

BatchPutItembidang

BatchPutItemdaftar bidang

`version`

Versi definisi template. Hanya `2018-05-29` didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `BatchPutItem` DynamoDB, ini harus diatur ke. `BatchPutItem` Nilai ini diperlukan.

`tables`

Tabel DynamoDB untuk menempatkan item di. Setiap entri tabel mewakili daftar item DynamoDB untuk disisipkan untuk tabel khusus ini. Setidaknya satu meja harus disediakan. Nilai ini diperlukan.

Hal-hal yang perlu diingat:

- Item yang dimasukkan sepenuhnya dikembalikan dalam respons, jika berhasil.
- Jika item belum dimasukkan dalam tabel, elemen null ditampilkan di blok data untuk tabel tersebut.
- Item yang disisipkan diurutkan per tabel, berdasarkan urutan penyediaannya di dalam templat pemetaan permintaan.

- Setiap `Put` perintah di dalam a `BatchPutItem` adalah atom, namun, batch dapat diproses sebagian. Jika batch diproses sebagian karena kesalahan, kunci yang belum diproses dikembalikan sebagai bagian dari hasil pemanggilan di dalam blok UnprocessedKeys.
- `BatchPutItem`terbatas pada 25 item.

Untuk contoh template pemetaan permintaan berikut:

```
{
  "version": "2018-05-29",
  "operation": "BatchPutItem",
  "tables": {
    "authors": [
        {
          "author_id": {
            "S": "a1"
          },
          "author_name": {
            "S": "a1_name"
          }
        },
    ],
    "posts": [
        {
          "author_id": {
            "S": "a1"
          },
          "post_id": {
            "S": "p2"
          },
          "post_title": {
            "S": "title"
          }
        }
    ],
  }
}
```

Hasil pemanggilan yang tersedia `$ctx.result` adalah sebagai berikut:

```
{
   "data": {
```

```
    "authors": [
        null
    ],
    "posts": [
       # Was inserted
       {
         "author_id": "a1",
         "post_id": "p2",
         "post_title": "title"
       }
    ]
  },
  "unprocessedItems": {
    "authors": [
       # This item was not processed due to an error
       {
         "author_id": "a1",
         "author_name": "a1_name"
       }
     ],
    "posts": []
  }
}
```

`$ctx.error`Berisi rincian tentang kesalahan. Data kunci, UnprocessEditems, dan setiap kunci tabel yang disediakan dalam template pemetaan permintaan dijamin akan hadir dalam hasil pemanggilan. Item yang telah dimasukkan ada di blok data. Item yang belum diproses ditandai sebagai null di dalam blok data dan ditempatkan di dalam blok UnProcessEditems.

Untuk contoh yang lebih lengkap, ikuti tutorial DynamoDB Batch dengan di sini AppSync Tutorial: DynamoDB batch resolvers.

TransactGetItems

Dokumen pemetaan `TransactGetItems` permintaan memungkinkan Anda memberi tahu AWS AppSync DynamoDB resolver untuk membuat `TransactGetItems` permintaan ke DynamoDB untuk mengambil beberapa item, berpotensi di beberapa tabel. Untuk template permintaan ini, Anda harus menentukan yang berikut:

- Nama tabel dari setiap item permintaan tempat untuk mengambil item dari
- Kunci dari setiap item permintaan untuk diambil dari setiap tabel

Batas `TransactGetItems` DynamoDB berlaku dan tidak ada ekspresi kondisi yang dapat diberikan.

Dokumen `TransactGetItems` pemetaan memiliki struktur sebagai berikut:

```
{
    "version": "2018-05-29",
    "operation": "TransactGetItems",
    "transactItems": [
       ## First request item
       {
           "table": "table1",
           "key": {
               "foo": ... typed value,
               "bar": ... typed value
           },
           "projection" : {
                ...
           }
       },
       ## Second request item
       {
           "table": "table2",
           "key": {
               "foo": ... typed value,
               "bar": ... typed value
           },
           "projection" : {
                ...
           }
       }
    ]
}
```

Bidang didefinisikan sebagai berikut:

TransactGetItemsbidang

TransactGetItemsdaftar bidang

`version`

Versi definisi template. Hanya `2018-05-29` didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `TransactGetItems` DynamoDB, ini harus diatur ke. `TransactGetItems` Nilai ini diperlukan.

`transactItems`

Item permintaan untuk disertakan. Nilai adalah array item permintaan. Setidaknya satu item permintaan harus disediakan. `transactItems`Nilai ini diperlukan.

`table`

Tabel DynamoDB untuk mengambil item dari. Nilai adalah string dari nama tabel. `table`Nilai ini diperlukan.

`key`

Kunci DynamoDB mewakili kunci utama item yang akan diambil. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan).

`projection`

Proyeksi yang digunakan untuk menentukan atribut yang akan dikembalikan dari operasi DynamoDB. Untuk informasi selengkapnya tentang proyeksi, lihat Proyeksi. Bidang ini bersifat opsional.

Hal-hal yang perlu diingat:

- Jika transaksi berhasil, urutan item yang diambil di `items` blok akan sama dengan urutan item permintaan.
- Transaksi dilakukan dengan all-or-nothing cara tertentu. Jika ada item permintaan yang menyebabkan kesalahan, seluruh transaksi tidak akan dilakukan dan rincian kesalahan akan dikembalikan.
- Item permintaan yang tidak dapat diambil bukanlah kesalahan. Sebagai gantinya, elemen null muncul di blok item di posisi yang sesuai.
- Jika kesalahan transaksi adalah TransactionCanceledException, `cancellationReasons` blok akan diisi. Urutan alasan pembatalan di `cancellationReasons` blok akan sama dengan urutan item permintaan.

- `TransactGetItems`terbatas pada 25 item permintaan.

Untuk contoh template pemetaan permintaan berikut:

```
{
    "version": "2018-05-29",
    "operation": "TransactGetItems",
    "transactItems": [
       ## First request item
       {
           "table": "posts",
           "key": {
               "post_id": {
                 "S": "p1"
               }
           }
       },
       ## Second request item
       {
           "table": "authors",
           "key": {
               "author_id": {
                 "S": a1
               }
           }
       }
    ]
}
```

Jika transaksi berhasil dan hanya item yang diminta pertama yang diambil, hasil pemanggilan yang tersedia adalah sebagai berikut: `$ctx.result`

```
{
    "items": [
       {
           // Attributes of the first requested item
           "post_id": "p1",
           "post_title": "title",
           "post_description": "description"
       },
       // Could not retrieve the second requested item
       null,
```

```
    ],
    "cancellationReasons": null
}
```

Jika transaksi gagal karena TransactionCanceledExceptiondisebabkan oleh item permintaan pertama, hasil pemanggilan yang tersedia `$ctx.result` adalah sebagai berikut:

```
{
    "items": null,
    "cancellationReasons": [
       {
           "type":"Sample error type",
           "message":"Sample error message"
       },
       {
           "type":"None",
           "message":"None"
       }
    ]
}
```

`$ctx.error`Berisi rincian tentang kesalahan. Item kunci dan CancellationReasons dijamin akan hadir di. `$ctx.result`

Untuk contoh yang lebih lengkap, ikuti tutorial Transaksi DynamoDB AppSync dengan tutorial di sini: DynamoDB transaction resolvers.

TransactWriteItems

Dokumen pemetaan `TransactWriteItems` permintaan memungkinkan Anda memberi tahu AWS AppSync DynamoDB resolver untuk membuat `TransactWriteItems` permintaan ke DynamoDB untuk menulis beberapa item, berpotensi ke beberapa tabel. Untuk template permintaan ini, Anda harus menentukan yang berikut:

- Nama tabel tujuan dari setiap item permintaan
- Pengoperasian setiap item permintaan untuk dilakukan. Ada empat jenis operasi yang didukung: PutItem, UpdateItem, DeleteItem, dan ConditionCheck
- Kunci dari setiap item permintaan untuk menulis

Batas DynamoDB `TransactWriteItems` berlaku.

Dokumen `TransactWriteItems` pemetaan memiliki struktur sebagai berikut:

```
{
    "version": "2018-05-29",
    "operation": "TransactWriteItems",
    "transactItems": [
       {
           "table": "table1",
           "operation": "PutItem",
           "key": {
               "foo": ... typed value,
               "bar": ... typed value
           },
           "attributeValues": {
               "baz": ... typed value
           },
           "condition": {
               "expression": "someExpression",
               "expressionNames": {
                   "#foo": "foo"
               },
               "expressionValues": {
                   ":bar": ... typed value
               },
               "returnValuesOnConditionCheckFailure": true|false
           }
       },
       {
           "table":"table2",
           "operation": "UpdateItem",
           "key": {
               "foo": ... typed value,
               "bar": ... typed value
           },
           "update": {
               "expression": "someExpression",
               "expressionNames": {
                   "#foo": "foo"
               },
               "expressionValues": {
                   ":bar": ... typed value
               }
           },
           "condition": {
```

```
                "expression": "someExpression",
                "expressionNames": {
                    "#foo":"foo"
                },
                "expressionValues": {
                    ":bar": ... typed value
                },
                "returnValuesOnConditionCheckFailure": true|false
            }
        },
        {
            "table": "table3",
            "operation": "DeleteItem",
            "key":{
                "foo": ... typed value,
                "bar": ... typed value
            },
            "condition":{
                "expression": "someExpression",
                "expressionNames": {
                    "#foo": "foo"
                },
                "expressionValues": {
                    ":bar": ... typed value
                },
                "returnValuesOnConditionCheckFailure": true|false
            }
        },
        {
            "table": "table4",
            "operation": "ConditionCheck",
            "key":{
                "foo": ... typed value,
                "bar": ... typed value
            },
            "condition":{
                "expression": "someExpression",
                "expressionNames": {
                    "#foo": "foo"
                },
                "expressionValues": {
                    ":bar": ... typed value
                },
                "returnValuesOnConditionCheckFailure": true|false
```

```
            }
        }
    ]
}
```

TransactWriteItemsbidang

TransactWriteItemsdaftar bidang

Bidang didefinisikan sebagai berikut:

`version`

Versi definisi template. Hanya `2018-05-29` didukung. Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `TransactWriteItems` DynamoDB, ini harus diatur ke. `TransactWriteItems` Nilai ini diperlukan.

`transactItems`

Item permintaan untuk disertakan. Nilai adalah array item permintaan. Setidaknya satu item permintaan harus disediakan. `transactItems`Nilai ini diperlukan.

Untuk`PutItem`, bidang didefinisikan sebagai berikut:

`table`

Tabel DynamoDB tujuan. Nilai adalah string dari nama tabel. `table`Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `PutItem` DynamoDB, ini harus diatur ke. `PutItem` Nilai ini diperlukan.

`key`

Kunci DynamoDB mewakili kunci utama item yang akan diletakkan. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Nilai ini diperlukan.

`attributeValues`

Sisa atribut item yang akan dimasukkan ke DynamoDB. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Bidang ini bersifat opsional.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan, `PutItem` permintaan akan menimpa entri yang ada untuk item tersebut. Anda dapat menentukan apakah akan mengambil kembali item yang ada saat pemeriksaan kondisi gagal. Untuk informasi selengkapnya tentang kondisi transaksional, lihat Ekspresi kondisi transaksi. Nilai ini bersifat opsional.

Untuk`UpdateItem`, bidang didefinisikan sebagai berikut:

`table`

Tabel DynamoDB untuk memperbarui. Nilai adalah string dari nama tabel. `table`Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `UpdateItem` DynamoDB, ini harus diatur ke. `UpdateItem` Nilai ini diperlukan.

`key`

Kunci DynamoDB mewakili kunci utama item yang akan diperbarui. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Nilai ini diperlukan.

`update`

`update`Bagian ini memungkinkan Anda menentukan ekspresi pembaruan yang menjelaskan cara memperbarui item di DynamoDB. Untuk informasi selengkapnya tentang cara menulis ekspresi pembaruan, lihat dokumentasi DynamoDB UpdateExpressions . Bagian ini diperlukan.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang

ditentukan, `UpdateItem` permintaan akan memperbarui entri yang ada terlepas dari statusnya saat ini. Anda dapat menentukan apakah akan mengambil kembali item yang ada saat pemeriksaan kondisi gagal. Untuk informasi selengkapnya tentang kondisi transaksional, lihat Ekspresi kondisi transaksi. Nilai ini bersifat opsional.

Untuk`DeleteItem`, bidang didefinisikan sebagai berikut:

`table`

Tabel DynamoDB untuk menghapus item. Nilai adalah string dari nama tabel. `table`Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `DeleteItem` DynamoDB, ini harus diatur ke. `DeleteItem` Nilai ini diperlukan.

`key`

Kunci DynamoDB mewakili kunci utama item yang akan dihapus. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Nilai ini diperlukan.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Jika tidak ada kondisi yang ditentukan, `DeleteItem` permintaan menghapus item terlepas dari keadaan saat ini. Anda dapat menentukan apakah akan mengambil kembali item yang ada saat pemeriksaan kondisi gagal. Untuk informasi selengkapnya tentang kondisi transaksional, lihat Ekspresi kondisi transaksi. Nilai ini bersifat opsional.

Untuk`ConditionCheck`, bidang didefinisikan sebagai berikut:

`table`

Tabel DynamoDB untuk memeriksa kondisi. Nilai adalah string dari nama tabel. `table`Nilai ini diperlukan.

`operation`

Operasi DynamoDB untuk melakukan. Untuk melakukan operasi `ConditionCheck` DynamoDB, ini harus diatur ke. `ConditionCheck` Nilai ini diperlukan.

`key`

Kunci DynamoDB mewakili kunci utama item untuk pemeriksaan kondisi. Item DynamoDB mungkin memiliki kunci hash tunggal, atau kunci hash dan kunci sortir, tergantung pada struktur tabel. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Mengetik sistem (pemetaan permintaan). Nilai ini diperlukan.

`condition`

Suatu kondisi untuk menentukan apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB. Anda dapat menentukan apakah akan mengambil kembali item yang ada saat pemeriksaan kondisi gagal. Untuk informasi selengkapnya tentang kondisi transaksional, lihat Ekspresi kondisi transaksi. Nilai ini diperlukan.

Hal-hal yang perlu diingat:

- Hanya kunci item permintaan yang dikembalikan dalam respons, jika berhasil. Urutan kunci akan sama dengan urutan item permintaan.
- Transaksi dilakukan dengan all-or-nothing cara tertentu. Jika ada item permintaan yang menyebabkan kesalahan, seluruh transaksi tidak akan dilakukan dan rincian kesalahan akan dikembalikan.
- Tidak ada dua item permintaan yang dapat menargetkan item yang sama. Kalau tidak, mereka akan menyebabkan TransactionCanceledExceptionkesalahan.
- Jika kesalahan transaksi adalah TransactionCanceledException, `cancellationReasons` blok akan diisi. Jika pemeriksaan kondisi item permintaan gagal dan Anda tidak `returnValuesOnConditionCheckFailure` menentukan`false`, item yang ada di tabel akan diambil dan disimpan `item` di posisi `cancellationReasons` blok yang sesuai.
- `TransactWriteItems`sterbatas pada 25 item permintaan.

Untuk contoh template pemetaan permintaan berikut:

```
{
    "version": "2018-05-29",
    "operation": "TransactWriteItems",
    "transactItems": [
       {
           "table": "posts",
```

```
            "operation": "PutItem",
            "key": {
                "post_id": {
                    "S": "p1"
                }
            },
            "attributeValues": {
                "post_title": {
                    "S": "New title"
                },
                "post_description": {
                    "S": "New description"
                }
            },
            "condition": {
                "expression": "post_title = :post_title",
                "expressionValues": {
                    ":post_title": {
                        "S": "Expected old title"
                    }
                }
            }
        },
        {
            "table":"authors",
            "operation": "UpdateItem",
            "key": {
                "author_id": {
                    "S": "a1"
                },
            },
            "update": {
                "expression": "SET author_name = :author_name",
                "expressionValues": {
                    ":author_name": {
                        "S": "New name"
                    }
                }
            },
        }
    ]
}
```

Jika transaksi berhasil, hasil pemanggilan yang tersedia `$ctx.result` adalah sebagai berikut:

```
{
    "keys": [
       // Key of the PutItem request
       {
           "post_id": "p1",
       },
       // Key of the UpdateItem request
       {
           "author_id": "a1"
       }
    ],
    "cancellationReasons": null
}
```

Jika transaksi gagal karena kegagalan pemeriksaan kondisi `PutItem` permintaan, hasil pemanggilan yang tersedia `$ctx.result` adalah sebagai berikut:

```
{
    "keys": null,
    "cancellationReasons": [
       {
           "item": {
               "post_id": "p1",
               "post_title": "Actual old title",
               "post_description": "Old description"
           },
           "type": "ConditionCheckFailed",
           "message": "The condition check failed."
       },
       {
           "type": "None",
           "message": "None"
       }
    ]
}
```

`$ctx.error`Berisi rincian tentang kesalahan. Kunci kunci dan CancellationReasons dijamin akan ada di. `$ctx.result`

Untuk contoh yang lebih lengkap, ikuti tutorial Transaksi DynamoDB AppSync dengan tutorial di sini: DynamoDB transaction resolvers.

Jenis sistem (pemetaan permintaan)

Saat menggunakan AWS AppSync DynamoDB resolver untuk memanggil AWS AppSync tabel DynamoDB Anda, perlu mengetahui jenis setiap nilai yang akan digunakan dalam panggilan itu. Ini karena DynamoDB mendukung lebih banyak tipe primitif daripada GraphQL atau JSON (seperti set dan data biner). AWS AppSync membutuhkan beberapa petunjuk saat menerjemahkan antara GraphQL dan DynamoDB, jika tidak maka harus membuat beberapa asumsi tentang bagaimana data disusun dalam tabel Anda.

Untuk informasi selengkapnya tentang tipe data DynamoDB, lihat deskriptor tipe Data DynamoDB dan dokumentasi tipe data.

Nilai DynamoDB diwakili oleh objek JSON yang berisi pasangan kunci-nilai tunggal. Kunci menentukan jenis DynamoDB, dan nilai menentukan nilai itu sendiri. Dalam contoh berikut, kunci `S` menunjukkan bahwa nilainya adalah string, dan nilainya `identifier` adalah nilai string itu sendiri.

```
{ "S" : "identifier" }
```

Perhatikan bahwa objek JSON tidak dapat memiliki lebih dari satu pasangan kunci-nilai. Jika lebih dari satu pasangan kunci-nilai ditentukan, dokumen pemetaan permintaan tidak diuraikan.

Nilai DynamoDB digunakan di mana saja dalam dokumen pemetaan permintaan di mana Anda perlu menentukan nilai. Beberapa tempat di mana Anda perlu melakukan ini termasuk: `key` dan `attributeValue` bagian, dan `expressionValues` bagian dari bagian ekspresi. Dalam contoh berikut, `identifier` nilai DynamoDB String sedang ditetapkan ke `id` bidang di bagian (mungkin dalam `key` `GetItem` dokumen pemetaan permintaan).

```
"key" : {
   "id" : { "S" : "identifier" }
}
```

Jenis yang Didukung

AWS AppSync mendukung skalar DynamoDB, dokumen, dan jenis set berikut:

Jenis string **S**

Nilai string tunggal. Nilai DynamoDB String dilambangkan dengan:

```
{ "S" : "some string" }
```

Contoh penggunaan adalah:

```
"key" : {
   "id" : { "S" : "some string" }
}
```

Jenis set string **SS**

Satu set nilai string. Nilai DynamoDB String Set dilambangkan dengan:

```
{ "SS" : [ "first value", "second value", ... ] }
```

Contoh penggunaan adalah:

```
"attributeValues" : {
   "phoneNumbers" : { "SS" : [ "+1 555 123 4567", "+1 555 234 5678" ] }
}
```

Jenis nomor **N**

Nilai numerik tunggal. Nilai Nomor DynamoDB dilambangkan dengan:

```
{ "N" : 1234 }
```

Contoh penggunaan adalah:

```
"expressionValues" : {
   ":expectedVersion" : { "N" : 1 }
}
```

Jenis set nomor **NS**

Satu set nilai angka. Nilai DynamoDB Number Set dilambangkan dengan:

```
{ "NS" : [ 1, 2.3, 4 ... ] }
```

Contoh penggunaan adalah:

```
"attributeValues" : {
   "sensorReadings" : { "NS" : [ 67.8, 12.2, 70 ] }
}
```

Jenis biner **B**

Nilai biner. Nilai biner DynamoDB dilambangkan dengan:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

Perhatikan bahwa nilainya sebenarnya adalah string, di mana string adalah representasi yang dikodekan base64 dari data biner. AWS AppSync mendekode string ini kembali ke nilai binernya sebelum mengirimnya ke DynamoDB. AWS AppSync menggunakan skema decoding base64 seperti yang didefinisikan oleh RFC 2045: karakter apa pun yang tidak ada dalam alfabet base64 diabaikan.

Contoh penggunaan adalah:

```
"attributeValues" : {
   "binaryMessage" : { "B" : "SGVsbG8sIFdvcmxkIQo=" }
}
```

Jenis set biner **BS**

Satu set nilai biner. Nilai DynamoDB Binary Set dilambangkan dengan:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

Perhatikan bahwa nilainya sebenarnya adalah string, di mana string adalah representasi yang dikodekan base64 dari data biner. AWS AppSync mendekode string ini kembali ke nilai binernya sebelum mengirimnya ke DynamoDB. AWS AppSync menggunakan skema decoding base64 seperti yang didefinisikan oleh RFC 2045: karakter apa pun yang tidak ada dalam alfabet base64 diabaikan.

Contoh penggunaan adalah:

```
"attributeValues" : {
   "binaryMessages" : { "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ] }
}
```

Jenis Boolean **`BOOL`**

Nilai Boolean. Nilai DynamoDB Boolean dilambangkan dengan:

```
{ "BOOL" : true }
```

Perhatikan bahwa hanya `true` dan `false` merupakan nilai yang valid.

Contoh penggunaan adalah:

```
"attributeValues" : {
   "orderComplete" : { "BOOL" : false }
}
```

Jenis daftar **`L`**

Daftar nilai DynamoDB lain yang didukung. Nilai Daftar DynamoDB dilambangkan dengan:

```
{ "L" : [ ... ] }
```

Perhatikan bahwa nilainya adalah nilai gabungan, di mana daftar dapat berisi nol atau lebih dari nilai DynamoDB yang didukung (termasuk daftar lainnya). Daftar ini juga dapat berisi campuran dari berbagai jenis.

Contoh penggunaan adalah:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

Jenis peta **`M`**

Mewakili kumpulan pasangan kunci-nilai yang tidak berurutan dari nilai DynamoDB lain yang didukung. Nilai DynamoDB Map dilambangkan dengan:

```
{ "M" : { ... } }
```

Perhatikan bahwa peta dapat berisi nol atau lebih pasangan nilai kunci. Kuncinya harus berupa string, dan nilainya dapat berupa nilai DynamoDB yang didukung (termasuk peta lainnya). Peta juga dapat berisi campuran dari berbagai jenis.

Contoh penggunaan adalah:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

Tipe nol **`NULL`**

Sebuah nilai nol. Nilai DynamoDB Null dilambangkan dengan:

```
{ "NULL" : null }
```

Contoh penggunaan adalah:

```
"attributeValues" : {
   "phoneNumbers" : { "NULL" : null }
}
```

Untuk informasi selengkapnya tentang setiap jenis, lihat dokumentasi DynamoDB.

Jenis sistem (pemetaan respons)

Saat menerima respons dari DynamoDB AWS AppSync , secara otomatis mengubahnya menjadi tipe primitif GraphQL dan JSON. Setiap atribut di DynamoDB diterjemahkan dan dikembalikan dalam konteks pemetaan respons.

Misalnya, jika DynamoDB mengembalikan berikut ini:

```
{
    "id" : { "S" : "1234" },
    "name" : { "S" : "Nadia" },
    "age" : { "N" : 25 }
```

```
}
```

Kemudian AWS AppSync DynamoDB resolver mengubahnya menjadi tipe GraphQL dan JSON sebagai:

```
{
    "id" : "1234",
    "name" : "Nadia",
    "age" : 25
}
```

Bagian ini menjelaskan cara AWS AppSync mengonversi skalar DynamoDB berikut, dokumen, dan jenis set:

Jenis string **S**

Nilai string tunggal. Nilai DynamoDB String dikembalikan sebagai string.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB String berikut:

```
{ "S" : "some string" }
```

AWS AppSync mengubahnya menjadi string:

```
"some string"
```

Jenis set string **SS**

Satu set nilai string. Nilai DynamoDB String Set dikembalikan sebagai daftar string.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB String Set berikut:

```
{ "SS" : [ "first value", "second value", ... ] }
```

AWS AppSync mengubahnya menjadi daftar string:

```
[ "+1 555 123 4567", "+1 555 234 5678" ]
```

Jenis nomor **N**

Nilai numerik tunggal. Nilai DynamoDB Number dikembalikan sebagai angka.

Misalnya, jika DynamoDB mengembalikan nilai Nomor DynamoDB berikut:

```
{ "N" : 1234 }
```

AWS AppSync mengubahnya menjadi angka:

```
1234
```

Jenis set nomor **NS**

Satu set nilai angka. Nilai DynamoDB Number Set dikembalikan sebagai daftar angka.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Number Set berikut:

```
{ "NS" : [ 67.8, 12.2, 70 ] }
```

AWS AppSync mengubahnya menjadi daftar angka:

```
[ 67.8, 12.2, 70 ]
```

Jenis biner **B**

Nilai biner. Nilai DynamoDB Binary dikembalikan sebagai string yang berisi representasi base64 dari nilai tersebut.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Binary berikut:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

AWS AppSync mengubahnya menjadi string yang berisi representasi base64 dari nilai:

```
"SGVsbG8sIFdvcmxkIQo="
```

Perhatikan bahwa data biner dikodekan dalam skema pengkodean base64 seperti yang ditentukan dalam RFC 4648 dan RFC 2045.

Jenis set biner **BS**

Satu set nilai biner. Nilai DynamoDB Binary Set dikembalikan sebagai daftar string yang berisi representasi nilai base64.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Binary Set berikut:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

AWS AppSync mengubahnya menjadi daftar string yang berisi representasi base64 dari nilai-nilai:

```
[ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ]
```

Perhatikan bahwa data biner dikodekan dalam skema pengkodean base64 seperti yang ditentukan dalam RFC 4648 dan RFC 2045.

Jenis Boolean **BOOL**

Nilai Boolean. Nilai DynamoDB Boolean dikembalikan sebagai Boolean.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Boolean berikut:

```
{ "BOOL" : true }
```

AWS AppSync mengubahnya menjadi Boolean:

```
true
```

Jenis daftar **L**

Daftar nilai DynamoDB lain yang didukung. Nilai Daftar DynamoDB dikembalikan sebagai daftar nilai, di mana setiap nilai batin juga dikonversi.

Misalnya, jika DynamoDB mengembalikan nilai Daftar DynamoDB berikut:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

AWS AppSync mengubahnya menjadi daftar nilai yang dikonversi:

```
[ "A string value", 1, [ "Another string value", "Even more string values!" ] ]
```

Jenis peta **`M`**

Kumpulan kunci/nilai dari nilai DynamoDB lain yang didukung. Nilai DynamoDB Map dikembalikan sebagai objek JSON, di mana setiap kunci/nilai juga dikonversi.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Map berikut:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

AWS AppSync mengubahnya menjadi objek JSON:

```
{
   "someString" : "A string value",
   "someNumber" : 1,
   "stringSet"  : [ "Another string value", "Even more string values!" ]
}
```

Tipe nol **`NULL`**

Sebuah nilai nol.

Misalnya, jika DynamoDB mengembalikan nilai DynamoDB Null berikut:

```
{ "NULL" : null }
```

AWS AppSync mengubahnya menjadi nol:

```
null
```

Filter

Saat menanyakan objek di DynamoDB menggunakan `Query` operasi `Scan` dan, Anda dapat secara opsional menentukan `filter` a yang mengevaluasi hasil dan hanya mengembalikan nilai yang diinginkan.

Bagian pemetaan filter dari dokumen `Query` atau `Scan` pemetaan memiliki struktur berikut:

```
"filter" : {
    "expression" : "filter expression"
    "expressionNames" : {
        "#name" : "name",
    },
    "expressionValues" : {
        ":value" : ... typed value
    },
}
```

Bidang didefinisikan sebagai berikut:

`expression`

Ekspresi kueri. Untuk informasi selengkapnya tentang cara menulis ekspresi filter, lihat dokumentasi DynamoDB dan QueryFilter ScanFilter DynamoDB. Bidang ini harus ditentukan.

`expressionNames`

Substitusi untuk placeholder nama atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci tersebut sesuai dengan placeholder nama yang digunakan dalam file. `expression` Nilai harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalam. `expression`

`expressionValues`

Substitusi untuk placeholder nilai atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nilai yang digunakan dalam`expression`, dan nilainya harus berupa nilai yang diketik. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Jenis Sistem (Permintaan Pemetaan). Ini harus ditentukan. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nilai atribut ekspresi yang digunakan dalam. `expression`

Contoh

Contoh berikut adalah bagian filter untuk template pemetaan, di mana entri diambil dari DynamoDB hanya dikembalikan jika judul dimulai dengan argumen. `title`

```
"filter" : {
```

```
    "expression" : "begins_with(#title, :title)",
    "expressionNames" : {
        "#title" : "title"
    },
    "expressionValues" : {
        ":title" : $util.dynamodb.toDynamoDBJson($context.arguments.title)
    }
}
```

Ekspresi kondisi

Saat Anda mengubah objek di DynamoDB dengan menggunakan operasi`PutItem`,`UpdateItem`, dan `DeleteItem` DynamoDB, Anda dapat secara opsional menentukan ekspresi kondisi yang mengontrol apakah permintaan harus berhasil atau tidak, berdasarkan status objek yang sudah ada di DynamoDB sebelum operasi dilakukan.

AWS AppSync DynamoDB resolver memungkinkan ekspresi kondisi untuk ditentukan `PutItem` dalam`UpdateItem`,, `DeleteItem` dan meminta dokumen pemetaan, dan juga strategi untuk mengikuti jika kondisi gagal dan objek tidak diperbarui.

Contoh 1

Dokumen `PutItem` pemetaan berikut tidak memiliki ekspresi kondisi. Akibatnya, ia menempatkan item di DynamoDB bahkan jika item dengan kunci yang sama sudah ada, sehingga menimpa item yang ada.

```
{
   "version" : "2017-02-28",
   "operation" : "PutItem",
   "key" : {
      "id" : { "S" : "1" }
   }
}
```

Contoh 2

Dokumen `PutItem` pemetaan berikut memang memiliki ekspresi kondisi yang memungkinkan operasi berhasil hanya jika item dengan kunci yang sama tidak ada di DynamoDB.

```
{
```

```
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key" : {
        "id" : { "S" : "1" }
    },
    "condition" : {
        "expression" : "attribute_not_exists(id)"
    }
}
```

Secara default, jika pemeriksaan kondisi gagal, penyelesai AWS AppSync DynamoDB mengembalikan kesalahan untuk mutasi dan nilai objek saat ini di DynamoDB di bidang di bagian respons GraphQL. `data` `error` Namun, AWS AppSync DynamoDB resolver menawarkan beberapa fitur tambahan untuk membantu pengembang menangani beberapa kasus tepi umum:

- Jika AWS AppSync DynamoDB resolver dapat menentukan bahwa nilai saat ini di DynamoDB cocok dengan hasil yang diinginkan, ia memperlakukan operasi seolah-olah berhasil pula.
- Alih-alih mengembalikan kesalahan, Anda dapat mengonfigurasi resolver untuk menjalankan fungsi Lambda khusus untuk memutuskan bagaimana penyelesai DynamoDB AWS AppSync harus menangani kegagalan.

Ini dijelaskan secara lebih rinci di bagian Handling a Condition Check Failure.

Untuk informasi selengkapnya tentang ekspresi kondisi DynamoDB, lihat dokumentasi DynamoDB. ConditionExpressions

Menentukan suatu kondisi

Dokumen pemetaan `PutItem` `UpdateItem`,, dan `DeleteItem` permintaan semuanya memungkinkan `condition` bagian opsional untuk ditentukan. Jika dihilangkan, tidak ada pemeriksaan kondisi yang dilakukan. Jika ditentukan, kondisi harus benar agar operasi berhasil.

`condition`Bagian memiliki struktur sebagai berikut:

```
"condition" : {
    "expression" : "someExpression"
    "expressionNames" : {
        "#foo" : "foo"
    },
    "expressionValues" : {
```

```
        ":bar" : ... typed value
    },
    "equalsIgnore" : [ "version" ],
    "consistentRead" : true,
    "conditionalCheckFailedHandler" : {
        "strategy" : "Custom",
        "lambdaArn" : "arn:..."
    }
}
```

Bidang berikut menentukan kondisi:

`expression`

Ekspresi pembaruan itu sendiri. Untuk informasi selengkapnya tentang cara menulis ekspresi kondisi, lihat dokumentasi DynamoDB ConditionExpressions . Bidang ini harus ditentukan.

`expressionNames`

Substitusi untuk placeholder nama atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nama yang digunakan dalam ekspresi, dan nilainya harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalam ekspresi.

`expressionValues`

Substitusi untuk placeholder nilai atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nilai yang digunakan dalam ekspresi, dan nilainya harus berupa nilai yang diketik. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Jenis Sistem (Permintaan Pemetaan). Ini harus ditentukan. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nilai atribut ekspresi yang digunakan dalam ekspresi.

Bidang yang tersisa memberi tahu AWS AppSync DynamoDB resolver cara menangani kegagalan pemeriksaan kondisi:

`equalsIgnore`

Ketika pemeriksaan kondisi gagal saat menggunakan `PutItem` operasi, penyelesai AWS AppSync DynamoDB membandingkan item yang saat ini ada di DynamoDB dengan item yang

coba ditulisnya. Jika mereka sama, itu memperlakukan operasi seolah-olah berhasil. Anda dapat menggunakan `equalsIgnore` bidang untuk menentukan daftar atribut yang AWS AppSync harus diabaikan saat melakukan perbandingan tersebut. Misalnya, jika satu-satunya perbedaan adalah `version` atribut, ia memperlakukan operasi seolah-olah berhasil. Bidang ini bersifat opsional.

`consistentRead`

Ketika pemeriksaan kondisi gagal, AWS AppSync dapatkan nilai item saat ini dari DynamoDB menggunakan pembacaan yang sangat konsisten. Anda dapat menggunakan bidang ini untuk memberi tahu penyelesai AWS AppSync DynamoDB agar menggunakan pembacaan yang konsisten pada akhirnya. Bidang ini opsional, dan defaultnya. `true`

`conditionalCheckFailedHandler`

Bagian ini memungkinkan Anda untuk menentukan bagaimana penyelesai AWS AppSync DynamoDB memperlakukan kegagalan pemeriksaan kondisi setelah membandingkan nilai saat ini di DynamoDB dengan hasil yang diharapkan. Bagian ini opsional. Jika dihilangkan, itu default ke strategi. `Reject`

`strategy`

Strategi yang diambil oleh AWS AppSync DynamoDB resolver setelah membandingkan nilai saat ini di DynamoDB dengan hasil yang diharapkan. Bidang ini diperlukan dan memiliki nilai yang mungkin berikut:

`Reject`

Mutasi gagal, dan kesalahan untuk mutasi dan nilai objek saat ini di DynamoDB di `data` bidang di bagian `error` respons GraphQL.

`Custom`

Penyelesai DynamoDB memanggil fungsi Lambda khusus untuk memutuskan cara menangani kegagalan pemeriksaan kondisi. AWS AppSync Ketika `strategy` diatur ke`Custom`, `lambdaArn` bidang harus berisi ARN dari fungsi Lambda untuk dipanggil.

`lambdaArn`

ARN dari fungsi Lambda untuk memanggil yang menentukan bagaimana penyelesai DynamoDB harus menangani kegagalan pemeriksaan AWS AppSync kondisi. Bidang ini hanya harus ditentukan ketika `strategy` diatur ke`Custom`. Untuk informasi selengkapnya tentang cara menggunakan fitur ini, lihat Menangani Kegagalan Pemeriksaan Kondisi.

Menangani kegagalan pemeriksaan kondisi

Secara default, ketika pemeriksaan kondisi gagal, penyelesai AWS AppSync DynamoDB mengembalikan kesalahan untuk mutasi dan nilai objek saat ini di DynamoDB di bidang di bagian respons GraphQL. `data` `error` Namun, AWS AppSync DynamoDB resolver menawarkan beberapa fitur tambahan untuk membantu pengembang menangani beberapa kasus tepi umum:

- Jika AWS AppSync DynamoDB resolver dapat menentukan bahwa nilai saat ini di DynamoDB cocok dengan hasil yang diinginkan, ia memperlakukan operasi seolah-olah berhasil pula.
- Alih-alih mengembalikan kesalahan, Anda dapat mengonfigurasi resolver untuk menjalankan fungsi Lambda khusus untuk memutuskan bagaimana penyelesai DynamoDB AWS AppSync harus menangani kegagalan.

Diagram alur untuk proses ini adalah:

Memeriksa hasil yang diinginkan

Ketika pemeriksaan kondisi gagal, penyelesai AWS AppSync DynamoDB melakukan permintaan DynamoDB `GetItem` untuk mendapatkan nilai item saat ini dari DynamoDB. Secara default, ini menggunakan pembacaan yang sangat konsisten, namun ini dapat dikonfigurasi menggunakan `consistentRead` bidang di `condition` blok dan membandingkannya dengan hasil yang diharapkan:

- Untuk `PutItem` operasi, AWS AppSync DynamoDB resolver membandingkan nilai saat ini terhadap nilai yang mencoba untuk menulis, tidak termasuk atribut yang tercantum dalam dari perbandingan. `equalsIgnore` Jika itemnya sama, ia memperlakukan operasi sebagai berhasil dan mengembalikan item yang diambil dari DynamoDB. Jika tidak, ia mengikuti strategi yang dikonfigurasi.

  Misalnya, jika dokumen pemetaan `PutItem` permintaan terlihat seperti berikut:

```
{
   "version" : "2017-02-28",
   "operation" : "PutItem",
   "key" : {
      "id" : { "S" : "1" }
   },
   "attributeValues" : {
```

```
      "name" : { "S" : "Steve" },
      "version" : { "N" : 2 }
   },
   "condition" : {
      "expression" : "version = :expectedVersion",
      "expressionValues" : {
          ":expectedVersion" : { "N" : 1 }
      },
      "equalsIgnore": [ "version" ]
   }
}
```

Dan item yang saat ini ada di DynamoDB terlihat seperti berikut:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

Penyelesai DynamoDB akan membandingkan item yang coba ditulisnya dengan nilai saat ini, melihat bahwa satu-satunya perbedaan adalah `version` bidang, tetapi karena dikonfigurasi untuk mengabaikan `version` bidang, ia memperlakukan operasi sebagai berhasil dan mengembalikan item yang diambil dari DynamoDB. AWS AppSync

- Untuk `DeleteItem` operasi, AWS AppSync DynamoDB resolver memeriksa untuk memverifikasi bahwa item dikembalikan dari DynamoDB. Jika tidak ada barang yang dikembalikan, itu memperlakukan operasi sebagai berhasil. Jika tidak, ia mengikuti strategi yang dikonfigurasi.
- Untuk `UpdateItem` operasi, penyelesai AWS AppSync DynamoDB tidak memiliki informasi yang cukup untuk menentukan apakah item yang saat ini ada di DynamoDB cocok dengan hasil yang diharapkan, dan oleh karena itu mengikuti strategi yang dikonfigurasi.

Jika status objek saat ini di DynamoDB berbeda dari hasil yang diharapkan, penyelesai AWS AppSync DynamoDB mengikuti strategi yang dikonfigurasi, untuk menolak mutasi atau memanggil fungsi Lambda untuk menentukan apa yang harus dilakukan selanjutnya.

Mengikuti strategi “tolak”

Saat mengikuti `Reject` strategi, penyelesai AWS AppSync DynamoDB mengembalikan kesalahan untuk mutasi, dan nilai objek saat ini di DynamoDB juga dikembalikan dalam bidang di bagian

respons GraphQL. `data` `error` Item yang dikembalikan dari DynamoDB dimasukkan melalui template pemetaan respons untuk menerjemahkannya ke dalam format yang diharapkan klien, dan difilter oleh set pilihan.

Misalnya, mengingat permintaan mutasi berikut:

```
mutation {
    updatePerson(id: 1, name: "Steve", expectedVersion: 1) {
        Name
        theVersion
    }
}
```

Jika item yang dikembalikan dari DynamoDB terlihat seperti berikut:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

Dan template pemetaan respons terlihat seperti berikut:

```
{
   "id" : $util.toJson($context.result.id),
   "Name" : $util.toJson($context.result.name),
   "theVersion" : $util.toJson($context.result.version)
}
```

Respons GraphQL terlihat seperti berikut:

```
{
  "data": null,
  "errors": [
    {
      "message": "The conditional request failed (Service: AmazonDynamoDBv2;
 Status Code: 400; Error Code: ConditionalCheckFailedException; Request ID:
 ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ)"
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      "data": {
        "Name": "Steve",
```

```
        "theVersion": 8
      },
      ...
    }
  ]
}
```

Juga, jika ada bidang dalam objek yang dikembalikan diisi oleh resolver lain dan mutasi telah berhasil, mereka tidak akan diselesaikan ketika objek dikembalikan di bagian. `error`

Mengikuti strategi “kustom”

Saat mengikuti `Custom` strategi, penyelesai AWS AppSync DynamoDB memanggil fungsi Lambda untuk memutuskan apa yang harus dilakukan selanjutnya. Fungsi Lambda memilih salah satu opsi berikut:

- `reject`mutasi. Ini memberitahu AWS AppSync DynamoDB resolver untuk berperilaku seolah-olah strategi yang dikonfigurasi `Reject` adalah, mengembalikan kesalahan untuk mutasi dan nilai objek saat ini di DynamoDB seperti yang dijelaskan di bagian sebelumnya.
- `discard`mutasi. Ini memberitahu AWS AppSync DynamoDB resolver untuk diam-diam mengabaikan kegagalan pemeriksaan kondisi dan mengembalikan nilai dalam DynamoDB.
- `retry`mutasi. Ini memberitahu AWS AppSync DynamoDB resolver untuk mencoba lagi mutasi dengan dokumen pemetaan permintaan baru.

Permintaan doa Lambda

Penyelesai AWS AppSync DynamoDB memanggil fungsi Lambda yang ditentukan dalam. `lambdaArn` Ini menggunakan `service-role-arn` konfigurasi yang sama pada sumber data. Muatan doa memiliki struktur sebagai berikut:

```
{
    "arguments": { ... },
    "requestMapping": {... },
    "currentValue": { ... },
    "resolver": { ... },
    "identity": { ... }
}
```

Bidang didefinisikan sebagai berikut:

`arguments`

Argumen dari mutasi GraphQL. Ini sama dengan argumen yang tersedia untuk dokumen pemetaan permintaan di`$context.arguments`.

`requestMapping`

Dokumen pemetaan permintaan untuk operasi ini.

`currentValue`

Nilai objek saat ini di DynamoDB.

`resolver`

Informasi tentang AWS AppSync resolver.

`identity`

Informasi tentang penelepon. Ini sama dengan informasi identitas yang tersedia untuk dokumen pemetaan permintaan di`$context.identity`.

Contoh lengkap dari muatan:

```
{
    "arguments": {
        "id": "1",
        "name": "Steve",
        "expectedVersion": 1
    },
    "requestMapping": {
        "version" : "2017-02-28",
        "operation" : "PutItem",
        "key" : {
           "id" : { "S" : "1" }
        },
        "attributeValues" : {
           "name" : { "S" : "Steve" },
           "version" : { "N" : 2 }
        },
        "condition" : {
           "expression" : "version = :expectedVersion",
           "expressionValues" : {
               ":expectedVersion" : { "N" : 1 }
           },
```

```
            "equalsIgnore": [ "version" ]
        }
    },
    "currentValue": {
        "id" : { "S" : "1" },
        "name" : { "S" : "Steve" },
        "version" : { "N" : 8 }
    },
    "resolver": {
        "tableName": "People",
        "awsRegion": "us-west-2",
        "parentType": "Mutation",
        "field": "updatePerson",
        "outputType": "Person"
    },
    "identity": {
        "accountId": "123456789012",
        "sourceIp": "x.x.x.x",
        "user": "AIDAAAAAAAAAAAAAAAAA",
        "userArn": "arn:aws:iam::123456789012:user/appsync"
    }
}
```

Tanggapan Doa Lambda

Fungsi Lambda dapat memeriksa payload pemanggilan dan menerapkan logika bisnis apa pun untuk memutuskan bagaimana penyelesai AWS AppSync DynamoDB harus menangani kegagalan. Ada tiga opsi untuk menangani kegagalan pemeriksaan kondisi:

- `reject`mutasi. Payload respons untuk opsi ini harus memiliki struktur ini:

  ```
  {
      "action": "reject"
  }
  ```

  Ini memberitahu AWS AppSync DynamoDB resolver untuk berperilaku seolah-olah strategi yang dikonfigurasi `Reject` adalah, mengembalikan kesalahan untuk mutasi dan nilai objek saat ini di DynamoDB, seperti yang dijelaskan pada bagian di atas.
- `discard`mutasi. Payload respons untuk opsi ini harus memiliki struktur ini:

  ```
  {
  ```

```
    "action": "discard"
}
```

Ini memberitahu AWS AppSync DynamoDB resolver untuk diam-diam mengabaikan kegagalan pemeriksaan kondisi dan mengembalikan nilai dalam DynamoDB.

- `retry`mutasi. Payload respons untuk opsi ini harus memiliki struktur ini:

```
{
    "action": "retry",
    "retryMapping": { ... }
}
```

Ini memberitahu AWS AppSync DynamoDB resolver untuk mencoba lagi mutasi dengan dokumen pemetaan permintaan baru. Struktur `retryMapping` bagian tergantung pada operasi DynamoDB, dan merupakan bagian dari dokumen pemetaan permintaan lengkap untuk operasi itu.

Untuk`PutItem`, `retryMapping` bagian tersebut memiliki struktur sebagai berikut. Untuk deskripsi `attributeValues` lapangan, lihat PutItem.

```
{
    "attributeValues": { ... },
    "condition": {
        "equalsIgnore" = [ ... ],
        "consistentRead" = true
    }
}
```

Untuk`UpdateItem`, `retryMapping` bagian tersebut memiliki struktur sebagai berikut. Untuk deskripsi `update` bagian ini, lihat UpdateItem.

```
{
    "update" : {
        "expression" : "someExpression"
        "expressionNames" : {
            "#foo" : "foo"
        },
        "expressionValues" : {
            ":bar" : ... typed value
        }
    },
```

```
    "condition": {
        "consistentRead" = true
    }
}
```

UntukDeleteItem, retryMapping bagian tersebut memiliki struktur sebagai berikut.

```
{
    "condition": {
        "consistentRead" = true
    }
}
```

Tidak ada cara untuk menentukan operasi atau kunci yang berbeda untuk dikerjakan. AWS AppSync DynamoDB resolver hanya memungkinkan percobaan ulang dari operasi yang sama pada objek yang sama. Juga, condition bagian ini tidak mengizinkan a conditionalCheckFailedHandler untuk ditentukan. Jika percobaan ulang gagal, penyelesai AWS AppSync DynamoDB mengikuti strategi. Reject

Berikut adalah contoh fungsi Lambda untuk menangani permintaan yang gagalPutItem. Logika bisnis melihat siapa yang membuat panggilan. Jika dibuat olehjeffTheAdmin, ia mencoba ulang permintaan, memperbarui version dan expectedVersion dari item yang saat ini ada di DynamoDB. Jika tidak, ia menolak mutasi.

```
exports.handler = (event, context, callback) => {
    console.log("Event: "+ JSON.stringify(event));

    // Business logic goes here.

    var response;
    if ( event.identity.user == "jeffTheAdmin" ) {
        response = {
            "action" : "retry",
            "retryMapping" : {
                "attributeValues" : event.requestMapping.attributeValues,
                "condition" : {
                    "expression" : event.requestMapping.condition.expression,
                    "expressionValues" :
 event.requestMapping.condition.expressionValues
                }
```

```
            }
        }
        response.retryMapping.attributeValues.version = { "N" :
 event.currentValue.version.N + 1 }
        response.retryMapping.condition.expressionValues[':expectedVersion'] =
 event.currentValue.version

    } else {
        response = { "action" : "reject" }
    }

    console.log("Response: "+ JSON.stringify(response))
    callback(null, response)
};
```

Ekspresi kondisi transaksi

Ekspresi kondisi transaksi tersedia dalam template pemetaan permintaan dari keempat jenis operasi di`TransactWriteItems`, yaitu,,`PutItem`, `DeleteItem``UpdateItem`, dan`ConditionCheck`.

Untuk`PutItem`,`DeleteItem`, dan`UpdateItem`, ekspresi kondisi transaksi adalah opsional. Untuk`ConditionCheck`, ekspresi kondisi transaksi diperlukan.

Contoh 1

Dokumen `DeleteItem` pemetaan transaksional berikut tidak memiliki ekspresi kondisi. Akibatnya, ia menghapus item di DynamoDB.

```
{
   "version": "2018-05-29",
   "operation": "TransactWriteItems",
   "transactItems": [
      {
         "table": "posts",
         "operation": "DeleteItem",
         "key": {
            "id": { "S" : "1" }
         }
      }
   ]
}
```

Contoh 2

Dokumen `DeleteItem` pemetaan transaksional berikut memang memiliki ekspresi kondisi transaksi yang memungkinkan operasi berhasil hanya jika penulis posting itu sama dengan nama tertentu.

```
{
   "version": "2018-05-29",
   "operation": "TransactWriteItems",
   "transactItems": [
      {
         "table": "posts",
         "operation": "DeleteItem",
         "key": {
            "id": { "S" : "1" }
         }
         "condition": {
            "expression": "author = :author",
            "expressionValues": {
               ":author": { "S" : "Chunyan" }
            }
         }
      }
   ]
}
```

Jika pemeriksaan kondisi gagal, itu akan menyebabkan `TransactionCanceledException` dan detail kesalahan akan dikembalikan`$ctx.result.cancellationReasons`. Perhatikan bahwa secara default, item lama di DynamoDB yang membuat pemeriksaan kondisi gagal akan dikembalikan. `$ctx.result.cancellationReasons`

Menentukan suatu kondisi

Dokumen pemetaan `PutItem``UpdateItem`,, dan `DeleteItem` permintaan semuanya memungkinkan `condition` bagian opsional untuk ditentukan. Jika dihilangkan, tidak ada pemeriksaan kondisi yang dilakukan. Jika ditentukan, kondisi harus benar agar operasi berhasil. `ConditionCheck`Harus memiliki `condition` bagian yang akan ditentukan. Syaratnya harus benar agar seluruh transaksi berhasil.

`condition`Bagian memiliki struktur sebagai berikut:

```
"condition": {
```

```
    "expression": "someExpression",
    "expressionNames": {
        "#foo": "foo"
    },
    "expressionValues": {
        ":bar": ... typed value
    },
    "returnValuesOnConditionCheckFailure": false
}
```

Bidang berikut menentukan kondisi:

`expression`

Ekspresi pembaruan itu sendiri. Untuk informasi selengkapnya tentang cara menulis ekspresi kondisi, lihat dokumentasi DynamoDB ConditionExpressions . Bidang ini harus ditentukan.

`expressionNames`

Substitusi untuk placeholder nama atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nama yang digunakan dalam ekspresi, dan nilainya harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalam ekspresi.

`expressionValues`

Substitusi untuk placeholder nilai atribut ekspresi, dalam bentuk pasangan kunci-nilai. Kunci sesuai dengan placeholder nilai yang digunakan dalam ekspresi, dan nilainya harus berupa nilai yang diketik. Untuk informasi selengkapnya tentang cara menentukan “nilai yang diketik”, lihat Jenis Sistem (pemetaan permintaan). Ini harus ditentukan. Bidang ini bersifat opsional, dan seharusnya hanya diisi dengan substitusi untuk placeholder nilai atribut ekspresi yang digunakan dalam ekspresi.

`returnValuesOnConditionCheckFailure`

Tentukan apakah akan mengambil item di DynamoDB kembali ketika pemeriksaan kondisi gagal. Item yang diambil akan masuk`$ctx.result.cancellationReasons[$index].item`, di `$index` mana indeks item permintaan yang gagal dalam pemeriksaan kondisi. Nilai ini default ke true.

Proyeksi

Saat membaca objek di DynamoDB menggunakan `GetItem`Scan,,,`Query`, `TransactGetItems` dan operasi`BatchGetItem`, Anda dapat secara opsional menentukan proyeksi yang mengidentifikasi atribut yang Anda inginkan. Proyeksi memiliki struktur berikut, yang mirip dengan filter:

```
"projection" : {
    "expression" : "projection expression"
    "expressionNames" : {
        "#name" : "name",
    }
}
```

Bidang didefinisikan sebagai berikut:

`expression`

Ekspresi proyeksi, yang merupakan string. Untuk mengambil atribut tunggal, tentukan namanya. Untuk beberapa atribut, nama harus berupa nilai yang dipisahkan koma. Untuk informasi selengkapnya tentang penulisan ekspresi proyeksi, lihat dokumentasi ekspresi proyeksi DynamoDB. Bidang ini wajib diisi.

`expressionNames`

Substitusi untuk placeholder nama atribut ekspresi dalam bentuk pasangan kunci-nilai. Kunci tersebut sesuai dengan placeholder nama yang digunakan dalam file. `expression` Nilai harus berupa string yang sesuai dengan nama atribut item di DynamoDB. Bidang ini opsional dan hanya boleh diisi dengan substitusi untuk placeholder nama atribut ekspresi yang digunakan dalam. `expression` Untuk informasi selengkapnya`expressionNames`, lihat dokumentasi DynamoDB.

Contoh 1

Contoh berikut adalah bagian proyeksi untuk template pemetaan VTL di mana hanya atribut `author` dan `id` dikembalikan dari DynamoDB:

```
"projection" : {
    "expression" : "#author, id",
    "expressionNames" : {
        "#author" : "author"
    }
```

```
}
```

> **Tip**
>
> Anda dapat mengakses set pemilihan permintaan GraphQL Anda menggunakan $context.info. selectionSetList. Bidang ini memungkinkan Anda untuk membingkai ekspresi proyeksi Anda secara dinamis sesuai dengan kebutuhan Anda.

> **Note**
>
> Saat menggunakan ekspresi proyeksi dengan `Scan` operasi `Query` dan, nilai untuk `select` harus`SPECIFIC_ATTRIBUTES`. Untuk informasi selengkapnya, lihat dokumentasi DynamoDB.

Referensi template pemetaan resolver untuk RDS

Template pemetaan resolver AWS AppSync RDS memungkinkan pengembang mengirim kueri SQL ke API Data untuk Amazon Aurora Tanpa Server dan mendapatkan kembali hasil kueri ini.

Meminta template pemetaan

Template pemetaan permintaan RDS cukup sederhana:

```
{
    "version": "2018-05-29",
    "statements": [],
    "variableMap": {},
    "variableTypeHintMap": {}
}
```

Berikut adalah representasi skema JSON dari template pemetaan permintaan RDS, setelah diselesaikan.

```
{
    "definitions": {},
    "$schema": "https://json-schema.org/draft-07/schema#",
    "$id": "https://example.com/root.json",
```

```
    "type": "object",
    "title": "The Root Schema",
    "required": [
        "version",
        "statements",
        "variableMap"
    ],
    "properties": {
        "version": {
            "$id": "#/properties/version",
            "type": "string",
            "title": "The Version Schema",
            "default": "",
            "examples": [
                "2018-05-29"
            ],
            "enum": [
                "2018-05-29"
            ],
            "pattern": "^(.*)$"
        },
        "statements": {
            "$id": "#/properties/statements",
            "type": "array",
            "title": "The Statements Schema",
            "items": {
                "$id": "#/properties/statements/items",
                "type": "string",
                "title": "The Items Schema",
                "default": "",
                "examples": [
                    "SELECT * from BOOKS"
                ],
                "pattern": "^(.*)$"
            }
        },
        "variableMap": {
            "$id": "#/properties/variableMap",
            "type": "object",
            "title": "The Variablemap Schema"
        },
        "variableTypeHintMap": {
            "$id": "#/properties/variableTypeHintMap",
            "type": "object",
```

```
            "title": "The variableTypeHintMap Schema"
        }
    }
}
```

Berikut ini adalah contoh template pemetaan permintaan dengan query statis:

```
{
    "version": "2018-05-29",
    "statements": [
        "select title, isbn13 from BOOKS where author = 'Mark Twain'"
    ]
}
```

Versi

Umum untuk semua template pemetaan permintaan, bidang versi mendefinisikan versi yang digunakan template. Bidang versi diperlukan. Nilai “2018-05-29" adalah satu-satunya versi yang didukung untuk template pemetaan Amazon RDS.

```
"version": "2018-05-29"
```

Pernyataan dan VariableMap

Array pernyataan adalah placeholder untuk kueri yang disediakan pengembang. Saat ini, hingga dua kueri per permintaan template pemetaan didukung. `variableMap`Ini adalah bidang opsional yang berisi alias yang dapat digunakan untuk membuat pernyataan SQL lebih pendek dan lebih mudah dibaca. Misalnya, hal berikut ini dimungkinkan:

```
{
"version": "2018-05-29",
    "statements": [
        "insert into BOOKS VALUES (:AUTHOR, :TITLE, :ISBN13)",
        "select * from BOOKS WHERE isbn13 = :ISBN13"
    ],
    "variableMap": {
        ":AUTHOR": $util.toJson($ctx.args.newBook.author),
        ":TITLE": $util.toJson($ctx.args.newBook.title),
        ":ISBN13": $util.toJson($ctx.args.newBook.isbn13)
    }
```

```
}
```

AWS AppSyncakan menggunakan nilai peta variabel untuk membuat SqlParameterizedkueri yang akan dikirim ke API Data Tanpa Server Amazon Aurora. Pernyataan SQL dijalankan dengan parameter yang disediakan dalam peta variabel, yang menghilangkan risiko injeksi SQL.

VariableTypeHintMap

`variableTypeHintMap`Ini adalah bidang opsional yang berisi tipe alias yang dapat digunakan untuk mengirim petunjuk tipe parameter SQL. Petunjuk jenis ini menghindari casting eksplisit dalam pernyataan SQL, membuatnya lebih pendek. Misalnya, hal berikut ini dimungkinkan:

```
{
    "version": "2018-05-29",
    "statements": [
        "insert into LOGINDATA VALUES (:ID, :TIME)",
        "select * from LOGINDATA WHERE id = :ID"
     ],
     "variableMap": {
        ":ID": $util.toJson($ctx.args.id),
        ":TIME": $util.toJson($ctx.args.time)
     },
     "variableTypeHintMap": {
        ":id": "UUID",
        ":time": "TIME"
     }
}
```

AWS AppSyncakan menggunakan nilai peta variabel untuk membuat kueri yang dikirim ke API Data Tanpa Server Amazon Aurora. Ini juga menggunakan `variableTypeHintMap` data dan mengirimkan informasi tipe ke RDS. Didukung RDS `typeHints` dapat ditemukan di sini.

Referensi Template Pemetaan Resolver untuk OpenSearch

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

AWS AppSync Resolver untuk Amazon OpenSearch Service memungkinkan Anda menggunakan GraphQL untuk menyimpan dan mengambil data di domain Layanan yang ada di akun Anda. OpenSearch Penyelesai ini berfungsi dengan memungkinkan Anda memetakan permintaan GraphQL yang masuk ke dalam permintaan Layanan, lalu memetakan OpenSearch respons Layanan kembali ke GraphQL. OpenSearch Bagian ini menjelaskan template pemetaan untuk operasi OpenSearch Layanan yang didukung.

Templat Pemetaan Permintaan

Sebagian besar templat pemetaan permintaan OpenSearch Layanan memiliki struktur umum di mana hanya beberapa bagian yang berubah. Contoh berikut menjalankan pencarian terhadap domain OpenSearch Layanan, di mana dokumen diatur di bawah indeks yang disebut`post`. Parameter pencarian didefinisikan di `body` bagian, dengan banyak klausa kueri umum yang didefinisikan di `query` bidang. Contoh ini akan mencari dokumen yang berisi `"Nadia""Bailey"`, atau, atau keduanya, di `author` bidang dokumen:

```
{
    "version":"2017-02-28",
    "operation":"GET",
    "path":"/post/_search",
    "params":{
        "headers":{},
        "queryString":{},
        "body":{
            "from":0,
            "size":50,
            "query" : {
                "bool" : {
                    "should" : [
                        {"match" : { "author" : "Nadia" }},
                        {"match" : { "author" : "Bailey" }}
                    ]
                }
            }
        }
    }
}
```

Templat Pemetaan Respon

Seperti sumber data lainnya, OpenSearch Layanan mengirimkan respons AWS AppSync yang perlu dikonversi ke GraphQL.

Sebagian besar kueri GraphQL mencari bidang `_source` dari respons Layanan. OpenSearch Karena Anda dapat melakukan pencarian untuk mengembalikan dokumen individual atau daftar dokumen, ada dua templat pemetaan respons umum yang digunakan dalam OpenSearch Layanan:

Daftar Hasil

```
[
    #foreach($entry in $context.result.hits.hits)
      #if( $velocityCount > 1 ) , #end
        $utils.toJson($entry.get("_source"))
    #end
]
```

Barang Individu

```
$utils.toJson($context.result.get("_source"))
```

`operation`lapangan

(REQUEST Mapping Template saja)

Metode HTTP atau kata kerja (GET, POST, PUT, HEAD atau DELETE) yang AWS AppSync mengirim ke domain OpenSearch Layanan. Baik kunci dan nilainya harus berupa string.

```
"operation" : "PUT"
```

`path`lapangan

(REQUEST Mapping Template saja)

Jalur pencarian untuk permintaan OpenSearch Layanan dari AWS AppSync. Ini membentuk URL untuk kata kerja HTTP operasi. Baik kunci dan nilainya harus berupa string.

```
"path" : "/<indexname>/_doc/<_id>"
"path" : "/<indexname>/_doc"
```

```
"path" : "/<indexname>/_search"
"path" : "/<indexname>/_update/<_id>
```

Ketika template pemetaan dievaluasi, jalur ini dikirim sebagai bagian dari permintaan HTTP, termasuk domain OpenSearch Layanan. Misalnya, contoh sebelumnya mungkin diterjemahkan ke:

```
GET https://opensearch-domain-name.REGION.es.amazonaws.com/indexname/type/_search
```

paramslapangan

(REQUEST Mapping Template saja)

Digunakan untuk menentukan tindakan apa yang dilakukan penelusuran Anda, paling umum dengan menetapkan nilai kueri di dalam badan. Namun, ada beberapa kemampuan lain yang dapat dikonfigurasi, seperti pemformatan respons.

- header

  Informasi header, sebagai pasangan kunci-nilai. Baik kunci dan nilainya harus berupa string. Sebagai contoh:

  ```
  "headers" : {
      "Content-Type" : "application/json"
  }
  ```

  > **Note**
  >
  > AWS AppSync saat ini hanya mendukung JSON sebagai file. `Content-Type`

- QueryString

  Pasangan nilai kunci yang menentukan opsi umum, seperti pemformatan kode untuk respons JSON. Baik kunci dan nilainya harus berupa string. Misalnya, jika Anda ingin mendapatkan JSON yang diformat dengan cantik, Anda akan menggunakan:

  ```
  "queryString" : {
      "pretty" : "true"
  }
  ```

- tubuh

Ini adalah bagian utama dari permintaan Anda, memungkinkan AWS AppSync untuk membuat permintaan pencarian yang terbentuk dengan baik ke domain OpenSearch Layanan Anda. Kuncinya harus berupa string yang terdiri dari sebuah objek. Beberapa demonstrasi ditunjukkan di bawah ini.

Contoh 1

Kembalikan semua dokumen dengan kota yang cocok dengan “seattle”:

```
"body":{
    "from":0,
    "size":50,
    "query" : {
        "match" : {
            "city" : "seattle"
        }
    }
}
```

Contoh 2

Kembalikan semua dokumen yang cocok dengan “washington” sebagai kota atau negara bagian:

```
"body":{
    "from":0,
    "size":50,
    "query" : {
        "multi_match" : {
            "query" : "washington",
            "fields" : ["city", "state"]
        }
    }
}
```

Melewati Variabel

(REQUEST Mapping Template saja)

Anda juga dapat meneruskan variabel sebagai bagian dari evaluasi dalam pernyataan VTL. Misalnya, Anda memiliki kueri GraphQL seperti berikut ini:

```
query {
    searchForState(state: "washington"){
        ...
    }
}
```

Template pemetaan dapat mengambil status sebagai argumen:

```
"body":{
    "from":0,
    "size":50,
    "query" : {
        "multi_match" : {
            "query" : "$context.arguments.state",
            "fields" : ["city", "state"]
        }
    }
}
```

Untuk daftar utilitas yang dapat Anda sertakan dalam VTL, lihat Header Permintaan Akses.

Referensi template pemetaan resolver untuk Lambda

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Anda dapat menggunakan templat pemetaan AWS AppSync resolver untuk membentuk permintaan dari AWS Lambda ke fungsi AWS AppSync Lambda yang terletak di akun Anda, dan respons dari fungsi Lambda Anda kembali ke. AWS AppSync Anda juga dapat menggunakan template pemetaan untuk memberikan petunjuk AWS AppSync tentang sifat operasi yang akan dipanggil. Bagian ini menjelaskan template pemetaan yang berbeda untuk operasi Lambda yang didukung.

Meminta template pemetaan

Template pemetaan permintaan Lambda cukup sederhana dan memungkinkan informasi konteks sebanyak mungkin untuk diteruskan ke fungsi Lambda Anda.

```
{
   "version": string,
   "operation": Invoke|BatchInvoke,
   "payload": any type
}
```

Berikut adalah representasi skema JSON dari template pemetaan permintaan Lambda, ketika diselesaikan.

```
{
    "definitions": {},
    "$schema": "https://json-schema.org/draft-06/schema#",
    "$id": "https://aws.amazon.com/appsync/request-mapping-template.json",
    "type": "object",
    "properties": {
        "version": {
            "$id": "/properties/version",
            "type": "string",
            "enum": [
                "2018-05-29"
            ],
            "title": "The Mapping template version.",
            "default": "2018-05-29"
        },
        "operation": {
            "$id": "/properties/operation",
            "type": "string",
            "enum": [
                "Invoke",
                "BatchInvoke"
            ],
            "title": "The Mapping template operation.",
            "description": "What operation to execute.",
            "default": "Invoke"
        },
        "payload": {}
    },
    "required": [
        "version",
        "operation"
    ],
    "additionalProperties": false
```

```
}
```

Berikut adalah contoh di mana kita meneruskan `field` nilai dan argumen bidang GraphQL dari konteksnya.

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments": $util.toJson($context.arguments)
    }
}
```

Seluruh dokumen pemetaan diteruskan sebagai masukan ke fungsi Lambda Anda, sehingga contoh sebelumnya sekarang akan terlihat seperti berikut:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments": {
            "id": "postId1"
        }
    }
}
```

Versi

Umum untuk semua template pemetaan permintaan, `version` mendefinisikan versi yang digunakan template. `version`ondiperlukan.

```
"version": "2018-05-29"
```

Operasi

Sumber data Lambda memungkinkan Anda menentukan dua operasi: `Invoke` dan. `BatchInvoke` `Invoke`Operasi ini memungkinkan AWS AppSync untuk memanggil fungsi Lambda Anda untuk setiap penyelesai bidang GraphQL. `BatchInvoke`menginstruksikan permintaan batch AWS AppSync untuk bidang GraphQL saat ini.

`operation`diperlukan.

Untuk`Invoke`, template pemetaan permintaan yang diselesaikan sama persis dengan payload input dari fungsi Lambda. Jadi contoh template berikut:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "arguments": $util.toJson($context.arguments)
    }
}
```

diselesaikan dan diteruskan ke fungsi Lambda, sebagai berikut:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "arguments": {
            "id": "postId1"
        }
    }
}
```

Untuk`BatchInvoke`, template pemetaan diterapkan untuk setiap penyelesai bidang dalam batch. Untuk keringkasan, AWS AppSync gabungkan semua `payload` nilai template pemetaan yang diselesaikan ke dalam daftar di bawah satu objek yang cocok dengan templat pemetaan.

Contoh template berikut menunjukkan penggabungan:

```
{
    "version": "2018-05-29",
    "operation": "BatchInvoke",
    "payload": $util.toJson($context)
}
```

Template ini diselesaikan ke dalam dokumen pemetaan berikut:

```
{
    "version": "2018-05-29",
    "operation": "BatchInvoke",
```

```
    "payload": [
        {...}, // context for batch item 1
        {...}, // context for batch item 2
        {...}  // context for batch item 3
    ]
}
```

di mana setiap elemen `payload` daftar sesuai dengan satu item batch. Fungsi Lambda juga diharapkan mengembalikan respons berbentuk daftar, cocok dengan urutan item yang dikirim dalam permintaan, sebagai berikut:

```
[
    { "data": {...}, "errorMessage": null, "errorType": null }, // result for batch
 item 1
    { "data": {...}, "errorMessage": null, "errorType": null }, // result for batch
 item 2
    { "data": {...}, "errorMessage": null, "errorType": null }  // result for batch
 item 3
]
```

`operation`diperlukan.

Muatan

`payload`Bidang adalah wadah yang dapat Anda gunakan untuk meneruskan JSON yang terbentuk dengan baik ke fungsi Lambda.

Jika `operation` bidang diatur ke`BatchInvoke`, AWS AppSync membungkus `payload` nilai yang ada ke dalam daftar.

`payload`adalah opsional.

Templat pemetaan respons

Seperti sumber data lainnya, fungsi Lambda Anda mengirimkan respons AWS AppSync yang harus dikonversi ke tipe GraphQL.

Hasil dari fungsi Lambda diatur pada `context` objek yang tersedia melalui properti Velocity Template Language (VTL). `$context.result`

Jika bentuk respons fungsi Lambda Anda sama persis dengan bentuk tipe GraphQL, Anda dapat meneruskan respons menggunakan templat pemetaan respons berikut:

```
$util.toJson($context.result)
```

Tidak ada bidang wajib atau batasan bentuk yang berlaku untuk template pemetaan respons. Namun, karena GraphQL diketik dengan kuat, template pemetaan yang diselesaikan harus sesuai dengan jenis GraphQL yang diharapkan.

Respons batch fungsi Lambda

Jika `operation` bidang diatur ke`BatchInvoke`, AWS AppSync mengharapkan daftar item kembali dari fungsi Lambda. AWS AppSync Untuk memetakan setiap hasil kembali ke item permintaan asli, daftar respons harus sesuai dengan ukuran dan urutan. Tidak apa-apa untuk memiliki `null` item dalam daftar respons; `$ctx.result` diatur ke null sesuai.

Resolver Lambda Langsung

Jika Anda ingin menghindari penggunaan template pemetaan sepenuhnya, AWS AppSync dapat memberikan payload default ke fungsi Lambda Anda dan default dari respons fungsi Lambda ke tipe GraphQL. Anda dapat memilih untuk menyediakan template permintaan, template respons, atau tidak, dan AWS AppSync menanganinya sesuai dengan itu.

Templat pemetaan permintaan Lambda langsung

Ketika template pemetaan permintaan tidak disediakan, AWS AppSync akan mengirim `Context` objek langsung ke fungsi Lambda Anda sebagai `Invoke` operasi. Untuk informasi lebih lanjut tentang struktur objek `Context`, lihat Referensi konteks template pemetaan penyelesai.

Templat pemetaan respons Lambda langsung

Ketika template pemetaan respons tidak disediakan, AWS AppSync lakukan salah satu dari dua hal setelah menerima respons fungsi Lambda Anda. Jika Anda tidak menyediakan templat pemetaan permintaan, atau jika Anda menyediakan templat pemetaan permintaan dengan versi “2018-05-29", maka logika respons berfungsi setara dengan templat pemetaan respons berikut:

```
#if($ctx.error)
     $util.error($ctx.error.message, $ctx.error.type, $ctx.result)
 #end
 $util.toJson($ctx.result)
```

Jika Anda menyediakan template dengan versi “2017-02-28", logika respons berfungsi setara dengan templat pemetaan respons berikut:

```
$util.toJson($ctx.result)
```

Secara dangkal, bypass template pemetaan beroperasi mirip dengan menggunakan templat pemetaan tertentu, seperti yang ditunjukkan pada contoh sebelumnya. Namun, di balik layar, evaluasi template pemetaan dielakkan sepenuhnya. Karena langkah evaluasi template dilewati, dalam beberapa skenario aplikasi mungkin mengalami lebih sedikit overhead dan latensi selama respons bila dibandingkan dengan fungsi Lambda dengan template pemetaan respons yang perlu dievaluasi.

Penanganan kesalahan khusus dalam respons Resolver Lambda Langsung

Anda dapat menyesuaikan respons kesalahan dari fungsi Lambda yang dipanggil Direct Lambda Resolvers dengan memunculkan pengecualian khusus. Contoh berikut menunjukkan cara membuat pengecualian kustom menggunakan JavaScript:

```
class CustomException extends Error {
    constructor(message) {
        super(message);
        this.name = "CustomException";
    }
}

throw new CustomException("Custom message");
```

Ketika pengecualian dinaikkan, `errorType` dan `errorMessage` adalah `name` dan`message`, masing-masing, dari kesalahan khusus yang dilemparkan.

Jika `errorType` ada`UnauthorizedException`, AWS AppSync mengembalikan pesan default (`"You are not authorized to make this call."`) bukan pesan kustom.

Berikut ini adalah contoh respons GraphQL yang menunjukkan kustom. `errorType`

```
{
  "data": {
    "query": null
  },
  "errors": [
    {
      "path": [
        "query"
      ],
      "data": null,
```

```
      "errorType": "CustomException",
      "errorInfo": null,
      "locations": [
        {
          "line": 5,
          "column": 10,
          "sourceName": null
        }
      ],
      "message": "Custom Message"
    }
  ]
}
```

Resolver Lambda Langsung: Batching diaktifkan

Anda dapat mengaktifkan batching untuk Resolver Lambda Langsung Anda dengan mengonfigurasi `maxBatchSize` pada resolver Anda. Bila `maxBatchSize` disetel ke nilai yang lebih besar dari 0 untuk penyelesai Lambda Langsung, AWS AppSync kirimkan permintaan dalam batch ke fungsi Lambda Anda dalam ukuran hingga. `maxBatchSize`

Pengaturan `maxBatchSize` ke 0 pada penyelesai Lambda Langsung mematikan batching.

Untuk informasi lebih lanjut tentang cara kerja batching dengan resolver Lambda, lihat. Kasus penggunaan lanjutan: Batching

Meminta template pemetaan

Ketika batching diaktifkan dan template pemetaan permintaan tidak disediakan, AWS AppSync mengirimkan daftar `Context` objek sebagai `BatchInvoke` operasi langsung ke fungsi Lambda Anda.

Templat pemetaan respons

Saat batching diaktifkan dan templat pemetaan respons tidak disediakan, logika respons setara dengan templat pemetaan respons berikut:

```
#if( $context.result && $context.result.errorMessage )
      $utils.error($context.result.errorMessage, $context.result.errorType,
      $context.result.data)
#else
      $utils.toJson($context.result.data)
```

```
#end
```

Fungsi Lambda harus mengembalikan daftar hasil dalam urutan yang sama dengan daftar `Context` objek yang dikirim. Anda dapat mengembalikan kesalahan individu dengan memberikan `errorMessage` dan `errorType` untuk hasil tertentu. Setiap hasil dalam daftar memiliki format berikut:

```
{
   "data" : { ... }, // your data
   "errorMessage" : { ... }, // optional, if included an error entry is added to the
 "errors" object in the AppSync response
   "errorType" : { ... } // optional, the error type
}
```

> **Note**
>
> Bidang lain dalam objek hasil saat ini diabaikan.

Menangani kesalahan dari Lambda

Anda dapat mengembalikan kesalahan untuk semua hasil dengan melempar pengecualian atau kesalahan dalam fungsi Lambda Anda. Jika permintaan payload atau ukuran respons untuk permintaan batch Anda terlalu besar, Lambda mengembalikan kesalahan. Dalam hal ini, Anda harus mempertimbangkan untuk mengurangi `maxBatchSize` atau mengurangi ukuran muatan respons.

Untuk informasi tentang penanganan kesalahan individu, lihatMengembalikan kesalahan individu.

Contoh fungsi Lambda

Dengan menggunakan skema di bawah ini, Anda dapat membuat Resolver Lambda Langsung untuk penyelesai bidang dan mengaktifkan pengelompokan `Post.relatedPosts` dengan menyetel ke lebih besar dari 0: `maxBatchSize`

```
schema {
    query: Query
    mutation: Mutation
}

type Query {
    getPost(id:ID!): Post
```

```
    allPosts: [Post]
}

type Mutation {
    addPost(id: ID!, author: String!, title: String, content: String, url: String):
 Post!
}

type Post {
    id: ID!
    author: String!
    title: String
    content: String
    url: String
    ups: Int
    downs: Int
    relatedPosts: [Post]
}
```

Dalam kueri berikut, fungsi Lambda akan dipanggil dengan kumpulan permintaan untuk diselesaikan: `relatedPosts`

```
query getAllPosts {
  allPosts {
    id
    relatedPosts {
      id
    }
  }
}
```

Implementasi sederhana dari fungsi Lambda disediakan di bawah ini:

```
const posts = {
  1: {
    id: '1',
    title: 'First book',
    author: 'Author1',
    url: 'https://amazon.com/',
    content:
      'SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT
 AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1',
    ups: '100',
```

```
    downs: '10',
  },
  2: {
    id: '2',
    title: 'Second book',
    author: 'Author2',
    url: 'https://amazon.com',
    content: 'SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT',
    ups: '100',
    downs: '10',
  },
  3: { id: '3', title: 'Third book', author: 'Author3', url: null, content: null, ups:
 null, downs: null },
  4: {
    id: '4',
    title: 'Fourth book',
    author: 'Author4',
    url: 'https://www.amazon.com/',
    content:
      'SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
 AUTHOR 4',
    ups: '1000',
    downs: '0',
  },
  5: {
    id: '5',
    title: 'Fifth book',
    author: 'Author5',
    url: 'https://www.amazon.com/',
    content: 'SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE
 TEXT AUTHOR 5 SAMPLE TEXT',
    ups: '50',
    downs: '0',
  },
}

const relatedPosts = {
  1: [posts['4']],
  2: [posts['3'], posts['5']],
  3: [posts['2'], posts['1']],
  4: [posts['2'], posts['1']],
  5: [],
}
```

```
exports.handler = async (event) => {
  console.log('event ->', event)
  // retrieve the ID of each post
  const ids = event.map((context) => context.source.id)
  // fetch the related posts for each post id
  const related = ids.map((id) => relatedPosts[id])

  // return the related posts; or an error if none were found
  return related.map((r) => {
    if (r.length > 0) {
      return { data: r }
    } else {
      return { data: null, errorMessage: 'Not found', errorType: 'ERROR' }
    }
  })
}
```

Referensi template pemetaan resolver untuk EventBridge

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Template pemetaan AWS AppSync resolver yang digunakan dengan sumber EventBridge data memungkinkan Anda mengirim acara khusus ke bus Amazon. EventBridge

Meminta template pemetaan

Templat pemetaan `PutEvents` permintaan memungkinkan Anda mengirim beberapa acara khusus ke bus EventBridge acara. Dokumen pemetaan memiliki struktur sebagai berikut:

```
{
    "version" : "2018-05-29",
    "operation" : "PutEvents",
    "events" : [{}]
}
```

Berikut ini adalah contoh template pemetaan permintaan untuk EventBridge:

```
{
    "version": "2018-05-29",
    "operation": "PutEvents",
    "events": [{
        "source": "com.mycompany.myapp",
        "detail": {
            "key1" : "value1",
            "key2" : "value2"
        },
        "detailType": "myDetailType1"
    },
    {
        "source": "com.mycompany.myapp",
        "detail": {
            "key3" : "value3",
            "key4" : "value4"
        },
        "detailType": "myDetailType2",
        "resources" : ["Resource1", "Resource2"],
        "time" : "2023-01-01T00:30:00.000Z"
    }

    ]
}
```

Templat pemetaan respons

Jika `PutEvents` operasi berhasil, respons dari EventBridge termasuk dalam`$ctx.result`:

```
#if($ctx.error)
  $util.error($ctx.error.message, $ctx.error.type, $ctx.result)
#end
  $util.toJson($ctx.result)
```

Kesalahan yang terjadi saat melakukan `PutEvents` operasi seperti `InternalExceptions` atau `Timeouts` akan muncul di`$ctx.error`. Untuk daftar EventBridge kesalahan umum, lihat referensi kesalahan EventBridge umum.

`result`Akan dalam format berikut:

```
{
    "Entries" [
```

```
        {
            "ErrorCode" : String,
            "ErrorMessage" : String,
            "EventId" : String
        }
    ],
    "FailedEntry" : number
}
```

- Entri

  Hasil acara yang dicerna, baik yang berhasil maupun yang tidak berhasil. Jika konsumsi berhasil, entri memiliki `EventID` di dalamnya. Jika tidak, Anda dapat menggunakan `ErrorCode` dan `ErrorMessage` untuk mengidentifikasi masalah dengan entri.

  Untuk setiap record, indeks elemen respon sama dengan indeks dalam array permintaan.

- FailedEntryCount

  Jumlah entri yang gagal. Nilai ini direpresentasikan sebagai bilangan bulat.

Untuk informasi lebih lanjut tentang tanggapan`PutEvents`, lihat PutEvents.

Contoh respon sampel 1

Contoh berikut adalah `PutEvents` operasi dengan dua peristiwa sukses:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        }
    ],
    "FailedEntryCount" : 0
}
```

Contoh respon sampel 2

Contoh berikut adalah `PutEvents` operasi dengan tiga peristiwa, dua keberhasilan dan satu gagal:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        },
        {
            "ErrorCode" : "SampleErrorCode",
            "ErrorMessage" : "Sample Error Message"
        }
    ],
    "FailedEntryCount" : 1
}
```

PutEventslapangan

- Versi

  Umum untuk semua template pemetaan permintaan, `version` bidang mendefinisikan versi yang digunakan template. Bidang ini wajib diisi. Nilai `2018-05-29` adalah satu-satunya versi yang didukung untuk template EventBridge pemetaan.

- Operasi

  Satu-satunya operasi yang didukung adalah`PutEvents`. Operasi ini memungkinkan Anda untuk menambahkan acara khusus ke bus acara Anda.

- Event

  Serangkaian acara yang akan ditambahkan ke bus acara. Array ini harus memiliki alokasi 1 - 10 item.

  `Event`Objek adalah objek JSON valid yang memiliki bidang berikut:

  - `"source"`: String yang mendefinisikan sumber acara.
  - `"detail"`: Objek JSON yang dapat Anda gunakan untuk melampirkan informasi tentang acara tersebut. Bidang ini bisa berupa peta kosong (`{ }`).
  - `"detailType`: Sebuah string yang mengidentifikasi jenis acara.
  - `"resources"`: Sebuah array JSON string yang mengidentifikasi sumber daya yang terlibat dalam acara tersebut. Bidang ini bisa berupa array kosong.

- `"time"`: Stempel waktu acara disediakan sebagai string. Ini harus mengikuti format stempel waktu RFC3339.

Cuplikan di bawah ini adalah beberapa contoh objek yang valid`Event`:

Contoh 1

```
{
    "source" : "source1",
    "detail" : {
        "key1" : [1,2,3,4],
        "key2" : "strval"
    },
    "detailType" : "sampleDetailType",
    "resources" : ["Resouce1", "Resource2"],
    "time" : "2022-01-10T05:00:10Z"
}
```

Contoh 2

```
{
    "source" : "source1",
    "detail" : {},
    "detailType" : "sampleDetailType"
}
```

Contoh 3

```
{
    "source" : "source1",
    "detail" : {
        "key1" : 1200
    },
    "detailType" : "sampleDetailType",
    "resources" : []
}
```

Referensi template pemetaan Resolver untuk sumber data None

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Template pemetaan AWS AppSync resolver yang digunakan dengan sumber data tipe None, memungkinkan Anda untuk membentuk permintaan untuk AWS AppSync operasi lokal.

Meminta template pemetaan

Template pemetaan sederhana dan memungkinkan Anda untuk meneruskan informasi konteks sebanyak mungkin melalui `payload` lapangan.

```
{
   "version": string,
   "payload": any type
}
```

Berikut adalah representasi skema JSON dari template pemetaan permintaan, setelah diselesaikan:

```
{
    "definitions": {},
    "$schema": "https://json-schema.org/draft-06/schema#",
    "$id": "https://aws.amazon.com/appsync/request-mapping-template.json",
    "type": "object",
    "properties": {
        "version": {
            "$id": "/properties/version",
            "type": "string",
            "enum": [
                "2018-05-29"
            ],
            "title": "The Mapping template version.",
            "default": "2018-05-29"
        },
        "payload": {}
    },
    "required": [
```

```
        "version"
    ],
    "additionalProperties": false
}
```

Berikut adalah contoh di mana argumen bidang diteruskan melalui properti konteks VTL: `$context.arguments`

```
{
    "version": "2018-05-29",
    "payload": $util.toJson($context.arguments)
}
```

Nilai `payload` bidang akan diteruskan ke template pemetaan respons dan tersedia di properti konteks VTL (). `$context.result`

Ini adalah contoh yang mewakili nilai interpolasi bidang: `payload`

```
{
    "id": "postId1"
}
```

Versi

Umum untuk semua template pemetaan permintaan, `version` bidang mendefinisikan versi yang digunakan oleh template.

Bidang `version` wajib diisi.

Contoh:

```
"version": "2018-05-29"
```

Muatan

`payload`Bidang adalah wadah yang dapat digunakan untuk meneruskan JSON yang terbentuk dengan baik ke template pemetaan respons.

`payload`Bidang ini opsional.

Templat pemetaan respons

Karena tidak ada sumber data, nilai `payload` bidang akan diteruskan ke template pemetaan respons dan disetel pada `context` objek yang tersedia melalui properti VTL. `$context.result`

Jika bentuk nilai `payload` bidang sama persis dengan bentuk tipe GraphQL, Anda dapat meneruskan respons menggunakan templat pemetaan respons berikut:

```
$util.toJson($context.result)
```

Tidak ada bidang wajib atau batasan bentuk yang berlaku untuk template pemetaan respons. Namun, karena GraphQL diketik dengan kuat, template pemetaan yang diselesaikan harus sesuai dengan jenis GraphQL yang diharapkan.

Referensi Template Pemetaan Resolver untuk HTTP

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Templat pemetaan AWS AppSync HTTP resolver memungkinkan Anda mengirim permintaan dari titik akhir HTTP apa pun, dan tanggapan dari titik akhir HTTP Anda kembali AWS AppSync ke. AWS AppSync Dengan menggunakan template pemetaan, Anda dapat memberikan petunjuk AWS AppSync tentang sifat operasi yang akan dipanggil. Bagian ini menjelaskan template pemetaan yang berbeda untuk resolver HTTP yang didukung.

Templat Pemetaan Permintaan

```
{
    "version": "2018-05-29",
    "method": "PUT|POST|GET|DELETE|PATCH",
    "params": {
        "query": Map,
        "headers": Map,
        "body": any
    },
    "resourcePath": string
```

```
}
```

Setelah template pemetaan permintaan HTTP diselesaikan, representasi skema JSON dari template pemetaan permintaan terlihat seperti berikut:

```
{
    "$id": "https://aws.amazon.com/appsync/request-mapping-template.json",
    "type": "object",
    "properties": {
        "version": {
        "$id": "/properties/version",
        "type": "string",
        "title": "The Version Schema ",
        "default": "",
        "examples": [
            "2018-05-29"
        ],
        "enum": [
            "2018-05-29"
        ]
        },
        "method": {
        "$id": "/properties/method",
        "type": "string",
        "title": "The Method Schema ",
        "default": "",
        "examples": [
            "PUT|POST|GET|DELETE|PATCH"
        ],
        "enum": [
            "PUT",
            "PATCH",
            "POST",
            "DELETE",
            "GET"
        ]
        },
        "params": {
        "$id": "/properties/params",
        "type": "object",
        "properties": {
            "query": {
            "$id": "/properties/params/properties/query",
```

```
            "type": "object"
            },
            "headers": {
            "$id": "/properties/params/properties/headers",
            "type": "object"
            },
            "body": {
            "$id": "/properties/params/properties/body",
            "type": "string",
            "title": "The Body Schema ",
            "default": "",
            "examples": [
                ""
            ]
            }
        }
        },
        "resourcePath": {
        "$id": "/properties/resourcePath",
        "type": "string",
        "title": "The Resourcepath Schema ",
        "default": "",
        "examples": [
            ""
        ]
        }
    },
    "required": [
        "version",
        "method",
        "resourcePath"
    ]
}
```

Berikut ini adalah contoh permintaan HTTP POST, dengan `text/plain` badan:

```
{
    "version": "2018-05-29",
    "method": "POST",
    "params": {
        "headers":{
        "Content-Type":"text/plain"
        },
```

```
        "body":"this is an example of text body"
    },
    "resourcePath": "/"
}
```

Versi

Minta template pemetaan saja

Mendefinisikan versi yang digunakan template. `version`onumum untuk semua templat pemetaan permintaan dan diperlukan.

```
"version": "2018-05-29"
```

Metode

Minta template pemetaan saja

Metode HTTP atau kata kerja (GET, POST, PUT, PATCH, atau DELETE) yang AWS AppSync mengirim ke titik akhir HTTP.

```
"method": "PUT"
```

ResourcePath

Minta template pemetaan saja

Jalur sumber daya yang ingin Anda akses. Seiring dengan titik akhir di sumber data HTTP, jalur sumber daya membentuk URL tempat AWS AppSync layanan membuat permintaan.

```
"resourcePath": "/v1/users"
```

Ketika template pemetaan dievaluasi, jalur ini dikirim sebagai bagian dari permintaan HTTP, termasuk titik akhir HTTP. Misalnya, contoh sebelumnya mungkin menerjemahkan ke yang berikut:

```
PUT <endpoint>/v1/users
```

Bidang Params

Minta template pemetaan saja

Digunakan untuk menentukan tindakan apa yang dilakukan penelusuran Anda, paling sering dengan menetapkan nilai kueri di dalam badan. Namun, ada beberapa kemampuan lain yang dapat dikonfigurasi, seperti pemformatan respons.

header

Informasi header, sebagai pasangan kunci-nilai. Baik kunci dan nilainya harus berupa string.

Sebagai contoh:

```
"headers" : {
    "Content-Type" : "application/json"
}
```

`Content-Type`Header yang didukung saat ini adalah:

```
text/*
application/xml
application/json
application/soap+xml
application/x-amz-json-1.0
application/x-amz-json-1.1
application/vnd.api+json
application/x-ndjson
```

Catatan: Anda tidak dapat mengatur header HTTP berikut:

```
HOST
CONNECTION
USER-AGENT
EXPECTATION
TRANSFER_ENCODING
CONTENT_LENGTH
```

kueri

Pasangan nilai kunci yang menentukan opsi umum, seperti pemformatan kode untuk respons JSON. Baik kunci dan nilainya harus berupa string. Contoh berikut menunjukkan bagaimana Anda dapat mengirim string query sebagai`?type=json`:

```
"query" : {
```

```
    "type" : "json"
}
```

tubuh

Tubuh berisi badan permintaan HTTP yang Anda pilih untuk disetel. Badan permintaan selalu berupa string yang dikodekan UTF-8 kecuali jenis konten menentukan charset.

```
"body":"body string"
```

Otoritas Sertifikat (CA) Diakui olehAWS AppSyncuntuk Titik Akhir HTTPS

> **Note**
>
> Mari Encrypt diterima melaluiidentrustdanisrgrootx1sertifikat. Tidak ada tindakan pada bagian Anda diperlukan jika Anda menggunakan Let's Encrypt.

Pada saat ini, sertifikat yang ditandatangani sendiri tidak didukung oleh resolver HTTP saat menggunakan HTTPS.AWS AppSync mengakui Otoritas Sertifikat berikut saat menyelesaikan sertifikat SSL/TLS untuk HTTPS:

Sertifikat akar yang dikenal diAWS AppSync

Nama	Tanggal	Sidik Jari SHA1
`digicertassuredidrootca`	21 Apr 2018	`05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43`
`trustcenterclass2caii`	21 Apr 2018	`AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E`
`thawtepremiumserverca`	21 Apr 2018	`E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66`
`cia-crt-g3-02-ca`	23 November 2016	`96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09`

Nama	Tanggal	Sidik Jari SHA1
swisssignplatinumg2ca	21 Apr 2018	56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66
swisssignsilverg2ca	21 Apr 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
thawteserverca	21 Apr 2018	9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
equifaxsecureebusinessca1	21 Apr 2018	AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
securetrustca	21 Apr 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
utnuserfirstclientauthemailca	21 Apr 2018	B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
thawtepersonalfreemailca	21 Apr 2018	E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
affirmtrustnetworkingca	21 Apr 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
entrustevca	21 Apr 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
utnuserfirsthardwareca	21 Apr 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
certumca	21 Apr 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
addtrustclass1ca	21 Apr 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
entrustrootcag2	21 Apr 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4

Nama	Tanggal	Sidik Jari SHA1
equifaxsecureca	21 Apr 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
quovadisrootca3	21 Apr 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
quovadisrootca2	21 Apr 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
digicertglobalroot g2	21 Apr 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
digicerthighassura nceevrootca	21 Apr 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
secomvalicertclass 1ca	21 Apr 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
equifaxsecureglobal ebusinessca1	21 Apr 2018	3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36
geotrustuniversalc a	21 Apr 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
deprecateditsecca	27 Jan 2012	12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
verisignclass3ca	21 Apr 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
thawteprimaryrootc ag3	21 Apr 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
thawteprimaryrootc ag2	21 Apr 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
deutschetelekomroo tca2	21 Apr 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF

Nama	Tanggal	Sidik Jari SHA1
buypassclass3ca	21 Apr 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
utnuserfirstobjectca	21 Apr 2018	E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
geotrustprimaryca	21 Apr 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
buypassclass2ca	21 Apr 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
baltimorecodesigningca	21 Apr 2018	30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD:49:27:08:7C:60:56:7B:0D
verisignclass1ca	21 Apr 2018	CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45:3E:64:09:EA:E8:7D:60:F1
baltimorecybertrustca	21 Apr 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
starfieldclass2ca	21 Apr 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
camerfirmachamberscommerceca	21 Apr 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
ttelesecglobalrootclass3ca	21 Apr 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
verisignclass3g5ca	21 Apr 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
ttelesecglobalrootclass2ca	21 Apr 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
trustcenteruniversalcai	21 Apr 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3

Nama	Tanggal	Sidik Jari SHA1
verisignclass3g4ca	21 Apr 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
verisignclass3g3ca	21 Apr 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
xrampglobalca	21 Apr 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
amzninternalrootca	12 Des 2008	A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
certplusclass3pprimaryca	21 Apr 2018	21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
certumtrustednetworkca	21 Apr 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
verisignclass3g2ca	21 Apr 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
globalsignr3ca	21 Apr 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
utndatacorpsgcca	21 Apr 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
secomscrootca2	21 Apr 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
gtecybertrustglobalca	21 Apr 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
secomscrootca1	21 Apr 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
affirmtrustcommercialca	21 Apr 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7

Nama	Tanggal	Sidik Jari SHA1
trustcenterclass4caii	21 Apr 2018	A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
verisignuniversalrootca	21 Apr 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
globalsignr2ca	21 Apr 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
certplusclass2primaryca	21 Apr 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
digicertglobalrootca	21 Apr 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
globalsignca	21 Apr 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
thawteprimaryrootca	21 Apr 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
starfieldrootg2ca	21 Apr 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
geotrustglobalca	21 Apr 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
soneraclass2ca	21 Apr 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
verisigntsaca	21 Apr 2018	20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
soneraclass1ca	21 Apr 2018	07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
quovadisrootca	21 Apr 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9

Nama	Tanggal	Sidik Jari SHA1
affirmtrustpremiumeccca	21 Apr 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
starfieldservicesrootg2ca	21 Apr 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
valicertclass2ca	21 Apr 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
comodoaaaca	21 Apr 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
aolrootca2	21 Apr 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
keynectisrootca	21 Apr 2018	9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
addtrustqualifiedca	21 Apr 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
aolrootca1	21 Apr 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
verisignclass2g3ca	21 Apr 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
addtrustexternalca	21 Apr 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
verisignclass2g2ca	21 Apr 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
geotrustprimarycag3	21 Apr 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycag2	21 Apr 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0

Nama	Tanggal	Sidik Jari SHA1
swisssigngoldg2ca	21 Apr 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
entrust2048ca	21 Apr 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
chunghwaepkirootca	21 Apr 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
camerfirmachambersignca	21 Apr 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
camerfirmachambersca	21 Apr 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
godaddyclass2ca	21 Apr 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
affirmtrustpremiumca	21 Apr 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
verisignclass1g3ca	21 Apr 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
secomevrootca1	21 Apr 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
verisignclass1g2ca	21 Apr 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
amzninternalinfosecag3	27 Feb 2015	B9:B1:CA:38:F7:BF:9C:D2:D4:95:E7:B6:5E:75:32:9B:A8:78:2E:F6
cia-crt-g3-01-ca	23 Nov 2016	2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
godaddyrootg2ca	21 Apr 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B

Nama	Tanggal	Sidik Jari SHA1
digicertassuredidrootca	21 Apr 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
microseceszignorootca2009	21 Apr 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
affirmtrustcommercial	21 Apr 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
comodoecccertificationauthority	21 Apr 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
cadisigrootr2	21 Apr 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
swisssignsilvercag2	21 Apr 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
securetrustca	21 Apr 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
cadisigrootr1	21 Apr 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
accvraiz1	21 Apr 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
entrustrootcertificationauthority	21 Apr 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
camerfirmaglobalchambersignroot	21 Apr 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
dstacescax6	21 Apr 2018	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
identrustpublicsectorrootca1	21 Apr 2018	BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD

Nama	Tanggal	Sidik Jari SHA1
starfieldrootcertificateauthorityg2	21 Apr 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
secureglobalca	21 Apr 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
eecertificationcentrerootca	21 Apr 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
opentrustrootcag3	21 Apr 2018	6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6
teliasonerarootcav1	21 Apr 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
autoridaddecertificacionfirmaprofesionalcifa62634068	21 Apr 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
opentrustrootcag2	21 Apr 2018	79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
opentrustrootcag1	21 Apr 2018	79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E
globalsigneccrootcar5	21 Apr 2018	1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
globalsigneccrootcar4	21 Apr 2018	69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
izenpecom	21 Apr 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19

Nama	Tanggal	Sidik Jari SHA1
turktrustelektroniksertifikahizmetsaglayicisih5	21 Apr 2018	C4:18:F6:4D:46:D1:DF:00:3D:27:30:13:72:43:A9:12:11:C6:75:FB
gdcatrustauthr5root	21 Apr 2018	0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
dtrustrootclass3ca22009	21 Apr 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
quovadisrootca3	21 Apr 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
quovadisrootca2	21 Apr 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
geotrustprimarycertificationauthorityg3	21 Apr 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycertificationauthorityg2	21 Apr 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
oistewisekeyglobalrootgbca	21 Apr 2018	0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
addtrustexternalroot	21 Apr 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
chambersofcommerceroot2008	21 Apr 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
digicertglobalrootg3	21 Apr 2018	7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E

Nama	Tanggal	Sidik Jari SHA1
comodoaaaservicesroot	21 Apr 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
digicertglobalrootg2	21 Apr 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
certinomisrootca	21 Apr 2018	9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C:01:B9:32:C5:34:E7:88:A8
oistewisekeyglobalrootgaca	21 Apr 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
dstrootcax3	21 Apr 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
certigna	21 Apr 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
digicerthighassuranceevrootca	21 Apr 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
soneraclass2rootca	21 Apr 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
trustcorrootcertca2	21 Apr 2018	B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
usertrustrsacertificationauthority	21 Apr 2018	2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
trustcorrootcertca1	21 Apr 2018	FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
geotrustuniversalca	21 Apr 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
certsignrootca	21 Apr 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B

Nama	Tanggal	Sidik Jari SHA1
amazonrootca4	21 Apr 2018	F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
amazonrootca3	21 Apr 2018	0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
amazonrootca2	21 Apr 2018	5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
verisignuniversalrootcertificationauthority	21 Apr 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
amazonrootca1	21 Apr 2018	8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
networksolutionscertificate authority	21 Apr 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
thawteprimaryrootcag3	21 Apr 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
affirmtrustnetworking	21 Apr 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
thawteprimaryrootcag2	21 Apr 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
trustcoreca1	21 Apr 2018	58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
deutschetelekomrootca2	21 Apr 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
godaddyrootcertificateauthorityg2	21 Apr 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B

Nama	Tanggal	Sidik Jari SHA1
entrustrootcertificationauthorityec1	21 Apr 2018	20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
szafirrootca2	21 Apr 2018	E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
tubitakkamusmsslkoksertifikasisurum1	21 Apr 2018	31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA
buypassclass3rootca	21 Apr 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
comodorsacertificationauthority	21 Apr 2018	AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
netlockaranyclassgoldfotanusitvany	21 Apr 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
securitycommunicationrootca2	21 Apr 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
dtrustrootclass3ca2ev2009	21 Apr 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
starfieldclass2ca	21 Apr 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
pscprocert	21 Apr 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
actalisauthenticationrootca	21 Apr 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
staatdernederlandenrootcag3	21 Apr 2018	D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC

Nama	Tanggal	Sidik Jari SHA1
cfcaevroot	21 Apr 2018	E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
digicerttrustedrootg4	21 Apr 2018	DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F:C8:3A:4D:7D:77:5D:05:E4
staatdernederlandenrootcag2	21 Apr 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
securitycommunicationevrootca1	21 Apr 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
globalsignrootcar3	21 Apr 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
globalsignrootcar2	21 Apr 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
certumtrustednetworkca2	21 Apr 2018	D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
acraizfnmtrcm	21 Apr 2018	EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
hellenicacademicandresearchinstitutionseccrootca2015	21 Apr 2018	9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
certplusrootcag2	21 Apr 2018	4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A
twcarootcertificationauthority	21 Apr 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
twcaglobalrootca	21 Apr 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65

Nama	Tanggal	Sidik Jari SHA1
certplusrootcag1	21 Apr 2018	22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
geotrustuniversalca2	21 Apr 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
baltimorecybertrustroot	21 Apr 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
buypassclass2rootca	21 Apr 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
certumtrustednetworkca	21 Apr 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
digicertassuredidrootg3	21 Apr 2018	F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
digicertassuredidrootg2	21 Apr 2018	A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F
isrgrootx1	21 Apr 2018	CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
entrustnetpremium2048secureserverca	21 Apr 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
certplusclass2primaryca	21 Apr 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
digicertglobalrootca	21 Apr 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
entrustrootcertificationauthorityg2	21 Apr 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4

Nama	Tanggal	Sidik Jari SHA1
starfieldservicesrootcertificateauthorityg2	21 Apr 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
thawteprimaryrootca	21 Apr 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
atostrustedroot2011	21 Apr 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
geotrustglobalca	21 Apr 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
luxtrustglobalroot2	21 Apr 2018	1E:0E:56:19:0A:D1:8B:25:98:B2:04:44:FF:66:8A:04:17:99:5F:3F
etugracertificationauthority	21 Apr 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
visaecommerceroot	21 Apr 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
quovadisrootca	21 Apr 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
identrustcommercialrootca1	21 Apr 2018	DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
staatdernederlandenevrootca	21 Apr 2018	76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
ttelesecglobalrootclass3	21 Apr 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
ttelesecglobalrootclass2	21 Apr 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9

Nama	Tanggal	Sidik Jari SHA1
comodocertificationauthority	21 Apr 2018	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
securitycommunicationrootca	21 Apr 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
quovadisrootca3g3	21 Apr 2018	48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
xrampglobalcaroot	21 Apr 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
securesignrootca11	21 Apr 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
affirmtrustpremium	21 Apr 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
globalsignrootca	21 Apr 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
swisssigngoldcag2	21 Apr 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
quovadisrootca2g3	21 Apr 2018	09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
affirmtrustpremiumecc	21 Apr 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
geotrustprimarycertificationauthority	21 Apr 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
quovadisrootca1g3	21 Apr 2018	1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67

Nama	Tanggal	Sidik Jari SHA1
hongkongpostrootca1	21 Apr 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
usertrustecccertificationauthority	21 Apr 2018	D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
cybertrustglobalroot	21 Apr 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
godaddyclass2ca	21 Apr 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
hellenicacademicandresearchinstitutionsrootca2015	21 Apr 2018	01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
ecacc	21 Apr 2018	28:90:3A:63:5B:52:80:FA:E6:77:4C:0B:6D:A7:D6:BA:A6:4A:F2:E8
hellenicacademicandresearchinstitutionsrootca2011	21 Apr 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
verisignclass3publicprimarycertificationauthorityg5	21 Apr 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
verisignclass3publicprimarycertificationauthorityg4	21 Apr 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A

Nama	Tanggal	Sidik Jari SHA1
verisignclass3publicprimary certificationauthorityg3	21 Apr 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
trustisfpsrootca	21 Apr 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
epkirootcertificationauthority	21 Apr 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
globalchambersignroot2008	21 Apr 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
camerfirmachambersofcommerceroot	21 Apr 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
mozillacert81.pem	13 Mar 2014	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
mozillacert99.pem	13 Mar 2014	F1:7F:6F:B6:31:DC:99:E3:A3:C8:7F:FE:1C:F1:81:10:88:D9:60:33
mozillacert145.pem	13 Mar 2014	10:1D:FA:3F:D5:0B:CB:BB:9B:B5:60:0C:19:55:A4:1A:F4:73:3A:04
mozillacert37.pem	13 Mar 2014	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert4.pem	13 Mar 2014	E3:92:51:2F:0A:CF:F5:05:DF:F6:DE:06:7F:75:37:E1:65:EA:57:4B
mozillacert70.pem	13 Mar 2014	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
mozillacert88.pem	13 Mar 2014	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D

Nama	Tanggal	Sidik Jari SHA1
mozillacert134.pem	13 Mar 2014	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
mozillacert26.pem	13 Mar 2014	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
mozillacert77.pem	13 Mar 2014	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert123.pem	13 Mar 2014	2A:B6:28:48:5E:78:FB:F3:AD:9E:79:10:DD:6B:DF:99:72:2C:96:E5
mozillacert15.pem	13 Mar 2014	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
mozillacert66.pem	13 Mar 2014	DD:E1:D2:A9:01:80:2E:1D:87:5E:84:B3:80:7E:4B:B1:FD:99:41:34
mozillacert112.pem	13 Mar 2014	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
mozillacert55.pem	13 Mar 2014	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
mozillacert101.pem	13 Mar 2014	99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39
mozillacert119.pem	13 Mar 2014	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
mozillacert44.pem	13 Mar 2014	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
mozillacert108.pem	13 Mar 2014	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
mozillacert95.pem	13 Mar 2014	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57

Nama	Tanggal	Sidik Jari SHA1
mozillacert141.pem	13 Mar 2014	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
mozillacert33.pem	13 Mar 2014	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
mozillacert0.pem	13 Mar 2014	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
mozillacert84.pem	13 Mar 2014	D3:C0:63:F2:19:ED:07:3E:34:AD:5D:75:0B:32:76:29:FF:D5:9A:F2
mozillacert130.pem	13 Mar 2014	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
mozillacert148.pem	13 Mar 2014	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
mozillacert22.pem	13 Mar 2014	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
mozillacert7.pem	13 Mar 2014	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
mozillacert73.pem	13 Mar 2014	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
mozillacert137.pem	13 Mar 2014	4A:65:D5:F4:1D:EF:39:B8:B8:90:4A:4A:D3:64:81:33:CF:C7:A1:D1
mozillacert11.pem	13 Mar 2014	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
mozillacert29.pem	13 Mar 2014	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
mozillacert62.pem	13 Mar 2014	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B

Nama	Tanggal	Sidik Jari SHA1
mozillacert126.pem	13 Mar 2014	25:01:90:19:CF:FB:D9:99:1C:B7:68:25:74:8D:94:5F:30:93:95:42
mozillacert18.pem	13 Mar 2014	79:98:A3:08:E1:4D:65:85:E6:C2:1E:15:3A:71:9F:BA:5A:D3:4A:D9
mozillacert51.pem	13 Mar 2014	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
mozillacert69.pem	13 Mar 2014	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
mozillacert115.pem	13 Mar 2014	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
mozillacert40.pem	13 Mar 2014	80:25:EF:F4:6E:70:C8:D4:72:24:65:84:FE:40:3B:8A:8D:6A:DB:F5
mozillacert58.pem	13 Mar 2014	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
mozillacert104.pem	13 Mar 2014	4F:99:AA:93:FB:2B:D1:37:26:A1:99:4A:CE:7F:F0:05:F2:93:5D:1E
mozillacert91.pem	13 Mar 2014	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
mozillacert47.pem	13 Mar 2014	1B:4B:39:61:26:27:6B:64:91:A2:68:6D:D7:02:43:21:2D:1F:1D:96
mozillacert80.pem	13 Mar 2014	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
mozillacert98.pem	13 Mar 2014	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
mozillacert144.pem	13 Mar 2014	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27

Nama	Tanggal	Sidik Jari SHA1
mozillacert36.pem	13 Mar 2014	23:88:C9:D3:71:CC:9E:96:3D:FF:7D:3C:A7:CE:FC:D6:25:EC:19:0D
mozillacert3.pem	13 Mar 2014	87:9F:4B:EE:05:DF:98:58:3B:E3:60:D6:33:E7:0D:3F:FE:98:71:AF
mozillacert87.pem	13 Mar 2014	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
mozillacert133.pem	13 Mar 2014	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
mozillacert25.pem	13 Mar 2014	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
mozillacert76.pem	13 Mar 2014	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert122.pem	13 Mar 2014	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
mozillacert14.pem	13 Mar 2014	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
mozillacert65.pem	13 Mar 2014	69:BD:8C:F4:9C:D3:00:FB:59:2E:17:93:CA:55:6A:F3:EC:AA:35:FB
mozillacert111.pem	13 Mar 2014	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
mozillacert129.pem	13 Mar 2014	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
mozillacert54.pem	13 Mar 2014	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
mozillacert100.pem	13 Mar 2014	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0

Nama	Tanggal	Sidik Jari SHA1
mozillacert118.pem	13 Mar 2014	7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
mozillacert151.pem	13 Mar 2014	AC:ED:5F:65:53:FD:25:CE:01:5F:1F:7A:48:3B:6A:74:9F:61:78:C6
mozillacert43.pem	13 Mar 2014	F9:CD:0E:2C:DA:76:24:C1:8F:BD:F0:F0:AB:B6:45:B8:F7:FE:D5:7A
mozillacert107.pem	13 Mar 2014	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
mozillacert94.pem	13 Mar 2014	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
mozillacert140.pem	13 Maret 2014	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
mozillacert32.pem	13 Maret 2014	60:D6:89:74:B5:C2:65:9E:8A:0F:C1:88:7C:88:D2:46:69:1B:18:2C
mozillacert83.pem	13 Maret 2014	A0:73:E5:C5:BD:43:61:0D:86:4C:21:13:0A:85:58:57:CC:9C:EA:46
mozillacert147.pem	13 Maret 2014	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
mozillacert21.pem	13 Maret 2014	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
mozillacert39.pem	13 Maret 2014	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
mozillacert6.pem	13 Maret 2014	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
mozillacert72.pem	13 Maret 2014	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B

Nama	Tanggal	Sidik Jari SHA1
mozillacert136.pem	13 Mar 2014	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert10.pem	13 Mar 2014	5F:3A:FC:0A:8B:64:F6:86:67:34:74:DF:7E:A9:A2:FE:F9:FA:7A:51
mozillacert28.pem	13 Mar 2014	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
mozillacert61.pem	13 Mar 2014	E0:B4:32:2E:B2:F6:A5:68:B6:54:53:84:48:18:4A:50:36:87:43:84
mozillacert79.pem	13 Maret 2014	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
mozillacert125.pem	13 Maret 2014	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert17.pem	13 Maret 2014	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
mozillacert50.pem	13 Maret 2014	8C:96:BA:EB:DD:2B:07:07:48:EE:30:32:66:A0:F3:98:6E:7C:AE:58
mozillacert68.pem	13 Maret 2014	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
mozillacert114.pem	13 Maret 2014	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert57.pem	13 Maret 2014	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
mozillacert103.pem	13 Maret 2014	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
mozillacert90.pem	13 Maret 2014	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC

Nama	Tanggal	Sidik Jari SHA1
mozillacert46.pem	13 Maret 2014	40:9D:4B:D9:17:B5:5C:27:B6:9B:64:CB:98:22:44:0D:CD:09:B8:89
mozillacert97.pem	13 Maret 2014	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
mozillacert143.pem	13 Maret 2014	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
mozillacert35.pem	13 Maret 2014	2A:C8:D5:8B:57:CE:BF:2F:49:AF:F2:FC:76:8F:51:14:62:90:7A:41
mozillacert2.pem	13 Maret 2014	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
mozillacert86.pem	13 Maret 2014	74:2C:31:92:E6:07:E4:24:EB:45:49:54:2B:E1:BB:C5:3E:61:74:E2
mozillacert132.pem	13 Maret 2014	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
mozillacert24.pem	13 Maret 2014	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
mozillacert9.pem	13 Maret 2014	F4:8B:11:BF:DE:AB:BE:94:54:20:71:E6:41:DE:6B:BE:88:2B:40:B9
mozillacert75.pem	13 Maret 2014	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
mozillacert121.pem	13 Maret 2014	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
mozillacert139.pem	13 Maret 2014	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
mozillacert13.pem	13 Maret 2014	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91

Nama	Tanggal	Sidik Jari SHA1
mozillacert64.pem	13 Maret 2014	62:7F:8D:78:27:65:63:99:D2:7D:7F:90:44:C9:FE:B3:F3:3E:FA:9A
mozillacert110.pem	13 Maret 2014	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
mozillacert128.pem	13 Maret 2014	A9:E9:78:08:14:37:58:88:F2:05:19:B0:6D:2B:0D:2B:60:16:90:7D
mozillacert53.pem	13 Maret 2014	7F:8A:B0:CF:D0:51:87:6A:66:F3:36:0F:47:C8:8D:8C:D3:35:FC:74
mozillacert117.pem	13 Maret 2014	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
mozillacert150.pem	13 Maret 2014	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
mozillacert42.pem	13 Maret 2014	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
mozillacert106.pem	13 Maret 2014	E7:A1:90:29:D3:D5:52:DC:0D:0F:C6:92:D3:EA:88:0D:15:2E:1A:6B
mozillacert93.pem	13 Maret 2014	31:F1:FD:68:22:63:20:EE:C6:3B:3F:9D:EA:4A:3E:53:7C:7C:39:17
mozillacert31.pem	13 Maret 2014	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
mozillacert49.pem	13 Maret 2014	61:57:3A:11:DF:0E:D8:7E:D5:92:65:22:EA:D0:56:D7:44:B3:23:71
mozillacert82.pem	13 Maret 2014	2E:14:DA:EC:28:F0:FA:1E:8E:38:9A:4E:AB:EB:26:C0:0A:D3:83:C3
mozillacert146.pem	13 Maret 2014	21:FC:BD:8E:7F:6C:AF:05:1B:D1:B3:43:EC:A8:E7:61:47:F2:0F:8A

Nama	Tanggal	Sidik Jari SHA1
mozillacert20.pem	13 Maret 2014	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert38.pem	13 Maret 2014	CB:A1:C5:F8:B0:E3:5E:B8:B9:45:12:D3:F9:34:A2:E9:06:10:D3:36
mozillacert5.pem	13 Maret 2014	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
mozillacert71.pem	13 Maret 2014	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
mozillacert89.pem	13 Maret 2014	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
mozillacert135.pem	13 Maret 2014	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
mozillacert27.pem	13 Maret 2014	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
mozillacert60.pem	13 Maret 2014	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
mozillacert78.pem	13 Maret 2014	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
mozillacert124.pem	13 Maret 2014	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert16.pem	13 Maret 2014	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
mozillacert67.pem	13 Maret 2014	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
mozillacert113.pem	13 Maret 2014	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31

Nama	Tanggal	Sidik Jari SHA1
mozillacert56.pem	13 Maret 2014	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
mozillacert102.pem	13 Maret 2014	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
mozillacert45.pem	13 Mar 2014	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert109.pem	13 Mar 2014	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
mozillacert96.pem	13 Mar 2014	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
mozillacert142.pem	13 Mar 2014	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
mozillacert34.pem	13 Mar 2014	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
mozillacert1.pem	13 Mar 2014	23:E5:94:94:51:95:F2:41:48:03:B4:D5:64:D2:A3:A3:F5:D8:8B:8C
mozillacert85.pem	13 Maret 2014	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
mozillacert131.pem	13 Maret 2014	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
mozillacert149.pem	13 Maret 2014	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
mozillacert23.pem	13 Maret 2014	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
mozillacert8.pem	13 Maret 2014	3E:2B:F7:F2:03:1B:96:F3:8C:E6:C4:D8:A8:5D:3E:2D:58:47:6A:0F

Nama	Tanggal	Sidik Jari SHA1
mozillacert74.pem	13 Maret 2014	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
mozillacert120.pem	13 Mar 2014	DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97:FE:2F:9D:F5:B7:D1:8A:41
mozillacert138.pem	13 Mar 2014	E1:9F:E3:0E:8B:84:60:9E:80:9B:17:0D:72:A8:C5:BA:6E:14:09:BD
mozillacert12.pem	13 Mar 2014	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
mozillacert63.pem	13 Mar 2014	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
mozillacert127.pem	13 Mar 2014	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
mozillacert19.pem	13 Mar 2014	B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB:B3:94:BD:63:7B:A7:82:B7
mozillacert52.pem	13 Mar 2014	8B:AF:4C:9B:1D:F0:2A:92:F7:DA:12:8E:B9:1B:AC:F4:98:60:4B:6F
mozillacert116.pem	13 Mar 2014	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
mozillacert41.pem	13 Mar 2014	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
mozillacert59.pem	13 Mar 2014	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
mozillacert105.pem	13 Mar 2014	77:47:4F:C6:30:E4:0F:4C:47:64:3F:84:BA:B8:C6:95:4A:8A:41:EC
mozillacert92.pem	13 Mar 2014	A3:F1:33:3F:E2:42:BF:CF:C5:D1:4E:8F:39:42:98:40:68:10:D1:A0

Nama	Tanggal	Sidik Jari SHA1
mozillacert30.pem	13 Mar 2014	E7:B4:F6:9D:61:EC:90:69:DB:7E:90:A7:40:1A:3C:F4:7D:4F:E8:EE
mozillacert48.pem	13 Mar 2014	A0:A1:AB:90:C9:FC:84:7B:3B:12:61:E8:97:7D:5F:D3:22:61:D3:CC
verisignc4g2.pem	20 Mar 2014	0B:77:BE:BB:CB:7A:A2:47:05:DE:CC:0F:BD:6A:02:FC:7A:BD:9B:52
verisignc2g3.pem	20 Mar 2014	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
verisignc1g6.pem	31 Des 2014	51:7F:61:1E:29:91:6B:53:82:FB:72:E7:44:D9:8D:C3:CC:53:6D:64
verisignc2g2.pem	20 Mar 2014	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
verisignroot.pem	20 Mar 2014	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
verisignc2g1.pem	20 Mar 2014	67:82:AA:E0:ED:EE:E2:1A:58:39:D3:C0:CD:14:68:0A:4F:60:14:2A
verisignc3g5.pem	20 Mar 2014	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
verisignc1g3.pem	20 Mar 2014	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
verisignc3g4.pem	20 Mar 2014	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
verisignc1g2.pem	20 Mar 2014	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
verisignc3g3.pem	20 Mar 2014	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6

Nama	Tanggal	Sidik Jari SHA1
verisignc1g1.pem	20 Mar 2014	90:AE:A2:69:85:FF:14:80:4C:43:49:52:EC:E9:60:84:77:AF:55:6F
verisignc3g2.pem	20 Mar 2014	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
verisignc3g1.pem	20 Mar 2014	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
verisignc2g6.pem	31 Des 2014	40:B3:31:A0:E9:BF:E8:55:BC:39:93:CA:70:4F:4E:C2:51:D4:1D:8F
verisignc4g3.pem	20 Mar 2014	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
gdroot-g2.pem	31 Des 2014	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
gd-class2-root.pem	31 Des 2014	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
gd_bundle-g2.pem	31 Des 2014	27:AC:93:69:FA:F2:52:07:BB:26:27:CE:FA:CC:BE:4E:F9:C3:19:B8
dstacescax6	18 Jun 2018	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
gd_bundle-g2.pem	18 Jun 2018	27:AC:93:69:FA:F2:52:07:BB:26:27:CE:FA:CC:BE:4E:F9:C3:19:B8
verisignc4g3.pem	18 Jun 2018	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
swisssignplatinumg2ca	21 Mei 2018	56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66

Nama	Tanggal	Sidik Jari SHA1
geotrustprimarycertificationauthorityg3	18 Jun 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycertificationauthorityg2	18 Jun 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
buypassclass2rootca	18 Jun 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
camerfirmachambersofcommerceroot	18 Jun 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
mozillacert20.pem	18 Jun 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert12.pem	18 Jun 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
mozillacert90.pem	18 Jun 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
mozillacert82.pem	18 Jun 2018	2E:14:DA:EC:28:F0:FA:1E:8E:38:9A:4E:AB:EB:26:C0:0A:D3:83:C3
mozillacert140.pem	18 Jun 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
mozillacert74.pem	18 Jun 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
mozillacert132.pem	18 Jun 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
mozillacert66.pem	18 Jun 2018	DD:E1:D2:A9:01:80:2E:1D:87:5E:84:B3:80:7E:4B:B1:FD:99:41:34

Nama	Tanggal	Sidik Jari SHA1
mozillacert124.pem	18 Jun 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert58.pem	18 Jun 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
securitycommunicationrootca2	18 Jun 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
mozillacert116.pem	18 Jun 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
mozillacert108.pem	18 Jun 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
certigna	18 Jun 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert3.pem	18 Jun 2018	87:9F:4B:EE:05:DF:98:58:3B:E3:60:D6:33:E7:0D:3F:FE:98:71:AF
verisignc1g1.pem	18 Jun 2018	90:AE:A2:69:85:FF:14:80:4C:43:49:52:EC:E9:60:84:77:AF:55:6F
verisignc4g2.pem	18 Jun 2018	0B:77:BE:BB:CB:7A:A2:47:05:DE:CC:0F:BD:6A:02:FC:7A:BD:9B:52
deutschetelekomrootca2	18 Jun 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
starfieldrootg2ca	21 Mei 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
comodoecccertificationauthority	18 Jun 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
digicertglobalrootg3	18 Jun 2018	7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E

Nama	Tanggal	Sidik Jari SHA1
digicertglobalroot g2	18 Jun 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73: FE:06:D1:CC:8D:4F:82:A4
mozillacert11.pem	18 Jun 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E: 4B:DF:B5:A8:99:B2:4D:43
mozillacert81.pem	18 Jun 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28: A2:59:3A:19:A7:0F:06:9E
mozillacert73.pem	18 Jun 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D: 92:F4:FE:39:D4:E7:0F:0E
szafirrootca2	18 Jun 2018	E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28: F3:9C:CC:CF:5E:B3:3F:DE
mozillacert131.pem	18 Jun 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69: F3:3C:2E:AF:47:4F:20:79
ecacc	18 Jun 2018	28:90:3A:63:5B:52:80:FA:E6:77:4C:0B: 6D:A7:D6:BA:A6:4A:F2:E8
mozillacert65.pem	18 Jun 2018	69:BD:8C:F4:9C:D3:00:FB:59:2E:17:93: CA:55:6A:F3:EC:AA:35:FB
turktrustelektroni ksertifik ahizmetsa glayicisih5	18 Jun 2018	C4:18:F6:4D:46:D1:DF:00:3D:27:30:13: 72:43:A9:12:11:C6:75:FB
mozillacert123.pem	18 Jun 2018	2A:B6:28:48:5E:78:FB:F3:AD:9E:79:10: DD:6B:DF:99:72:2C:96:E5
mozillacert57.pem	18 Jun 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F: CB:34:6E:B2:58:B2:8A:58
mozillacert115.pem	18 Jun 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62: 32:17:65:CF:17:D8:94:E9

Nama	Tanggal	Sidik Jari SHA1
mozillacert49.pem	18 Jun 2018	61:57:3A:11:DF:0E:D8:7E:D5:92:65:22:EA:D0:56:D7:44:B3:23:71
mozillacert107.pem	18 Jun 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
verisignclass3g4ca	21 Apr 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
securetrustca	18 Jun 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
mozillacert2.pem	18 Jun 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
buypassclass2ca	21 Apr 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
secomscrootca2	21 Apr 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
secomscrootca1	21 Apr 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
trustisfpsrootca	18 Jun 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
hongkongpostrootca1	18 Jun 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
certsignrootca	18 Jun 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
geotrustprimaryca	21 Apr 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
twcaglobalrootca	18 Jun 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65

Nama	Tanggal	Sidik Jari SHA1
camerfirmachambersca	21 Apr 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
mozillacert10.pem	18 Jun 2018	5F:3A:FC:0A:8B:64:F6:86:67:34:74:DF:7E:A9:A2:FE:F9:FA:7A:51
mozillacert80.pem	18 Jun 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
mozillacert72.pem	18 Jun 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
comodoaaaca	21 Apr 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert130.pem	18 Jun 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
mozillacert64.pem	18 Jun 2018	62:7F:8D:78:27:65:63:99:D2:7D:7F:90:44:C9:FE:B3:F3:3E:FA:9A
mozillacert122.pem	18 Jun 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
mozillacert56.pem	18 Jun 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
equifaxsecureebusinessca1	21 Apr 2018	AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
camerfirmachambersignca	21 Apr 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
mozillacert114.pem	18 Jun 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert48.pem	18 Jun 2018	A0:A1:AB:90:C9:FC:84:7B:3B:12:61:E8:97:7D:5F:D3:22:61:D3:CC

Nama	Tanggal	Sidik Jari SHA1
pscprocert	18 Jun 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
mozillacert106.pem	18 Jun 2018	E7:A1:90:29:D3:D5:52:DC:0D:0F:C6:92:D3:EA:88:0D:15:2E:1A:6B
mozillacert1.pem	18 Jun 2018	23:E5:94:94:51:95:F2:41:48:03:B4:D5:64:D2:A3:A3:F5:D8:8B:8C
eecertificationcentrerootca	18 Jun 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
digicertglobalrootca	18 Jun 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
thawteprimaryrootcag3	18 Jun 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
thawteprimaryrootcag2	18 Jun 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
entrustrootcertificationauthorityec1	18 Jun 2018	20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
valicertclass2ca	21 Apr 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
globalchambersignroot2008	18 Jun 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
amazonrootca4	18 Jun 2018	F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
gd-class2-root.pem	18 Jun 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4

Nama	Tanggal	Sidik Jari SHA1
amazonrootca3	18 Jun 2018	0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
amazonrootca2	18 Jun 2018	5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
securitycommunicationrootca	18 Jun 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
amazonrootca1	18 Jun 2018	8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
acraizfnmtrcm	18 Jun 2018	EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
quovadisrootca3g3	18 Jun 2018	48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
certplusrootcag2	18 Jun 2018	4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A
certplusrootcag1	18 Jun 2018	22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
mozillacert71.pem	18 Jun 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
mozillacert63.pem	18 Jun 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
mozillacert121.pem	18 Jun 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
ttelesecglobalrootclass3ca	21 Apr 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
mozillacert55.pem	18 Jun 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12

Nama	Tanggal	Sidik Jari SHA1
mozillacert113.pem	18 Jun 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
baltimorecybertrustca	21 Apr 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
mozillacert47.pem	18 Jun 2018	1B:4B:39:61:26:27:6B:64:91:A2:68:6D:D7:02:43:21:2D:1F:1D:96
mozillacert105.pem	18 Jun 2018	77:47:4F:C6:30:E4:0F:4C:47:64:3F:84:BA:B8:C6:95:4A:8A:41:EC
mozillacert39.pem	18 Jun 2018	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
usertrustecccertificationauthority	18 Jun 2018	D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
mozillacert0.pem	18 Jun 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
securitycommunicationevrootca1	18 Jun 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
verisignc3g5.pem	18 Jun 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
globalsignr3ca	21 Apr 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
trustcoreca1	18 Jun 2018	58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
equifaxsecureglobalebusinessca1	21 Apr 2018	3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36
geotrustuniversalca	18 Jun 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79

Nama	Tanggal	Sidik Jari SHA1
affirmtrustpremiumca	21 Apr 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
staatdernederlandenrootcag3	18 Jun 2018	D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC
staatdernederlandenrootcag2	18 Jun 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
mozillacert70.pem	18 Jun 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
secomevrootca1	21 Apr 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
geotrustglobalca	18 Jun 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
mozillacert62.pem	18 Jun 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert120.pem	18 Jun 2018	DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97:FE:2F:9D:F5:B7:D1:8A:41
mozillacert54.pem	18 Jun 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
mozillacert112.pem	18 Jun 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
mozillacert46.pem	18 Jun 2018	40:9D:4B:D9:17:B5:5C:27:B6:9B:64:CB:98:22:44:0D:CD:09:B8:89
swisssigngoldcag2	18 Jun 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert104.pem	18 Jun 2018	4F:99:AA:93:FB:2B:D1:37:26:A1:99:4A:CE:7F:F0:05:F2:93:5D:1E

Nama	Tanggal	Sidik Jari SHA1
mozillacert38.pem	18 Jun 2018	CB:A1:C5:F8:B0:E3:5E:B8:B9:45:12:D3:F9:34:A2:E9:06:10:D3:36
certplusclass3ppri maryca	21 Apr 2018	21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
entrustrootcertifi cationauthorityg2	18 Jun 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
godaddyrootg2ca	21 Apr 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
cfcaevroot	18 Jun 2018	E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
verisignc3g4.pem	18 Jun 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
geotrustuniversalc a2	18 Jun 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
starfieldservicesr ootg2ca	21 Apr 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
digicerthighassura nceevrootca	18 Jun 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
entrustnetpremium2 048secureserverca	18 Jun 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
camerfirmaglobalch ambersignroot	18 Jun 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
verisignclass3g3ca	21 Apr 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
godaddyclass2ca	18 Jun 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4

Nama	Tanggal	Sidik Jari SHA1
mozillacert61.pem	18 Jun 2018	E0:B4:32:2E:B2:F6:A5:68:B6:54:53:84:48:18:4A:50:36:87:43:84
mozillacert53.pem	18 Jun 2018	7F:8A:B0:CF:D0:51:87:6A:66:F3:36:0F:47:C8:8D:8C:D3:35:FC:74
atostrustedroot2011	18 Jun 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
mozillacert111.pem	18 Jun 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
staatdernederlandenevrootca	18 Jun 2018	76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
mozillacert45.pem	18 Jun 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert103.pem	18 Jun 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
mozillacert37.pem	18 Jun 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert29.pem	18 Jun 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
izenpecom	18 Jun 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
comodorsacertificationauthority	18 Jun 2018	AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
mozillacert99.pem	18 Jun 2018	F1:7F:6F:B6:31:DC:99:E3:A3:C8:7F:FE:1C:F1:81:10:88:D9:60:33
mozillacert149.pem	18 Jun 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1

Nama	Tanggal	Sidik Jari SHA1
utnuserfirstobjectca	21 Apr 2018	E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
verisignc3g3.pem	18 Jun 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
dstrootcax3	18 Jun 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
addtrustexternalroot	18 Jun 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
certumtrustednetworkca	18 Jun 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
affirmtrustpremiumecc	18 Jun 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
starfieldclass2ca	18 Jun 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
actalisauthenticationrootca	18 Jun 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
verisignclass2g3ca	21 Apr 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
isrgrootx1	18 Jun 2018	CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
godaddyrootcertificateauthorityg2	18 Jun 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
mozillacert60.pem	18 Jun 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
chunghwaepkirootca	21 Apr 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0

Nama	Tanggal	Sidik Jari SHA1
mozillacert52.pem	18 Jun 2018	8B:AF:4C:9B:1D:F0:2A:92:F7:DA:12:8E:B9:1B:AC:F4:98:60:4B:6F
microseceszignorootca2009	18 Jun 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
securesignrootca11	18 Jun 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
mozillacert110.pem	18 Jun 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
mozillacert44.pem	18 Jun 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
mozillacert102.pem	18 Jun 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
mozillacert36.pem	18 Jun 2018	23:88:C9:D3:71:CC:9E:96:3D:FF:7D:3C:A7:CE:FC:D6:25:EC:19:0D
mozillacert28.pem	18 Jun 2018	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
baltimorecybertrustroot	18 Jun 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
amzninternalrootca	12 Des 2008	A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
mozillacert98.pem	18 Jun 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
mozillacert148.pem	18 Jun 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
verisignc3g2.pem	18 Jun 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F

Nama	Tanggal	Sidik Jari SHA1
quovadisrootca2g3	18 Jun 2018	09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
geotrustprimarycertificatio nauthority	18 Jun 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
opentrustrootcag3	18 Jun 2018	6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6
opentrustrootcag2	18 Jun 2018	79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
opentrustrootcag1	18 Jun 2018	79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E
verisignclass3ca	21 Apr 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
globalsignca	21 Apr 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
ttelesecglobalroot class2ca	21 Apr 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
verisignclass1g3ca	21 Apr 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
verisignuniversalr ootca	21 Apr 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
soneraclass2ca	21 Apr 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
starfieldservicesr ootcertif icateauthorityg2	18 Jun 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F

Nama	Tanggal	Sidik Jari SHA1
mozillacert51.pem	18 Jun 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
mozillacert43.pem	18 Jun 2018	F9:CD:0E:2C:DA:76:24:C1:8F:BD:F0:F0:AB:B6:45:B8:F7:FE:D5:7A
mozillacert101.pem	18 Jun 2018	99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39
mozillacert35.pem	18 Jun 2018	2A:C8:D5:8B:57:CE:BF:2F:49:AF:F2:FC:76:8F:51:14:62:90:7A:41
globalsignr2ca	21 Apr 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
mozillacert27.pem	18 Jun 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
affirmtrustpremium	18 Jun 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
mozillacert19.pem	18 Jun 2018	B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB:B3:94:BD:63:7B:A7:82:B7
mozillacert97.pem	18 Jun 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
netlockaranyclassgoldfotanusitvany	18 Jun 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
mozillacert89.pem	18 Jun 2018	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
verisignroot.pem	18 Jun 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
mozillacert147.pem	18 Jun 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4

Nama	Tanggal	Sidik Jari SHA1
aolrootca2	21 Apr 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
cia-crt-g3-01-ca	23 Nov 2016	2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
aolrootca1	21 Apr 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
verisignc3g1.pem	18 Jun 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert139.pem	18 Jun 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
soneraclass2rootca	18 Jun 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
swisssignsilverg2ca	21 Apr 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
thawteprimaryrootca	18 Jun 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
gdcatrustauthr5root	18 Jun 2018	0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
trustcenterclass4caii	21 Apr 2018	A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
usertrustrsacertificationauthority	18 Jun 2018	2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
digicertassuredidrootg3	18 Jun 2018	F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
digicertassuredidrootg2	18 Jun 2018	A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F

Nama	Tanggal	Sidik Jari SHA1
mozillacert50.pem	18 Jun 2018	8C:96:BA:EB:DD:2B:07:07:48:EE:30:32:66:A0:F3:98:6E:7C:AE:58
mozillacert42.pem	18 Jun 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
mozillacert100.pem	18 Jun 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
mozillacert34.pem	18 Jun 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
affirmtrustcommercialca	21 Apr 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert26.pem	18 Jun 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
globalsigneccrootcar5	18 Jun 2018	1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
globalsigneccrootcar4	18 Jun 2018	69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
buypassclass3rootca	18 Jun 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
mozillacert18.pem	18 Jun 2018	79:98:A3:08:E1:4D:65:85:E6:C2:1E:15:3A:71:9F:BA:5A:D3:4A:D9
mozillacert96.pem	18 Jun 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
verisignc2g6.pem	18 Jun 2018	40:B3:31:A0:E9:BF:E8:55:BC:39:93:CA:70:4F:4E:C2:51:D4:1D:8F
secomvalicertclass1ca	21 Apr 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E

Nama	Tanggal	Sidik Jari SHA1
mozillacert88.pem	18 Jun 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
accvraiz1	18 Jun 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
mozillacert146.pem	18 Jun 2018	21:FC:BD:8E:7F:6C:AF:05:1B:D1:B3:43:EC:A8:E7:61:47:F2:0F:8A
mozillacert138.pem	18 Jun 2018	E1:9F:E3:0E:8B:84:60:9E:80:9B:17:0D:72:A8:C5:BA:6E:14:09:BD
verisignclass3g2ca	21 Apr 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
dtrustrootclass3ca2ev2009	18 Jun 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
xrampglobalca	21 Apr 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
mozillacert9.pem	18 Jun 2018	F4:8B:11:BF:DE:AB:BE:94:54:20:71:E6:41:DE:6B:BE:88:2B:40:B9
verisignuniversalrootcertificationauthority	18 Jun 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
tubitakkamusmsslkoksertifikasisurum1	18 Jun 2018	31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA
mozillacert41.pem	18 Jun 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
mozillacert33.pem	18 Jun 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D

Nama	Tanggal	Sidik Jari SHA1
mozillacert25.pem	18 Jun 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
mozillacert17.pem	18 Jun 2018	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
mozillacert95.pem	18 Jun 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
affirmtrustpremiumeccca	21 Apr 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
mozillacert87.pem	18 Jun 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
mozillacert145.pem	18 Jun 2018	10:1D:FA:3F:D5:0B:CB:BB:9B:B5:60:0C:19:55:A4:1A:F4:73:3A:04
mozillacert79.pem	18 Jun 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
mozillacert137.pem	18 Jun 2018	4A:65:D5:F4:1D:EF:39:B8:B8:90:4A:4A:D3:64:81:33:CF:C7:A1:D1
digicertassuredidrootca	18 Jun 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
addtrustqualifiedca	21 Apr 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert129.pem	18 Jun 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
verisignclass2g2ca	21 Apr 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
baltimorecodesigningca	21 Apr 2018	30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD:49:27:08:7C:60:56:7B:0D

Nama	Tanggal	Sidik Jari SHA1
luxtrustglobalroot2	18 Jun 2018	1E:0E:56:19:0A:D1:8B:25:98:B2:04:44:FF:66:8A:04:17:99:5F:3F
visaecommerceroot	18 Jun 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
oistewisekeyglobalrootgbca	18 Jun 2018	0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
mozillacert8.pem	18 Jun 2018	3E:2B:F7:F2:03:1B:96:F3:8C:E6:C4:D8:A8:5D:3E:2D:58:47:6A:0F
comodocertificationauthority	18 Jun 2018	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
cia-crt-g3-02-ca	23 Nov 2016	96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09
verisignc1g6.pem	18 Jun 2018	51:7F:61:1E:29:91:6B:53:82:FB:72:E7:44:D9:8D:C3:CC:53:6D:64
trustcenterclass2caii	21 Apr 2018	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
quovadisrootca1g3	18 Jun 2018	1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67
mozillacert40.pem	18 Jun 2018	80:25:EF:F4:6E:70:C8:D4:72:24:65:84:FE:40:3B:8A:8D:6A:DB:F5
cadisigrootr2	18 Jun 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
cadisigrootr1	18 Jun 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
mozillacert32.pem	18 Jun 2018	60:D6:89:74:B5:C2:65:9E:8A:0F:C1:88:7C:88:D2:46:69:1B:18:2C

Nama	Tanggal	Sidik Jari SHA1
utndatacorpsgcca	21 Apr 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
mozillacert24.pem	18 Jun 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
addtrustclass1ca	21 Apr 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
mozillacert16.pem	18 Jun 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
affirmtrustnetworkingca	21 Apr 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
mozillacert94.pem	18 Jun 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
mozillacert86.pem	18 Jun 2018	74:2C:31:92:E6:07:E4:24:EB:45:49:54:2B:E1:BB:C5:3E:61:74:E2
mozillacert144.pem	18 Jun 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
mozillacert78.pem	18 Jun 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
mozillacert136.pem	18 Jun 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert128.pem	18 Jun 2018	A9:E9:78:08:14:37:58:88:F2:05:19:B0:6D:2B:0D:2B:60:16:90:7D
verisignclass1g2ca	21 Apr 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47

Nama	Tanggal	Sidik Jari SHA1
hellenicacademicandresearchinstitutionsrootca2015	18 Jun 2018	01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
soneraclass1ca	21 Apr 2018	07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
hellenicacademicandresearchinstitutionsrootca2011	18 Jun 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
certumtrustednetworkca2	18 Jun 2018	D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
equifaxsecureca	21 Apr 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
thawteserverca	21 Apr 2018	9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
mozillacert7.pem	18 Jun 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
affirmtrustnetworking	18 Jun 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
deprecateditsecca	27 Jan 2012	12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
globalsignrootcar3	18 Jun 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
globalsignrootcar2	18 Jun 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE

Nama	Tanggal	Sidik Jari SHA1
quovadisrootca	18 Jun 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
mozillacert31.pem	18 Jun 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
entrustrootcertificationauthority	18 Jun 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert23.pem	18 Jun 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
mozillacert15.pem	18 Jun 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
verisignc2g3.pem	18 Jun 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
mozillacert93.pem	18 Jun 2018	31:F1:FD:68:22:63:20:EE:C6:3B:3F:9D:EA:4A:3E:53:7C:7C:39:17
mozillacert151.pem	18 Jun 2018	AC:ED:5F:65:53:FD:25:CE:01:5F:1F:7A:48:3B:6A:74:9F:61:78:C6
mozillacert85.pem	18 Jun 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
certplusclass2primaryca	18 Jun 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
mozillacert143.pem	18 Jun 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
mozillacert77.pem	18 Jun 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert135.pem	18 Jun 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18

Nama	Tanggal	Sidik Jari SHA1
mozillacert69.pem	18 Jun 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
mozillacert127.pem	18 Jun 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
mozillacert119.pem	18 Jun 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
geotrustprimarycag3	21 Apr 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
identrustpublicsectorrootca1	18 Jun 2018	BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
geotrustprimarycag2	21 Apr 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
trustcorrootcertca2	18 Jun 2018	B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
mozillacert6.pem	18 Jun 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
trustcorrootcertca1	18 Jun 2018	FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
networksolutionscertificate authority	18 Jun 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
twcarootcertificationauthority	18 Jun 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
addtrustexternalca	21 Apr 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68

Nama	Tanggal	Sidik Jari SHA1
verisignclass3g5ca	21 Apr 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
autoridaddecertificacionfirmaprofesionalcifa62634068	18 Jun 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
hellenicacademicandresearchinstitutionseccrootca2015	18 Jun 2018	9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
verisigntsaca	21 Apr 2018	20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
utnuserfirsthardwareca	21 Apr 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
identrustcommercialrootca1	18 Jun 2018	DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
dtrustrootclass3ca22009	18 Jun 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
epkirootcertificationauthority	18 Jun 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert30.pem	18 Jun 2018	E7:B4:F6:9D:61:EC:90:69:DB:7E:90:A7:40:1A:3C:F4:7D:4F:E8:EE
teliasonerarootcav1	18 Jun 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
buypassclass3ca	21 Apr 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57

Nama	Tanggal	Sidik Jari SHA1
mozillacert22.pem	18 Jun 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
mozillacert14.pem	18 Jun 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
verisignc2g2.pem	18 Jun 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
certumca	21 Apr 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
mozillacert92.pem	18 Jun 2018	A3:F1:33:3F:E2:42:BF:CF:C5:D1:4E:8F:39:42:98:40:68:10:D1:A0
mozillacert150.pem	18 Jun 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
mozillacert84.pem	18 Jun 2018	D3:C0:63:F2:19:ED:07:3E:34:AD:5D:75:0B:32:76:29:FF:D5:9A:F2
ttelesecglobalroot class3	18 Jun 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
globalsignrootca	18 Jun 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
ttelesecglobalroot class2	18 Jun 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
mozillacert142.pem	18 Jun 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
mozillacert76.pem	18 Jun 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert134.pem	18 Jun 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62

Nama	Tanggal	Sidik Jari SHA1
mozillacert68.pem	18 Jun 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
etugracertificationauthority	18 Jun 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert126.pem	18 Jun 2018	25:01:90:19:CF:FB:D9:99:1C:B7:68:25:74:8D:94:5F:30:93:95:42
keynectisrootca	21 Apr 2018	9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
mozillacert118.pem	18 Jun 2018	7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
quovadisrootca3	18 Jun 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
quovadisrootca2	18 Jun 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
mozillacert5.pem	18 Jun 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
verisignc1g3.pem	18 Jun 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
cybertrustglobalroot	18 Jun 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
amzninternalinfosecag3	27 Feb 2015	B9:B1:CA:38:F7:BF:9C:D2:D4:95:E7:B6:5E:75:32:9B:A8:78:2E:F6
starfieldrootcertificateauthorityg2	18 Jun 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
entrust2048ca	21 Apr 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31

Nama	Tanggal	Sidik Jari SHA1
swisssignsilvercag 2	18 Jun 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25: 93:DF:A7:F0:40:D1:1D:CB
affirmtrustcommerc ial	18 Jun 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80: DC:E9:6E:2C:C7:B2:78:B7
certinomisrootca	18 Jun 2018	9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C: 01:B9:32:C5:34:E7:88:A8
xrampglobalcaroot	18 Jun 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30: 54:F3:4C:52:B7:E5:58:C6
secureglobalca	18 Jun 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46: 1E:3B:9C:C4:5F:F5:3A:1B
swisssigngoldg2ca	21 Apr 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6: 45:25:3A:6F:9F:1A:27:61
mozillacert21.pem	18 Jun 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25: 93:DF:A7:F0:40:D1:1D:CB
mozillacert13.pem	18 Jun 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B: A9:03:D0:06:B7:97:09:91
verisignc2g1.pem	18 Jun 2018	67:82:AA:E0:ED:EE:E2:1A:58:39:D3:C0: CD:14:68:0A:4F:60:14:2A
mozillacert91.pem	18 Jun 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22: 93:D9:DF:F5:4B:81:C0:04
oistewisekeyglobal rootgaca	18 Jun 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39: 6A:46:46:B0:44:1B:0F:A9
mozillacert83.pem	18 Jun 2018	A0:73:E5:C5:BD:43:61:0D:86:4C:21:13: 0A:85:58:57:CC:9C:EA:46
entrustevca	21 Apr 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37: D4:4D:F5:D4:67:49:52:F9

Nama	Tanggal	Sidik Jari SHA1
mozillacert141.pem	18 Jun 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
mozillacert75.pem	18 Jun 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
mozillacert133.pem	18 Jun 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
mozillacert67.pem	18 Jun 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
mozillacert125.pem	18 Jun 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert59.pem	18 Jun 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
thawtepremiumserverca	21 Apr 2018	E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66
mozillacert117.pem	18 Jun 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
utnuserfirstclientauthemailca	21 Apr 2018	B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
entrustrootcag2	21 Apr 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
mozillacert109.pem	18 Jun 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
digicerttrustedrootg4	18 Jun 2018	DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F:C8:3A:4D:7D:77:5D:05:E4
gdroot-g2.pem	18 Jun 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B

Nama	Tanggal	Sidik Jari SHA1
comodoaaaservicesroot	18 Jun 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert4.pem	18 Jun 2018	E3:92:51:2F:0A:CF:F5:05:DF:F6:DE:06:7F:75:37:E1:65:EA:57:4B
verisignclass3publicprimarycertificationauthorityg5	18 Jun 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
chambersofcommerceroot2008	18 Jun 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
verisignclass3publicprimarycertificationauthorityg4	18 Jun 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
verisignclass3publicprimarycertificationauthorityg3	18 Jun 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
thawtepersonalfreemailca	21 Apr 2018	E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
verisignc1g2.pem	18 Jun 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
gtecybertrustglobalca	21 Apr 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
trustcenteruniversalcai	21 Apr 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3

Nama	Tanggal	Sidik Jari SHA1
camerfirmachambers commerceca	21 Apr 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0: DB:72:2E:31:30:61:F0:B1
verisignclass1ca	21 Apr 2018	CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45: 3E:64:09:EA:E8:7D:60:F1

Changelog template pemetaan penyelesai

> **Note**
>
> Kami sekarang terutama mendukung runtime APPSYNC_JS dan dokumentasinya. Harap pertimbangkan untuk menggunakan runtime APPSYNC_JS dan panduannya di sini.

Template penyelesai dan pemetaan fungsi diberi versi. Versi template pemetaan, seperti`2018-05-29`) menentukan hal berikut: * Bentuk yang diharapkan dari konfigurasi permintaan sumber data yang disediakan oleh template permintaan* Perilaku eksekusi template pemetaan permintaan dan template pemetaan respons

Versi direpresentasikan menggunakan format YYYY-MM-DD, di kemudian hari sesuai dengan versi yang lebih baru. Halaman ini mencantumkan perbedaan antara versi template pemetaan yang saat ini didukung. AWS AppSync

Topik

- Ketersediaan Operasi Sumber Data Per Versi Matriks
- Mengubah Versi pada Templat Pemetaan Unit Resolver
- Mengubah Versi pada Fungsi
- 2018-05-29
- 2017-02-28

Ketersediaan Operasi Sumber Data Per Versi Matriks

Operasi/Versi yang Didukung	2017-02-28	2018-05-29
AWS LambdaMemohon	Ya	Ya
AWS Lambda BatchInvoke	Ya	Ya
Tidak ada Datasource	Ya	Ya
Amazon OpenSearch DAPATKAN	Ya	Ya
OpenSearch POS Amazon	Ya	Ya
Amazon OpenSearch PUT	Ya	Ya
Amazon OpenSearch HAPUS	Ya	Ya
Amazon OpenSearch DAPATKAN	Ya	Ya
DynamoDB GetItem	Ya	Ya
Pemindaian DynamoDB	Ya	Ya
Query DynamoDB	Ya	Ya
DynamoDB DeleteItem	Ya	Ya
DynamoDB PutItem	Ya	Ya
DynamoDB BatchGetItem	Tidak	Ya
DynamoDB BatchPutItem	Tidak	Ya
DynamoDB BatchDeleteItem	Tidak	Ya
HTTP	Tidak	Ya
Amazon RDS	Tidak	Ya

Catatan: Hanya versi 2018-05-29 yang saat ini didukung dalam fungsi.

Mengubah Versi pada Templat Pemetaan Unit Resolver

Untuk penyelesai Unit, versi ditentukan sebagai bagian dari isi template pemetaan permintaan. Untuk memperbarui versi, cukup perbarui `version` bidang ke versi baru.

Misalnya, untuk memperbarui versi pada AWS Lambda template:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments": $utils.toJson($context.arguments)
    }
}
```

Anda perlu memperbarui bidang versi dari `2017-02-28` menjadi `2018-05-29` sebagai berikut:

```
{
    "version": "2018-05-29", ## Note the version
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments": $utils.toJson($context.arguments)
    }
}
```

Mengubah Versi pada Fungsi

Untuk fungsi, versi ditentukan sebagai `functionVersion` bidang pada objek fungsi. Untuk memperbarui versi, cukup perbarui`functionVersion`. Catatan: Saat ini, `2018-05-29` hanya didukung untuk fungsi.

Berikut ini adalah contoh perintah CLI untuk memperbarui versi fungsi yang ada:

```
aws appsync update-function \
--api-id REPLACE_WITH_API_ID \
--function-id REPLACE_WITH_FUNCTION_ID \
--data-source-name "PostTable" \
```

```
--function-version "2018-05-29" \
--request-mapping-template "{...}" \
--response-mapping-template "\$util.toJson(\$ctx.result)"
```

Catatan: Disarankan untuk menghilangkan bidang versi dari template pemetaan permintaan fungsi karena tidak akan dihormati. Jika Anda menentukan versi di dalam template pemetaan permintaan fungsi, nilai versi akan diganti dengan nilai bidang. `functionVersion`

2018-05-29

Perubahan Perilaku

- Jika hasil pemanggilan sumber data`null`, template pemetaan respons dijalankan.
- Jika pemanggilan sumber data menghasilkan kesalahan, sekarang terserah Anda untuk menangani kesalahan, hasil evaluasi template pemetaan respons akan selalu ditempatkan di dalam blok respons GraphQL. `data`

Penalaran

- Hasil `null` pemanggilan memiliki arti, dan dalam beberapa kasus penggunaan aplikasi, kami mungkin ingin menangani `null` hasil dengan cara khusus. Misalnya, aplikasi mungkin memeriksa apakah catatan ada di tabel Amazon DynamoDB untuk melakukan beberapa pemeriksaan otorisasi. Dalam hal ini, hasil `null` pemanggilan berarti pengguna mungkin tidak diotorisasi. Menjalankan template pemetaan respons sekarang menyediakan kemampuan untuk memunculkan kesalahan yang tidak sah. Perilaku ini memberikan kontrol yang lebih besar kepada perancang API.

Diberikan template pemetaan respons berikut:

```
$util.toJson($ctx.result)
```

Sebelumnya dengan`2017-02-28`, jika `$ctx.result` kembali null, template pemetaan respons tidak dijalankan. Dengan`2018-05-29`, kita sekarang dapat menangani skenario ini. Misalnya, kita dapat memilih untuk memunculkan kesalahan otorisasi sebagai berikut:

```
# throw an unauthorized error if the result is null
#if ( $util.isNull($ctx.result) )
    $util.unauthorized()
```

```
#end
$util.toJson($ctx.result)
```

Catatan: Kesalahan yang kembali dari sumber data terkadang tidak fatal atau bahkan diharapkan, itulah sebabnya template pemetaan respons harus diberi fleksibilitas untuk menangani kesalahan pemanggilan dan memutuskan apakah akan mengabaikannya, menaikkannya kembali, atau membuang kesalahan yang berbeda.

Diberikan template pemetaan respons berikut:

```
$util.toJson($ctx.result)
```

Sebelumnya, dengan`2017-02-28`, jika terjadi kesalahan pemanggilan, template pemetaan respons dievaluasi dan hasilnya ditempatkan secara otomatis di blok respons `errors` GraphQL. Dengan`2018-05-29`, kita sekarang dapat memilih apa yang harus dilakukan dengan kesalahan, menaikkannya kembali, memunculkan kesalahan yang berbeda, atau menambahkan kesalahan saat mengembalikan data.

Naikkan kembali Kesalahan Pemanggilan

Dalam template respons berikut, kami memunculkan kesalahan yang sama yang kembali dari sumber data.

```
#if ( $ctx.error )
    $util.error($ctx.error.message, $ctx.error.type)
#end
$util.toJson($ctx.result)
```

Jika terjadi kesalahan pemanggilan (misalnya, `$ctx.error` ada) responsnya terlihat seperti berikut:

```
{
    "data": {
        "getPost": null
    },
    "errors": [
        {
            "path": [
                "getPost"
            ],
            "errorType": "DynamoDB:ConditionalCheckFailedException",
            "message": "Conditional check failed exception...",
```

```
            "locations": [
                {
                    "line": 5,
                    "column": 5
                }
            ]
        }
    ]
}
```

Naikkan Kesalahan yang Berbeda

Dalam template respons berikut, kami memunculkan kesalahan kustom kami sendiri setelah memproses kesalahan yang kembali dari sumber data.

```
#if ( $ctx.error )
    #if ( $ctx.error.type.equals("ConditionalCheckFailedException") )
        ## we choose here to change the type and message of the error for
 ConditionalCheckFailedExceptions
        $util.error("Error while updating the post, try again. Error:
 $ctx.error.message", "UpdateError")
    #else
        $util.error($ctx.error.message, $ctx.error.type)
    #end
#end
$util.toJson($ctx.result)
```

Jika terjadi kesalahan pemanggilan (misalnya, `$ctx.error` ada) responsnya terlihat seperti berikut:

```
{
    "data": {
        "getPost": null
    },
    "errors": [
        {
            "path": [
                "getPost"
            ],
            "errorType": "UpdateError",
            "message": "Error while updating the post, try again. Error: Conditional
 check failed exception...",
            "locations": [
```

```
                {
                    "line": 5,
                    "column": 5
                }
            ]
        }
    ]
}
```

Menambahkan Kesalahan untuk Mengembalikan Data

Dalam template respons berikut, kami menambahkan kesalahan yang sama yang kembali dari sumber data saat mengembalikan data kembali ke dalam respons. Hal ini juga dikenal sebagai respon parsial.

```
#if ( $ctx.error )
    $util.appendError($ctx.error.message, $ctx.error.type)
    #set($defaultPost = {id: "1", title: 'default post'})
    $util.toJson($defaultPost)
#else
    $util.toJson($ctx.result)
#end
```

Jika terjadi kesalahan pemanggilan (misalnya, `$ctx.error` ada) responsnya terlihat seperti berikut:

```
{
    "data": {
        "getPost": {
            "id": "1",
            "title: "A post"
        }
    },
    "errors": [
        {
            "path": [
                "getPost"
            ],
            "errorType": "ConditionalCheckFailedException",
            "message": "Conditional check failed exception...",
            "locations": [
                {
                    "line": 5,
```

```
                    "column": 5
                }
            ]
        }
    ]
}
```

Bermigrasi dari 2017-02-28 ke 2018-05-29

Migrasi dari 2017-02-28 ke 2018-05-29 sangat mudah. Ubah bidang versi pada template pemetaan permintaan resolver atau pada objek versi fungsi. Namun, perhatikan bahwa eksekusi 2018-05-29 berperilaku berbeda dari 2017-02-28, perubahan diuraikan di sini.

Mempertahankan perilaku eksekusi yang sama dari 2017-02-28 hingga 2018-05-29

Dalam beberapa kasus, dimungkinkan untuk mempertahankan perilaku eksekusi yang sama dengan versi 2017-02-28 saat menjalankan template berversi 2018-05-29.

Contoh: DynamoDB PutItem

Diberikan sebagai berikut 2017-02-28 PutItem DynamoDB permintaan template:

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key": {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "attributeValues" : {
        "baz" : ... typed value
    },
    "condition" : {
       ...
    }
}
```

Dan template respons berikut:

```
$util.toJson($ctx.result)
```

Migrasi ke 2018-05-29 mengubah template ini sebagai berikut:

```
{
    "version" : "2018-05-29", ## Note the new 2018-05-29 version
    "operation" : "PutItem",
    "key": {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "attributeValues" : {
        "baz" : ... typed value
    },
    "condition" : {
       ...
    }
}
```

Dan mengubah template respons sebagai berikut:

```
## If there is a datasource invocation error, we choose to raise the same error
## the field data will be set to null.
#if($ctx.error)
  $util.error($ctx.error.message, $ctx.error.type, $ctx.result)
#end

## If the data source invocation is null, we return null.
#if($util.isNull($ctx.result))
  #return
#end

$util.toJson($ctx.result)
```

Sekarang tanggung jawab Anda untuk menangani kesalahan, kami memilih untuk memunculkan kesalahan yang sama menggunakan `$util.error()` yang dikembalikan dari DynamoDB. Anda dapat menyesuaikan cuplikan ini untuk mengonversi templat pemetaan Anda ke 2018-05-29, perhatikan bahwa jika templat respons Anda berbeda, Anda harus memperhitungkan perubahan perilaku eksekusi.

Contoh: DynamoDB GetItem

Diberikan sebagai berikut 2017-02-28 GetItem DynamoDB permintaan template:

```
{
```

```
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "consistentRead" : true
}
```

Dan template respons berikut:

```
## map table attribute postId to field Post.id
$util.qr($ctx.result.put("id", $ctx.result.get("postId")))

$util.toJson($ctx.result)
```

Migrasi ke 2018-05-29 mengubah template ini sebagai berikut:

```
{
    "version" : "2018-05-29", ## Note the new 2018-05-29 version
    "operation" : "GetItem",
    "key" : {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "consistentRead" : true
}
```

Dan mengubah template respons sebagai berikut:

```
## If there is a datasource invocation error, we choose to raise the same error
#if($ctx.error)
  $util.error($ctx.error.message, $ctx.error.type)
#end

## If the data source invocation is null, we return null.
#if($util.isNull($ctx.result))
  #return
#end

## map table attribute postId to field Post.id
$util.qr($ctx.result.put("id", $ctx.result.get("postId")))
```

```
$util.toJson($ctx.result)
```

Dalam versi 2017-02-28, jika pemanggilan sumber data adalah, `null` artinya tidak ada item dalam tabel DynamoDB yang cocok dengan kunci kami, template pemetaan respons tidak akan dijalankan. Mungkin baik-baik saja untuk sebagian besar kasus, tetapi jika Anda `$ctx.result` mengharapkannya tidak`null`, Anda sekarang harus menangani skenario itu.

2017-02-28

Karakteristik

- Jika hasil pemanggilan sumber data adalah**`null`**, template pemetaan respons tidak dijalankan.
- Jika pemanggilan sumber data menghasilkan kesalahan, template pemetaan respons dijalankan dan hasil yang dievaluasi ditempatkan di dalam blok respons GraphQL. `errors.data`

Jenis referensi

Bagian ini digunakan sebagai referensi untuk jenis skema.

Jenis skalar diAWS AppSync

Jenis objek GraphQL memiliki nama dan bidang, dan bidang tersebut dapat memiliki sub-bidang. Pada akhirnya, bidang tipe objek harus diselesaikanskalarjenis, yang mewakili daun kueri. Untuk informasi selengkapnya tentang jenis objek dan skalar, lihatSkema dan tipedi situs web GraphQL.

Selain set default skalar GraphQL,AWS AppSyncjuga memungkinkan Anda menggunakanditetapkan layananskalar yang dimulai denganAWSawalan.AWS AppSynctidak mendukung penciptaanditentukan pengguna(kustom) skalar. Anda harus menggunakan default atauAWSskalar.

Anda tidak dapat menggunakanAWSsebagai awalan untuk jenis objek kustom.

Bagian berikut adalah referensi untuk pengetikan skema.

Skalar default

GraphQL mendefinisikan skalar default berikut:

Daftar skalar default

`ID`

Pengenal unik untuk suatu objek. Skalar ini diserialisasikan seperti a`String`tetapi tidak dimaksudkan untuk dapat dibaca manusia.

`String`

Urutan karakter UTF-8.

`Int`

Nilai integer antara - (2^{31}) dan 2^{31}-1.

`Float`

Nilai floating point IEEE 754.

`Boolean`

Nilai Boolean, juga`true`atau`false`.

AWS AppSyncskalar

AWS AppSyncmendefinisikan skalar berikut:

AWS AppSyncdaftar skalar

`AWSDate`

DiperpanjangTanggal ISO 8601string dalam format`YYYY-MM-DD`.

`AWSTime`

DiperpanjangISO 8601 kalistring dalam format`hh:mm:ss.sss`.

`AWSDateTime`

DiperpanjangISO 8601 tanggal dan waktustring dalam format`YYYY-MM-DDThh:mm:ss.sssZ`.

> **Note**
>
> The`AWSDate`,`AWSTime`, dan`AWSDateTime`skalar secara opsional dapat mencakup a*zona waktu offset*. Misalnya, nilainya`1970-01-01Z`,`1970-01-01-07:00`, dan`1970-01-01+05:30`Semua berlaku untuk`AWSDate`. Offset zona waktu harus berupa`Z`(UTC) atau offset dalam jam dan menit (dan, opsional, detik). Sebagai contoh, `±hh:mm:ss`. Bidang detik di offset zona waktu dianggap valid meskipun bukan bagian dari standar ISO 8601.

`AWSTimestamp`

Nilai integer yang mewakili jumlah detik sebelum atau sesudah`1970-01-01-T00:00Z`.

`AWSEmail`

Alamat email dalam format`local-part@domain-part`sebagaimana didefinisikan olehRFC 822.

`AWSJSON`

Sebuah string JSON. Setiap konstruksi JSON yang valid secara otomatis diurai dan dimuat dalam kode resolver sebagai peta, daftar, atau nilai skalar daripada sebagai string input literal. String yang tidak dikutip atau JSON yang tidak valid menghasilkan kesalahan validasi GraphQL.

`AWSPhone`

Nomor telepon. Nilai ini disimpan sebagai string. Nomor telepon dapat berisi spasi atau tanda hubung untuk memisahkan grup digit. Nomor telepon tanpa kode negara diasumsikan sebagai nomor AS/Amerika Utara yang mengikutiRencana Penomoran Amerika Utara (NANP).

`AWSURL`

URL seperti yang didefinisikan olehRFC 1738. Misalnya, `https://www.amazon.com/dp/B000NZW3KC/` atau `mailto:example@example.com`. URL harus berisi skema (`http`,`mailto`) dan tidak dapat berisi dua garis miring ke depan (`//`) di bagian jalan.

`AWSIPAddress`

Alamat IPv4 atau IPv6 yang valid. Alamat IPv4 diharapkan dalam notasi quad-dotted (`123.12.34.56`). Alamat IPv6 diharapkan dalam format non-kurung, dipisahkan titik dua (`1a2b:3c4b::1234:4567`). Anda dapat menyertakan akhiran CIDR opsional (`123.45.67.89/16`) untuk menunjukkan subnet mask.

Contoh penggunaan skema

Contoh berikut skema GraphQL menggunakan semua skalar kustom sebagai “objek” dan menunjukkan permintaan resolver dan template respons untuk operasi put, get, dan list dasar. Terakhir, contoh menunjukkan bagaimana Anda dapat menggunakan ini saat menjalankan kueri dan mutasi.

```
type Mutation {
    putObject(
        email: AWSEmail,
        json: AWSJSON,
        date: AWSDate,
        time: AWSTime,
        datetime: AWSDateTime,
        timestamp: AWSTimestamp,
        url: AWSURL,
        phoneno: AWSPhone,
        ip: AWSIPAddress
    ): Object
}

type Object {
    id: ID!
```

```
    email: AWSEmail
    json: AWSJSON
    date: AWSDate
    time: AWSTime
    datetime: AWSDateTime
    timestamp: AWSTimestamp
    url: AWSURL
    phoneno: AWSPhone
    ip: AWSIPAddress
}

type Query {
    getObject(id: ID!): Object
    listObjects: [Object]
}

schema {
    query: Query
    mutation: Mutation
}
```

Inilah yang menjadi template permintaan`putObject`mungkin terlihat seperti. SEBUAH`putObject`menggunakan a`PutItem`operasi untuk membuat atau memperbarui item di tabel Amazon DynamoDB Anda. Perhatikan bahwa cuplikan kode ini tidak memiliki tabel Amazon DynamoDB yang dikonfigurasi sebagai sumber data. Ini hanya digunakan sebagai contoh:

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key" : {
        "id": $util.dynamodb.toDynamoDBJson($util.autoId()),
    },
    "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
}
```

Template respon untuk`putObject`mengembalikan hasil:

```
$util.toJson($ctx.result)
```

Inilah yang menjadi template permintaan`getObject`mungkin terlihat seperti. SEBUAH`getObject`menggunakan a`GetItem`operasi untuk mengembalikan satu set atribut untuk

item yang diberikan kunci utama. Perhatikan bahwa cuplikan kode ini tidak memiliki tabel Amazon DynamoDB yang dikonfigurasi sebagai sumber data. Ini hanya digunakan sebagai contoh:

```
{
    "version": "2017-02-28",
    "operation": "GetItem",
    "key": {
        "id": $util.dynamodb.toDynamoDBJson($ctx.args.id),
    }
}
```

Template respon untuk`getObject`mengembalikan hasil:

```
$util.toJson($ctx.result)
```

Inilah yang menjadi template permintaan`listObjects`mungkin terlihat seperti. SEBUAH`listObjects`menggunakan a`Scan`operasi untuk mengembalikan satu atau lebih item dan atribut. Perhatikan bahwa cuplikan kode ini tidak memiliki tabel Amazon DynamoDB yang dikonfigurasi sebagai sumber data. Ini hanya digunakan sebagai contoh:

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
}
```

Template respon untuk`listObjects`mengembalikan hasil:

```
$util.toJson($ctx.result.items)
```

Berikut ini adalah beberapa contoh penggunaan skema ini dengan kueri GraphQL:

```
mutation CreateObject {
    putObject(email: "example@example.com"
        json: "{\"a\":1, \"b\":3, \"string\": 234}"
        date: "1970-01-01Z"
        time: "12:00:34."
        datetime: "1930-01-01T16:00:00-07:00"
        timestamp: -123123
        url:"https://amazon.com"
        phoneno: "+1 555 764 4377"
```

```
        ip: "127.0.0.1/8"
    ) {
        id
        email
        json
        date
        time
        datetime
        url
        timestamp
        phoneno
        ip
    }
}

query getObject {
    getObject(id:"0d97daf0-48e6-4ffc-8d48-0537e8a843d2"){
        email
        url
        timestamp
        phoneno
        ip
    }
}

query listObjects {
    listObjects {
        json
        date
        time
        datetime
    }
}
```

Antarmuka dan serikat pekerja di GraphQL

Sistem tipe GraphQL mendukungAntarmuka. Antarmuka memperlihatkan sekumpulan bidang tertentu yang harus disertakan tipe untuk mengimplementasikan antarmuka.

Sistem tipe GraphQL juga mendukungSerikat pekerja. Serikat pekerja identik dengan antarmuka, kecuali bahwa mereka tidak mendefinisikan sekumpulan bidang yang umum. Serikat pekerja umumnya lebih disukai daripada antarmuka ketika tipe yang mungkin tidak berbagi hierarki logis.

Bagian berikut adalah referensi untuk pengetikan skema.

Contoh antarmuka

Kita bisa mewakili sebuah`Event`antarmuka yang mewakili segala jenis aktivitas atau pengumpulan orang. Beberapa jenis acara yang mungkin`Concert`,`Conference`, dan`Festival`. Semua jenis ini memiliki karakteristik umum, termasuk nama, tempat di mana acara berlangsung, dan tanggal mulai dan berakhir. Tipe-tipe ini juga memiliki perbedaan; a`Conference`menawarkan daftar pembicara dan lokakarya, sementara`Concert`menampilkan band pertunjukan.

Dalam Schema Definition Language (SDL),`Event`antarmuka didefinisikan sebagai berikut:

```
interface Event {
        id: ID!
        name : String!
        startsAt: String
        endsAt: String
        venue: Venue
        minAgeRestriction: Int
}
```

Dan masing-masing jenis mengimplementasikan`Event`antarmuka sebagai berikut:

```
type Concert implements Event {
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
    minAgeRestriction: Int
    performingBand: String
}

type Festival implements Event {
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
    minAgeRestriction: Int
    performers: [String]
}
```

```
type Conference implements Event {
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
    minAgeRestriction: Int
    speakers: [String]
    workshops: [String]
}
```

Antarmuka berguna untuk mewakili elemen yang mungkin dari beberapa jenis. Misalnya, kami dapat mencari semua acara yang terjadi di tempat tertentu. Mari kita tambahkan`findEventsByVenue`bidang ke skema sebagai berikut:

```
schema {
    query: Query
}

type Query {
    # Retrieve Events at a specific Venue
    findEventsAtVenue(venueId: ID!): [Event]
}

type Venue {
    id: ID!
    name: String
    address: String
    maxOccupancy: Int
}

type Concert implements Event {
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
    minAgeRestriction: Int
    performingBand: String
}

interface Event {
```

```
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
    minAgeRestriction: Int
}

type Festival implements Event {
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
    minAgeRestriction: Int
    performers: [String]
}

type Conference implements Event {
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
    minAgeRestriction: Int
    speakers: [String]
    workshops: [String]
}
```

The`findEventsByVenue`mengembalikan daftar`Event`. Karena bidang antarmuka GraphQL umum untuk semua jenis implementasi, dimungkinkan untuk memilih bidang apa pun di`Event`antarmuka (`id`,`name`,`startsAt`,`endsAt`,`venue`, dan`minAgeRestriction`). Selain itu, Anda dapat mengakses bidang pada jenis implementasi apa pun menggunakan GraphQLpecahan, selama Anda menentukan jenisnya.

Mari kita periksa contoh query GraphQL yang menggunakan antarmuka.

```
query {
  findEventsAtVenue(venueId: "Madison Square Garden") {
    id
    name
    minAgeRestriction
    startsAt
```

```
    ... on Festival {
      performers
    }

    ... on Concert {
      performingBand
    }

    ... on Conference {
      speakers
      workshops
    }
  }
}
```

Kueri sebelumnya menghasilkan satu daftar hasil, dan server dapat mengurutkan peristiwa berdasarkan tanggal mulai secara default.

```
{
  "data": {
    "findEventsAtVenue": [
      {
        "id": "Festival-2",
        "name": "Festival 2",
        "minAgeRestriction": 21,
        "startsAt": "2018-10-05T14:48:00.000Z",
        "performers": [
          "The Singers",
          "The Screamers"
        ]
      },
      {
        "id": "Concert-3",
        "name": "Concert 3",
        "minAgeRestriction": 18,
        "startsAt": "2018-10-07T14:48:00.000Z",
        "performingBand": "The Jumpers"
      },
      {
        "id": "Conference-4",
        "name": "Conference 4",
        "minAgeRestriction": null,
```

```
        "startsAt": "2018-10-09T14:48:00.000Z",
        "speakers": [
          "The Storytellers"
        ],
        "workshops": [
          "Writing",
          "Reading"
        ]
      }
    ]
  }
}
```

Karena hasil dikembalikan sebagai kumpulan peristiwa tunggal, menggunakan antarmuka untuk mewakili karakteristik umum sangat membantu untuk menyortir hasil.

Contoh serikat

Seperti yang dinyatakan sebelumnya, serikat pekerja tidak mendefinisikan kumpulan bidang umum. Hasil pencarian mungkin mewakili berbagai jenis. Menggunakan`Events`skema, Anda dapat mendefinisikan`SearchResult`serikat sebagai berikut:

```
type Query {
    # Retrieve Events at a specific Venue
    findEventsAtVenue(venueId: ID!): [Event]
    # Search across all content
    search(query: String!): [SearchResult]
}

union SearchResult = Conference | Festival | Concert | Venue
```

Dalam hal ini, untuk menanyakan bidang apa pun di`SearchResult`serikat pekerja, Anda harus menggunakan fragmen:

```
query {
  search(query: "Madison") {
    ... on Venue {
      id
      name
      address
    }
```

```
    ... on Festival {
      id
      name
      performers
    }

    ... on Concert {
      id
      name
      performingBand
    }

    ... on Conference {
      speakers
      workshops
    }
  }
}
```

Ketik resolusi diAWS AppSync

Resolusi tipe adalah mekanisme dimana mesin GraphQL mengidentifikasi nilai yang diselesaikan sebagai jenis objek tertentu.

Kembali ke contoh pencarian serikat pekerja, asalkan kueri kami menghasilkan hasil, setiap item dalam daftar hasil harus menampilkan dirinya sebagai salah satu jenis yang mungkin`SearchResult`serikat didefinisikan (yaitu,`Conference`,`Festival`,`Concert`, atau`Venue`).

Karena logika untuk mengidentifikasi`Festival`dari a`Venue`atau`Conference`tergantung pada persyaratan aplikasi, mesin GraphQL harus diberi petunjuk untuk mengidentifikasi kemungkinan jenis kami dari hasil mentah.

denganAWS AppSync, petunjuk ini diwakili oleh bidang meta bernama`__typename`, yang nilainya sesuai dengan nama tipe objek yang diidentifikasi.`__typename`diperlukan untuk tipe pengembalian yang merupakan antarmuka atau serikat pekerja.

Contoh resolusi tipe

Mari kita gunakan kembali skema sebelumnya. Anda dapat mengikuti dengan menavigasi ke konsol dan menambahkan yang berikut di bawah`Skema`halaman:

```
schema {
```

```
    query: Query
}

type Query {
    # Retrieve Events at a specific Venue
    findEventsAtVenue(venueId: ID!): [Event]
    # Search across all content
    search(query: String!): [SearchResult]
}

union SearchResult = Conference | Festival | Concert | Venue

type Venue {
    id: ID!
    name: String!
    address: String
    maxOccupancy: Int
}

interface Event {
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
    minAgeRestriction: Int
}

type Festival implements Event {
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
    minAgeRestriction: Int
    performers: [String]
}

type Conference implements Event {
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
```

```
    minAgeRestriction: Int
    speakers: [String]
    workshops: [String]
}

type Concert implements Event {
    id: ID!
    name: String!
    startsAt: String
    endsAt: String
    venue: Venue
    minAgeRestriction: Int
    performingBand: String
}
```

Mari kita lampirkan resolver ke`Query.search`lapangan. Di`Resolvers`bagian, pilihLampirkan, buat yang baruSumber Datadari jenisTIDAK ADA, dan kemudian beri namaStubDataSource. Demi contoh ini, kita akan berpura-pura kita mengambil hasil dari sumber eksternal, dan kode keras hasil yang diambil dalam template pemetaan permintaan.

Di panel templat pemetaan permintaan, masukkan yang berikut ini:

```
{
    "version" : "2018-05-29",
    "payload":
    ## We are effectively mocking our search results for this example
    [
        {
            "id": "Venue-1",
            "name": "Venue 1",
            "address": "2121 7th Ave, Seattle, WA 98121",
            "maxOccupancy": 1000
        },
        {
            "id": "Festival-2",
            "name": "Festival 2",
            "performers": ["The Singers", "The Screamers"]
        },
        {
            "id": "Concert-3",
            "name": "Concert 3",
            "performingBand": "The Jumpers"
        },
```

```
        {
            "id": "Conference-4",
            "name": "Conference 4",
            "speakers": ["The Storytellers"],
            "workshops": ["Writing", "Reading"]
        }
    ]
}
```

Jika aplikasi mengembalikan nama tipe sebagai bagian dari`id`bidang, logika resolusi tipe harus mengurai`id`bidang untuk mengekstrak nama tipe dan kemudian menambahkan`__typename`bidang untuk setiap hasil. Anda dapat melakukan logika itu di template pemetaan respons sebagai berikut:

> **Note**
>
> Anda juga dapat melakukan tugas ini sebagai bagian dari fungsi Lambda Anda, jika Anda menggunakan sumber data Lambda.

```
#foreach ($result in $context.result)
    ## Extract type name from the id field.
    #set( $typeName = $result.id.split("-")[0] )
    #set( $ignore = $result.put("__typename", $typeName))
#end
$util.toJson($context.result)
```

Jalankan kueri berikut:

```
query {
  search(query: "Madison") {
    ... on Venue {
      id
      name
      address
    }

    ... on Festival {
        id
      name
      performers
    }
```

```
    ... on Concert {
      id
      name
      performingBand
    }

    ... on Conference {
      speakers
      workshops
    }
  }
}
```

Kueri menghasilkan hasil sebagai berikut:

```
{
  "data": {
    "search": [
      {
        "id": "Venue-1",
        "name": "Venue 1",
        "address": "2121 7th Ave, Seattle, WA 98121"
      },
      {
        "id": "Festival-2",
        "name": "Festival 2",
        "performers": [
          "The Singers",
          "The Screamers"
        ]
      },
      {
        "id": "Concert-3",
        "name": "Concert 3",
        "performingBand": "The Jumpers"
      },
      {
        "speakers": [
          "The Storytellers"
        ],
        "workshops": [
          "Writing",
```

```
          "Reading"
        ]
      }
    ]
  }
}
```

Logika resolusi tipe bervariasi tergantung pada aplikasi. Misalnya, Anda dapat memiliki logika pengenal berbeda yang memeriksa keberadaan bidang tertentu atau bahkan kombinasi bidang. Artinya, Anda bisa mendeteksi keberadaan`performers`bidang untuk mengidentifikasi`Festival`atau kombinasi dari`speakers`dan`workshops`bidang untuk mengidentifikasi`Conference`. Pada akhirnya, terserah Anda untuk menentukan logika yang ingin Anda gunakan.

Pemecahan Masalah dan Kesalahan Umum

Bagian ini membahas beberapa kesalahan umum dan cara memecahkan masalah mereka.

Pemetaan Kunci DynamoDB Salah

Jika operasi GraphQL Anda mengembalikan pesan galat berikut, itu mungkin karena struktur template pemetaan permintaan Anda tidak cocok dengan struktur kunci Amazon DynamoDB:

```
The provided key element does not match the schema (Service: AmazonDynamoDBv2; Status
 Code: 400; Error Code
```

Misalnya, jika tabel DynamoDB Anda memiliki kunci hash yang `"id"` dipanggil dan template Anda `"PostID"` mengatakan, seperti pada contoh berikut, ini menghasilkan kesalahan sebelumnya, karena tidak cocok. `"id"` `"PostID"`

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "PostID" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    }
}
```

Resolver Hilang

Jika Anda menjalankan operasi GraphQL, seperti kueri, dan mendapatkan respons nol, ini mungkin karena Anda tidak memiliki resolver yang dikonfigurasi.

Misalnya, jika Anda mengimpor skema yang mendefinisikan `getCustomer(userId: ID!):` bidang, dan Anda belum mengonfigurasi resolver untuk bidang ini, maka ketika Anda menjalankan kueri seperti`getCustomer(userId:"ID123"){...}`, Anda akan mendapatkan respons seperti berikut:

```
{
    "data": {
    "getCustomer": null
```

```
    }
}
```

Kesalahan Template Pemetaan

Jika template pemetaan Anda tidak dikonfigurasi dengan benar, Anda akan menerima respons GraphQL yang ada. `errorType` `MappingTemplate` `message`Bidang harus menunjukkan di mana masalahnya ada di template pemetaan Anda.

Misalnya, jika Anda tidak memiliki `operation` bidang dalam templat pemetaan permintaan, atau jika nama `operation` bidang salah, Anda akan mendapatkan respons seperti berikut:

```
{
    "data": {
        "searchPosts": null
    },
    "errors": [
        {
        "path": [
            "searchPosts"
        ],
        "errorType": "MappingTemplate",
        "locations": [
            {
            "line": 2,
            "column": 3
            }
        ],
        "message": "Value for field '$[operation]' not found."
        }
    ]
}
```

Jenis Pengembalian Salah

Jenis pengembalian dari sumber data Anda harus cocok dengan tipe objek yang ditentukan dalam skema Anda, jika tidak, Anda mungkin melihat kesalahan GraphQL seperti:

```
"errors": [
    {
```

```
    "path": [
        "posts"
    ],
    "locations": null,
    "message": "Can't resolve value (/posts) : type mismatch error, expected type LIST,
 got OBJECT"
    }
]
```

Misalnya ini dapat terjadi dengan definisi kueri berikut:

```
type Query {
    posts: [Post]
}
```

Yang mengharapkan DAFTAR `[Posts]` objek. Misalnya jika Anda memiliki fungsi Lambda di Node.JS dengan sesuatu seperti berikut:

```
const result = { data: data.Items.map(item => { return item ; }) };
callback(err, result);
```

Ini akan menimbulkan kesalahan seperti `result` objek. Anda perlu mengubah panggilan balik ke `result.data` atau mengubah skema Anda untuk tidak mengembalikan LIST.

Memproses permintaan yang tidak valid

Ketika AWS AppSync tidak dapat memproses dan mengirim permintaan (karena data yang tidak benar seperti sintaks tidak valid) ke penyelesai bidang, payload respons akan mengembalikan data bidang dengan nilai yang disetel ke dan kesalahan yang relevan. `null`

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