Guía para desarrolladores

AWS AppSync

AWS AppSync: Guía para desarrolladores

Table of Contents

¿Qué es AWS AppSync? 1
Características de AWS AppSync 1
¿Es la primera vez que usa AWS AppSync? 2
Servicios relacionados 2
Precios de AWS AppSync 2
GraphQL y arquitectura de AWS AppSync 4
¿Qué es una API? 5
Clientes 5
Recursos 5
¿Qué es REST? 6
Interfaz uniforme 6
Ausencia de estado 7
Sistema por capas 7
Capacidad de almacenamiento en caché 7
¿Qué es una API RESTful? 7
¿Cómo funcionan las API RESTful? 8
¿Por qué usar GraphQL en lugar de REST? 8
Componentes de una API de GraphQL 10
Esquemas 11
Orígenes de datos 29
Solucionadores 46
Propiedades adicionales de GraphQL 56
Declarativo 56
Jerárquico 57
Introspectivo 58
Tipado fuerte 59
Introducción: Cómo crear su primera API de GraphQL 60
Paso 1: lanzar un esquema 61
Paso 2: explorar la consola 65
Diseñador del esquema 65
Orígenes de datos 67
Consultas 67
Configuración 67
Paso 3: agregar datos con una mutación de GraphQL 68

Paso 4: recuperar datos con una consulta de GraphQL 73
Secciones complementarias 76
Integration 76
Lectura complementaria 77
Diseño de API de GraphQL 78
Estructuración de una API de GraphQL (API en blanco o importadas) 78
Paso 1: diseño del esquema 79
Paso 2: Asociar un origen de datos 108
Paso 3: Configurar solucionadores 120
Paso 4: Uso de una API: ejemplo de CDK 178
Datos en tiempo real 196
Directivas de suscripción del esquema de GraphQL 196
Uso de argumentos de suscripción 199
Creación de API pub/sub genéricas con tecnología de WebSockets sin servidor 203
Filtrado de suscripciones mejorado 206
Cancelación de la suscripción de las conexiones 218
Creación de un cliente WebSocket en tiempo real 222
API fusionadas 238
API fusionadas y federación 240
Resolución de conflictos de la API fusionada 242
Configuración de esquemas 250
Configuración de modos de autorización 250
Configuración de roles de ejecución 251
Configuración de API fusionadas entre cuentas mediante AWS RAM 253
Fusión 255
Asistencia adicional para las API fusionadas 256
Limitaciones de las API fusionadas 257
Creación de API fusionadas 257
Introspección de RDS 259
Uso de la característica de introspección (consola) 260
Uso de la característica de introspección (API) 264
Creación de una aplicación cliente 267
Tutoriales de solucionadores (JavaScript) 270
Tutorial: Solucionadores de JavaScript de DynamoDB 270
Creación de la API de GraphQL 271
Definición de una API de publicación básica 271

Configuración de la tabla de Amazon DynamoDB 272
Configuración del solucionador addPost (PutItem de Amazon DynamoDB) 273
Configuración del solucionador getPost (GetItem de Amazon DynamoDB) 276
Creación de una mutación updatePost (UpdateItem de Amazon DynamoDB) 279
Cree mutaciones de votación (UpdateItem de Amazon DynamoDB) 284
Configuración de un solucionador deletePost (DeleteItem de Amazon DynamoDB) 287
Configuración de un solucionador allPost (Scan de Amazon DynamoDB) 293
Configuración de un solucionador allPostsByAuthor (Query de Amazon DynamoDB) 298
Uso de conjuntos 303
Conclusión 310
Tutorial: Solucionadores de Lambda 310
Creación de una función de Lambda 310
Configure un origen de datos para Lambda 312
Cree un esquema de GraphQL 313
Configure solucionadores 121
Pruebe la API de GraphQL 315
Devolución de errores 316
Caso de uso avanzado: agrupación en lotes 319
Tutorial: Solucionadores locales 329
Creación de la aplicación pub/sub 329
Envíe y suscríbase a mensajes 330
Tutorial: Combinación de solucionadores de GraphQL 331
Esquema de ejemplo 332
Modificación de datos mediante solucionadores 333
DynamoDB y OpenSearch Service 333
Tutorial: Solucionadores de Amazon OpenSearch Service 336
Cree un nuevo dominio de OpenSearch Service 336
Configure un origen de datos para OpenSearch Service 337
Conexión de un solucionador 339
Modificación de las búsquedas 340
Adición de datos a OpenSearch Service 342
Recuperación de un solo documento 343
Ejecución de consultas y mutaciones 344
Prácticas recomendadas 344
Tutorial: Solucionadores de transacciones de DynamoDB 345
Permisos 345

Origen de datos 346
Transacciones 347
Tutorial: Solucionadores por lotes de DynamoDB 355
Lotes en una única tabla 355
Lotes en varias tablas 360
Control de errores 368
Tutorial: Solucionadores de HTTP 374
Creación de una API de REST 374
Creación de la API de GraphQL 375
Creación de un esquema de GraphQL 375
Configure el origen de datos HTTP 376
Configuración de los solucionadores 155
Invocación de servicios de AWS 380
Tutorial: Aurora PostgreSQL con API de datos 382
Creación de clústeres 382
Habilitación de la API de datos 383
Creación de la base de datos y la tabla 383
Creación de un esquema de GraphQL 384
Solucionadores para RDS 386
Eliminación de su clúster 394
Tutoriales de solucionadores (VTL) 395
Tutorial: Solucionadores de DynamoDB 396
Configuración de las tablas de DynamoDB 396
Creación de la API de GraphQL 375
Definición de una API de publicación básica 398
Configuración del origen de datos para las tablas de DynamoDB 399
Configuración del solucionador addPost (PutItem de DynamoDB) 400
Configuración del solucionador getPost (GetItem de DynamoDB) 405
Cree una mutación updatePost (UpdateItem de DynamoDB) 408
Modificación del solucionador updatePost (UpdateItem en DynamoDB) 412
Cree mutaciones upvotePost y downvotePost (UpdateItem de DynamoDB) 418
Configuración de un solucionador deletePost (DeleteItem en DynamoDB) 422
Configuración del solucionador allPost (Scan en DynamoDB) 429
Configuración del solucionador allPostsByAuthor (Query de DynamoDB) 434
Uso de conjuntos 303
Uso de listas y mapas 448

Conclusión 451
Tutorial: Solucionadores de Lambda 452
Creación de una función de Lambda 452
Configure un origen de datos para Lambda 454
Cree un esquema de GraphQL 375
Configure solucionadores 155
Pruebe la API de GraphQL 458
Devolución de errores 460
Caso de uso avanzado: agrupación en lotes 462
Tutorial: Solucionadores de Amazon OpenSearch Service 473
Configuración en un clic 473
Cree un nuevo dominio de OpenSearch Service 473
Configure el origen de datos para OpenSearch Service 474
Conexión de un solucionador 476
Modificación de las búsquedas 478
Adición de datos a OpenSearch Service 479
Recuperación de un solo documento 480
Ejecución de consultas y mutaciones 480
Prácticas recomendadas 481
Tutorial: Solucionadores locales 482
Creación de la aplicación buscapersonas 482
Envío y suscripción a notificaciones 483
Tutorial: Combinación de solucionadores de GraphQL 484
Esquema de ejemplo 485
Modificación de datos mediante solucionadores 486
DynamoDB y OpenSearch Service 487
Tutorial: Solucionadores por lotes de DynamoDB 491
Permisos 491
Origen de datos 492
Lotes en una única tabla 493
Lotes en varias tablas 497
Control de errores 504
Tutorial: Solucionadores de transacciones de DynamoDB 510
Permisos 491
Origen de datos 492
Transacciones 513

Tutorial: Solucionadores de HTTP 522
Configuración en un clic 473
Creación de una API de REST 374
Creación de la API de GraphQL 375
Creación de un esquema de GraphQL 375
Configure el origen de datos HTTP 376
Configuración de solucionadores 155
Invocación de servicios de AWS 528
Tutorial: Aurora sin servidor 530
Crear un clúster 530
Habilitar la API de datos 383
Creación de una base de datos y tabla 531
Esquema de GraphQL 532
Configuración de solucionadores 155
Ejecutar mutaciones 538
Ejecutar consultas 539
Saneamiento de la entrada 540
Tutorial: Solucionadores de canalizaciones 542
Configuración en un clic 473
Configuración manual 543
Prueba de la API de GraphQL 458
Tutorial: Delta Sync 557
Configuración en un clic 473
Esquema 558
Mutaciones 561
Consultas Sync 561
Ejemplo 562
Configuración y ajustes 569
Almacenamiento en caché y compresión 569
Tipos de instancias 570
Comportamiento del almacenamiento en caché 571
Cifrado de caché 572
Expulsión de caché 572
Expulsar una entrada en caché 573
Expulsar una entrada de caché en función de su identidad 574
Compresión de respuestas de API 576

Configuración de nombres de dominio personalizados 577
Registro y configuración de un nombre de dominio 578
Creación de un nombre de dominio personalizado en AWS AppSync 578
Nombres de dominio personalizados comodín en AWS AppSync 579
Detección de conflictos y sincronización 580
Orígenes de datos versionados 580
Detección y resolución de conflictos 584
Operaciones de sincronización 594
Supervisión y registro 595
Ajustes y configuración 595
CloudWatch métricas 597
CloudWatch registros 609
Referencia de tipos de registros 614
Analizar sus CloudWatch registros con Logs Insights 616
Analice sus registros con Service OpenSearch 618
Migración del formato de registro 618
Rastreo con AWS X-Ray 618
Ajustes y configuración 595
Rastreo de su API con X-Ray 619
Registro de llamadas a la API de AWS AppSync mediante AWS CloudTrail 622
Información de AWS AppSync en CloudTrail 622
Descripción de las entradas de los archivos de registro de AWS AppSync 623
Uso de API privadas de AWS AppSync 626
Creación de API privadas de AWS AppSync 628
Creación de un punto de conexión de interfaz para AWS AppSync 629
Ejemplos avanzados 630
Uso de políticas de IAM para limitar la creación de API públicas 634
Configurar la complejidad de ejecución, la profundidad de las consultas y la introspección de GraphQL con AWS AppSync 635
Uso de la característica de introspección 635
Configuración de los límites de profundidad de las consultas 637
Configuración de los límites de recuento de solucionadores 639
Uso de variables de entorno en AWS AppSync 640
Configuración de variables de entorno (consola) 641
Configuración de variables de entorno (API) 642
Configuración de variables de entorno (CFN) 643

variables de entorno y API combinadas 644
Recuperación de variables de entorno 644
Autorización y autenticación 646
Tipos de autorización 646
Autorización API_KEY 647
Autorización de AWS_LAMBDA 649
Elusión de las limitaciones de autorización de los tokens SiGv4 y OIDC 654
Autorización AWS_IAM 655
Autorización OPENID_CONNECT 657
Autorización AMAZON_COGNITO_USER_POOLS 659
Uso de modos de autorización adicionales 660
Control de acceso detallado 662
Filtrado de información 665
Acceso al origen de datos 666
Casos de uso de autorizaciones 667
Información general 667
Lectura de datos 668
Escritura de datos 672
Registros públicos y privados 674
Datos en tiempo real 675
Uso de AWS WAF para proteger las API 679
Integre una API de AppSync con AWS WAF 680
Creación de reglas para una ACL web 681
Seguridad 685
Protección de datos 686
Cifrado en movimiento 687
Validación de conformidad 687
Seguridad de la infraestructura 688
Resiliencia 689
Administración de identidades y accesos 689
Público 690
Autenticación con identidades 691
Administración de acceso mediante políticas 694
¿Cómo AWS AppSync funciona con IAM 697
Políticas basadas en identidad 704
Resolución de problemas 716

Registrar las llamadas a AWS AppSync la API con AWS CloudTrail 719
AWS AppSync información en CloudTrail 720
Descripción AWS AppSync de las entradas de los archivos de registro 721
Prácticas recomendadas 481
Métodos de autenticación 723
Uso de TLS para solucionadores de HTTP 724
Utilice roles con el menor número de permisos posible 724
Prácticas recomendadas sobre políticas de IAM 724
Referencia de solucionadores (JavaScript) 726
Descripción general de los solucionadores de JavaScript 726
Características de la versión ejecutable compatibles 727
Solucionadores de unidad 727
Anatomía de un solucionador de canalización de JavaScript 727
Escritura de código 732
Utilidades 735
Agrupación, TypeScript y mapas de origen 738
Pruebas 744
Migración de VTL a JavaScript 747
Elección entre acceso directo a los orígenes de datos y proxies a través de un origen de datos de Lambda 750
Referencia al objeto del contexto del solucionador 752
Acceso a `context` 752
Características de la versión ejecutable de JavaScript para solucionadores y funciones 763
Características de la versión ejecutable compatibles 763
Utilidades integradas 771
Módulos integrados 774
Utilidades de tiempo de ejecución 797
Aplicaciones auxiliares de tiempo en util.time 798
Aplicaciones auxiliares de DynamoDB en util.dynamodb 800
Aplicaciones auxiliares para HTTP en util.http 806
Aplicaciones auxiliares de transformación en util.transform 807
Aplicaciones auxiliares de cadena en util.str 821
Extensiones 821
Aplicaciones auxiliares para XML en util.xml 825
Referencia a la función de solucionador de JavaScript para DynamoDB 827
GetItem 827

PutItem 829
UpdateItem 832
DeleteItem 837
Consulta 840
Examen 845
Sync (Sincronizar) 849
BatchGetItem 852
BatchDeleteItem 855
BatchPutItem 858
TransactGetItems 860
TransactWriteItems 863
Sistema de tipos (mapeo de solicitud) 870
Sistema de tipos (mapeo de respuestas) 875
Filtros 879
Expresiones de condición 881
Expresiones de condición de transacción 893
Proyecciones 896
Referencia a la función de solucionador de JavaScript para OpenSearch 898
Solicitud 898
Respuesta 899
Campo `operation` 899
Campo `path` 899
Campo `params` 900
Variables de transferencia 902
JavaScript referencia de la función de resolución para Lambda 903
Objeto Request (solicitud) 903
Objeto de respuesta 907
Repuesta de la función de Lambda en lotes 907
JavaScript referencia de la función de resolución para la fuente EventBridge de datos 907
Solicitud 898
Respuesta 908
Campo `PutEvents` 910
Referencia a la función de solucionador de JavaScript para el origen de datos None 911
Solicitud 898
Carga 905
Respuesta 908

JavaScript referencia de función de resolución para HTTP 913
Solicitud 898
Método 914
ResourcePath 914
Campo params 914
Respuesta 908
JavaScript referencia de la función de resolución para Amazon RDS 916
Plantilla con etiquetas SQL 916
Creación de instrucciones 917
Recuperación de datos 918
Funciones de utilidad 919
Conversión 927
Referencia de plantillas de mapeo de solucionador (VTL) 929
Información general sobre las plantillas de mapeo de solucionador 929
Solucionadores de unidad 930
Solucionadores de canalización 173
Ejemplo de plantilla de de 935
Reglas de deserialización de plantillas de mapeo evaluadas 937
Guía de programación de plantillas de mapeo de solucionador 939
Configuración 940
Variables 941
Llamada a métodos 943
Strings 944
Bucles 945
Matrices 946
Comprobaciones condicionales 947
Operadores 948
Contexto 950
Filtrado 950
Referencia de contexto de las plantillas de mapeo del solucionador 955
Acceso a `$context` 955
Sanear datos entrantes 965
Referencia de utilidad de la plantilla de mapeo de solucionador 967
Aplicaciones auxiliares de utilidades en $util 968
AWS AppSync directivas 980
Aplicaciones auxiliares de tiempo en $util.time 981

Aplicaciones auxiliares de lista en $util.list 984
Aplicaciones auxiliares de mapas en $util.map 985
Aplicaciones auxiliares de DynamoDB en $util.dynamodb 985
Auxiliares de Amazon RDS en $util.rds 995
Aplicaciones auxiliares para HTTP en $util.http 998
Aplicaciones auxiliares para XML en $util.xml 1000
Aplicaciones auxiliares de transformación en $util.transform 1002
Aplicaciones auxiliares de matemáticas en $util.math 1016
Aplicaciones auxiliares de cadena en $util.str 1017
Extensiones 1018
Referencia de las plantillas de mapeo de solucionador para DynamoDB 1031
GetItem 1032
PutItem 1034
UpdateItem 1037
DeleteItem 1044
Consulta 1047
Examen 1051
Sync (Sincronizar) 1056
BatchGetItem 1059
BatchDeleteItem 1063
BatchPutItem 1067
TransactGetItems 1070
TransactWriteItems 1074
Sistema de tipos (mapeo de solicitud) 1083
Sistema de tipos (mapeo de respuestas) 1088
Filtros 1092
Expresiones de condición 1094
Expresiones de condición de transacción 1106
Proyecciones 1109
Referencia de plantillas de mapeo de solucionador para RDS 1110
Plantilla de mapeo de solicitudes 1110
Versión 1112
Instrucciones y VariableMap 1112
VariableTypeHintMap 1113
Referencia de plantillas de mapeo de solucionador para OpenSearch 1114
Plantilla de mapeo de solicitudes 1110

Plantilla de mapeo de respuestas 899
Campo operation 899
Campo path 899
Campo params 900
Variables de transferencia 902
Referencia de plantillas de mapeo de solucionador para Lambda 1118
Plantilla de mapeo de solicitudes 1110
Plantilla de mapeo de respuestas 899
Repuesta de la función de Lambda en lotes 1124
Solucionadores de Lambda directos 1124
Referencia de plantilla de mapeo de Resolver para EventBridge 1131
Plantilla de mapeo de solicitudes 1110
Plantilla de mapeo de respuestas 899
Campo PutEvents 910
Referencia de plantillas de mapeo de solucionador para el origen de datos None 1135
Plantilla de mapeo de solicitudes 1110
Versión 1112
Carga 1123
Plantilla de mapeo de respuestas 899
Referencia de plantillas de mapeo de solucionador para HTTP 1138
Plantilla de mapeo de solicitudes 1110
Versión 1112
Método 1141
ResourcePath 1141
Campo params 900
Entidades de certificación (CA) reconocidas por AWS AppSync para los puntos de conexión HTTPS 1143
Registro de cambios de plantillas de mapeo de solucionador 1212
Disponibilidad de operación de un origen de datos por matriz de la versión 1213
Cambio de la versión en una plantilla de mapeo de solucionador de unidad 1214
Cambio de la versión en una función 1214
2018-05-29 1215
2017-02-28 1222
Referencia de tipos 1223
Tipos escalares 1223
Escalares predeterminados 1223

Escalares AWS AppSync 1224
Ejemplo de uso de esquema 1225
Interfaces y uniones en GraphQL 1229
Ejemplos de interfaces 1229
Ejemplos de uniones 1233
Resolución Type en AWS AppSync 1234
Ejemplo de resolución Type 1235
Solución de problemas y errores comunes 1240
Mapeo de clave de DynamoDB incorrecto 1240
Falta el solucionador 1240
Errores en la plantilla de mapeo 1241
Tipos de retorno incorrectos 1241
Procesamiento de solicitudes no válidas 1242
..... mccxliii

¿Qué es AWS AppSync?

AWS AppSync permite a los desarrolladores conectar sus aplicaciones y servicios a datos y eventos con API de GraphQL y Pub/Sub seguras, sin servidor y de alto rendimiento. Puede hacer lo siguiente con AWS AppSync:

- Acceder a los datos de uno o más orígenes de datos desde un único punto de conexión de la API de GraphQL.
- Combinar API de GraphQL de múltiples orígenes en una única API de GraphQL fusionada.
- Publicar actualizaciones de datos en tiempo real en sus aplicaciones.
- Aprovechar las funcionalidades de seguridad, monitorización, registro y rastreo integrados, con el almacenamiento en caché opcional para una baja latencia.
- Pagar únicamente por las solicitudes de API y por los mensajes en tiempo real que se entreguen.

Temas

- Características de AWS AppSync
- ¿Es la primera vez que usa AWS AppSync?
- Servicios relacionados
- Precios de AWS AppSync

Características de AWS AppSync

- Acceso y consulta de datos simplificados, con tecnología de GraphQL
- WebSockets sin servidor para suscripciones a GraphQL y canales pub/sub
- Almacenamiento en caché del lado del servidor para que los datos estén disponibles en cachés en memoria de alta velocidad para una latencia baja
- Compatibilidad con JavaScript y TypeScript para escribir lógica de negocio
- Seguridad empresarial con API privadas para restringir el acceso a las API y la integración con AWS WAF
- Controles de autorización integrados, compatibles con claves de API, IAM, Amazon Cognito, proveedores de OpenID Connect y autorización de Lambda para lógica personalizada.
- API combinadas para admitir casos de uso federados

Para obtener más información sobre cada una de estas funcionalidades, consulte Características de AWS AppSync.

¿Es la primera vez que usa AWS AppSync?

Si es la primera vez que usa AWS AppSync, le recomendamos que empiece leyendo las siguientes secciones:

- Si no está familiarizado con GraphQL, consulte la Introducción: Cómo crear su primera API de GraphQL.
- Si va a crear aplicaciones que consumen API de GraphQL, consulte Creación de una aplicación cliente y the section called “Datos en tiempo real”.
- Si quiere obtener información sobre el solucionador de GraphQL, consulte lo siguiente:

  JavaScript/TypeScript

  - Tutoriales de solucionadores (JavaScript)
  - Referencia de solucionadores (JavaScript)

  VTL

  - Tutoriales de solucionadores (VTL)
  - Referencia de plantillas de mapeo de solucionador (VTL)
- Si busca ejemplos de proyectos de AWS AppSync, actualizaciones y mucho más, consulte el blog de AppSync.

Servicios relacionados

Si va a crear una aplicación web o móvil desde cero, considere la posibilidad de usar AWS Amplify. Amplify aprovecha AWS AppSync y otros servicios de AWS para ayudarlo a crear aplicaciones web y móviles más sólidas y potentes con menos trabajo.

Precios de AWS AppSync

AWS AppSync tiene un precio basado en millones de solicitudes y actualizaciones. El almacenamiento en caché tiene un coste adicional. Para obtener más información, consulte Precios de AWS AppSync.

A continuación, se enumeran las excepciones a los precios generales de AWS AppSync:

- El almacenamiento en caché de API en AWS AppSync no está disponible en la Capa gratuita de AWS.
- Las solicitudes no se cobran por errores de autorización y autenticación.
- Las llamadas a métodos que requieran claves de API no se cobrarán si faltan las claves de API o no son válidas.

GraphQL y arquitectura de AWS AppSync

> **Note**
>
> Esta guía asume que el usuario tiene un conocimiento práctico del estilo arquitectónico REST. Recomendamos revisar este y otros temas de front-end antes de trabajar con GraphQL y AWS AppSync.

GraphQL es un lenguaje de consulta y manipulación para las API. GraphQL proporciona una sintaxis flexible e intuitiva para describir los requisitos e interacciones de los datos. Permite a los desarrolladores solicitar exactamente lo que se necesita y obtener resultados predecibles. También permite acceder a muchos orígenes en una sola solicitud, lo que reduce la cantidad de llamadas de red y los requisitos de ancho de banda y, por lo tanto, ahorra batería y los ciclos de CPU que las aplicaciones consumen.

La actualización de los datos se simplifica con mutaciones, lo que permite a los desarrolladores describir cómo deberían cambiar los datos. GraphQL también facilita la configuración rápida de soluciones en tiempo real mediante suscripciones. Todas estas características combinadas, junto con potentes herramientas para desarrolladores, hacen que GraphQL sea esencial para administrar los datos de las aplicaciones.

GraphQL es una alternativa a REST. La arquitectura RESTful es actualmente una de las soluciones más populares para la comunicación cliente-servidor. Se centra en el concepto de que sus recursos (datos) están expuestos por una URL. Estas URL se pueden usar para acceder a los datos y manipularlos mediante operaciones CRUD (crear, leer, actualizar, eliminar) en forma de métodos HTTP como `GET`, `POST`, y `DELETE`. La ventaja de REST es que es relativamente fácil de aprender e implementar. Puede configurar rápidamente las API RESTful para llamar a una amplia gama de servicios.

Sin embargo, la tecnología se complica cada vez más. A medida que las aplicaciones, las herramientas y los servicios comienzan a escalarse para un público de alcance mundial, la necesidad de arquitecturas rápidas y escalables adquiere una importancia capital. REST tiene muchas deficiencias en cuanto a operaciones escalables. Vea este caso de uso como ejemplo.

En las secciones siguientes, revisaremos algunos de los conceptos que rodean a las API RESTful. A continuación, presentaremos GraphQL y cómo funciona.

Para obtener más información sobre GraphQL y las ventajas de migrar a AWS, consulte la Guía de decisiones sobre implementaciones de GraphQL.

Temas

- ¿Qué es una API?
- ¿Qué es REST?
- ¿Por qué usar GraphQL en lugar de REST?
- Componentes de una API de GraphQL
- Propiedades adicionales de GraphQL

¿Qué es una API?

Una interfaz de programación de aplicaciones (API) define las reglas que se deben seguir para comunicarse con otros sistemas de software. Los desarrolladores exponen o crean las API para que otras aplicaciones puedan comunicarse con sus aplicaciones mediante programación. Por ejemplo, la aplicación de plantillas de horarios expone una API que solicita el nombre completo del empleado y un intervalo de fechas. Cuando recibe esta información, procesa internamente la plantilla de horarios del empleado y devuelve el número de horas trabajadas en ese intervalo de fechas.

Puede pensar en una API web como una puerta de enlace entre los clientes y los recursos de la web.

Clientes

Los clientes son usuarios que desean acceder a la información desde la web. El cliente puede ser una persona o un sistema de software que utilice la API. Por ejemplo, los desarrolladores pueden escribir programas que accedan a la información del tiempo de un sistema meteorológico. O se puede acceder a los mismos datos desde un navegador al visitar directamente el sitio web meteorológico.

Recursos

Los recursos son la información que las diferentes aplicaciones proporcionan a sus clientes. Los recursos pueden ser imágenes, vídeos, texto, números o cualquier tipo de datos. La máquina que proporciona el recurso al cliente también se denomina servidor. Las organizaciones utilizan las API para compartir recursos y proporcionar servicios web y, a la vez, mantener la seguridad, el control y la autenticación. Además, las API ayudan a determinar qué clientes tienen acceso a recursos internos específicos.

¿Qué es REST?

A un alto nivel, la transferencia de estado representacional (REST) es una arquitectura de software que impone condiciones sobre el funcionamiento de una API. REST se creó inicialmente como una guía para gestionar la comunicación en una red compleja como Internet. Una arquitectura basada en REST puede emplearse para apoyar una comunicación fiable y de alto rendimiento a escala. Puede implementarla y modificarla fácilmente, lo que aporta visibilidad y portabilidad multiplataforma a cualquier sistema de API.

Los desarrolladores de API pueden diseñarlas utilizando diversas arquitecturas distintas. Las API que siguen el estilo arquitectónico REST se denominan API de REST. Los servicios web que implementan la arquitectura REST se denominan servicios web RESTful. El término API RESTful generalmente se refiere a las API web RESTful. Sin embargo, los términos API de REST y API RESTful pueden usarse de manera intercambiable.

A continuación, se indican algunos de los principios del estilo arquitectónico de REST:

Interfaz uniforme

La interfaz uniforme es fundamental para el diseño de cualquier servicio web RESTful. Indica que el servidor transfiere la información en un formato estándar. El recurso formateado se denomina representación en REST. Este formato puede ser diferente de la representación interna del recurso en la aplicación de servidor. Por ejemplo, el servidor puede almacenar datos como texto pero enviarlos en un formato de representación HTML.

La interfaz uniforme impone cuatro restricciones arquitectónicas:

1. Las solicitudes deben identificar recursos. Lo hacen mediante un identificador de recursos uniforme.
2. Los clientes tienen suficiente información en la representación del recurso para modificarlo o eliminarlo si así lo desean. El servidor cumple esta condición mediante el envío de metadatos que describen el recurso con más detalle.
3. Los clientes reciben información sobre cómo seguir procesando la representación. El servidor lo consigue enviando mensajes autodescriptivos que contienen metadatos sobre la mejor manera en la que el cliente puede utilizarlos.
4. Los clientes reciben información sobre todos los demás recursos relacionados que necesitan para completar una tarea. El servidor lo consigue enviando hipervínculos en la representación para que los clientes puedan descubrir más recursos de forma dinámica.

Ausencia de estado

En la arquitectura REST, la ausencia de estado se refiere a un método de comunicación en el que el servidor completa cada una de las solicitudes del cliente independientemente de las solicitudes anteriores. Los clientes pueden solicitar recursos en cualquier orden y cada solicitud no tiene estado o está aislada de las otras. Esta restricción de diseño de la API de REST implica que el servidor puede comprender y cumplir completamente la solicitud en todo momento.

Sistema por capas

En una arquitectura de sistema por capas, el cliente puede conectarse a otros intermediarios autorizados entre el cliente y el servidor, y seguirá recibiendo respuestas del servidor. Los servidores también pueden transferir las solicitudes a otros servidores. Puede diseñar su servicio web RESTful para que se ejecute en varios servidores con varios niveles, como seguridad, aplicaciones y lógica empresarial, y que funcionen juntos para cumplir con las solicitudes de los clientes. Estas capas permanecen invisibles para el cliente.

Capacidad de almacenamiento en caché

Los servicios web RESTful admiten el almacenamiento en caché, que es el proceso de almacenar respuestas en el cliente o en un intermediario para mejorar el tiempo de respuesta del servidor. Por ejemplo, supongamos que visita un sitio web que tiene imágenes comunes de encabezado y pie de página en todas las páginas. Cada vez que visita una nueva página web, el servidor debe volver a enviar las mismas imágenes. Para evitarlo, el cliente guarda en caché o almacena estas imágenes después de la primera respuesta y, a continuación, las utiliza directamente desde la caché. Los servicios web RESTful controlan el almacenamiento en caché mediante respuestas de API que se definen a sí mismas como almacenables en caché o no almacenables en caché.

¿Qué es una API RESTful?

La API RESTful es una interfaz que dos sistemas informáticos utilizan para intercambiar información de forma segura a través de Internet. La mayoría de las aplicaciones empresariales tienen que comunicarse con otras aplicaciones internas y de terceros para realizar diversas tareas. Por ejemplo, para generar nóminas mensuales, su sistema de cuentas interno debe compartir los datos con el sistema bancario del cliente para automatizar la facturación y comunicarse con una aplicación interna de plantillas de horarios. Las API RESTful respaldan este intercambio de información porque siguen estándares de comunicación de software seguros, fiables y eficientes.

¿Cómo funcionan las API RESTful?

La función básica de una API RESTful es la misma que la de navegar por Internet. El cliente contacta con el servidor mediante la API cuando necesita un recurso. Los desarrolladores de las API explican cómo el cliente debe usar la API de REST en la documentación de la API de la aplicación de servidor. Estos son los pasos generales para cualquier llamada a la API de REST:

1. El cliente envía una solicitud al servidor. El cliente sigue la documentación de la API para aplicar a la solicitud un formato que el servidor comprenda.
2. El servidor autentica al cliente y confirma que este tiene derecho a realizar la solicitud.
3. El servidor recibe la solicitud y la procesa internamente.
4. El servidor devuelve una respuesta al cliente. La respuesta contiene información que indica al cliente si la solicitud se ha realizado correctamente. La respuesta también incluye la información que el cliente haya solicitado.

Los detalles de solicitud y respuesta de la API de REST varían ligeramente según cómo diseñen la API los desarrolladores.

¿Por qué usar GraphQL en lugar de REST?

REST es uno de los estilos arquitectónicos fundamentales de las API web. Sin embargo, cuanto más se interconecte el mundo, la necesidad de desarrollar aplicaciones robustas y escalables se convertirá en un tema cada vez más urgente. Si bien REST es actualmente el estándar del sector para la creación de las API web, las implementaciones RESTful presentan varios inconvenientes recurrentes que se han identificado:

1. Solicitudes de datos: al utilizar las API RESTful, normalmente se solicitan los datos que se necesitan a través de puntos de conexión. El problema surge cuando hay datos que pueden no estar tan bien empaquetados. Es posible que los datos que necesite estén ocultos tras múltiples niveles de abstracción y que la única forma de obtenerlos sea mediante varios puntos de conexión, lo que implica realizar diversas solicitudes para extraer todos los datos.
2. Obtención excesiva o insuficiente de datos: para agravar los problemas que plantean las solicitudes múltiples, los datos de cada punto de conexión están estrictamente definidos, por lo que se obtendrá como resultado todos los datos que se hayan definido para esa API, incluso si técnicamente no los deseaba.

Esto puede provocar una obtención excesiva de datos, lo que significa que nuestras solicitudes devuelven datos superfluos. Por ejemplo, supongamos que solicita datos del personal de la empresa y desea saber los nombres de los empleados de un departamento determinado. El punto de conexión que devuelve los datos contendrá los nombres, pero también puede contener otras informaciones, como el cargo o la fecha de nacimiento. Como la API es fija, no se puede solicitar simplemente los nombres, sino que el resto de los datos se incluyen también.

La situación opuesta, en la que no devolvemos suficientes datos, se denomina obtención insuficiente de datos. Para obtener todos los datos solicitados, tal vez deba realizar varias solicitudes al servicio. En función de cómo estén estructurados los datos, puede encontrarse con consultas ineficientes que den lugar a problemas como el temido problema n+1.

3. Iteraciones de desarrollo lentas: muchos desarrolladores adaptan sus API RESTful para adaptarlas al flujo de sus aplicaciones. Sin embargo, a medida que sus aplicaciones crecen, es posible que tanto el front-end como el backend requieran cambios importantes. Como resultado, es posible que las API ya no se adapten a la forma de los datos de una manera eficaz o significativa. Esto hace que las iteraciones de los productos sean más lentas debido a la necesidad de modificar la API.
4. Rendimiento a escala: debido a estos problemas agravados, la escalabilidad se verá afectada en numerosos ámbitos. El rendimiento de las aplicaciones puede verse afectado porque las solicitudes devuelvan demasiados datos o muy pocos (lo que se traduce en más solicitudes). Ambas situaciones provocan una carga innecesaria en la red, lo que se traduce en un rendimiento deficiente. Por parte de los desarrolladores, la velocidad de desarrollo puede reducirse porque las API son fijas y ya no se ajustan a los datos que solicitan.

El argumento de venta de GraphQL es la superación de los inconvenientes de REST. Estas son algunas de las soluciones clave que GraphQL ofrece a los desarrolladores:

1. Puntos de conexión únicos: GraphQL utiliza un único punto de conexión para consultar datos. No es necesario crear varias API para que se adapten a la forma de los datos. El resultado de esto es un menor número de solicitudes que circulan por la red.
2. Obtención de datos: GraphQL resuelve los problemas perennes de la obtención excesiva o insuficiente de datos simplemente definiendo los datos que necesita. GraphQL permite dar forma a los datos para que se ajusten a sus necesidades y solo reciba lo que haya pedido.
3. Abstracción: las API de GraphQL contienen componentes y sistemas que describen los datos mediante un estándar independiente del lenguaje. En otras palabras, la forma y la estructura de

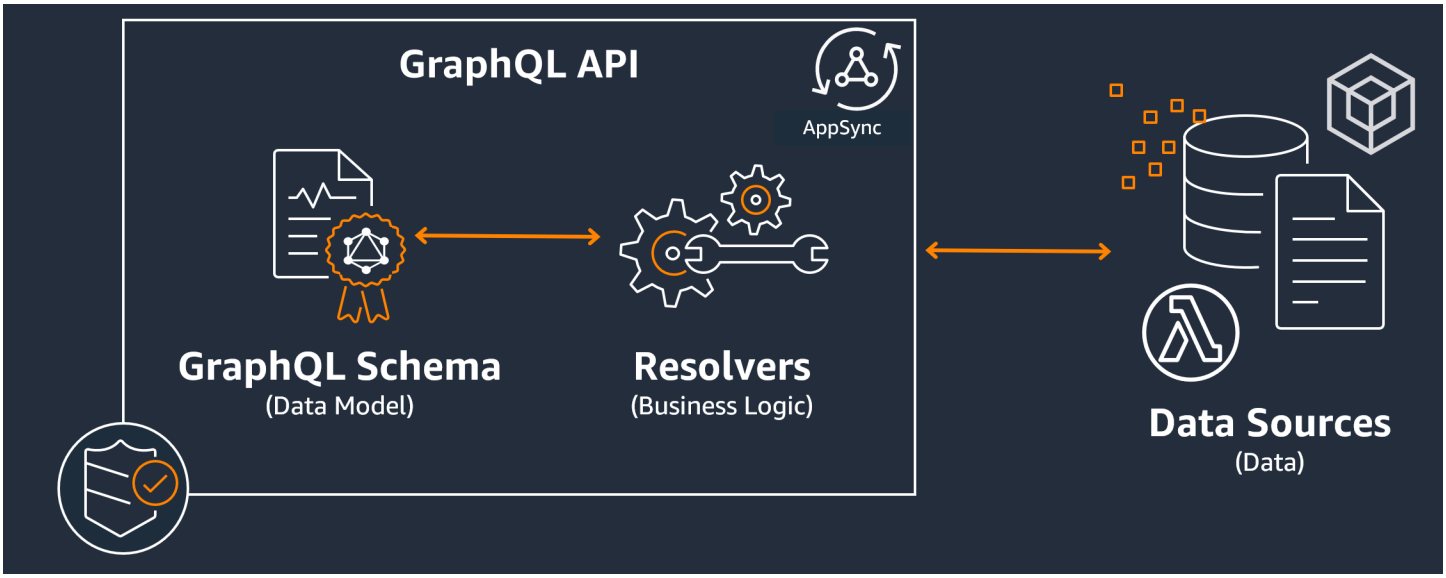
los datos están estandarizadas para que tanto el frontend como el backend sepan cómo se van a enviar a través de la red. Esto permite a los desarrolladores de ambas partes trabajar con los sistemas de GraphQL y no cerca de ellos.

4. Iteraciones rápidas: debido a la estandarización de los datos, es posible que los cambios realizados en un extremo del desarrollo no sean necesarios en el otro. Por ejemplo, es posible que los cambios de presentación del frontend no provoquen variaciones importantes en el backend, ya que GraphQL permite modificar fácilmente la especificación de los datos. Simplemente puede definir o modificar la forma de los datos para adaptarlos a las necesidades de la aplicación a medida que crecen. La consecuencia de esto es la reducción del trabajo de desarrollo potencial.

Estas son solo algunas de las ventajas de GraphQL. En las siguientes secciones, aprenderá a estructurar GraphQL y las propiedades que lo convierten en una alternativa única a REST.

Componentes de una API de GraphQL

Una API GraphQL estándar se compone de un único esquema que gestiona la forma de los datos que se van a consultar. El esquema está vinculado a uno o más orígenes de datos como una base de datos o una función de Lambda. Entre las dos se encuentran uno o más solucionadores que se encargan de la lógica empresarial de las solicitudes. Todos los componentes tienen efectos muy relevantes en la implementación de GraphQL. En las siguientes secciones se presentarán estos tres componentes y el papel que desempeñan en el servicio GraphQL.



Temas

- Esquemas

- Orígenes de datos
- Solucionadores

Esquemas

El esquema GraphQL es la base de una API de GraphQL. Sirve como esquema que define la forma de los datos. También es un contrato entre el cliente y el servidor que define cómo se recuperarán o modificarán los datos.

Los esquemas de GraphQL se escriben en el lenguaje de definición de esquemas (SDL). SDL está compuesto por tipos y campos con una estructura establecida:

- Tipos: los tipos son la forma en que GraphQL define la forma y el comportamiento de los datos. GraphQL admite una multitud de tipos que se explicarán más adelante en esta sección. Cada tipo que se defina en su esquema tendrá su propio ámbito. Dentro del ámbito habrá uno o más campos que pueden contener un valor o una lógica que se utilice en el servicio GraphQL. Los tipos cumplen muchos roles diferentes, siendo las más comunes los objetos o los escalares (tipos de valores primitivos).
- Campos: los campos existen dentro del ámbito de un tipo y contienen el valor que se solicita al servicio GraphQL. Se parecen mucho a las variables de otros lenguajes de programación. La forma de los datos que defina en sus campos determinará cómo se estructuren los datos en una operación de solicitud/respuesta. Esto permite a los desarrolladores predecir lo que se va a devolver sin saber cómo se implementa el backend del servicio.

Para visualizar el aspecto que tendría un esquema, revisemos el contenido de un esquema de GraphQL simple. En el código de producción, el esquema normalmente estará en un archivo llamado `schema.graphql` o `schema.json`. Supongamos que estamos analizando un proyecto que implementa un servicio de GraphQL. Este proyecto almacena los datos del personal de la empresa y el archivo `schema.graphql` se utiliza para recuperar los datos del personal y añadir personal nuevo a una base de datos. El código tiene este aspecto:

schema.graphql

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

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

Podemos ver que hay tres tipos definidos en el esquema: `Person`, `Query` y `Mutation`. Si nos fijamos en `Person`, podemos adivinar que es el esquema de la instancia del empleado de una empresa, lo que convertiría a este tipo en un objeto. Dentro de su alcance, vemos `id`, `name` y `age`. Estos son los campos que definen las propiedades de una `Person`. Esto significa que nuestro origen de datos almacena el `name` de cada `Person` como un tipo escalar (primitivo) `String` y `age` como un tipo escalar (primitivo) `Int`. El `id` actúa como identificador único y especial para cada `Person`. También es un valor obligatorio, tal como lo indica el símbolo `!`.

Los dos tipos de objetos siguientes se comportan de forma diferente. GraphQL reserva unas palabras clave para tipos de objetos especiales que definen cómo se rellenarán los datos en el esquema. Un tipo `Query` recuperará los datos del origen. En nuestro ejemplo, la consulta podría recuperar objetos de `Person` de una base de datos. Esto puede recordar a las operaciones `GET` en la terminología RESTful. Una `Mutation` modificará los datos. En nuestro ejemplo, la mutación puede añadir más objetos de `Person` a la base de datos. Esto puede recordarle a operaciones que cambian de estado, como `PUT` o `POST`. Los comportamientos de todos los tipos de objetos especiales se explicarán más adelante en esta sección.

Supongamos que la `Query` de nuestro ejemplo va a recuperar algo de la base de datos. Si nos fijamos en los campos de `Query`, vemos uno llamado `people`. El valor del campo es `[Person]`. Esto significa que queremos recuperar alguna instancia de `Person` de la base de datos. Sin embargo, la adición de corchetes significa que queremos devolver una lista de todas las instancias de `Person` y no solo una específica.

El tipo `Mutation` es responsable de realizar operaciones que cambian el estado, como la modificación de datos. Una mutación es responsable de realizar alguna operación de cambio de estado en el origen de datos. En nuestro ejemplo, nuestra mutación contiene una operación llamada `addPerson` que agrega un nuevo objeto de `Person` a la base de datos. La mutación utiliza una `Person` y espera una entrada para los campos `id`, `name` y `age`.

En este punto, quizás se pregunte cómo funcionan las operaciones como `addPerson` sin una implementación de código, dado que en teoría tiene algún comportamiento y se parece mucho a una

función con un nombre y parámetros de función. Actualmente, no funcionará porque un esquema solo sirve como declaración. Para implementar el comportamiento de `addPerson`, deberíamos agregarle un solucionador. Un solucionador es una unidad de código que se ejecuta siempre que se llama a su campo asociado (en este caso, la operación `addPerson`). Si quiere usar una operación, deberá añadir la implementación del solucionador en algún momento. En cierto modo, puede pensarse en la operación de esquema como la declaración de la función y en el solucionador como en la definición. Los solucionadores se explicarán en otra sección.

En este ejemplo se muestran solo las formas más sencillas en que un esquema puede manipular datos. Creará aplicaciones complejas, sólidas y escalables gracias a las funciones de GraphQL y AWS AppSync. En la siguiente sección, definiremos todos los diferentes tipos y comportamientos de campo que puede utilizar en su esquema.

Tipos de GraphQL

GraphQL admite muchos tipos diferentes. Como vio en la sección anterior, los tipos definen la forma o el comportamiento de los datos. Son los componentes fundamentales de un esquema de GraphQL.

Los tipos se pueden clasificar como entradas y salidas. Los de entrada son tipos que se pueden pasar como argumento para los tipos de objetos especiales (`Query`, `Mutation`, etc.), mientras que los tipos de salida se utilizan estrictamente para almacenar y devolver datos. A continuación, se muestra una lista de tipos y sus categorizaciones:

- Objetos: un objeto contiene campos que describen una entidad. Por ejemplo, un objeto podría ser algo así como un `book` con campos que describen sus características, como `authorName`, `publishingYear`, etc. Son estrictamente tipos de salida.
- Escalares: son tipos primitivos como int, string, etc. Por lo general, se asignan a los campos. Usando el campo `authorName` como ejemplo, se le podría asignar el escalar `String` para almacenar un nombre como «John Smith». Los escalares pueden ser tanto de entrada como de salida.
- Entradas: las entradas permiten transferir un grupo de campos como argumento. Están estructurados de forma muy similar a los objetos, pero se pueden transferir como argumentos a objetos especiales. Las entradas permiten definir escalares, enumeraciones y otras entradas incluidas en su ámbito. Las entradas solo pueden ser tipos de entrada.
- Objetos especiales: los objetos especiales realizan operaciones que cambian de estado y se encargan de la mayor parte del trabajo pesado del servicio. Hay tres tipos de objetos especiales: consulta, mutación y suscripción. Las consultas suelen obtener datos; las mutaciones manipulan

los datos; las suscripciones se abren y mantienen una conexión bidireccional entre los clientes y los servidores para una comunicación constante. Los objetos especiales no son ni entradas ni salidas debido a su funcionalidad.

- Enumeraciones: listas predefinidas de valores legales. Si llama a una enumeración, sus valores solo pueden ser lo que esté definido en su ámbito. Por ejemplo, si tuviera una enumeración llamada `trafficLights` que representara una lista de señales de tráfico, podría tener valores como `redLight` y `greenLight`, pero no `purpleLight`. Un semáforo real solo tendrá un número determinado de señales, por lo que podría usar la enumeración para definirlas y hacer que sean los únicos valores legales para referirse a `trafficLight`. Las enumeraciones pueden ser tanto de entrada como de salida.
- Uniones/interfaces: las uniones permiten devolver uno o más elementos en una solicitud en función de los datos que haya solicitado el cliente. Por ejemplo, si tuviera un tipo `Book` con un campo `title` y un tipo `Author` con un campo `name`, podría crear una unión entre ambos tipos. Si su cliente quisiera consultar la frase "Julio César" en una base de datos, la unión podría devolver Julio César (la obra de William Shakespeare) del `title` del `Book` y Julio César (el autor de Commentarii de Bello Gallico) del `name` de `Author`. Las uniones solo pueden ser tipos de salida.

  Las interfaces son conjuntos de campos que los objetos deben implementar. Es parecido a las interfaces de lenguajes de programación como Java, donde hay que implementar los campos definidos en la interfaz. Por ejemplo, supongamos que creó una interfaz llamada `Book` que contenía un campo `title`. Digamos que más tarde ha creado un tipo llamado `Novel` que ha implementado `Book`. `Novel` tendría que incluir un campo `title`. Sin embargo, `Novel` también podría incluir otros campos que no estén en la interfaz, como `pageCount` de `ISBN`. Las interfaces solo pueden ser tipos de salida.

En las siguientes secciones se explicará cómo funciona cada tipo en GraphQL.

Objects

Los objetos de GraphQL son el tipo principal que verá en el código de producción. En GraphQL, puede pensar en un objeto como en una agrupación de campos diferentes (de forma aprecia a las variables en otros lenguajes) en la que cada campo se define por un tipo (normalmente un escalar u otro objeto) que puede contener un valor. Los objetos representan una unidad de datos que se puede recuperar o manipular a partir de la implementación del servicio.

Los tipos de objetos se declaran mediante la palabra clave `Type`. Modifiquemos un poco nuestro ejemplo de esquema:

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

type Occupation {
  title: String
}
```

Los tipos de objetos que aparecen aquí son `Person` y `Occupation`. Cada objeto tiene sus propios campos con sus propios tipos. Una característica de GraphQL es la capacidad de establecer campos de otros tipos. Puede ver que el campo `occupation` de `Person` contiene un tipo de objeto `Occupation`. Podemos hacer esta asociación porque GraphQL solo describe los datos y no la implementación del servicio.

Escalares

Los escalares son esencialmente tipos primitivos que contienen valores. En AWS AppSync, hay dos tipos de escalares: los escalares y los escalares predeterminados de GraphQL y los escalares de AWS AppSync. Los escalares se utilizan normalmente para almacenar valores de campo dentro de los tipos de objetos. Entre los tipos de GraphQL predeterminados se incluyen `Int`, `Float`, `String`, `Boolean` y `ID`. Volvamos al ejemplo anterior:

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

type Occupation {
  title: String
}
```

Si elegimos los campos `title` y `name`, ambos contienen un escalar `String`. `Name` podría devolver un valor de cadena como "`John Smith`" y el título podría devolver algo como "`firefighter`". Algunas implementaciones de GraphQL también admiten escalares personalizados que utilizan la palabra clave `Scalar` e implementan el comportamiento del tipo. Sin embargo, AWS AppSync

actualmente no admite escalares personalizados. Para obtener una lista de escalares, consulte Tipos escalares en AWS AppSync.

Entradas

Debido al concepto de tipos de entrada y salida, existen ciertas restricciones para la transferencia de argumentos. Los tipos que normalmente deben transferirse, especialmente los objetos, están restringidos. Puede usar el tipo de entrada para omitir esta regla. Las entradas son tipos que contienen escalares, enumeraciones y otros tipos de entrada.

Las entradas se definen mediante la palabra clave `input`:

```
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
}
```

Como ve, podemos tener entradas independientes que imitan el tipo original. Estas entradas se utilizarán a menudo en sus operaciones de campo de la siguiente manera:

```
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
}
```

Observe cómo seguimos transfiriendo `occupationInput` en lugar de `Occupation` para crear una `Person`.

Este es solo uno de los escenarios sobre entradas. No es necesario que copien los objetos a escala 1:1 y, en el código de producción, lo más probable es que no lo utilice de esta manera. Una buena práctica es aprovechar los esquemas de GraphQL y definir solo lo que se necesite introducir como argumentos.

Además, se pueden usar las mismas entradas en varias operaciones, pero no recomendamos hacerlo. Lo ideal es que cada operación contenga su propia copia única de las entradas en caso de que cambien los requisitos del esquema.

Objetos especiales

GraphQL reserva algunas palabras clave para objetos especiales que definen parte de la lógica empresarial sobre la forma en que su esquema va a recuperar o manipular los datos. Como máximo, puede haber una de cada una de estas palabras clave en un esquema. Actúan como puntos de entrada para todos los datos solicitados que sus clientes ejecutan en el servicio de GraphQL.

Los objetos especiales también se definen con la palabra clave `type`. Aunque se utilizan de forma diferente a los tipos de objetos normales, su implementación es muy parecida.

Queries

> Las consultas son muy similares a las operaciones `GET` en el sentido de que realizan una búsqueda de solo lectura para obtener datos de su origen. En GraphQL, la `Query` define todos los puntos de entrada para los clientes que realizan solicitudes a su servidor. Siempre habrá una `Query` en tu implementación de GraphQL.

Aquí están la `Query` y los tipos de objetos modificados que hemos utilizado en nuestro ejemplo de esquema anterior:

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

Nuestra `Query` contiene un campo llamado `people` que devuelve una lista de instancias de `Person` del origen de datos. Supongamos que necesitamos cambiar el comportamiento de nuestra aplicación y que ahora necesitamos devolver una lista de solo las instancias de `Occupation` para un propósito diferente. Simplemente podríamos añadirlo a la consulta:

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

En GraphQL, podemos tratar nuestra consulta como el único origen de las solicitudes. Como puede ver, esto es potencialmente mucho más simple que las implementaciones RESTful, que pueden usar diferentes puntos de conexión para lograr lo mismo (`.../api/1/people` y `.../api/1/occupations`).

Suponiendo que tengamos una implementación de resolución para esta consulta, ahora podemos realizar una consulta real. Si bien el tipo de `Query` existe, debemos llamarlo explícitamente para que se ejecute en el código de la aplicación. Esto se puede hacer con la palabra clave `query`:

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

```
   }
}
```

Como puede ver, esta consulta se llama `getItems` y devuelve `people` (una lista de objetos de `Person`) y `occupations` (una lista de objetos de `Occupation`). En `people`, devolvemos solo el campo `name` de cada `Person`, mientras que devolvemos el campo `title` de cada `Occupation`. La respuesta puede tener un aspecto similar al siguiente:

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

La respuesta de ejemplo muestra cómo los datos siguen la forma de la consulta. Cada entrada recuperada aparece dentro del ámbito del campo. `people` y `occupations` devuelven los elementos como listas separadas. Si bien es útil, puede ser más conveniente modificar la consulta para que devuelva una lista con los nombres y las ocupaciones de las personas:

```
query getItems {
   people {
```

```
      name
      occupation {
        title
      }
}
```

Se trata de una modificación legal porque nuestro tipo `Person` contiene un campo `occupation` de tipo `Occupation`. Cuando se incluye dentro del ámbito de `people`, devolvemos el `name` de cada `Person` junto con la correspondiente `Occupation` mediante `title`. La respuesta puede tener un aspecto similar al siguiente:

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

Mutations

Las mutaciones son parecidas a las operaciones que cambian el estado, como PUT o POST. Realizan una operación de escritura para modificar los datos del origen y, a continuación, obtienen la respuesta. Definen los puntos de entrada para las solicitudes de modificación de datos. A diferencia de las consultas, una mutación puede incluirse en el esquema o no según las necesidades del proyecto. Esta es la mutación del ejemplo del esquema:

```
type Mutation {
```

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

El campo `addPerson` representa un punto de entrada que añade una `Person` al origen de datos. `addPerson` es el nombre del campo; `id`, `name` y `age` son los parámetros; y `Person` es el tipo de retorno. Volviendo al tipo de `Person`:

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

Hemos añadido el campo `occupation`. Sin embargo, no podemos establecer este campo directamente en `Occupation` porque los objetos no se pueden transferir como argumentos; son estrictamente tipos de salida. En lugar de ello, deberíamos pasar una entrada con los mismos campos que un argumento:

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

También podemos actualizar fácilmente nuestro `addPerson` para incluirlo como parámetro al crear nuevas instancias de `Person`:

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

Este es el esquema actualizado:

```
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
}
```

Tenga en cuenta que `occupation` transferirá el campo `title` desde `occupationInput` para completar la creación de la `Person` objeto en lugar del objeto `Occupation` original. Suponiendo que tengamos una implementación de resolución para `addPerson`, ahora podemos realizar una mutación real. Si bien el tipo de `Mutation` existe, debemos llamarlo explícitamente para que se ejecute en el código de la aplicación. Esto se puede hacer con la palabra clave `mutation`:

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

Esta mutación se llama `createPerson` y `addPerson` es la operación. Para crear una nueva `Person`, podemos introducir los argumentos para `id`, `name`, `age` y `occupation`. En el ámbito de `addPerson`, también podemos ver otros campos como `name`, `age`, etc. Esta es su respuesta; estos son los campos que se devolverán una vez finalizada la operación `addPerson`. Esta es la parte final del ejemplo:

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

```
}
```

Si se usa esta mutación, el resultado podría ser parecido a este:

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

Como puede ver, la respuesta ha devuelto los valores que solicitamos en el mismo formato que se definió en nuestra mutación. Se recomienda devolver todos los valores que se hayan modificado para reducir la confusión y la necesidad de realizar más consultas en el futuro. Las mutaciones permiten incluir varias operaciones dentro de su ámbito. Se ejecutarán secuencialmente en el orden indicado en la mutación. Por ejemplo, si creamos otra operación denominada `addOccupation` que agregue títulos de trabajo al origen de datos, podemos llamarla así en la mutación después de `addPerson`. `addPerson` se gestionará primero y, a continuación, `addOccupation`.

Subscriptions

Las suscripciones utilizan WebSockets para abrir una conexión bidireccional duradera entre el servidor y sus clientes. Normalmente, un cliente se suscribe o escucha al servidor. Siempre que el servidor realice un cambio en el lado del servidor o realice un evento, el cliente suscrito recibirá las actualizaciones. Este tipo de protocolo resulta útil cuando hay varios clientes suscritos y es necesario notificarles los cambios que se produzcan en el servidor o en otros clientes. Por ejemplo, las suscripciones se pueden utilizar para actualizar los feeds de las redes sociales. Puede haber dos usuarios, el usuario A y el usuario B, que estén suscritos a las actualizaciones de notificaciones automáticas cada vez que reciban mensajes directos. El usuario A del cliente A podría enviar un mensaje directo al usuario B del cliente B. El cliente del usuario A enviaría el mensaje directo, que sería procesado por el servidor. A continuación, el servidor enviaría el mensaje directo a la cuenta del usuario B y, al mismo tiempo, enviaría una notificación automática al cliente B.

Este es un ejemplo de `Subscription` que podríamos añadir al ejemplo del esquema:

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

El campo `personAdded` enviará un mensaje a los clientes suscritos cada vez que `Person` se añada una nueva al nuevo origen de datos. Suponiendo que tengamos una implementación de solucionador para `personAdded`, ahora podemos usar la suscripción. Si bien el tipo de `Subscription` existe, debemos llamarlo explícitamente para que se ejecute en el código de la aplicación. Esto se puede hacer con la palabra clave `subscription`:

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

La suscripción se llama `personAddedOperation` y la operación es `personAdded`. `personAdded` devolverá los campos `id` e `name` de las nuevas instancias de `Person`. Observando el ejemplo de la mutación, añadimos una operación `Person` utilizando esta operación:

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

Si nuestros clientes estaban suscritos a las actualizaciones de la `Person` recién añadida, es posible que vean lo siguiente cuando se ejecute `addPerson`:

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

A continuación se muestra un resumen de lo que ofrecen las suscripciones:

Las suscripciones son canales bidireccionales que permiten al cliente y al servidor recibir actualizaciones rápidas pero constantes. Por lo general, utilizan el protocolo WebSocket, que crea conexiones estandarizadas y seguras.

Las suscripciones son ágiles, ya que reducen la sobrecarga de configuración de la conexión. Una vez suscrito, un cliente puede seguir utilizando esa suscripción durante largos períodos de tiempo. Por lo general, utilizan los recursos informáticos de forma eficiente, ya que permiten a los desarrolladores personalizar la duración de la suscripción y configurar la información que se va a solicitar.

En general, las suscripciones permiten al cliente realizar varias suscripciones a la vez. Al pertenecer a AWS AppSync, las suscripciones solo se utilizan para recibir actualizaciones del servicio de AWS AppSync en tiempo real. No se pueden usar para realizar consultas o mutaciones.

La principal alternativa a las suscripciones es el sondeo, que envía consultas a intervalos establecidos para solicitar datos. Este proceso suele ser menos eficiente que las suscripciones y supone una gran carga tanto para el cliente como para el backend.

Lo que no se ha mencionado en nuestro ejemplo de esquema es que los tipos de objetos especiales también deben definirse en una raíz de `schema`. Por lo tanto, si exporta un esquema AWS AppSync, este podría tener este aspecto:

schema.graphql

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

.
.
.

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

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

Enumeraciones

Las enumeraciones son escalares especiales que limitan los argumentos legales que un tipo o campo puede tener. Esto significa que siempre que se defina una enumeración en el esquema, su tipo o campo asociado se limitará a los valores de la enumeración. Las enumeraciones se serializan como escalares de cadena. Tenga en cuenta que los diferentes lenguajes de programación pueden gestionar las enumeraciones de GraphQL de forma diferente. Por ejemplo, JavaScript no admite enumeraciones nativas, por lo que los valores de enumeración se pueden asignar a valores int.

Las enumeraciones se definen mediante la palabra clave `enum`: A continuación se muestra un ejemplo:

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

Al llamar a la enumeración `trafficLights`, los argumentos solo pueden ser `solidRed`, `solidYellow`, `solidGreen`, etc. Es habitual usar enumeraciones para representar cosas que tienen un número distinto pero limitado de opciones.

Uniones/interfaces

Consulte Interfaces y uniones en GraphQL.

Campos de GraphQL

Los campos existen dentro del ámbito de un tipo y contienen el valor que se solicita al servicio GraphQL. Se parecen mucho a las variables de otros lenguajes de programación. Por ejemplo, este es un tipo de objeto de `Person`:

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

```
}
```

En este caso, los campos son `name` y `age`, y contienen un valor `String` y `Int`, respectivamente. Los campos de objeto, como los que se muestran arriba, se pueden utilizar como entradas en los campos (operaciones) de las consultas y las mutaciones. Por ejemplo, consulte la `Query` siguiente:

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

El campo `people` solicita todas las instancias de `Person` del origen de datos. Cuando agrega o recupera una `Person` en su servidor de GraphQL, puede esperar que los datos sigan el formato de sus tipos y campos, es decir, la estructura de sus datos en el esquema determina cómo se estructurarán en su respuesta:

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

Los campos desempeñan un rol importante en la estructuración de los datos. Hay un par de propiedades adicionales que se explican a continuación y que se pueden aplicar a los campos para una mayor personalización.

Lists

Las listas devuelven todos los elementos de un tipo específico. Se puede agregar una lista al tipo de campo mediante corchetes `[]`:

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

En `Query`, los corchetes que rodean a `Person` indican que desea devolver todas las instancias de `Person` del origen de datos en forma de matriz. En la respuesta, los valores `name` y `age` de cada `Person` se devolverán como una lista única y delimitada:

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

No está limitado a los tipos de objetos especiales. También puede utilizar listas en los campos de los tipos de objetos normales.

Valores no nulos

Los valores no nulos indican un campo que no puede ser nulo en la respuesta. Para establecer un campo como no nulo, utilice el símbolo `!`:

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

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

El campo `name` no puede ser explícitamente nulo. Si consultara el origen de datos y proporcionara una entrada nula para este campo, se generaría un error.

Puede combinar listas y valores no nulos. Compare estas consultas:

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

.
.
.

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

.
.
.

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

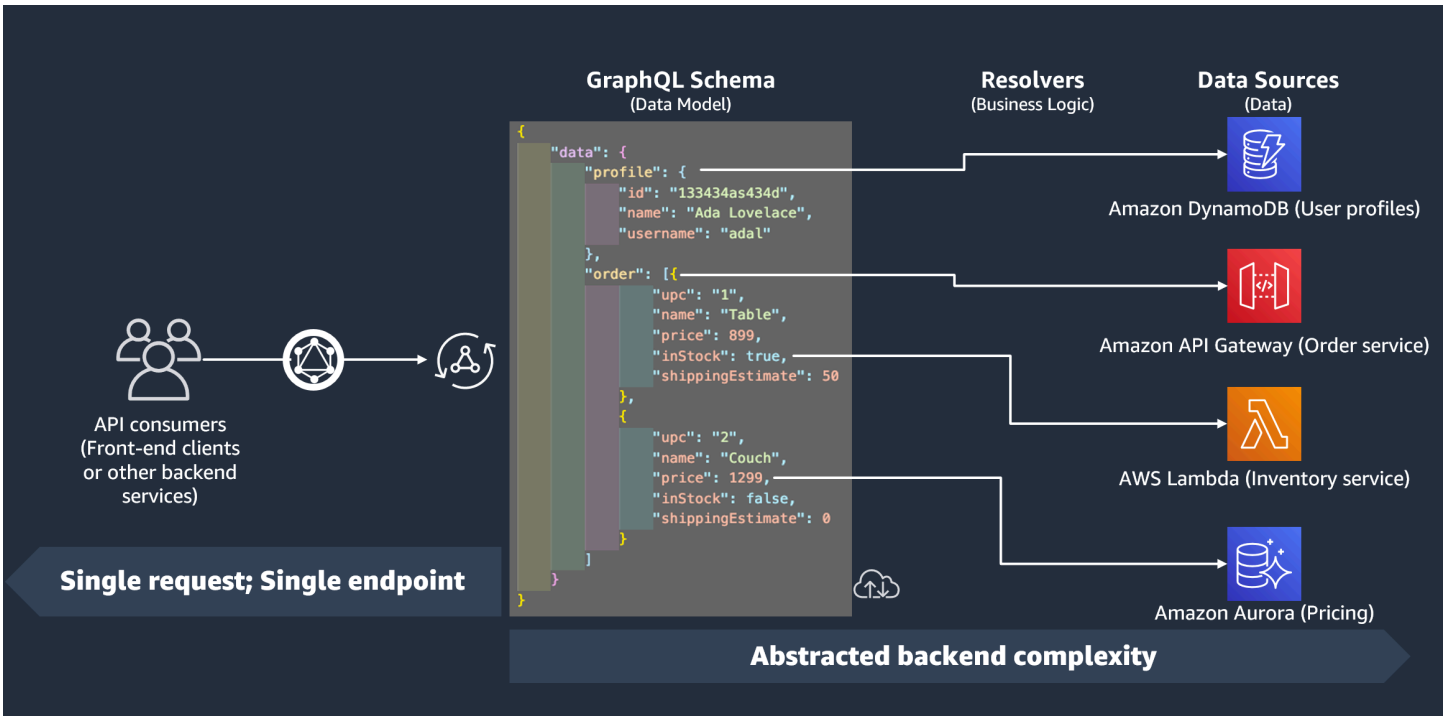
En el caso de uso 1, la lista no puede contener elementos nulos. En el caso de uso 2, la lista en sí no se puede establecer como nula. En el caso de uso 3, la lista y sus elementos no pueden ser nulos. Sin embargo, en cualquier caso, aún puede devolver listas vacías.

Como puede ver, en GraphQL hay muchos componentes móviles. En esta sección, mostramos la estructura de un esquema simple y los diferentes tipos y campos que admite un esquema. En la sección siguiente, descubrirá los demás componentes de una API de GraphQL y cómo funcionan con el esquema.

Orígenes de datos

En la sección anterior, hemos aprendido que un esquema define la forma de los datos. Sin embargo, no llegamos a explicar de dónde procedían esos datos. En proyectos reales, el esquema es como

una puerta de enlace que gestiona todas las solicitudes realizadas al servidor. Cuando se realiza una solicitud, el esquema actúa como el único punto de conexión que interactúa con el cliente. El esquema accederá a los datos del origen de datos, los procesará y los retransmitirá al cliente. Vea la infografía siguiente:



AWS AppSync y GraphQL implementan magníficamente las soluciones Backend For Frontend (BFF). Funcionan en grupo para reducir la complejidad a gran escala al abstraer el backend. Si su servicio utiliza orígenes de datos o microservicios diferentes, para abstraer de manera básica parte de la complejidad, puede definir la forma de los datos de cada origen (subgráfico) en un único esquema (supergráfico). Esto significa que su API de GraphQL no se limita a usar un origen de datos. Puede asociar cualquier número de orígenes de datos a su API de GraphQL y especificar en su código cómo van a interactuar con el servicio.

Como puede ver en la infografía, el esquema de GraphQL contiene toda la información que los clientes necesitan para solicitar datos. Esto significa que todo se puede procesar en una sola solicitud en lugar de en varias, como es el caso con REST. Estas solicitudes pasan por el esquema, que es el único punto de conexión del servicio. Cuando se procesan las solicitudes, un solucionador (elemento que se explica en la sección siguiente) ejecuta su código para procesar los datos del origen de datos correspondiente. Cuando se devuelva la respuesta, el subgráfico vinculado al origen de datos se rellenará con los datos del esquema.

AWS AppSync admite muchos tipos de orígenes de datos diferentes. En la siguiente tabla, describiremos cada tipo, enumeraremos algunas de las ventajas de cada uno y proporcionaremos enlaces útiles para obtener más contexto.

Origen de datos	Descripción	Ventajas	Información complementaria

Amazon DynamoDB	“Amazon DynamoDB es un servicio de base de datos NoSQL totalmente administrado que ofrece un rendimiento rápido y predecible, así como una perfecta escalabilidad. DynamoDB le permite delegar las cargas administrativas que supone tener que utilizar y escalar bases de datos distribuidas, para que no tenga que preocuparse del aprovisionamiento, la instalación ni la configuración del hardware, ni tampoco de las tareas de replicación, aplicación de parches de software o escalado de clústeres. DynamoDB también ofrece el cifrado en reposo, que elimina la carga y la complejid	• Rendimiento a escala: DynamoDB está diseñado en torno a un rendimiento uniforme a cualquier escala. Esto es posible mediante el uso de particiones. DynamoDB particionará automáticamente las tablas en varias asignaciones que se almacenarán en múltiples SSD en varios nodos. Por lo general, esto aumentará el rendimiento de la red y reducirá la latencia.
		• Capacidad a escala: DynamoDB supervisa el tráfico y permite escalar automáticamente el rendimiento si
			• Documentación oficial de DynamoDB
			  • Particiones
			  • Escalado automático
			  • Tolerancia a errores
			  • Supervisión
			  • Seguridad
			• GraphQL y DynamoDB
			• Operaciones de solucionadores para DynamoDB
			• Modelo de precios

Origen de datos	Descripción	Ventajas	Información complementaria
	ad operativa que conlleva la protección de información confidencial.”	la red permanece sobrecargada durante períodos prolongados. • Disponibilidad y tolerancia a errores: DynamoDB es compatible con varias regiones aisladas físicamente, cada una de las cuales contiene varias zonas de disponibilidad aisladas físicamente. DynamoDB cambiará automáticamente a una zona de respaldo en caso de que se interrumpa el servicio. También puede hacer copias de seguridad y replicar los datos manualmente para garantizar la seguridad de estos. • Registro y monitorización: DynamoDB proporciona varias herramientas analíticas para	

Origen de datos	Descripción	Ventajas	Información complementaria
		las tablas. Puede monitorizar el rendimiento de la tabla y crear alarmas que le notifiquen los cambios drásticos que se produzcan en el servicio. • Seguridad: DynamoDB sigue protocolos estrictos para garantizar que sus datos cumplan con los requisitos de seguridad de su organización. • Integración con AWS AppSync: DynamoDB se integra perfectam ente con nuestro servicio. Puede crear nuevas tablas de DynamoDB y generar automátic amente un esquema a partir de ellas para agilizar el proceso de desarrollo. También ofrecemos toda una colección de	

Origen de datos	Descripción	Ventajas	Información complementaria
		operaciones para solicitar fácilmente datos de las tablas de DynamoDB existentes en su cuenta en su solucionador.	

Origen de datos	Descripción	Ventajas	Información complementaria

AWS Lambda	“AWS Lambda es un servicio informático que permite ejecutar código sin aprovisionar ni administrar servidores.
	
	Lambda ejecuta el código en una infraestructura de computación de alta disponibilidad y realiza todas las tareas de administración de los recursos de computación, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, así como las funciones de registro. Con Lambda, lo único que tiene que hacer es suministrar el código en uno de los tiempos de ejecución de lenguaje compatibles con Lambda.”
		• Modelo de pago por uso: Lambda solo le cobra cuando utiliza sus recursos. También le permiten escalar la cantidad de recursos utilizados en función de las necesidades de su aplicación.
		• Escalado automático: a veces, la aplicación puede necesitar más potencia de procesamiento para un proceso concreto. Lambda le permite escalar automáticamente los recursos informáticos para adaptarlos a las necesidades de su aplicación.
		• Tiempos de implementación más rápidos: puede optimizar su proceso de desarrollo mediante un paquete de
			• Documentación oficial
			  • Escalado
			  • Implementación
			  • Tiempos de ejecución
			• Tutorial sobre solucionadores de Lambda
			• Modelo de precios

Origen de datos	Descripción	Ventajas	Información complementaria
		implementación. Utilice un paquete para cargar el código de función en el servicio de Lambda. A continuación, puede utilizar sus entornos de ejecución para probar y ejecutar sus funciones. • Versatilidad: Lambda se puede utilizar en multitud de casos de uso. Puede integrar Lambda perfectam ente tanto con servicios de terceros como con servicios de AWS. Algunos ejemplos son los canales de CI/CD y los servicios de correo masivo. • Integración con AWS AppSync: puede invocar fácilmente las funciones de Lambda en su solucionador para	

Origen de datos	Descripción	Ventajas	Información complementaria
		gestionar solicitudes. Nuestro servicio proporciona una operación de solicitud simplificada para realizar llamadas Lambda. Permitimos tanto llamadas individuales como en lotes.	

Origen de datos	Descripción	Ventajas	Información complementaria
OpenSearch	“Amazon OpenSearch Service es un servicio administrado que facilita la implementación, la operación y el escalado de clústeres de OpenSearch en la nube de AWS. Amazon OpenSearch Service es compatible con OpenSearch y con la version heredada de Elasticsearch OSS (con hasta 7.10, la versión final de código abierto del software). Al crear un clúster, tiene la opción de elegir qué motor de búsqueda utilizar. OpenSearch es un motor de búsqueda y análisis totalmente de código abierto para casos de uso como análisis de registros, monitoreo de aplicaciones en tiempo real y análisis de secuencias de clics. Para más información, consulte	• Escalado: puede escalar fácilmente el servicio para adaptarlo a sus requisitos de servicio a través de OpenSearch Serverless. • Ingesta de datos: puede utilizar OpenSearch Ingestion para importar, procesar y analizar datos. Existen muchas aplicaciones para la ingesta de datos, que puede encontrar aquí. • Seguridad: OpenSearch puede gestionar su configuración de seguridad de AWS, que incluye IAM, CloudTrail, VPC, autenticación, etc. • Disponibilidad: OpenSearch también admite diversas regiones y zonas de disponibilidad en su servicio.	• Documentación oficial • Sin servidor • Modelo de precios

Origen de datos	Descripción	Ventajas	Información complementaria
	la documentación de OpenSearch. Amazon OpenSearch Service le proporciona todos los recursos para su clúster y lo inicia. También detecta y sustituye automáticamente los nodos de OpenSearch Service que tienen algún error. De este modo, reduce la sobrecarga asociada con las infraestructuras autoadministradas. Puede escalar el clúster con una única llamada a la API o con algunos clics en la consola.”	• Integración con AWS AppSync: en AWS AppSync, puede utilizar las API de GraphQL para almacenar y recuperar datos de los dominios de OpenSearch Service existentes en su cuenta.	

Origen de datos	Descripción	Ventajas	Información complementaria
Puntos de conexión HTTP	Puede usar puntos de conexión HTTP como orígenes de datos. AWS AppSync puede enviar solicitudes a los puntos de conexión con la información relevante, como los parámetros y la carga. La respuesta HTTP estará expuesta al solucionador, que devolverá la respuesta final cuando finalice sus operaciones.	• Útil para aplicaciones sencillas que no estén tan integradas con servicios como Lambda.	• Referencia de solucionadores

Origen de datos	Descripción	Ventajas	Información complementaria

Amazon EventBridge	“EventBridge es un servicio sin servidor que utiliza eventos para conectar los componentes de la aplicación entre sí, lo que facilita la creación de aplicaciones escalables basadas en eventos. Utilícelo para enrutar eventos desde orígenes como aplicaciones propias, servicios de AWS y software de terceros a aplicaciones de consumo en toda su organización. EventBridge proporciona una forma sencilla y coherente de incorporar, filtrar, transformar y entregar eventos para que pueda crear nuevas aplicaciones rápidamente.”	• Arquitectura basada en eventos: puede aprovechar la arquitectura basada en eventos.
		• Programación: puede utilizar el Programador de EventBridge para automatizar sus tareas y reglas mediante expresiones cron o establecer intervalos de tiempo como alternativa a los patrones de eventos.
		• Canalizaciones: con EventBridge Pipes, puede sustituir el bus de eventos por una canalización que incluya patrones de eventos de filtrado adicionales y enriquecimiento mediante transformaciones de datos antes de enviar el evento al destino.
			• Documentación oficial
			  • Canalizaciones
			  • Programador
			• Referencia de solucionadores
			• Modelo de precios

Origen de datos	Descripción	Ventajas	Información complementaria
		• Integración con AWS AppSync: AWS AppSync permite enviar eventos a buses de eventos mediante su solucionador.	

Origen de datos	Descripción	Ventajas	Información complementaria

Bases de datos relacionales	“Amazon Relational Database Service (Amazon RDS) es un servicio web que facilita la configuración, la operación y la escala de una base de datos relacional en la nube de AWS. Proporciona una capacidad rentable y de tamaño ajustable para una base de datos relacional estándar y se ocupa de las tareas de administración de bases de datos comunes.”	• Administrar de forma sencilla: periódicamente, RDS realiza el mantenimiento de sus recursos. En la mayoría de los casos, estas tareas de mantenimiento incluyen actualizaciones del hardware subyacente, del sistema operativo (SO) subyacente o de la versión del motor de base de datos de la instancia de base de datos. En circunstancias normales, puede decidir cuándo realizar las actualizaciones (excepto con los parches de seguridad, entre otros casos).
		• Recomendaciones: la característica de recomendaciones RDS proporciona sugerencias
			• Documentación oficial
			  • Características
			  • Mantenimiento
			  • Recomendaciones
			  • Opciones de almacenamiento
			  • Disponibilidad.
			  • Seguridad
			• Modelo de precios

Origen de datos	Descripción	Ventajas	Información complementaria
		automatizadas para solucionar posibles problemas en su instancia. • Disponibilidad: RDS está disponible en diferentes regiones físicas de todo el mundo. Puede distribuir fácilment e las necesidad es de su base de datos entre distintos nodos para ofrecer un mejor servicio a sus clientes. • Personalización: RDS está diseñado para cumplir con los requisitos de las grandes corporaci ones. RDS ofrece varias opciones de computación, implementación rápida, escalabil idad y almacenam iento. • Seguridad: RDS está integrado con varias herramien tas y servicios para mantener la	

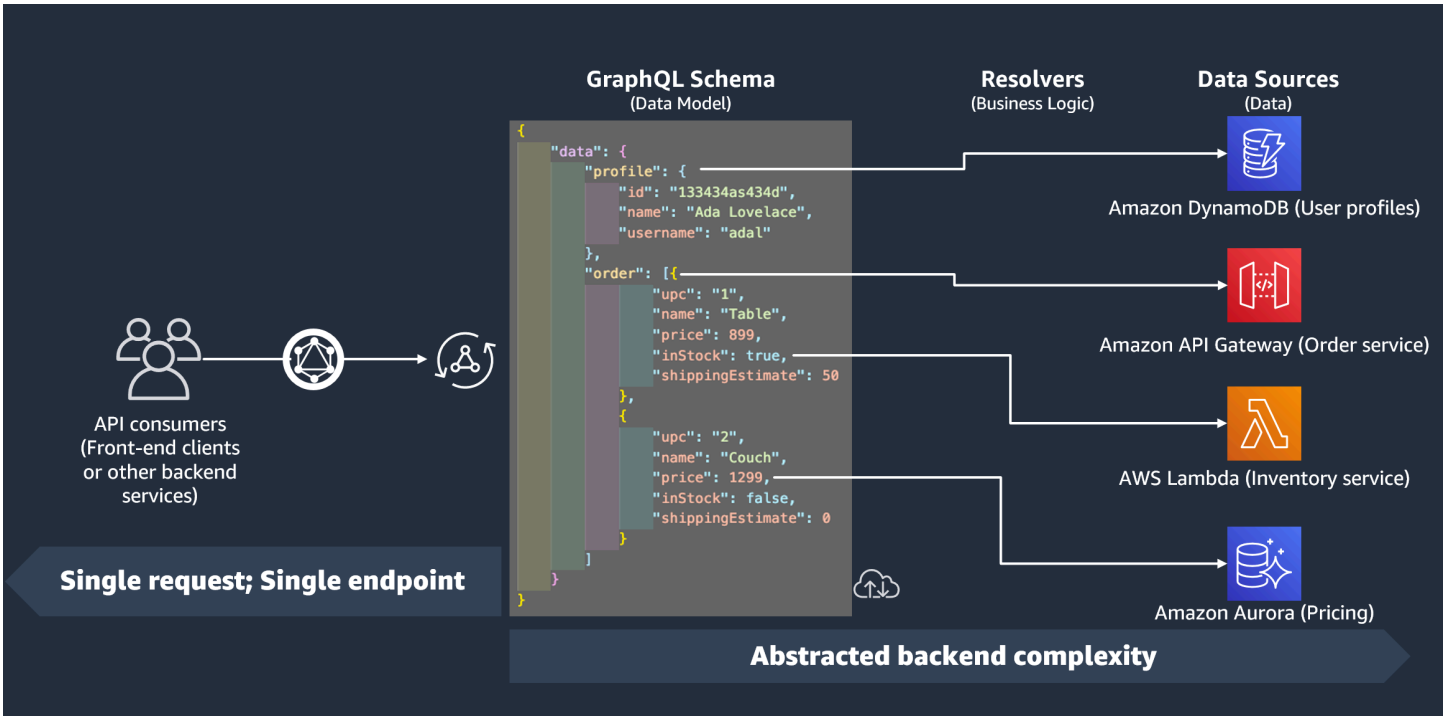
Origen de datos	Descripción	Ventajas	Información complementaria
		seguridad de las bases de datos en los niveles de usuario, base de datos y red. • Integración con AWS AppSync: si busca una solución de backend avanzada, AWS AppSync le permite enviar, procesar, almacenar y devolver datos utilizando su instancia como origen de datos.	
Origen de datos none	Si no planea usar un servicio de origen de datos, puede configurarlo en none. Un origen de datos none, aunque se siga considerando explícitamente un origen de datos, no es un medio de almacenamiento. A pesar de ello, sigue siendo útil en algunos casos para la manipulación y transferencia de datos.	• Potencialmente útil para acciones como la conversión de datos • Útil cuando se resuelve algo localmente	• Referencia de solucionadores

> **Tip**
>
> Para obtener más información sobre cómo interactúan los orígenes de datos con AWS AppSync, consulte Asociar un origen de datos.

Solucionadores

En las secciones anteriores, ha aprendido acerca de los componentes del esquema y el origen de datos. Ahora, debemos abordar cómo interactúan el esquema y los orígenes de datos. Todo comienza con el solucionador.

Un solucionador es una unidad de código que gestiona cómo se resolverán los datos de ese campo cuando se realice una solicitud al servicio. Los solucionadores se adjuntan a campos específicos dentro de los tipos de su esquema. Por lo general, se utilizan para implementar las operaciones de cambio de estado de las operaciones de los campos de consulta, mutación y suscripción. El solucionador procesará la solicitud del cliente y, a continuación, devolverá el resultado, que puede ser un grupo de tipos de salida, como objetos o escalares:



Tiempo de ejecución del solucionador

En AWS AppSync, primero debe especificar un tiempo de ejecución para el solucionador. El tiempo de ejecución de un solucionador indica el entorno en el que este se ejecuta. También indica el idioma

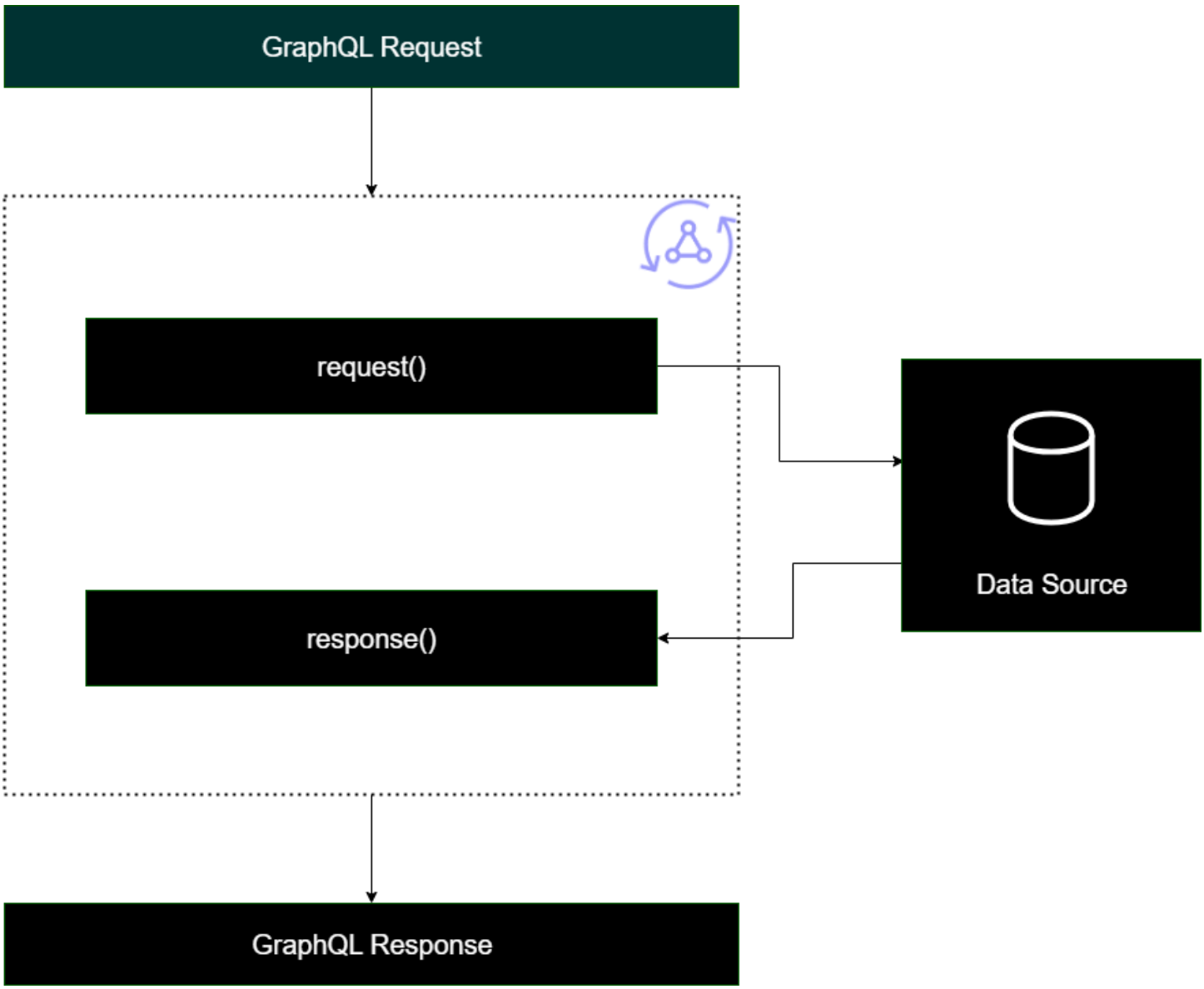
en el que se escribirán los solucionadores. Actualmente, AWS AppSync admite APPSYNC_JS para JavaScript y Velocity Template Language (VTL). Consulte las características de tiempo de ejecución de JavaScript para ver los solucionadores y las funciones para JavaScript o la Referencia de la utilidad de plantillas de asignación de solucionadores para VTL.

Estructura del solucionador

En cuanto al código, los solucionadores se pueden estructurar de dos maneras. Hay solucionadores de unidad y de canalización.

Solucionadores de unidad

Un solucionador de unidad se compone de código que define un solo controlador de solicitudes y respuestas que se ejecutan con un origen de datos. El controlador de solicitudes toma un objeto de contexto como argumento y devuelve la carga de la solicitud utilizada para llamar al origen de datos. El controlador de respuestas recibe una carga útil del origen de datos con el resultado de la solicitud ejecutada. El controlador de respuestas transforma la carga útil en una respuesta de GraphQL para resolver el campo de GraphQL.



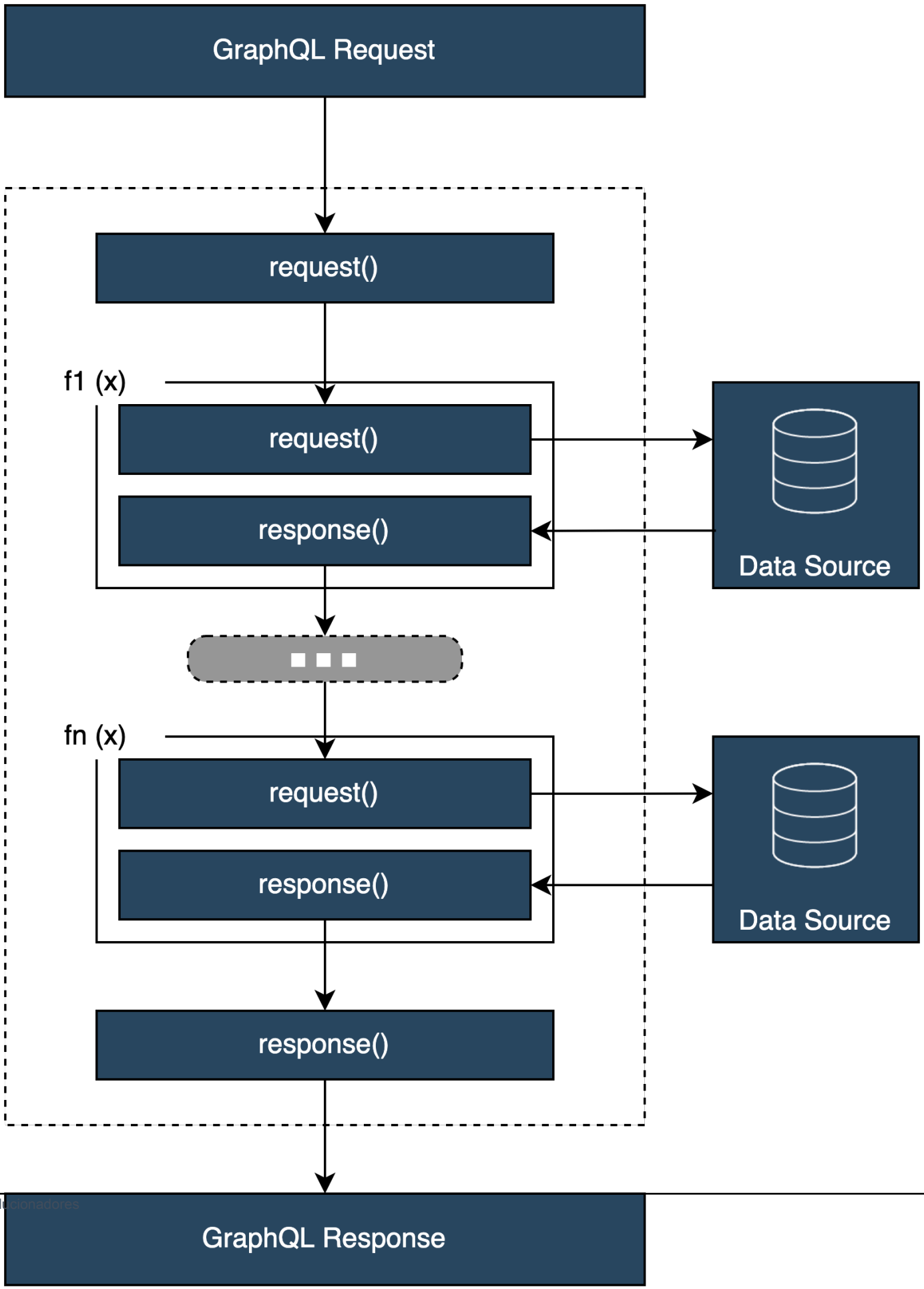
Solucionadores de canalización

Al implementar los solucionadores de canalización, estos siguen una estructura general:

- Paso anterior: cuando el cliente realiza una solicitud, los datos de esta se envían a los solucionadores de los campos de esquema que se usan (normalmente, las consultas, las mutaciones y las suscripciones). El solucionador empezará a procesar los datos de la solicitud con un controlador previo paso a paso, que permite realizar algunas operaciones de preprocesamiento antes de que los datos pasen por el solucionador.
- Funciones: una vez ejecutado el paso anterior, la solicitud se pasa a la lista de funciones. La primera función de la lista se ejecuta conforme al origen de datos. Una función es un subconjunto

del código de su solucionador, que contiene su propio controlador de solicitudes y respuestas. Un controlador de solicitudes toma los datos de la solicitud y realiza operaciones con el origen de datos. El controlador de respuestas procesa la respuesta del origen de datos antes de devolverla a la lista. Si hay más de una función, los datos de la solicitud se envían a la siguiente función de la lista que se ejecutará. Las funciones de la lista se ejecutan en serie en el orden definido por el desarrollador. Una vez ejecutadas todas las funciones, el resultado final se envía al paso posterior.

- Paso posterior: el paso posterior es una función de controlador que permite realizar algunas operaciones finales en la respuesta de la función final antes de pasarla a la respuesta de GraphQL.


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

Estructura del controlador del solucionador

Los controladores suelen ser funciones denominadas `Request` y `Response`:

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

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

En un solucionador de unidad, solo habrá un conjunto de estas funciones. En un solucionador de canalización, habrá un conjunto de estas funciones para el paso anterior y posterior y un conjunto adicional para cada función. Para visualizar cómo podría verse esto, revisemos un simple tipo `Query`:

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

Se trata de una consulta sencilla con un campo denominado `helloWorld` de tipo `String`. Supongamos que siempre queremos que este campo devuelva la cadena “Hello World”. Para implementar este comportamiento, necesitamos añadir el solucionador a este campo. En un solucionador de unidad, podríamos añadir algo parecido a esto:

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

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

La `request` puede dejarse en blanco porque no vamos a solicitar ni procesar datos. También podemos suponer que nuestro origen de datos es `None`, lo que indica que este código no necesita realizar ninguna invocación. La respuesta simplemente devuelve “Hello World”. Para probar este solucionador, necesitamos realizar una solicitud utilizando el tipo de consulta:

```
query helloWorldTest {
```

```
  helloWorld
}
```

Esta es una consulta llamada `helloWorldTest` que devuelve el campo `helloWorld`. Cuando se ejecuta, el solucionador del campo `helloWorld` también ejecuta y devuelve la respuesta:

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

Devolver constantes como esta es lo más sencillo que puede hacer. En realidad, devolverá entradas, listas y más. Este es un ejemplo más complicado:

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

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

Aquí devolvemos una lista de `Books`. Supongamos que utilizamos una tabla de DynamoDB para almacenar datos de libros. Nuestros controladores pueden tener un aspecto similar al siguiente:

```
/**
 * 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;
}
```

Nuestra solicitud ha utilizado una operación de escaneo integrada para buscar todas las entradas de la tabla, ha almacenado los resultados en el contexto y, a continuación, los ha transferido a la respuesta. La respuesta ha tomado los elementos resultantes y los ha devuelto dentro de ella:

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

        ...

      ]
    }
  }
}
```

Contexto del solucionador

En un solucionador, cada paso de la cadena de controladores debe conocer el estado de los datos de los pasos anteriores. El resultado de un controlador se puede almacenar y transferir a otro como argumento. GraphQL define cuatro argumentos básicos del solucionador:

Argumentos básicos del solucionador	Descripción
`obj`, `root`, `parent`, etc.:	El resultado del nivel superior.
`args`	Los argumentos proporcionados al campo en la consulta GraphQL.
`context`	Un valor que se proporciona a todos los solucionadores y que contiene informaci ón contextual importante, como el usuario

Argumentos básicos del solucionador	Descripción
	registrado actualmente o el acceso a una base de datos.
`info`	Un valor que contiene información específica del campo relevante para la consulta actual, así como los detalles del esquema.

En AWS AppSync, el argumento `context` (ctx) puede contener todos los datos mencionados anteriormente. Es un objeto que se crea por solicitud y contiene datos como las credenciales de autorización, los datos de los resultados, los errores, los metadatos de la solicitud, etc. El contexto facilita a los programadores la manipulación de los datos procedentes de otras partes de la solicitud. Vuelva a leer este fragmento de código:

```
/**
 * 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;
}
```

La solicitud recibe el contexto (ctx) como argumento; este es el estado de la solicitud. Realiza un análisis de todos los elementos de una tabla y, a continuación, vuelve a almacenar el resultado en el contexto en `result`. A continuación, el contexto se transfiere al argumento de respuesta, que accede al `result` y devuelve su contenido.

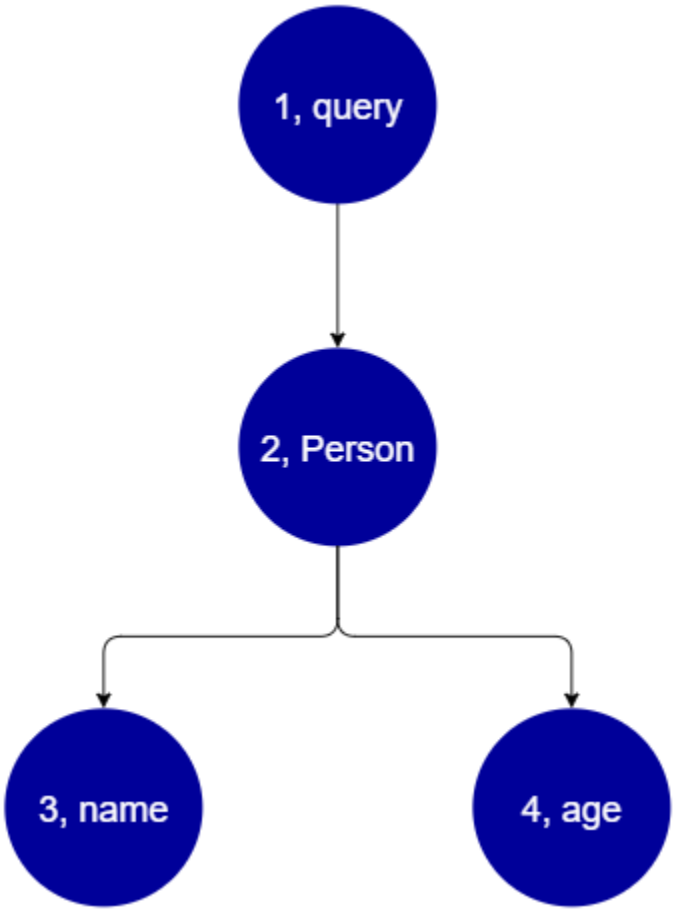
Solicitudes y análisis

Cuando realiza una consulta a su servicio GraphQL, este debe pasar por un proceso de análisis y validación antes de ejecutarse. Su solicitud se analizará y traducirá en un árbol sintáctico abstracto. El contenido del árbol se valida mediante la ejecución de varios algoritmos de validación en función

de su esquema. Tras el paso de validación, se recorren y procesan los nodos del árbol. Se invoca a los solucionadores, los resultados se almacenan en el contexto y se devuelve la respuesta. Tomemos por ejemplo esta consulta:

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

Devolvemos `Person` con los campos `name` y `age`. Al ejecutar esta consulta, el árbol tendrá un aspecto parecido al siguiente:



En el árbol, parece que esta solicitud buscará la raíz para la `Query` en el esquema. Dentro de la consulta, se resolverá el campo `Person`. Por los ejemplos anteriores, sabemos que podría ser una entrada del usuario, una lista de valores, etc. Lo más probable es que la `Person` esté vinculada a un tipo de objeto que contenga los campos que necesitamos (`name` y `age`). Cuando se encuentran

estos dos campos secundarios, se resuelven en el orden indicado (`name` y después `age`). Cuando el árbol esté completamente resuelto, la solicitud estará completa y se devolverá al cliente.

Propiedades adicionales de GraphQL

GraphQL consta de varios principios de diseño para mantener la simplicidad y la solidez a escala.

Declarativo

GraphQL es declarativo, lo que significa que el usuario describirá (dará forma) a los datos declarando únicamente los campos que desee consultar. La respuesta solo devolverá los datos de estas propiedades. Por ejemplo, esta es una operación que recupera un objeto `Book` de una tabla de DynamoDB con el valor de `id` ISBN 13 de *`9780199536061:`*

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

La respuesta devolverá los campos de la carga (`name`, `year` y `author`) y nada más:

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

Gracias a este principio de diseño, GraphQL elimina los continuos problemas de obtención excesiva e insuficiente de datos a las que se enfrentan las API de REST en los sistemas complejos. Esto se traduce en una recopilación de datos más eficiente y en un mejor rendimiento de la red.

Jerárquico

GraphQL es flexible en el sentido de que el usuario puede configurar los datos solicitados para adaptarlos a las necesidades de la aplicación. Los datos solicitados siempre siguen los tipos y la sintaxis de las propiedades definidas en la API de GraphQL. Por ejemplo, en el siguiente fragmento de código se muestra la operación de `getBook` con un nuevo ámbito de campo denominado `quotes` que devuelve todas las cadenas de citas almacenadas y las páginas enlazadas al `Book` *`9780199536061`*:

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

Al ejecutar esta consulta, se devuelve el resultado siguiente:

```
{
  "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
          }
      ]
    }
  }
}
```

Como puede ver, los campos `quotes` vinculados al libro solicitado se han devuelto como una matriz con el mismo formato descrito en nuestra consulta. Aunque no se ha mostrado aquí, GraphQL tiene la ventaja adicional de no ser exigente en cuanto a la ubicación de los datos que recupera. Los `Books` y las `quotes` podría almacenarse por separado, pero GraphQL seguirá recuperando la información mientras exista la asociación. Esto significa que su consulta puede recuperar múltiples datos independientes en una sola solicitud.

Introspectivo

GraphQL se documenta automáticamente o es declarativo. Admite varias operaciones integradas que permiten a los usuarios ver los tipos y campos subyacentes dentro del esquema. Por ejemplo, este es un tipo `Foo` con un campo `date` y `description`:

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

Podríamos usar la operación `_type` para buscar los metadatos de escritura que se encuentran debajo del esquema:

```
{
  __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
      }
    }
  }
}
```

Se devolverá una respuesta:

```
{
  "__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
      },
    ]
  }
}
```

Esta característica se puede utilizar para averiguar qué tipos y campos admite un esquema de GraphQL en particular. GraphQL admite una amplia variedad de estas operaciones introspectivas. Para obtener más información, consulte Introspección.

Tipado fuerte

GraphQL admite el tipado fuerte a través de su sistema de tipos y campos. Cuando defina algo en su esquema, debe tener un tipo que pueda validarse antes de su tiempo de ejecución. También debe seguir la especificación de sintaxis de GraphQL. Este concepto no es diferente de la programación en otros lenguajes. Por ejemplo, este es el tipo `Foo` de antes:

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

Podemos ver que ese `Foo` es el objeto que se va a crear. Dentro de una instancia de `Foo`, habrá un campo `date` y otro `description`, ambos del tipo primitivo (escalar) `String`. Sintácticamente, vemos que `Foo` se ha declarado y que sus campos están dentro de su ámbito. Esta combinación de comprobación de tipos y sintaxis lógica garantiza que la API de GraphQL sea concisa y evidente por sí misma. La especificación de tipos y sintaxis de GraphQL se encuentra aquí.

Introducción: Cómo crear su primera API de GraphQL

Puede usar la consola de AWS AppSync para configurar y lanzar una API de GraphQL. Las API de GraphQL suelen requerir tres componentes:

1. Esquema de GraphQL: tu esquema de GraphQL es la base para crear la API. Define los tipos y campos que puede solicitar al ejecutar una operación. Para rellenar el esquema con datos, debe conectar los orígenes de datos a la API de GraphQL. En esta guía rápida, crearemos un esquema utilizando un modelo predefinido.
2. Orígenes de datos: son los recursos que contienen los datos para rellenar tu API de GraphQL. Puede ser una tabla de DynamoDB, una función de Lambda, etc. AWS AppSync admite diversos orígenes de fuentes de datos para crear API de GraphQL robustas y escalables. Los orígenes de datos están vinculados a los campos del esquema. Siempre que se realiza una solicitud en un campo, los datos del origen rellenan el campo. Este mecanismo lo controla el solucionador. En esta guía rápida, crearemos un origen de datos utilizando un modelo predefinido junto con el esquema.
3. Solucionadores: los solucionadores se ocupan de vincular el campo del esquema al origen de datos. Recuperan los datos del origen y, a continuación, devuelven el resultado en función de lo que haya definido el campo AWS. AppSync es compatible tanto con JavaScript como con VTL para escribir solucionadores para las API de GraphQL. En esta guía rápida, los solucionadores se generarán automáticamente en función del esquema y el origen de datos. No vamos a tratar este tema en profundidad en esta sección.

AWS AppSync admite la creación y la configuración de todos los componentes de GraphQL. Al abrir la consola, puede usar los siguientes métodos para crear su API:

1. Diseñar una API de GraphQL personalizada generándola mediante un modelo predefinido y configurando una nueva tabla de DynamoDB (origen de datos) para respaldarla.
2. Diseñar una API de GraphQL con un esquema en blanco y sin orígenes de datos ni solucionadores.
3. Usar una tabla de DynamoDB para importar datos y generar los tipos y campos de su esquema.
4. Usar las capacidades de WebSocket de AWS AppSync y la arquitectura Pub/Sub para desarrollar API en tiempo real.
5. Usar las API de GraphQL (API de origen) existentes para vincularlas a una API fusionada.

> **Note**
>
> Es aconsejable consultar la sección Diseñar un esquema antes de trabajar con herramientas más avanzadas. En estas guías se explican ejemplos más sencillos que puede utilizar como base teórica para crear aplicaciones más complejas en AWS AppSync.

AWS AppSync también admite varias opciones que no son de consola para crear API de GraphQL. Entre ellas se incluyen:

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

En el siguiente ejemplo, se muestra cómo crear los componentes básicos de una API de GraphQL mediante modelos predefinidos y DynamoDB.

Temas

- Paso 1: lanzar un esquema
- Paso 2: explorar la consola
- Paso 3: agregar datos con una mutación de GraphQL
- Paso 4: recuperar datos con una consulta de GraphQL
- Secciones complementarias

Paso 1: lanzar un esquema

En este ejemplo, creará una API Todo que permitirá a los usuarios crear elementos Todo como recordatorios de tareas diarias, como *`terminar un trabajo`* o *`pasar a recoger la compra`*. Esta API muestra cómo utilizar las operaciones de GraphQL cuando el estado persiste en una tabla de DynamoDB.

A nivel teórico, hay tres pasos principales para crear su primera API de GraphQL. Debe definir el esquema (tipos y campos), asociar los orígenes de datos a los campos y, a continuación, escribir el solucionador que gestiona la lógica empresarial. Sin embargo, la experiencia con la consola sugiere

que es mejor cambiar este orden. Empezaremos por definir cómo queremos que nuestro origen de datos interactúe con nuestro esquema y, más adelante, definiremos el esquema y el solucionador.

Para crear su API de GraphQL

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
2. En el Dashboard (Panel), elija Create API (Crear API).
3. Mantenga seleccionada la API de GraphQL y elija Diseñar desde cero. A continuación, haga clic en Next.
4. En el Nombre de la API, cambie el nombre rellenado previamente a **`Todo API`**. Después, seleccione Siguiente.

   > **Note**
   >
   > Hay más opciones presentes aquí, pero no las usaremos en este ejemplo.

5. En la sección Especificar recursos de GraphQL, haga lo siguiente:

   a. Seleccione Crear tipo respaldado por una tabla de DynamoDB ahora.

      > **Note**
      >
      > Esto significa que vamos a crear una nueva tabla de DynamoDB para asociarla como origen de datos.

   b. En el campo Nombre del modelo, escriba **`Todo`**.

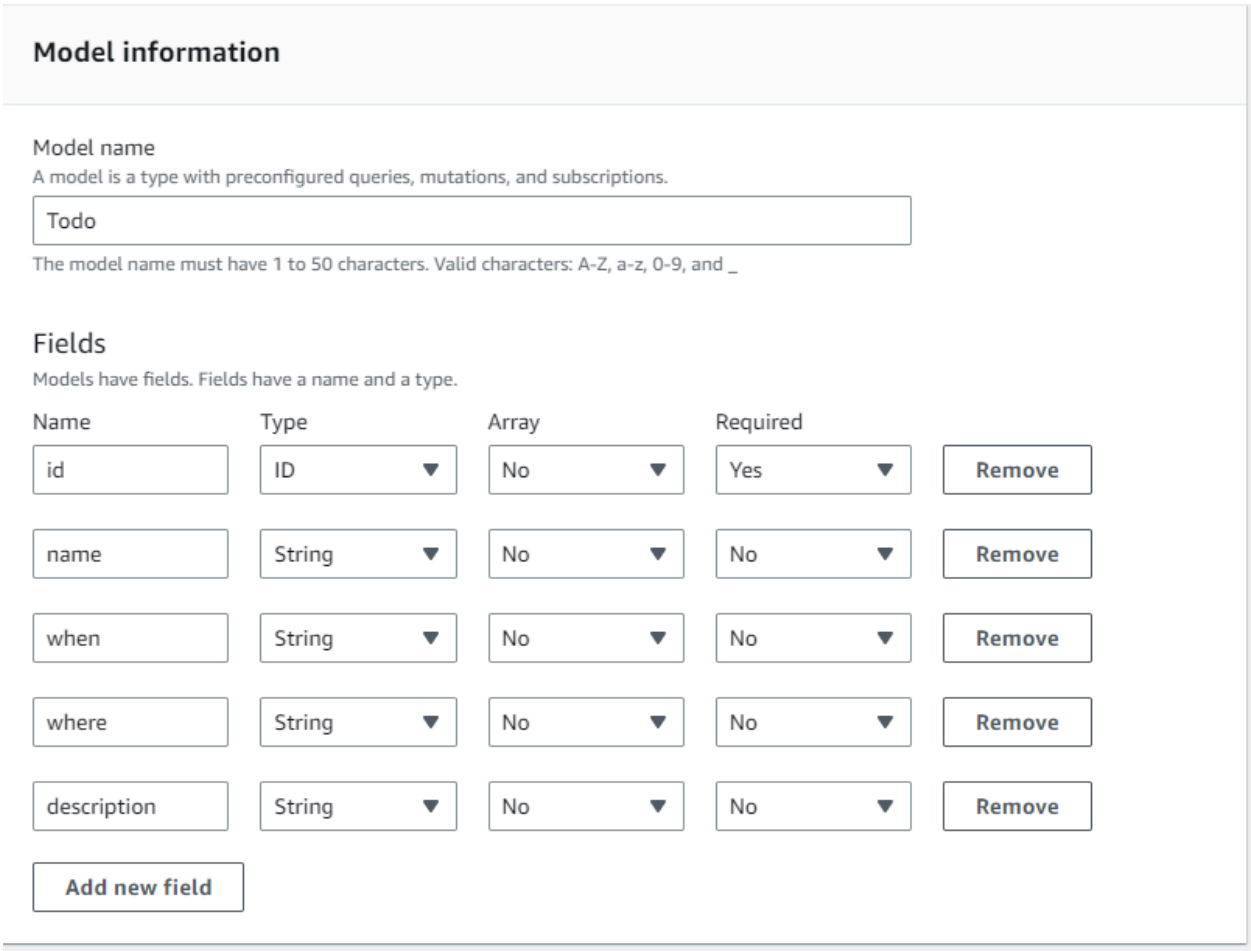
      > **Note**
      >
      > Nuestro primer requisito es definir nuestro esquema. Este Nombre del modelo será el nombre del tipo, de modo que lo que realmente está haciendo es crear un `type` llamado `Todo` que existirá en el esquema:
      >
      > ```
      > type Todo {}
      > ```

   c. En Campos, haga lo siguiente:

      i. Cree un campo llamado **`id`**, con el tipo `ID` y el valor obligatorio establecido en `Yes`.

> **Note**
>
> Estos son los campos que existirán dentro del ámbito de su tipo `Todo`. Aquí, el nombre de su campo se llamará `id` con un tipo de `ID!`:
>
> ```
> type Todo {
>  id: ID!
> }
> ```
>
> AWS AppSync admite varios valores escalares para distintos casos de uso.

ii. Con Agregar nuevo campo, cree cuatro campos adicionales con los valores `Name` definidos como **`name`**, **`when`**, **`where`** y **`description`**. Sus valores `Type` serán `String`, y los valores `Array` y `Required` se establecerán en `No`. Tendrá un aspecto similar al siguiente:



> **Note**
>
> El tipo completo y sus campos tendrán un aspecto similar al siguiente:
>
> ```
> type Todo {
>  id: ID!
>  name: String
>  when: String
>  where: String
>  description: String
> }
> ```
>
> Como vamos a crear un esquema con este modelo predefinido, también se rellenará con varias mutaciones reutilizables basadas en el tipo (por ejemplo, `create`, `delete` y `update`) para ayudarle a rellenar el origen de datos fácilmente.

d. En Configurar la tabla del modelo, introduzca un nombre de tabla, como **`TodoAPITable`**. Establezca la clave principal en `id`.

> **Note**
>
> Básicamente, estamos creando una nueva tabla de DynamoDB llamada *`TodoAPITable`* que se asociará a la API como nuestro origen de datos principal. Nuestra clave principal se establece en el campo obligatorio `id` que definimos anteriormente. Tenga en cuenta que esta nueva tabla está en blanco y no contiene nada excepto la clave de partición.

e. Elija Next (Siguiente).

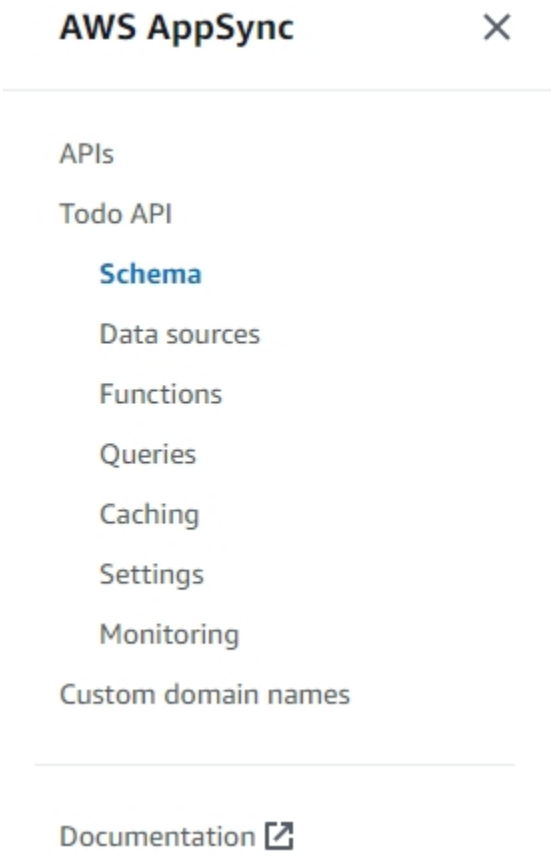
6. Revise los cambios y seleccione Crear API. Espere un momento a que el servicio AWS AppSync termine de crear su API.

Ha creado correctamente una API de GraphQL con su esquema y su origen de datos de DynamoDB. Para resumir los pasos anteriores, hemos creado una API de GraphQL completamente nueva. Hemos definido el nombre de la API y, a continuación, hemos agregado nuestra definición de esquema agregando nuestro primer tipo. Hemos definido el tipo y sus campos y, a continuación,

hemos optado por asociar un origen de datos a uno de los campos creando una nueva tabla de DynamoDB sin datos.

Paso 2: explorar la consola

Antes de agregar datos a nuestra tabla de DynamoDB, debemos revisar las características básicas de la experiencia de la consola de AWS AppSync. La pestaña de consola de AWS AppSync en la parte izquierda de la página permite a los usuarios navegar fácilmente a cualquiera de los principales componentes u opciones de configuración que proporciona AWS AppSync:



Diseñador del esquema

Seleccione Esquema para ver el esquema que acaba de crear. Si revisa el contenido del esquema, verá que ya incluye un gran número de operaciones auxiliares para agilizar el proceso de desarrollo. En el editor de esquemas, si se desplaza por el código, llegará finalmente al modelo que definió en la sección anterior:

```
type Todo {
```

```
  id: ID!
  name: String
  when: String
  where: String
  description: String
}
```

Su modelo ha pasado a ser el tipo en el que se ha basado la creación de todo el esquema. Empezaremos a agregar datos a nuestro origen de datos mediante mutaciones que se han generado automáticamente a partir de este tipo.

Estos son algunos consejos y datos adicionales sobre el editor de esquemas:

1. El editor de código tiene funcionalidades de análisis y de comprobación de errores que puede utilizar al escribir sus propias aplicaciones.
2. En la parte derecha de la consola se muestran los tipos de GraphQL que se han creado y los solucionadores en diferentes tipos de nivel superior, como, por ejemplo, las consultas.
3. Al agregar nuevos tipos a un esquema (por ejemplo, `type User {...}`), puede hacer que AWS AppSync aprovisione los recursos de DynamoDB en su lugar. Entre estos se incluyen la clave principal, la clave de clasificación y el diseño del índice adecuados que coincidan mejor con su patrón de acceso a los datos de GraphQL. Si elige Create Resources (Crear recursos) en la parte superior y elige uno de estos tipos definidos por el usuario en el menú, puede elegir diferentes opciones de campo en el diseño de esquema. Hablaremos de esto en la sección de diseño de un esquema.

Configuración del solucionador

En el diseñador de esquemas, la sección Solucionadores contiene todos los tipos y campos del esquema. Si se desplaza por la lista de campos, verá que puede adjuntar solucionadores a determinados campos si selecciona Asociar. De este modo, se abrirá un editor de código en el que podrá escribir el código de su solucionador. AWS AppSync admite tiempos de ejecución de VTL y JavaScript, que se pueden cambiar en la parte superior de la página seleccionando Acciones y, a continuación, Actualice el tiempo de ejecución. En la parte inferior de la página, también puede crear funciones que ejecutarán varias operaciones en secuencia. Sin embargo, los solucionadores son un tema avanzado y no lo trataremos en esta sección.

Orígenes de datos

Seleccione Orígenes de datos para ver su tabla de DynamoDB. Al elegir la opción `Resource` (si está disponible), puede ver la configuración de su origen de datos. En nuestro ejemplo, esto lleva a la consola DynamoDB. Desde allí, puede editar sus datos. También puede editar directamente algunos de los datos seleccionando el origen de datos y, a continuación, seleccionando Editar. Si alguna vez necesita eliminar el origen de datos, puede elegir el origen de datos y, a continuación, seleccionar Eliminar. Por último, puede crear nuevos orígenes de datos. Para ello, seleccione Crear origen de datos y, a continuación, configure el nombre y el tipo. Tenga en cuenta que esta opción sirve para vincular el servicio AWS AppSync a un recurso existente. Aún queda pendiente crear el recurso en su cuenta mediante el servicio correspondiente para que AWS AppSync lo reconozca.

Consultas

Seleccione Consultas para ver sus consultas y mutaciones. Cuando creamos nuestra API de GraphQL con nuestro modelo, AWS AppSync generó automáticamente algunas mutaciones y consultas auxiliares con el fin de realizar pruebas. En el editor de consultas, en el lado izquierdo se encuentra el Explorador. Esta es una lista que muestra todas sus mutaciones y consultas. Aquí puede activar fácilmente las operaciones y los campos que desea utilizar haciendo clic en los valores de sus nombres. Esto hará que el código aparezca automáticamente en la parte central del editor. Aquí puede editar sus mutaciones y consultas modificando los valores. En la parte inferior del editor, tiene el editor de Variables de consulta, que le permite introducir los valores de los campos para las variables de entrada de sus operaciones. Al seleccionar Ejecutar en la parte superior del editor, aparecerá una lista desplegable para seleccionar la consulta o mutación que se va a ejecutar. El resultado de esta ejecución aparecerá en el lado derecho de la página. Al volver a la sección Explorador de la parte superior, puede elegir una operación (Consulta, Mutación o Suscripción) y, a continuación, elegir el símbolo + para agregar una nueva instancia de esa operación en particular. En la parte superior de la página, habrá otra lista desplegable que contiene el modo de autorización para la ejecución de sus consultas. Sin embargo, no abordaremos esa característica en esta sección (para obtener más información, consulte Seguridad).

Configuración

Seleccione Ajustes para ver algunas opciones de configuración de su API de GraphQL. Aquí puede habilitar algunas opciones, como el registro, el rastreo y la funcionalidad de firewall de aplicaciones web. También puede agregar nuevos modos de autorización para proteger sus datos de filtraciones no deseadas al público en general. Sin embargo, estas opciones son más avanzadas y no las abordaremos en esta sección.

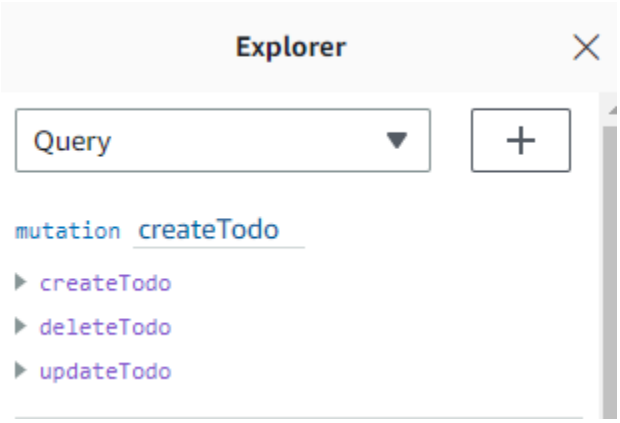
> **Note**
>
> Nota: El modo de autorización predeterminado, `API_KEY`, utiliza una clave de API para probar la aplicación. Esta es la autorización básica que se otorga a todas las API de GraphQL recién creadas. Es aconsejable utilizar un método de producción diferente. Para ilustrar el ejemplo que nos ocupa, solo usaremos la clave de API. Para obtener más información acerca de los modos de autorización, consulte Seguridad.

Paso 3: agregar datos con una mutación de GraphQL

El siguiente paso es agregar datos a su tabla de DynamoDB, que está vacía, mediante una mutación de GraphQL. Las mutaciones son uno de los tipos de operaciones fundamentales en GraphQL. Se definen en el esquema y permiten manipular los datos del origen de datos. En cuanto a las API de REST, son muy similares a operaciones como `PUT` o`POST`.

Para agregar su origen de datos

1. Si aún no lo ha hecho, inicie sesión en la AWS Management Console y abra la consola de AppSync.
2. Seleccione la API de la tabla.
3. En la pestaña de la izquierda, seleccione Consultas.
4. En la pestaña del Explorador situada a la izquierda de la tabla, debería ver varias mutaciones y consultas ya definidas en el editor de consultas:



> **Note**
>
> De hecho, esta mutación se encuentra en su esquema como el tipo `Mutation`. Tiene el código:
>
> ```
> type Mutation {
>  createTodo(input: CreateTodoInput!): Todo
>  updateTodo(input: UpdateTodoInput!): Todo
>  deleteTodo(input: DeleteTodoInput!): Todo
> }
> ```
>
> Como puede ver, las operaciones aquí son similares al contenido del editor de consultas.

AWS AppSync las generó automáticamente a partir del modelo que definimos anteriormente. En este ejemplo, se utilizará la mutación `createTodo` para agregar entradas a nuestra tabla *`TodoAPITable`*.

5. Seleccione la operación `createTodo` expandiéndola debajo de la mutación `createTodo`:

Active las casillas de verificación de todos los campos, como se muestra en la imagen anterior.

> **Note**
>
> Los atributos que ve aquí son los diferentes elementos modificables de la mutación. Su `input` se puede interpretar como el parámetro de `createTodo`. Las distintas opciones

con casillas de verificación son los campos que se devolverán en la respuesta una vez que se haya realizado una operación.

6. En el editor de código situado en el centro de la pantalla, verá que la operación aparece debajo de la mutación `createTodo`:

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

> **Note**
>
> Para explicar este fragmento correctamente, también debemos observar el código del esquema. La declaración `mutation createTodo($createtodoinput: CreateTodoInput!){}` es la mutación con una de sus operaciones, `createTodo`. La mutación completa se encuentra en el esquema:
>
> ```
> type Mutation {
>  createTodo(input: CreateTodoInput!): Todo
>  updateTodo(input: UpdateTodoInput!): Todo
>  deleteTodo(input: DeleteTodoInput!): Todo
> }
> ```
>
> Volviendo a la declaración de mutación del editor, el parámetro es un objeto llamado `$createtodoinput` con un tipo de entrada obligatorio de `CreateTodoInput`. Tenga en cuenta que `CreateTodoInput` (y todas las entradas de la mutación) también están definidas en el esquema. Por ejemplo, este es el código reutilizable de `CreateTodoInput`:
>
> ```
> input CreateTodoInput {
>  name: String
>  when: String
>  where: String
>  description: String
> ```

```
}
```

Contiene los campos que definimos en nuestro modelo: `name`, `when`, `where` y `description`.
Volviendo al código del editor, en `createTodo(input: $createtodoinput) {}`, declaramos la entrada como `$createtodoinput`, que también se usó en la declaración de mutación. Hacemos esto porque esto permite a GraphQL validar nuestras entradas con los tipos proporcionados y garantizar que se utilizan con las entradas correctas.
La parte final del código del editor muestra los campos que se devolverán en la respuesta después de realizar una operación:

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

En la pestaña de Variables de consulta situada debajo de este editor, habrá un objeto `createtodoinput` genérico que puede contener los siguientes datos:

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

Note

Aquí es donde asignamos los valores de la entrada mencionada anteriormente:

```
input CreateTodoInput {
 name: String
```

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

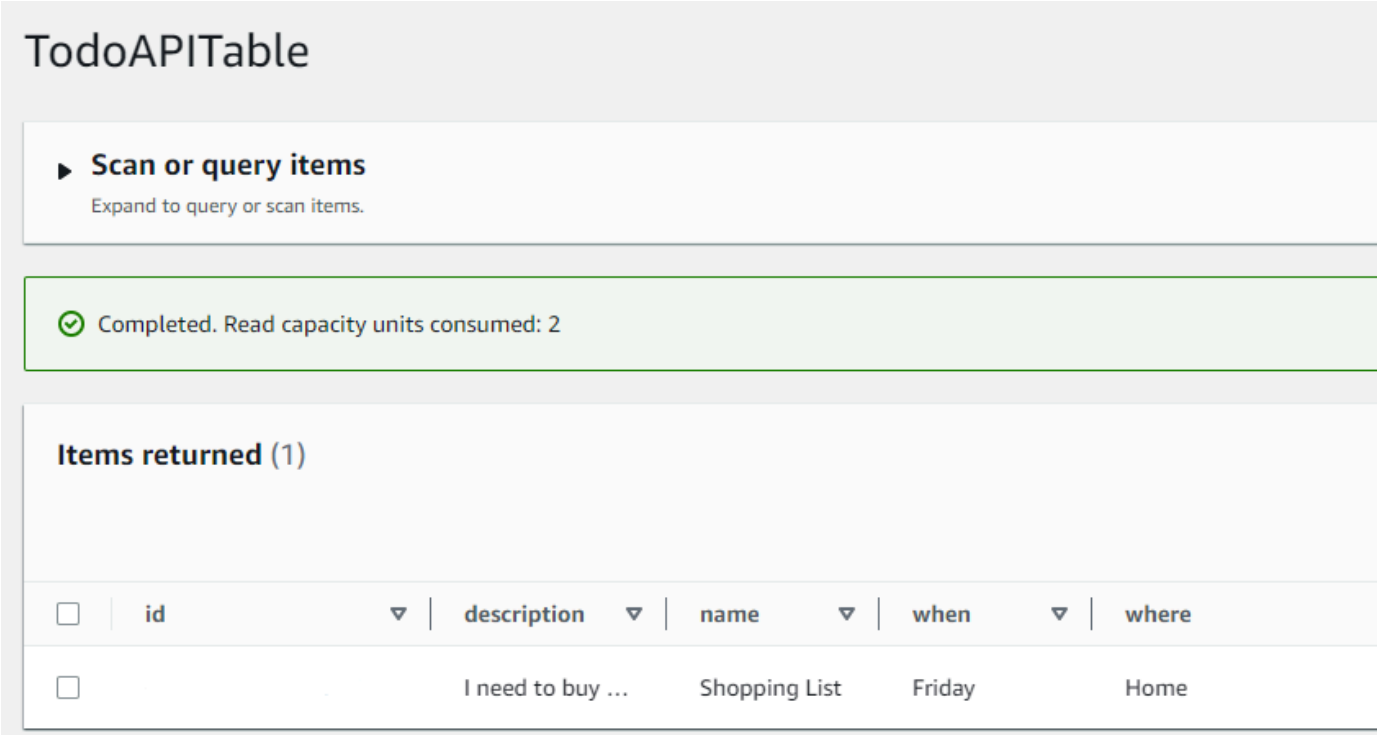
Cambie `createtodoinput` agregando la información que queremos incluir en nuestra tabla de DynamoDB. En este caso, queríamos crear algunos elementos Todo como recordatorios:

```
{
  "createtodoinput": {
    "name": "Shopping List",
    "when": "Friday",
    "where": "Home",
    "description": "I need to buy eggs"
  }
}
```

7. Seleccione Ejecutar en la parte superior del editor. En la lista desplegable, seleccione createTodo. En la parte derecha del editor, debería ver la respuesta. Puede tener un aspecto similar al siguiente:

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

Si navega hasta el servicio DynamoDB, ahora verá una entrada en el origen de datos con esta información:



Para resumir la operación, el motor GraphQL analizó el registro y un solucionador lo insertó en la tabla de Amazon DynamoDB. De nuevo, puede comprobarlo en la consola de DynamoDB. Tenga en cuenta que no ha sido necesario transferir un valor `id`. Se genera un `id` y se devuelve en los resultados. Esto se debe a que en el ejemplo se utilizó una función `autoId()` en un solucionador de GraphQL para el conjunto de claves de partición de los recursos de DynamoDB. Veremos cómo puede crear solucionadores en una sección diferente. Tome nota del valor `id` devuelto; lo usará en la siguiente sección para recuperar datos con una consulta de GraphQL.

Paso 4: recuperar datos con una consulta de GraphQL

Ahora que ya tiene un registro en su base de datos, puede obtener resultados al ejecutar una consulta. Una consulta es otra de las operaciones fundamentales de GraphQL. Se usa para analizar y recuperar información de su origen de datos. En cuanto a las API de REST, es similar a la operación `GET`. La principal ventaja de las consultas de GraphQL es la capacidad de especificar los requisitos de datos exactos de su aplicación para que pueda obtener los datos relevantes en el momento adecuado.

Para consultar su origen de datos

1. Si aún no lo ha hecho, inicie sesión en la AWS Management Console y abra la consola de AppSync.
2. Seleccione la API de la tabla.
3. En la pestaña de la izquierda, seleccione Consultas.
4. En la pestaña Explorador situada a la izquierda de la tabla, en `query` `listTodos`, expanda la operación `getTodo`:

5. En el editor de código, debería ver el código de operación:

```
query listTodos {
  getTodo(id: "") {
    description
    id
    name
    when
    where
  }
```

En `(id:"")`, introduzca el valor que guardó en el resultado de la operación de mutación. En nuestro ejemplo, sería:

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

```
  }
```

6. Seleccione Ejecutar y, a continuación, listTodos. El resultado aparecerá a la derecha del editor. Nuestro ejemplo tenía un aspecto similar al siguiente:

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

> **Note**
>
> Las consultas solo devuelven los campos especificados. Puede anular la selección de los campos que no necesite eliminándolos del campo de devolución:
>
> ```
> {
>     description
>     id
>     name
>     when
>     where
>   }
> ```
>
> También puede anular la selección de la casilla de la pestaña Explorador situada junto al campo que desea eliminar.

7. Asimismo, puede probar la operación `listTodos` repitiendo los pasos para crear una entrada en el origen de datos y, a continuación, repetir los pasos de consulta con la operación `listTodos`. A continuación, se muestra un ejemplo en el que agregamos una segunda tarea:

```
{
  "createtodoinput": {
    "name": "Second Task",
    "when": "Monday",
```

```
    "where": "Home",
    "description": "I need to mow the lawn"
  }
}
```

Al llamar a la operación `listTodos`, devolvió las entradas antiguas y nuevas:

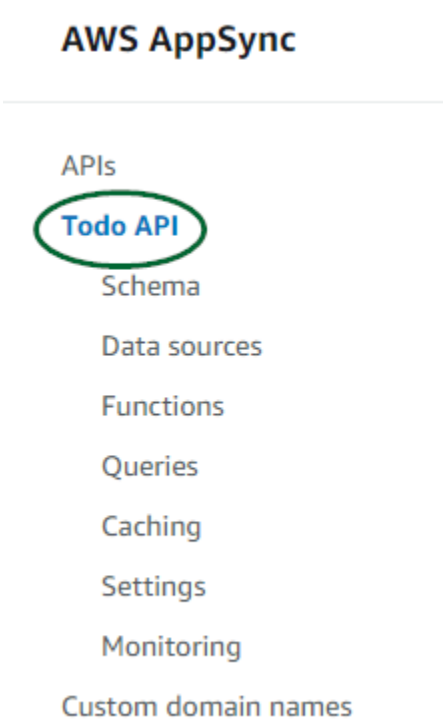
```
{
  "data": {
    "listTodos": {
      "items": [
        {
          "id": "abcdefgh-1234-1234-1234-abcdefghijkl",
          "name": "Shopping List",
          "when": "Friday",
          "where": "Home",
          "description": "I need to buy eggs"
        },
        {
          "id": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",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

Secciones complementarias

Estas secciones sirven de referencia para temas más avanzados de AWS AppSync. Recomendamos continuar con la sección Lectura complementaria antes de nada.

Integration

En la pestaña de la consola, si elige el nombre de su API, aparecerá la página Integración:



En ella, se resumen los pasos para configurar la API y se describen los siguientes pasos para crear una aplicación cliente. En la sección Integrar en su aplicación, se proporcionan detalles para usar la cadena de herramientas de AWS Amplify para automatizar el proceso de conexión de su API con aplicaciones de iOS, Android y JavaScript a través de la configuración y la generación de código. La cadena de herramientas de Amplify es totalmente compatible con la creación de proyectos desde su estación de trabajo local, incluidos el aprovisionamiento de GraphQL y los flujos de trabajo para CI/CD.

La sección Muestras de cliente también incluye ejemplos de aplicaciones cliente (p. ej., JavaScript, iOS o Android) para probar una experiencia integral. Puede clonar y descargar estos ejemplos y el archivo de configuración que tiene la información necesaria (como su URL de punto de conexión) que precisa para comenzar. Siga las instrucciones de la página Cadena de herramientas de AWS Amplify para ejecutar su aplicación.

Lectura complementaria

- Diseño de API de GraphQL: esta es una guía completa para crear su GraphQL utilizando un esquema en blanco sin orígenes de datos ni solucionadores.

Diseño de API de GraphQL

AWS AppSync le permite crear API de GraphQL utilizando la experiencia de la consola. Puede echar un vistazo a esto en la sección Lanzamiento de un esquema de ejemplo. Sin embargo, esa guía no mostraba todo el catálogo de opciones y configuraciones que podría utilizar en AWS AppSync.

Si decide crear una API de GraphQL en la consola, puede explorar varias opciones. Si ha leído nuestra guía de Lanzamiento de un esquema de ejemplo, le mostramos cómo crear una API a partir de un modelo predefinido. En las secciones siguientes, le guiaremos por el resto de las opciones y configuraciones para crear las API de GraphQL en AWS AppSync.

En esta sección, analizará los conceptos siguientes:

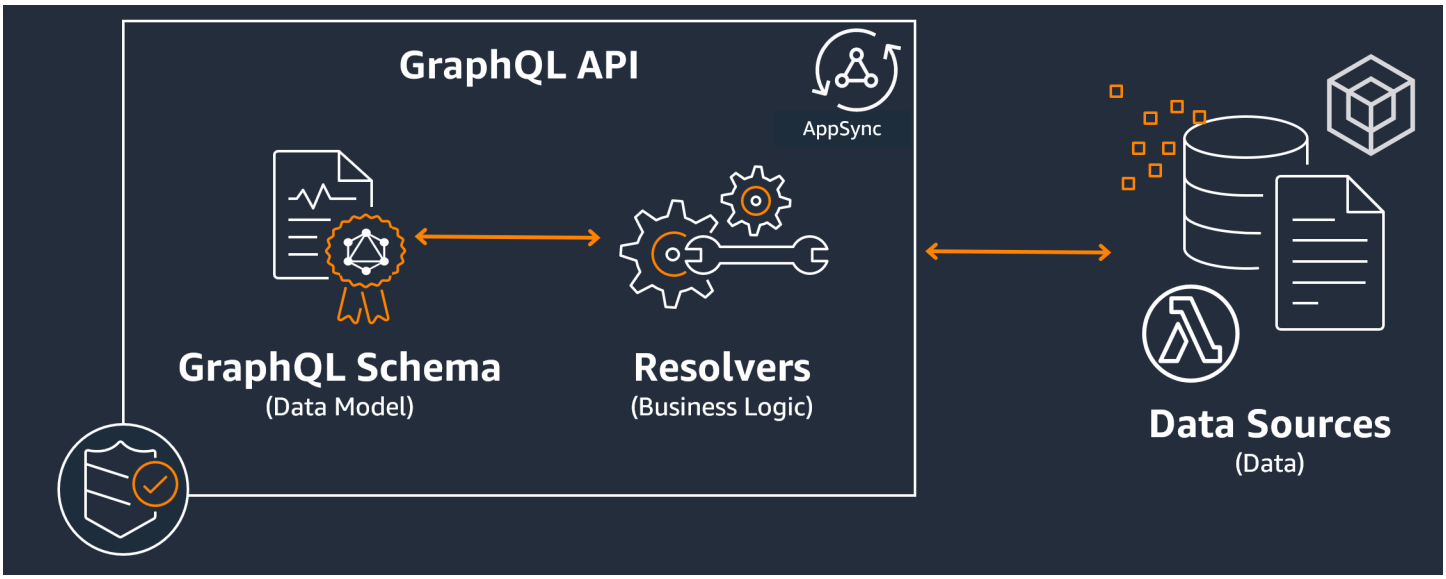
1. Blank APIs or imports: esta guía explica todo el proceso de creación de una API de GraphQL. Aprenderá a crear un GraphQL a partir de una plantilla en blanco sin modelo, a configurar los orígenes de datos del esquema y a añadir su primer solucionador a un campo.
2. Real-time data: esta guía muestra las posibles opciones para crear una API utilizando el motor WebSocket de AWS AppSync.
3. Merged APIs: esta guía muestra cómo crear nuevas API de GraphQL asociando y fusionando datos de varias API de GraphQL existentes.
4. the section called “Introspección de RDS”: esta guía muestra cómo integrar sus tablas de Amazon RDS mediante una API de datos.

Estructuración de una API de GraphQL (API en blanco o importadas)

Para crear su API de GraphQL a partir de una plantilla en blanco, antes sería útil revisar los conceptos relacionados con GraphQL. Hay tres componentes fundamentales de una API de GraphQL:

1. El esquema es el archivo que contiene la forma y la definición de los datos. Cuando un cliente realice una solicitud a su servicio GraphQL, los datos devueltos seguirán la especificación del esquema. Para obtener más información, consulte Esquemas.
2. El origen de datos se adjunta a su esquema. Cuando se realiza una solicitud, aquí es donde se recuperan y modifican los datos. Para obtener más información, consulte Data sources.

3. El solucionador se encuentra entre el esquema y el origen de datos. Cuando se realiza una solicitud, el solucionador realiza la operación con los datos del origen y, a continuación, devuelve el resultado como respuesta. Para obtener más información, consulte Resolvers.



AWS AppSync gestiona sus API permitiéndole crear, editar y almacenar el código para sus esquemas y solucionadores. Los orígenes de datos procederán de repositorios externos, como bases de datos, tablas de DynamoDB y funciones de Lambda. Si utiliza un servicio de AWS para almacenar tus datos o tiene pensado hacerlo, AWS AppSync ofrece una experiencia prácticamente perfecta para asociar los datos de tus cuentas de AWS a sus API de GraphQL.

En la siguiente sección, aprenderá a crear cada uno de estos componentes mediante el servicio de AWS AppSync.

Temas

- Paso 1: diseño del esquema
- Paso 2: Asociar un origen de datos
- Paso 3: Configurar solucionadores
- Paso 4: Uso de una API: ejemplo de CDK

Paso 1: diseño del esquema

El esquema GraphQL es la base de cualquier implementación de servidor GraphQL. Cada API de GraphQL se define mediante un solo esquema que contiene tipos y campos que describen cómo

se rellenarán los datos de las solicitudes. Los datos que fluyen a través de la API y las operaciones realizadas deben validarse con respecto al esquema.

En general, el sistema de tipos de GraphQL describe las funcionalidades de un servidor GraphQL y se utiliza para determinar si una consulta es válida. El sistema de tipos de un servidor suele denominarse esquema de ese servidor y puede constar de diferentes tipos de objetos, tipos escalares, tipos de entradas, etc. GraphQL es declarativo y tiene establecimiento inflexible de tipos, lo que significa que los tipos estarán bien definidos en tiempo de ejecución y solo devolverán lo que se haya especificado.

AWS AppSync permite definir y configurar esquemas de GraphQL. En la siguiente sección se describe cómo crear esquemas de GraphQL desde cero utilizando los servicios de AWS AppSync.

Estructuración de un esquema de GraphQL

> **Tip**
>
> Recomendamos consultar la sección Esquemas antes de continuar.

GraphQL es una poderosa herramienta para implementar servicios de API. Según el sitio web de GraphQL, GraphQL es lo siguiente:

“GraphQL es un lenguaje de consulta para las API y un tiempo de ejecución para responder a esas consultas con los datos existentes. GraphQL proporciona una descripción completa y comprensible de los datos de su API, da a los clientes la posibilidad de pedir exactamente lo que necesitan y nada más, facilita la evolución de las API a lo largo del tiempo y habilita potentes herramientas para desarrolladores.”

Esta sección cubre la primera parte de la implementación de GraphQL, el esquema. Siguiendo la cita anterior, un esquema cumple la función de “proporcionar una descripción completa y comprensible de los datos de su API”. En otras palabras, un esquema GraphQL es una representación textual de los datos, las operaciones y las relaciones entre ellos de su servicio. El esquema se considera el punto de entrada principal para la implementación del servicio GraphQL. Como era de esperar, suele ser una de las primeras cosas que hace en su proyecto. Recomendamos consultar la sección Esquemas antes de continuar.

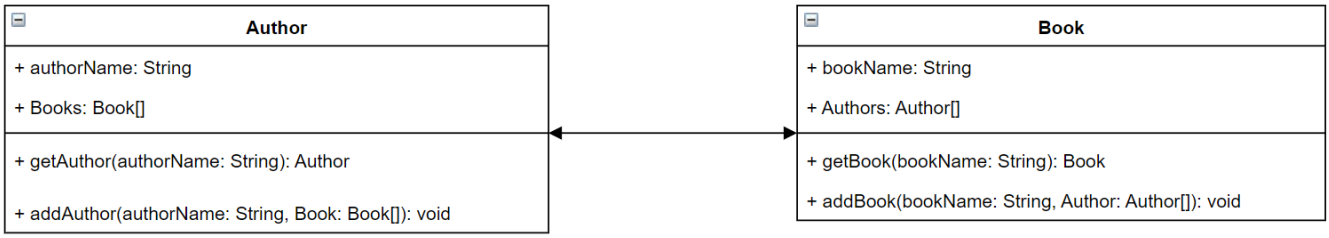
Para citar la sección Esquemas, los esquemas de GraphQL se escriben en el lenguaje de definición de esquema (SDL). SDL está compuesto por tipos y campos con una estructura establecida:

- Tipos: los tipos son la forma en que GraphQL define la forma y el comportamiento de los datos. GraphQL admite una multitud de tipos que se explicarán más adelante en esta sección. Cada tipo que se defina en su esquema tendrá su propio ámbito. Dentro del ámbito habrá uno o más campos que pueden contener un valor o una lógica que se utilice en el servicio GraphQL. Los tipos cumplen muchos roles diferentes, siendo las más comunes los objetos o los escalares (tipos de valores primitivos).
- Campos: los campos existen dentro del ámbito de un tipo y contienen el valor que se solicita al servicio GraphQL. Se parecen mucho a las variables de otros lenguajes de programación. La forma de los datos que defina en sus campos determinará cómo se estructuren los datos en una operación de solicitud/respuesta. Esto permite a los desarrolladores predecir lo que se va a devolver sin saber cómo se implementa el backend del servicio.

Los esquemas más simples contendrán tres categorías de datos diferentes:

1. Raíces del esquema: las raíces definen los puntos de entrada de su esquema. Señala los campos que realizarán alguna operación con los datos, como añadir, eliminar o modificar algo.
2. Tipos: son tipos básicos que se utilizan para representar la forma de los datos. Casi se puede pensar en ellos como objetos o representaciones abstractas de algo con características definidas. Por ejemplo, podría crear un objeto `Person` que represente a una persona en una base de datos. Las características de cada persona se definirán dentro de `Person` como campos. Pueden ser cualquier cosa, como el nombre, la edad, el trabajo, la dirección, etc. de la persona.
3. Tipos de objetos especiales: son los tipos que definen el comportamiento de las operaciones del esquema. Cada tipo de objeto especial se define una vez por cada esquema. Primero se colocan en la raíz del esquema y, a continuación, se definen en el cuerpo del esquema. Cada campo de un tipo de objeto especial define una sola operación que debe implementar el solucionador.

Para poner esto en perspectiva, imagine que está creando un servicio que almacena autores y los libros que han escrito. Cada autor tiene un nombre y una serie de libros que han escrito. Cada libro tiene un nombre y una lista de autores asociados. También queremos poder añadir o recuperar libros y autores. Una representación UML simple de esta relación puede tener este aspecto:



En GraphQL, las entidades `Author` y `Book` representan dos tipos de objetos diferentes en el esquema:

```
type Author {
}

type Book {
}
```

`Author` contiene `authorName` y `Books`, mientras que `Book` contiene `bookName` y `Authors`. Estos se pueden representar como los campos incluidos en el ámbito de sus tipos:

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

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

Como puede ver, las representaciones de tipos se parecen mucho al diagrama. Sin embargo, los métodos son algo más complicados. Se colocarán como campo en uno de algunos tipos de objetos especiales. La categorización de los objetos especiales depende de su comportamiento. GraphQL contiene tres tipos de objetos especiales fundamentales: consultas, mutaciones y suscripciones. Para obtener más información sobre los objetos, consulte Objetos especiales.

Como tanto `getAuthor` como `getBook` solicitan datos, se colocarán en un tipo de objeto especial de `Query`:

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

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

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

Las operaciones están vinculadas a la consulta, que a su vez está vinculada al esquema. Cuando se añade una raíz de esquema, se define el tipo de objeto especial (en este caso, `Query`) como uno de los puntos de entrada. Esto se puede hacer con la palabra clave `schema`:

```
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
}
```

Si nos fijamos en los dos últimos métodos, `addAuthor` y `addBook` añaden datos a la base de datos, por lo que se definirán en un tipo de objeto `Mutation` especial. Sin embargo, por la página Tipos, también sabemos que no se permiten entradas que hagan referencia directamente a objetos

porque son estrictamente tipos de salida. En este caso, no podemos usar `Author` ni `Book`, por lo que necesitamos crear un tipo de entrada con los mismos campos. En este ejemplo, añadimos `AuthorInput` y`BookInput`, que aceptan los mismos campos de sus tipos correspondientes. A continuación, creamos nuestra mutación usando las entradas como parámetros:

```
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
}
```

Repasemos lo que acabamos de hacer:

1. Hemos creado un esquema con los tipos `Book` y `Author` para representar nuestras entidades.

2. Hemos añadido los campos que contienen las características de nuestras entidades.
3. Hemos añadido una consulta para recuperar esta información de la base de datos.
4. Hemos añadido una mutación para manipular los datos de la base de datos.
5. Hemos añadido tipos de entrada para reemplazar los parámetros de nuestro objeto en la mutación y cumplir con las reglas de GraphQL.
6. Hemos añadido la consulta y la mutación a nuestro esquema de raíz para que la implementación de GraphQL entienda la ubicación del tipo de raíz.

Como puede ver, el proceso de creación de un esquema requiere muchos conceptos del modelado de datos (especialmente del modelado de bases de datos) en general. Puede considerarse que el esquema se ajusta a la forma de los datos de la fuente. También sirve como modelo que el solucionador implementará. En las secciones siguientes, aprenderá a crear un esquema utilizando varias herramientas y servicios respaldados por AWS.

> **Note**
>
> Los ejemplos de las secciones siguientes no están pensados para ejecutarse en una aplicación real. Con ellos solo se pretende mostrar los comandos para que pueda crear sus propias aplicaciones.

Creación de esquemas

Su esquema estará en un archivo llamado `schema.graphql`. AWS AppSync permite a los usuarios crear nuevos esquemas para sus API de GraphQL mediante varios métodos. En este ejemplo, crearemos una API en blanco junto con un esquema en blanco.

Console

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   a. En el Panel, elija Crear API.
   b. En Opciones de API, seleccione API de GraphQL, Diseñar desde cero y, a continuación, Siguiente.
      i. En Nombre de la API, cambie el nombre previamente cumplimentado por el que su aplicación necesite.

ii. Para obtener los detalles de contacto, puede introducir un punto de contacto para identificar al administrador de la API. Se trata de un campo opcional.

iii. En Configuración de API privada, puede habilitar las características de API privadas. Solo se puede acceder a una API privada desde un punto de conexión de VPC (VPCE) configurado. Para obtener más información, consulte API privadas.

No le recomendamos habilitar esta característica para este ejemplo. Seleccione Siguiente después de revisar sus entradas.

c. En Crear un tipo de GraphQL, puede elegir crear una tabla de DynamoDB para utilizarla como origen de datos u omitir este paso y hacerlo más adelante.

Para este ejemplo, elija Crear recursos de GraphQL más adelante. Crearemos un recurso en una sección aparte.

d. Revise la información indicada y, a continuación, seleccione Crear API.

2. Estará en el panel de control de tu API específica. Lo notará porque el nombre de la API aparecerá en la parte superior del panel de control. Si este no es el caso, puede seleccionar API en la barra lateral y, a continuación, elegir la API en el panel de API.

- En la barra lateral situada debajo del nombre de la API, seleccione Esquema.

3. En el Editor de esquemas, puede configurar su archivo `schema.graphql`. Puede estar vacío o lleno de tipos generados a partir de un modelo. A la derecha, tiene la sección Solucionadores para asociar solucionadores a los campos del esquema. En esta sección no nos fijaremos en los solucionadores.

CLI

> **Note**
>
> Cuando utilice la CLI, asegúrese de tener los permisos correctos para acceder y crear recursos en el servicio. Es posible que desee establecer políticas de privilegios mínimos para los usuarios que no sean administradores y que necesiten acceder al servicio. Para obtener más información acerca de las políticas de AWS AppSync, consulte Identity and access management en AWS AppSync.
> Además, le recomendamos leer primero la versión de consola si aún no lo ha hecho.

1. Si aún no lo ha hecho, instale la CLI de AWS y añadas su configuración.

2. Cree un objeto de API de GraphQL ejecutando el comando `create-graphql-api`.

   Deberá escribir dos parámetros para este comando concreto:

   1. El `name` de su API.
   2. El `authentication-type` o el tipo de credenciales utilizado para acceder a la API (IAM, OIDC, etc.).

   > **Note**
   >
   > Otros parámetros, como `Region`, deben configurarse pero normalmente se utilizarán de forma predeterminada en los valores de configuración de la CLI.

   Un comando de ejemplo puede tener este aspecto:

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

   Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

   ```
   {
       "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**
>
> Se trata de un comando opcional que toma un esquema existente y lo carga en el servicio de AWS AppSync mediante un blob de base 64. No utilizaremos este comando para este ejemplo.

Ejecute el comando `start-schema-creation`

Deberá escribir dos parámetros para este comando concreto:

1. Su `api-id` del paso anterior.
2. La `definition` del esquema es un blob binario codificado en base 64.

Un comando de ejemplo puede tener este aspecto:

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

Se devolverá un resultado:

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

Este comando no devolverá el resultado final después del procesamiento. Para ver el resultado, debe usar un comando diferente, `get-schema-creation-status`. Tenga en cuenta que estos dos comandos son asíncronos, por lo que puede comprobar el estado de la salida incluso mientras se crea el esquema.

CDK

> **Tip**
>
> Antes de usar el CDK, le recomendamos que consulte la documentación oficial de este, junto con la referencia del CDK de AWS AppSync.

Los pasos que se indican a continuación solo mostrarán un ejemplo general del fragmento de código utilizado para añadir un recurso concreto. No se pretende que esta sea una solución funcional en su código de producción. También, se presupone que ya tiene una aplicación en funcionamiento.

1. El punto de partida del CDK es ligeramente diferente. Lo ideal es que el archivo `schema.graphql` ya esté creado. Solo necesita crear un nuevo archivo con la extensión de archivo `.graphql`. Puede ser un archivo vacío.
2. En general, puede que tenga que añadir la directiva de importación al servicio que utilice. Por ejemplo, puede seguir estas formas:

```
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'
```

Para añadir una API de GraphQLs tu archivo de pila debe importar el servicio de AWS AppSync:

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

> **Note**
>
> Esto significa que vamos a importar todo el servicio con la palabra clave `appsync`. Para usar esto en tu aplicación, sus constructos de AWS AppSync utilizarán el formato `appsync.construct_name`. Por ejemplo, si quisiéramos crear una API de GraphQL, diríamos `new appsync.GraphqlApi(args_go_here)`. En el siguiente paso se describe esto.

3. La API de GraphQL más básica incluirá un `name` para la API y la ruta de `schema`.

```
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**
>
> Repasemos lo que hace este fragmento de código. Dentro del ámbito de `api`, creamos una nueva API de GraphQL llamando a `appsync.GraphqlApi(scope: Construct, id: string, props: GraphqlApiProps)`. El alcance es `this`, que hace referencia al objeto actual. El identificador es *`API_ID`*, que será el nombre del recurso de su API de GraphQL en AWS CloudFormation cuando se cree. `GraphqlApiProps` contiene el `name` de la API de GraphQL y el `schema`. `schema` generará un esquema (`SchemaFile.fromAsset`) buscando la ruta absoluta (`__dirname`) del archivo `.graphql` (*`schema_name.graphql`*). En un escenario real, es probable que su archivo de esquema esté dentro de la aplicación del CDK. Para usar los cambios realizados en la API de GraphQL, deberá volver a implementar la aplicación.

Adición de tipos a los esquemas

Ahora que ha añadido su esquema, puede empezar a agregar los tipos tanto de entrada como de salida. Tenga en cuenta que los tipos que aparecen aquí no deben usarse en código real; son solo ejemplos que le ayudarán a entender el proceso.

En primer lugar, crearemos un tipo de objeto. En el código real, no es necesario empezar con estos tipos. Puede crear cualquier tipo que desee en cualquier momento siempre que siga las reglas y la sintaxis de GraphQL.

> **Note**
>
> Las siguientes secciones usarán el editor de esquemas, así que manténgalo abierto.

Console

- Puede crear un tipo de objeto utilizando la palabra clave `type` junto con el nombre del tipo:

```
type Type_Name_Goes_Here {}
```

Dentro del ámbito del tipo, puede añadir campos que representen las características del objeto:

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

A continuación se muestra un ejemplo:

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

> **Note**
>
> En este paso, hemos añadido un tipo de objeto genérico con un campo `id` obligatorio almacenado como `ID`, un campo `title` almacenado como `String` y un campo `date` almacenado como `AWSDateTime`. Para ver una lista de tipos y campos, y lo que hacen, consulte Esquemas. Para ver una lista de escalares y lo que hacen, consulte Referencia de tipos.

CLI

> **Note**
>
> Le recomendamos leer primero la versión de consola si aún no lo ha hecho.

- Puede crear un tipo de objeto ejecutando el comando `create-type`.

  Para este comando en particular, deberá introducir varios parámetros:

  1. El `api-id` de su API.
  2. La `definition` o el contenido de su tipo. En el ejemplo de la consola, esto era:

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

3. El `format` de su entrada. En este ejemplo, usaremos `SDL`.

Un comando de ejemplo puede tener este aspecto:

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

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

```
{
    "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**
>
> En este paso, hemos añadido un tipo de objeto genérico con un campo `id` obligatorio almacenado como `ID`, un campo `title` almacenado como `String` y un campo `date` almacenado como `AWSDateTime`. Para ver una lista de tipos y campos, y lo que hacen, consulte Esquemas. Para ver una lista de escalares y lo que hacen, consulte Referencia de tipos.
>
> Por otra parte, puede que se haya dado cuenta de que introducir la definición directamente funciona para tipos más pequeños pero no es factible para añadir tipos más grandes o múltiples. Puede optar por añadir todo el contenido en un archivo `.graphql` y, a continuación, pasarlo como entrada.

CDK

> **Tip**
>
> Antes de usar el CDK, le recomendamos que consulte la documentación oficial de este, junto con la referencia del CDK de AWS AppSync.
> Los pasos que se indican a continuación solo mostrarán un ejemplo general del fragmento de código utilizado para añadir un recurso concreto. No se pretende que esta sea una solución funcional en su código de producción. También, se presupone que ya tiene una aplicación en funcionamiento.

Para añadir un tipo, debe añadirlo a su archivo `.graphql`. El ejemplo de la consola era:

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

Puede añadir sus tipos directamente al esquema como cualquier otro archivo.

> **Note**
>
> Para usar los cambios realizados en la API de GraphQL, deberá volver a implementar la aplicación.

El tipo de objeto tiene campos que son de tipo escalar, como cadenas y números enteros. AWS AppSync también permite usar tipos escalares mejorados, como `AWSDateTime`, además de los escalares básicos de GraphQL. Además, todos los campos que terminen con un signo de exclamación son obligatorios.

El tipo escalar `ID` en concreto es un identificador exclusivo que puede ser `String` o `Int`. Puede controlarlos en el código de resolución para la asignación automática.

Existen similitudes entre los tipos de objetos especiales, como `Query` y los tipos de objetos "normales", como en el ejemplo anterior, en el sentido de que ambos utilizan la palabra clave `type` y se consideran objetos. Sin embargo, en el caso de los tipos de objetos especiales (`Query`,

`Mutation`, y `Subscription`), su comportamiento es muy diferente, ya que se exponen como puntos de entrada a la API. También se centran más en dar forma a las operaciones que a los datos. Para obtener más información, consulte la sección sobre tipos de consultas y mutaciones.

En cuanto a los tipos de objetos especiales, el siguiente paso podría ser añadir uno o más para realizar operaciones con los datos con forma. En un escenario real, cada esquema de GraphQL debe tener al menos un tipo de consulta raíz para solicitar datos. Puede considerar la consulta como uno de los puntos de entrada (o puntos de conexión) de su servidor GraphQL. Añadamos una consulta como ejemplo.

Console

- Para crear una consulta, basta con añadirla al archivo de esquema como cualquier otro tipo. Una consulta requeriría un tipo `Query` y una entrada en la raíz de la manera siguiente:

```
schema {
  query: Name_of_Query
}

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

Tenga en cuenta que, en la mayoría de los casos, *`Name_of_Query`* simplemente se llamará `Query` en un entorno de producción. Se recomienda mantenerlo en este valor. Dentro del tipo de consulta, puede añadir campos. Cada campo realizará una operación en la solicitud. El resultado es que la mayoría de estos campos, si no todos, se asociarán a un solucionador. Sin embargo, en esta sección o abordaremos eso. En cuanto al formato de la operación de campo, podría ser como este:

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

A continuación se muestra un ejemplo:

```
schema {
  query: Query
}

type Query {
```

```
  getObj: [Obj_Type_1]
}

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

> **Note**
>
> En este paso, agregamos un tipo `Query` y lo definimos en nuestra raíz de `schema`. Nuestro tipo `Query` definió un campo `getObj` que devuelve una lista de objetos `Obj_Type_1`. Tenga en cuenta que `Obj_Type_1` es el objeto del paso anterior. En el código de producción, sus operaciones de campo normalmente funcionarán con datos moldeados por objetos como `Obj_Type_1`. Además, campos como `getObj` suelen tener un solucionador para ejecutar la lógica empresarial. Este tema se tratará en otra sección.
>
> Como comentario adicional, AWS AppSync añade automáticamente una raíz de esquema durante las exportaciones, por lo que técnicamente no tiene que añadirla directamente al esquema. Nuestro servicio procesará automáticamente los esquemas duplicados. Lo añadimos aquí como práctica recomendada.

CLI

> **Note**
>
> Le recomendamos leer primero la versión de consola si aún no lo ha hecho.

1. Cree una raíz de `schema` con una definición `query` ejecutando el comando `create-type`.

   Para este comando en particular, deberá introducir varios parámetros:

   1. El `api-id` de su API.
   2. La `definition` o el contenido de su tipo. En el ejemplo de la consola, esto era:

      ```
      schema {
      ```

```
  query: Query
}
```

3. El `format` de su entrada. En este ejemplo, usaremos `SDL`.

Un comando de ejemplo puede tener este aspecto:

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

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

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

> **Note**
>
> Tenga en cuenta que si no ha introducido algo correctamente en el comando `create-type`, puede actualizar la raíz del esquema (o cualquier tipo del esquema) ejecutando el comando `update-type`. En este ejemplo, cambiaremos temporalmente la raíz del esquema para que contenga una definición de `subscription`.
> Para este comando en particular, deberá introducir varios parámetros:
>
> 1. El `api-id` de su API.
> 2. El `type-name` de su tipo. En el ejemplo de la consola, esto era `schema`.
> 3. La `definition` o el contenido de su tipo. En el ejemplo de la consola, esto era:
>
> ```
> schema {
>   query: Query
> }
> ```

El esquema después de añadir una `subscription` será como el siguiente:

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

4. El `format` de su entrada. En este ejemplo, usaremos `SDL`.

Un comando de ejemplo puede tener este aspecto:

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

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

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

En este ejemplo, seguirá funcionando la adición de archivos preformateados.

2. Cree un tipo de `Query` ejecutando el comando `create-type`.

Para este comando en particular, deberá introducir varios parámetros:

1. El `api-id` de su API.
2. La `definition` o el contenido de su tipo. En el ejemplo de la consola, esto era:

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

3. El `format` de su entrada. En este ejemplo, usaremos `SDL`.

Un comando de ejemplo puede tener este aspecto:

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

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

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

> **Note**
>
> En este paso, hemos añadido un tipo `Query` y lo hemos definido en su raíz de `schema`. Nuestro tipo `Query` ha definido un campo `getObj` que devuelve una lista de objetos `Obj_Type_1`.
> En el código de raíz de `schema` `query: Query`, la parte `query:` indica que se ha definido una consulta en el esquema, mientras que la parte `Query` indica el nombre real del objeto especial.

CDK

> **Tip**
>
> Antes de usar el CDK, le recomendamos que consulte la documentación oficial de este, junto con la referencia del CDK de AWS AppSync.
> Los pasos que se indican a continuación solo mostrarán un ejemplo general del fragmento de código utilizado para añadir un recurso concreto. No se pretende que esta

sea una solución funcional en su código de producción. También, se presupone que ya tiene una aplicación en funcionamiento.

Deberá añadir la consulta y la raíz del esquema al archivo `.graphql`. Nuestro caso era parecido al ejemplo siguiente, pero en este lo sustituirá con su código de esquema actual:

```
schema {
  query: Query
}

type Query {
  getObj: [Obj_Type_1]
}

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

Puede añadir sus tipos directamente al esquema como cualquier otro archivo.

> **Note**
>
> Actualizar la raíz del esquema es opcional. Lo hemos añadido a este ejemplo como práctica recomendada.
> Para usar los cambios realizados en la API de GraphQL, deberá volver a implementar la aplicación.

Ahora ya ha visto un ejemplo de creación tanto de objetos como de objetos especiales (consultas). También ha visto cómo se pueden interconectar para describir datos y operaciones. Puede tener esquemas con solo la descripción de los datos y una o más consultas. Sin embargo, nos gustaría añadir otra operación para añadir datos al origen de datos. Vamos a añadir otro tipo de objeto especial denominado `Mutation` que modifica los datos.

Console

- Una mutación se llamará `Mutation`. Como en `Query`, las operaciones de campo de dentro de `Mutation` describirán una operación y se asociarán a un solucionador. Tenga en cuenta también que debemos definirla en la raíz `schema` porque es un tipo de objeto especial. Aquí tiene un ejemplo de mutación:

```
schema {
  mutation: Name_of_Mutation
}

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

Una mutación típica aparecerá en la raíz en forma de consulta. La mutación se define mediante la palabra clave `type` junto con el nombre. Por lo general, *`Name_of_Mutation`* se llamará `Mutation`, por lo que recomendamos mantenerlo así. Cada campo realizará también una operación. En cuanto al formato de la operación de campo, podría ser como este:

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

A continuación se muestra un ejemplo:

```
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**
>
> En este paso, hemos añadido un tipo `Mutation` con un campo `addObj`. Vamos a resumir lo que hace este campo:
>
> ```
> addObj(id: ID!, title: String, date: AWSDateTime): Obj_Type_1
> ```
>
> `addObj` está utilizando el objeto `Obj_Type_1` para realizar una operación. Esto es evidente debido a los campos, pero la sintaxis lo demuestra en el tipo de retorno `:` `Obj_Type_1`. Dentro de `addObj`, acepta los campos `id`, `title` y `date` del objeto `Obj_Type_1` como parámetros. Como puede ver, es muy parecido a la declaración de un método. Sin embargo, aún no hemos descrito el comportamiento de nuestro método. Como se ha afirmado antes, el esquema solo está ahí para definir cuáles serán los datos y las operaciones, no cómo funcionan. La lógica empresarial real se implementará más adelante, cuando creemos nuestros primeros solucionadores.
> Una vez que haya terminado con su esquema, hay una opción para exportarlo como archivo `schema.graphql`. En el editor de esquemas, puede elegir Exportar esquema para descargar el archivo en un formato compatible.
> Como comentario adicional, AWS AppSync añade automáticamente una raíz de esquema durante las exportaciones, por lo que técnicamente no tiene que añadirla directamente al esquema. Nuestro servicio procesará automáticamente los esquemas duplicados. Lo añadimos aquí como práctica recomendada.

CLI

> **Note**
>
> Le recomendamos leer primero la versión de consola si aún no lo ha hecho.

1. Actualice su esquema de raíz ejecutando el comando `update-type`.

Para este comando en particular, deberá introducir varios parámetros:

1. El `api-id` de su API.
2. El `type-name` de su tipo. En el ejemplo de la consola, esto era `schema`.
3. La `definition` o el contenido de su tipo. En el ejemplo de la consola, esto era:

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

4. El `format` de su entrada. En este ejemplo, usaremos `SDL`.

Un comando de ejemplo puede tener este aspecto:

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

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

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

2. Cree un tipo de `Mutation` ejecutando el comando `create-type`.

Para este comando en particular, deberá introducir varios parámetros:

1. El `api-id` de su API.
2. La `definition` o el contenido de su tipo. En el ejemplo de la consola, esto era:

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

3. El `format` de su entrada. En este ejemplo, usaremos `SDL`.

Un comando de ejemplo puede tener este aspecto:

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

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

```
{
    "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**
>
> Antes de usar el CDK, le recomendamos que consulte la documentación oficial de este, junto con la referencia del CDK de AWS AppSync.
> Los pasos que se indican a continuación solo mostrarán un ejemplo general del fragmento de código utilizado para añadir un recurso concreto. No se pretende que esta sea una solución funcional en su código de producción. También, se presupone que ya tiene una aplicación en funcionamiento.

Deberá añadir la consulta y la raíz del esquema al archivo `.graphql`. Nuestro caso era parecido al ejemplo siguiente, pero en este lo sustituirá con su código de esquema actual:

```
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**
>
> Actualizar la raíz del esquema es opcional. Lo hemos añadido a este ejemplo como práctica recomendada.
> Para usar los cambios realizados en la API de GraphQL, deberá volver a implementar la aplicación.

Consideraciones opcionales: uso de enumeraciones como estados

Llegados a este punto, ya sabe cómo hacer un esquema básico. Sin embargo, hay muchas cosas que se pueden añadir para aumentar la funcionalidad del esquema. Una característica común que se encuentra en las aplicaciones es el uso de enumeraciones como estados. Puede usar una enumeración para forzar la elección de un valor específico de un conjunto de valores cuando se invoque. Esto es adecuado para cosas que sabe que no van a cambiar drásticamente durante largos períodos de tiempo. Hipotéticamente, podríamos añadir una enumeración que devuelva el código de estado o cadena en la respuesta.

Como ejemplo, supongamos que vamos a crear una aplicación de redes sociales que almacene los datos de las publicaciones de un usuario en el backend. Nuestro esquema contiene un tipo `Post` que representa los datos de una publicación individual:

```
type Post {
```

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

Nuestra `Post` contendrá un `id` único, un título de `title`, una `date` de publicación y una enumeración denominada `PostStatus` que representa el estado de la publicación a medida que la aplicación la procesa. Para nuestras operaciones, tendremos una consulta que devolverá todos los datos de la publicación:

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

También tendremos una mutación que añade publicaciones al origen de datos:

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

Si observamos nuestro esquema, la enumeración `PostStatus` podría tener varios estados. Tal vez que queramos que los tres estados básicos se llamen `success` (publicación procesada correctamente), `pending` (publicación procesada) y `error` (publicación que no se puede procesar). Para añadir la enumeración, podemos hacer lo siguiente:

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

El esquema completo podría ser como este:

```
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
}
```

Si un usuario añade una `Post` en la aplicación, se llamará a la operación `addPost` para procesar esos datos. A medida que el solucionador adjunto a `addPost` procese los datos, actualizará continuamente `poststatus` con el estado de la operación. Cuando se consulte, `Post` contendrá el estado final de los datos. Tenga en cuenta que solo estamos describiendo cómo queremos que funcionen los datos en el esquema. Estamos haciendo muchas suposiciones sobre la implementación de nuestros solucionadores, que implementarán la lógica empresarial real para gestionar los datos a fin de cumplir con la solicitud.

Consideraciones opcionales: suscripciones

Las suscripciones en AWS AppSync se invocan como respuesta a una mutación. Se puede configurar con un tipo `Subscription` y una directiva `@aws_subscribe()` en el esquema para indicar qué mutaciones invocan una o varias suscripciones. Para obtener más información sobre cómo configurar suscripciones, consulte Datos en tiempo real.

Consideraciones opcionales: relaciones y paginación

Supongamos que tiene un millón de `Posts` almacenadas en una tabla de DynamoDB y desea devolver algunos de esos datos. Sin embargo, la consulta de ejemplo anterior solo devuelve todas las publicaciones. No le interesa buscarlas todas cada vez que realice una solicitud. Más bien, le interesa paginarlas. Realice los cambios siguientes en su esquema:

- En el campo `getPosts`, añada dos argumentos de entrada: `nextToken` (iterador) y `limit` (límite de iteración).
- Añada un nuevo tipo de `PostIterator` que contenga los campos `Posts` (recupera la lista de objetos `Post`) y `nextToken` (iterador).
- Modifique `getPosts` para que devuelva `PostIterator` y no una lista de objetos de `Post`.

```
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
}
```

El tipo `PostIterator` le permite devolver una parte de la lista de objetos de `Post` y un `nextToken` para obtener la parte siguiente. Dentro de `PostIterator`, hay una lista de elementos de `Post` (`[Post]`) que se devuelve con un token de paginación (`nextToken`). En AWS AppSync, se

conectaría a Amazon DynamoDB a través de un solucionador y se generaría automáticamente como un token cifrado. Así se convierte el valor del argumento `limit` en el parámetro `maxResults` y el argumento `nextToken` en el parámetro `exclusiveStartKey`. Para ver ejemplos y muestras de plantillas integradas en la consola de AWS AppSync, consulte Referencia de solucionadores (JavaScript).

Paso 2: Asociar un origen de datos

Los orígenes de datos son recursos de la cuenta de AWS con los que pueden interactuar las API de GraphQL.AWS AppSync admite varios orígenes de datos, como AWS Lambda, Amazon DynamoDB, bases de datos relacionales (Amazon Aurora sin servidor), Amazon OpenSearch Service y puntos de conexión HTTP. Una API de AWS AppSync se puede configurar para interactuar con varios orígenes de datos, lo que permite agregar datos en una sola ubicación. AWS AppSync puede usar los recursos de AWS existentes de su cuenta o aprovisionar tablas de DynamoDB en su nombre a partir de una definición de esquema.

La sección siguiente muestra cómo asociar un origen de datos a la API de GraphQL.

Tipos de orígenes de datos

Ahora que ha creado un esquema en la consola de AWS AppSync y la ha guardado, puede añadir un origen de datos. Al crear una API por primera vez, existe la opción de aprovisionar una tabla de Amazon DynamoDB durante la creación del esquema predefinido. Sin embargo, en esta sección no hablaremos de esa opción. Puede ver un ejemplo de esto en la sección Lanzamiento de un esquema.

Ahora, analizaremos todos los orígenes de datos compatibles con AWS AppSync. Para elegir la solución adecuada para su aplicación se tienen en cuenta numerosos factores. Las siguientes secciones proporcionarán más contexto para cada origen de datos. Para obtener información general sobre los orígenes de datos, consulte Orígenes de datos.

Amazon DynamoDB

Amazon DynamoDB es una de las principales soluciones de almacenamiento de AWS para aplicaciones escalables. El componente principal de DynamoDB es la tabla, que es simplemente una recopilación de datos. Por lo general, las tablas se crean a partir de entidades como `Book` o `Author`. La información de las entradas de la tabla se almacena como elementos, que son grupos de campos únicos para cada entrada. Un elemento completo representa una fila o un registro de la base de datos. Por ejemplo, un elemento de una entrada de `Book` puede incluir `title` y `author` junto con

sus valores. Cada uno de los campos, como el de `title` y el de `author`, se denominan atributos, que son similares a los valores de las columnas en las bases de datos relacionales.

Como puede suponer, las tablas se utilizarán para almacenar datos de su aplicación. AWS AppSync permite enlazar las tablas de DynamoDB a la API de GraphQL para manipular datos. Tomemos este caso de uso del blog Frontend web y móvil. Esta aplicación permite a los usuarios registrarse en una aplicación de redes sociales. Los usuarios pueden unirse a grupos y cargar publicaciones que se difunden a otros usuarios suscritos al grupo. Su aplicación almacena información de usuarios, de publicaciones y de grupos de usuarios en DynamoDB. La API de GraphQL (gestionada por AWS AppSync) interactúa con la tabla de DynamoDB. Cuando un usuario realiza cambios en el sistema que se vaya a reflejar en el frontend, la API de GraphQL los recupera y los difunde a otros usuarios en tiempo real.

AWS Lambda

Lambda es un servicio basado en eventos que crea automáticamente los recursos necesarios para ejecutar código como respuesta a un evento. Lambda utiliza funciones, es decir, instrucciones de grupo que contienen el código, las dependencias y las configuraciones para ejecutar un recurso. Las funciones se ejecutan automáticamente cuando detectan un disparador, es decir, un grupo de actividades que invocan la función. Un disparador puede ser cualquier cosa, por ejemplo una aplicación que realiza una llamada a la API, un servicio de AWS de la cuenta que activa un recurso, etc. Cuando se activen, las funciones procesarán eventos, que son documentos JSON que contienen los datos que se van a modificar.

Lambda es ideal para ejecutar código sin tener que aprovisionar los recursos para ejecutarlo. Tomemos este caso de uso del blog Frontend web y móvil. Este caso de uso es algo parecido al que se muestra en la sección de DynamoDB. En esta aplicación, la API de GraphQL es responsable de definir las operaciones para acciones como la incorporación de publicaciones (mutaciones) y la obtención de esos datos (consultas). Para implementar la funcionalidad de sus operaciones (por ejemplo, `getPostsByAuthor ( author: String ! ) : [ Post ]` o `getPost ( id: String ! ) : Post`), utilizan las funciones de Lambda para procesar las solicitudes entrantes. En Opción 2: AWS AppSync con el solucionador de Lambda, utilizan el servicio de AWS AppSync para mantener su esquema y enlazar un origen de datos de Lambda a una de las operaciones. Cuando se llama a la operación, Lambda interactúa con el Amazon RDS proxy para ejecutar la lógica empresarial en la base de datos.

Amazon RDS

Amazon RDS permite crear y configurar rápidamente bases de datos relacionales. En Amazon RDS, creará una instancia de base de datos genérica que servirá como entorno de base de datos aislado en la nube. En esta instancia, utilizará un motor de base de datos, que es de hecho el software de RDBMS (PostgreSQL, MySQL, etc.). El servicio aligera gran parte del trabajo de backend, ya que proporciona escalabilidad mediante el uso de la infraestructura y los servicios de seguridad de AWS, por ejemplo la aplicación de parches y el cifrado, y reduce los costes administrativos de las implementaciones.

Tomemos el mismo caso de uso de la sección Lambda. En Opción 3: AWS AppSync con el solucionador de Amazon RDS, se presenta otra opción que consiste en enlazar la API de GraphQL de AWS AppSync directamente a Amazon RDS. Mediante una API de datos, asocian la base de datos con la API de GraphQL. Un solucionador se adjunta a un campo (normalmente una consulta, una mutación o una suscripción) e implementa las instrucciones SQL necesarias para acceder a la base de datos. Cuando el cliente realiza una solicitud de llamada del campo, el solucionador ejecuta las instrucciones y devuelve la respuesta.

Amazon EventBridge

En EventBridge, creará buses de eventos, que son canalizaciones que reciben eventos de los servicios o de las aplicaciones que adjuntes (el origen de los eventos) y los procesan de conformidad con un conjunto de reglas. Un evento es un cambio de estado de un entorno de ejecución, mientras que una regla es un conjunto de filtros para eventos. Una regla sigue un patrón de eventos o los metadatos del cambio de estado de un evento (ID, región, número de cuenta, ARN, etc.). Cuando un evento coincida con el patrón de eventos, EventBridge enviará el evento por la canalización al servicio de destino (el destino) y activará la acción especificada en la regla.

EventBridge es útil para direccionar las operaciones que cambian de estado a algún otro servicio. Tomemos este caso de uso del blog Frontend web y móvil. El ejemplo muestra una solución de comercio electrónico en la que varios equipos mantienen servicios diversos. Uno de estos servicios proporciona al cliente actualizaciones de los pedidos en cada paso de la entrega (pedido realizado, en curso, enviado, entregado, etc.) en el frontend. Sin embargo, el equipo de frontend que gestiona este servicio no tiene acceso directo a los datos del sistema de pedidos, ya que los mantiene un equipo de backend independiente. El sistema de pedidos del equipo de backend también se describe como una caja negra, por lo que es difícil obtener información sobre la forma en que estructuran sus datos. Sin embargo, el equipo de backend creó un sistema que publicaba los datos de los pedidos a través de un bus de eventos gestionado por EventBridge. Para acceder a los datos procedentes del bus de eventos y dirigirlos al frontend, el equipo de frontend ha creado un nuevo objetivo que

apunta a su API de GraphQL ubicada en AWS AppSync. También han creado una regla para enviar solo los datos relevantes para la actualización del pedido. Cuando se realiza una actualización, los datos del bus de eventos se envían a la API de GraphQL. El esquema de la API procesa los datos y, a continuación, los pasa al frontend.

Orígenes de datos none

Si no tiene pensado utilizar un origen de datos, puede configurarlo como `none`. Un origen de datos `none`, aunque se siga considerando explícitamente un origen de datos, no es un medio de almacenamiento. Por lo general, un solucionador invocará uno o más orígenes de datos en algún momento para procesar la solicitud. Sin embargo, hay situaciones en las que tal vez no sea necesario manipular un origen de datos. Si se configura el origen de datos en `none`, se ejecutará la solicitud, se omitirá el paso de invocación de datos y, a continuación, se ejecutará la respuesta.

Tomemos el mismo caso de uso de la sección EventBridge. En el esquema, la mutación procesa la actualización del estado y, a continuación, la envía a los suscriptores. Al recordar cómo funcionan los solucionadores, normalmente hay al menos una invocación al origen de datos. Sin embargo, en este escenario, el bus de eventos ya ha enviado los datos automáticamente. Esto significa que no es necesario que la mutación realice una invocación al origen de datos, sino que el estado del pedido puede gestionarse simplemente de forma local. La mutación se establece en `none`, que actúa como un valor de transferencia sin invocar el origen de datos. A continuación, se rellena el esquema con los datos, que se envían a los suscriptores.

OpenSearch

Amazon OpenSearch Service es un conjunto de herramientas para implementar la búsqueda de texto completo, la visualización de datos y el registro. Puede utilizar este servicio para consultar los datos estructurados que ha cargado.

En este servicio, creará instancias de OpenSearch. Estos se denominan nodos. En un nodo, agregará al menos un índice. Conceptualmente, los índices se parece un poco a las tablas de bases de datos relacionales. (Sin embargo, OpenSearch no es compatible con ACID, por lo que no debe usarse de esa manera). Rellenará el índice con los datos que cargue al servicio OpenSearch. Cuando se carguen los datos, se indexarán en una o más particiones que existan en el índice. Una partición es como una subdivisión del índice que contiene algunos de sus datos y se puede consultar por separado de otras particiones. Una vez cargados, los datos se estructurarán como archivos JSON denominados documentos. A continuación, puede consultar los datos del documento en el nodo.

Puntos de conexión HTTP

Puede usar puntos de conexión HTTP como orígenes de datos. AWS AppSync puede enviar solicitudes a los puntos de conexión con la información relevante, como los parámetros y la carga. La respuesta HTTP estará expuesta al solucionador, que devolverá la respuesta final cuando finalice sus operaciones.

Adición de un origen de datos

Si ha creado un origen de datos, puede enlazarlo al servicio de AWS AppSync y, más específicamente, a la API.

Console

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   a. En el panel, elija su API.
   b. En la barra lateral, seleccione Origen de datos.
2. Elija Crear origen de datos.
   a. Asigne un nombre al origen de datos. También puede asignarle una descripción, pero eso es opcional.
   b. Elija el tipo de origen de datos.
   c. Para DynamoDB, elija su región y, a continuación, la tabla dentro de la región. Para dictar las reglas de interacción con su tabla, elija crear un nuevo rol de tabla genérico o importe un rol existente para la tabla. Puede habilitar el control de versiones, que permite crear automáticamente versiones de los datos para cada solicitud cuando hay varios clientes intentando actualizar los datos al mismo tiempo. El control de versiones se utiliza para conservar y mantener múltiples variantes de datos con el fin de detectar y resolver conflictos. También puede habilitar la generación automática de esquemas, que toma el origen de datos y genera algunas de las operaciones CRUD, `List` y `Query` necesarias para acceder a ella en el esquema.

      En el caso de OpenSearch, tendrá que elegir su región y, a continuación, el dominio (clúster) de la región. Para dictar las reglas de interacción con su dominio, elija crear un nuevo rol de tabla genérico o importe un rol existente para la tabla.

Para Lambda, elija su región y, a continuación, el ARN de la función de Lambda de la región. Para dictar las reglas de interacción con su función de Lambda, elija crear un nuevo rol de tabla genérico o importe un rol existente para la tabla.

Para HTTP, introduzca su punto de conexión HTTP.

Para EventBridge, elija su región y, a continuación, el bus de eventos de la región. Para dictar las reglas de interacción con su bus de eventos, elija crear un nuevo rol de tabla genérico o importe un rol existente para la tabla.

Para RDS, elija su región y, a continuación, el almacén secreto (nombre de usuario y contraseña), el nombre de la base de datos y el esquema.

Para none, añada un origen de datos sin un origen de datos real. Esto sirve para gestionar los solucionadores de forma local y no a través de un origen de datos real.

> **Note**
>
> Si importa roles existentes, estos necesitan una política de confianza. Para obtener más información, consulte la política de confianza de IAM.

3. Seleccione Crear.

> **Note**
>
> Como alternativa, si va a crear un origen de datos de DynamoDB, puede ir a la página Esquema de la consola, elegir Crear recursos en la parte superior de la página y, a continuación, rellenar un modelo predefinido para convertirlo en una tabla. En esta opción, rellenará o importará el tipo base, configurará los datos básicos de la tabla, incluida la clave de partición, y revisará los cambios del esquema.

CLI

- Cree su origen de datos ejecutando el comando `create-data-source`.

  Para este comando en particular, deberá introducir varios parámetros:

  1. El `api-id` de su API.

2. El `name` de su tabla.
3. El `type` de su origen de datos. Según el tipo de origen de datos que elija, es posible que deba introducir una etiqueta `service-role-arn` y `-config`.

Un comando de ejemplo puede tener este aspecto:

```
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**
>
> Antes de usar el CDK, le recomendamos que consulte la documentación oficial de este, junto con la referencia del CDK de AWS AppSync.
> Los pasos que se indican a continuación solo mostrarán un ejemplo general del fragmento de código utilizado para añadir un recurso concreto. No se pretende que esta sea una solución funcional en su código de producción. También, se presupone que ya tiene una aplicación en funcionamiento.

Para añadir un origen de datos concreto, añada el constructo a su archivo de pila. Puede encontrar una lista de los tipos de orígenes de datos aquí:

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

1. En general, puede que tenga que añadir la directiva de importación al servicio que utilice. Por ejemplo, puede seguir estas formas:

```
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'
```

Por ejemplo, así es cómo se importan los servicios AWS AppSync y DynamoDB:

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

2. Algunos servicios, como RDS, requieren una configuración adicional en el archivo de pila antes de crear el origen de datos (por ejemplo, la creación de VPC, las funciones y las credenciales de acceso). Consulte los ejemplos de las páginas de CDK correspondientes para obtener más información.
3. Para la mayoría de los orígenes de datos, especialmente para los servicios de AWS, creará una nueva instancia del origen de datos en su archivo de pila. Normalmente, será como esta:

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

Por ejemplo, a continuación se muestra un ejemplo de tabla de 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**
>
> La mayoría de los orígenes de datos tendrán al menos un accesorio obligatorio (se indicará sin el símbolo ?). Consulte la documentación del CDK para ver qué accesorios se necesitan.

4. A continuación, debe enlazar el origen de datos a la API de GraphQL. El método recomendado es añadirlo al crear una función para su solucionador de canalizaciones. Por ejemplo, el fragmento de código siguiente es una función que analiza todos los elementos de una tabla de 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,
});
```

En los accesorios de `dataSource`, puede llamar a la API de GraphQL (`add_api`) y usar uno de sus métodos integrados (`addDynamoDbDataSource`) para realizar la asociación entre la tabla y la API de GraphQL. Los argumentos son el nombre de este enlace que existirá en la consola AWS AppSync (`data_source_name_in_console` en este ejemplo) y el método de tabla (`add_ddb_table`). Encontrará más información sobre este tema en la sección siguiente cuando empiece a crear solucionadores.

Existen métodos alternativos para enlazar un origen de datos. Técnicamente, podría añadir `api` a la lista de accesorios de la función de tabla. Por ejemplo, este es el fragmento de código del paso 3, pero con un accesorio `api` que contiene una API de GraphQL:

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

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

 ...
```

```
  api: add_api
});
```

Como alternativa, puede llamar al constructo `GraphqlApi` por separado:

```
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);
```

Recomendamos crear la asociación únicamente en los accesorios de la función.
De lo contrario, tendrá que enlazar la función de solucionador al origen de datos manualmente en la consola AWS AppSync (si desea seguir utilizando el valor de consola `data_source_name_in_console`) o crear una asociación independiente en la función con otro nombre como `data_source_name_in_console_2`. Esto se debe a las limitaciones en la forma en que los accesorios procesan la información.

> **Note**
>
> Deberá volver a implementar la aplicación para ver los cambios.

Política de confianza de IAM

Si utiliza un rol de IAM ya existente para el origen de datos, deberá conceder a ese rol los permisos adecuados para llevar a cabo operaciones en su recurso de AWS, por ejemplo `PutItem` en una tabla de Amazon DynamoDB. También tiene que modificar la política de confianza de ese rol para permitir que AWS AppSync la use en el acceso de recursos tal como se muestra en el ejemplo de política siguiente:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
        "Effect": "Allow",
        "Principal": {
            "Service": "appsync.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}
```

También puede añadir condiciones a su política de confianza para limitar el acceso al origen de datos según lo desee. Actualmente, las claves `SourceArn` y `SourceAccount` se pueden utilizar en estas condiciones. Por ejemplo, la política siguiente limita el acceso a su origen de datos a la cuenta `123456789012`:

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

Como alternativa, puede limitar el acceso a un origen de datos a una API específica, por ejemplo `abcdefghijklmnopq`, mediante la política siguiente:

```
{
  "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"
        }
      }
    }
  ]
}
```

Puede limitar el acceso a todas las API de AWS AppSync desde una región específica, por ejemplo `us-east-1`, mediante la política siguiente:

```
{
  "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/*"
        }
      }
    }
  ]
}
```

En la sección siguiente (Configuración de los solucionadores), añadiremos nuestra lógica empresarial de solucionador y la asociaremos a los campos de nuestro esquema para procesar los datos de nuestro origen de datos.

Para obtener más información sobre la configuración de la política de roles, consulte Modificación de un rol en la Guía del usuario de IAM.

Para obtener más información sobre el acceso entre cuentas a los solucionadores de AWS Lambda de AWSAppSync, consulte Creación de solucionadores de AWS Lambda entre cuentas para AWS AppSync.

Paso 3: Configurar solucionadores

En las secciones anteriores, aprendió a crear el esquema y la fuente de datos de GraphQL y, a continuación, los vinculó en el AWS AppSync servicio. En su esquema, tal vez haya establecido uno o más campos (operaciones) en la consulta y la mutación. Si bien el esquema describía los tipos de datos que las operaciones solicitarían al origen de datos, no se implementó el comportamiento de esas operaciones con respecto a los datos.

El comportamiento de una operación se implementa siempre en el solucionador, que esta vinculado al campo que realiza la operación. Para obtener más información sobre el funcionamiento general de los solucionadores, consulte la página Solucionadores.

En AWS AppSync, la resolución está vinculada a un tiempo de ejecución, que es el entorno en el que se ejecuta la resolución. Los tiempos de ejecución determinan el lenguaje en el que se escribirá la resolución. Actualmente, se admiten dos tiempos de ejecución: APPSYNC_JS (JavaScript) y Apache Velocity Template Language (VTL).

Al implementar los solucionadores, estos siguen una estructura general:

- Paso anterior: cuando el cliente realiza una solicitud, los datos de esta se envían a los solucionadores de los campos de esquema que se usan (normalmente, las consultas, las mutaciones y las suscripciones). El solucionador empezará a procesar los datos de la solicitud con un controlador previo paso a paso, que permite realizar algunas operaciones de preprocesamiento antes de que los datos pasen por el solucionador.
- Funciones: una vez ejecutado el paso anterior, la solicitud se pasa a la lista de funciones. La primera función de la lista se ejecuta conforme al origen de datos. Una función es un subconjunto del código de su solucionador, que contiene su propio controlador de solicitudes y respuestas. Un controlador de solicitudes toma los datos de la solicitud y realiza operaciones con el origen de datos. El controlador de respuestas procesa la respuesta del origen de datos antes de devolverla a la lista. Si hay más de una función, los datos de la solicitud se envían a la siguiente función de la lista que se ejecutará. Las funciones de la lista se ejecutan en serie en el orden definido por el desarrollador. Una vez ejecutadas todas las funciones, el resultado final se envía al paso posterior.

- Paso posterior: el paso posterior es una función de controlador que permite realizar algunas operaciones finales en la respuesta de la función final antes de pasarla a la respuesta de GraphQL.

Este flujo es un ejemplo de un solucionador de canalizaciones. Los solucionadores de canalizaciones son compatibles con ambos tiempos de ejecución. Sin embargo, esta es una explicación simplificada de lo que pueden hacer los solucionadores de canalizaciones. Además, solo describimos una posible configuración del solucionador. Para obtener más información sobre las configuraciones de resolución compatibles, consulte la descripción general de los resolutores de APPSYNC_JS o la descripción general de la plantilla de mapeo de JavaScript resolutores de VTL.

Como puede ver, los solucionadores son modulares. Para que los componentes del solucionador funcionen correctamente, deben poder observar el estado de la ejecución desde otros componentes. En la sección Solucionadores ya hemos visto que a cada componente del solucionador se le puede pasar información crítica sobre el estado de la ejecución en forma de un conjunto de argumentos (`args`, `context`, etc.). En AWS AppSync, esto lo gestiona estrictamente el. `context` Se trata de un contenedor de la información acerca del campo que se está solucionando. Aquí se puede incluir de todo, desde los argumentos que se pasan hasta los resultados, pasando por los datos de autorización, los datos del encabezado, etc. Para obtener más información sobre el contexto, consulte Referencia al objeto del contexto del solucionador para APPSYNC_JS o Referencia de contexto de las plantillas de mapeo del solucionador para VTL.

El contexto no es la única herramienta que puede utilizar para implementar su resolución. AWS AppSync admite una amplia gama de utilidades para la generación de valor, el manejo de errores, el análisis, la conversión, etc. Puede ver una lista de utilidades aquí para APPSYNC_JS o aquí para VTL.

En las siguientes secciones, verá cómo configurar solucionadores en su API de GraphQL.

Temas

- Configuración de solucionadores (JavaScript)
- Configuración de solucionadores (VTL)

Configuración de solucionadores (JavaScript)

Los solucionadores de GraphQL conectan los campos de un esquema de tipo a un origen de datos. Los solucionadores son el mecanismo mediante el cual se atienden las solicitudes.

Los solucionadores de AWS AppSync utilizan JavaScript para convertir una expresión de GraphQL en un formato que pueda utilizar el origen de datos. Como alternativa, las plantillas de asignación se pueden escribir en Apache Velocity Template Language (VTL) para convertir una expresión de GraphQL en un formato que el origen de datos pueda utilizar.

En esta sección se describe cómo configurar solucionadores mediante JavaScript. La sección Tutoriales de solucionadores (JavaScript) proporciona tutoriales detallados sobre cómo implementar solucionadores mediante JavaScript. La sección Referencia de solucionadores (JavaScript) proporciona una explicación de las operaciones de utilidad que se pueden utilizar con los solucionadores de JavaScript.

Recomendamos seguir esta guía antes de intentar utilizar alguno de los tutoriales mencionados anteriormente.

En esta sección se mostrará cómo crear y configurar solucionadores para consultas y mutaciones.

> **Note**
>
> En esta guía se supone que ha creado su esquema y que tiene al menos una consulta o mutación. Si busca suscripciones (datos en tiempo real), consulte esta guía.

En esta sección, proporcionaremos algunos pasos generales para configurar solucionadores junto con un ejemplo que usa el esquema siguiente:

```
// 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]
}
```

Creación de solucionadores de consultas básicos

En esta sección se mostrará cómo crear un solucionador de consultas básico.

Console

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   a. En el panel de API, seleccione su API de GraphQL.
   b. En la barra lateral, seleccione Esquema.
2. Introduzca los detalles del esquema y el origen de datos. Consulte las secciones Diseño del esquema y Asociar un origen de datos para obtener más información.
3. Junto al editor Esquemas, hay una ventana llamada Solucionadores. Este cuadro contiene una lista de los tipos y campos definidos en la ventana Esquema. Puede asociar solucionadores a los campos. Lo más probable es que asocie solucionadores a sus operaciones de campo. En esta sección se analizarán las configuraciones de consultas sencillas. En el tipo Consulta, seleccione Asociar junto al campo de la consulta.
4. En la página Asociar solucionador, en Tipo de solucionador, puede elegir entre solucionadores de canalización o unitarios. Para obtener más información sobre estos tipos, consulte Solucionadores. En esta guía se utilizará `pipeline resolvers`.

   > **Tip**
   >
   > Al crear solucionadores de canalización, sus orígenes de datos se asociarán a las funciones de canalización. Las funciones se crean después de crear el propio solucionador de canalizaciones, por lo que no existe la opción de configurarlo en esta página. Si utiliza un solucionador unitario, el origen de datos se vincula directamente al solucionador, por lo que deberá configurarlo en esta página.

   En Tiempo de ejecución del solucionador, seleccione `APPSYNC_JS` para habilitar el tiempo de ejecución de JavaScript.

5. Puede habilitar el almacenamiento en caché para esta API. También recomendamos desactivar esta característica por el momento. Seleccione Crear.
6. En la página Editar solucionador, hay un editor de código llamado Código de solucionador que le permite implementar la lógica para el controlador y la respuesta del solucionador (antes y después de los pasos). Para obtener más información, consulte Descripción general de los solucionadores de JavaScript.

> **Note**
>
> En nuestro ejemplo, vamos a dejar la solicitud en blanco y la respuesta establecida para devolver el último resultado del origen de datos del contexto:
>
> ```
> import {util} from '@aws-appsync/utils';
>
> export function request(ctx) {
>     return {};
> }
>
> export function response(ctx) {
>     return ctx.prev.result;
> }
> ```

Bajo esta sección, hay una tabla llamada Funciones. Las funciones permiten implementar código que se puede reutilizar en varios solucionadores. En lugar de tener que reescribir o copiar el código constantemente, puede almacenar el código fuente como una función para añadirla a un solucionador siempre que lo necesite.

Las funciones constituyen la mayor parte de la lista de operaciones de una canalización. Al utilizar varias funciones en un solucionador, se establece el orden de las funciones y estas se ejecutan en ese orden en secuencia. Se ejecutan después de la función de solicitud y antes de que comience la función de respuesta.

Para añadir una función nueva, en Funciones, seleccione Añadir función y, a continuación, Crear nueva función. Como alternativa, puede que vea el botón Crear función para seleccionarlo.

a. Elija un origen de datos. Será el origen de datos en el que actuará el solucionador.

> **Note**
>
> En nuestro ejemplo, asociamos un solucionador para `getPost`, que recupera un objeto `Post` según el `id`. Supongamos que ya hemos configurado una tabla de DynamoDB para este esquema. Su clave de partición está establecida en `id` y está vacía.

b. Introduzca un `Function name`.

c. En Código de función, deberá implementar el comportamiento de la función. Esto puede resultar confuso, pero cada función tendrá su propio controlador local de solicitudes y respuestas. Se ejecuta la solicitud, luego se invoca el origen de datos para gestionar la solicitud y, a continuación, el controlador de respuestas procesa la respuesta del origen de datos. El resultado se almacena en el objeto de contexto. Después, se ejecutará la siguiente función de la lista o se pasará al controlador de respuestas posterior al paso si es el último.

> **Note**
>
> En nuestro ejemplo, asociamos un solucionador a `getPost`, que obtiene una lista de objetos `Post` del origen de datos. Nuestra función de solicitud solicitará los datos de nuestra tabla, la tabla pasará su respuesta al contexto (ctx) y, a continuación, la respuesta devolverá el resultado en el contexto. El punto fuerte de AWS AppSync reside en su interconexión con otros servicios de AWS. Dado que utilizamos DynamoDB, tenemos un conjunto de operaciones para simplificar este tipo de acciones. También tenemos algunos ejemplos repetitivos para otros tipos de orígenes de datos.
> Nuestro código será como este:

```
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;
}
```

En este paso, hemos añadido dos funciones:

- `request`: el controlador de solicitudes realiza la operación de recuperación en el origen de datos. El argumento contiene el objeto de contexto (`ctx`) o datos que están disponibles para todos los solucionadores que realizan una operación determinada. Por ejemplo, puede contener datos de autorización, los nombres de los campos que se están resolviendo, etc. La instrucción return realiza una operación `Scan` (vea ejemplos aquí). Dado que trabajamos con DynamoDB, podemos usar algunas de las operaciones de ese servicio. El escaneo realiza una búsqueda básica de todos los elementos de nuestra tabla. El resultado de esta operación se almacena en el objeto de contexto como un contenedor de `result` antes de pasarlo al controlador de respuestas. La `request` se ejecuta antes de la respuesta en la canalización.
- `response`: el controlador de respuestas que devuelve la salida de `request`. El argumento es el objeto de contexto actualizado y la declaración de retorno es `ctx.prev.result`. En este punto de la guía, es posible que no esté familiarizado con este valor. `ctx` hace referencia al objeto de contexto. `prev`, a la operación anterior en la canalización, que era nuestra `request`. `result` contiene los resultados del solucionador a medida que se desplaza por la canalización. Si lo junta todo, `ctx.prev.result` devuelve el resultado de la última operación realizada, que fue el controlador de solicitudes.

d. Cuando haya terminado, elija Crear.

7. De vuelta a la pantalla de solucionador, en Funciones, seleccione el menú desplegable Añadir función y añada la función a su lista de funciones.
8. Seleccione Guardar para actualizar el solucionador.

CLI

Para añadir su función

- Cree una función para su solucionador de canalizaciones mediante el comando `create-function`.

  Para este comando en particular, deberá introducir varios parámetros:

  1. El `api-id` de su API.
  2. El `name` de la función de la consola de AWS AppSync.
  3. El `data-source-name` o el nombre del origen de datos que la función utilizará. Ya debe estar creado y vinculado a su API de GraphQL en el servicio de AWS AppSync.
  4. El `runtime` o el entorno y el idioma de la función. Para JavaScript, el nombre debe ser `APPSYNC_JS` y el tiempo de ejecución, `1.0.0`.
  5. El `code` o los controladores de solicitud y respuesta de su función. Si bien puede escribirlo manualmente, es mucho más fácil agregarlo a un archivo.txt (o un formato similar) y luego pasarlo como argumento.

     > **Note**
     >
     > Nuestro código de consulta estará en un archivo que se transfiere como argumento:
     >
     > ```
     > 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;
     > }
     > ```

  Un comando de ejemplo puede tener este aspecto:

```
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}
```

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

```
{
    "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**
>
> Asegúrese de grabar el `functionId` en algún lugar, ya que se utilizará para asociar la función al solucionador.

Para crear su primer solucionador

- Cree una función de canalización para `Query` ejecutando el comando `create-resolver`.

  Para este comando en particular, deberá introducir varios parámetros:

  1. El `api-id` de su API.

2. El `type-name` o el tipo de objeto especial de su esquema (consulta, mutación, suscripción).
3. El `field-name` o la operación de campo de dentro del tipo de objeto especial al que desee asociar el solucionador.
4. El `kind`, que especifica un solucionador unitario o de canalización. Configúrelo en `PIPELINE` para habilitar las funciones de canalización.
5. La `pipeline-config` o las funciones que se van a asociar al solucionador. Asegúrese de conocer los valores de `functionId` de sus funciones. El orden de la lista es importante.
6. El `runtime`, que era `APPSYNC_JS` (JavaScript). La `runtimeVersion` actualmente es `1.0.0`.
7. El `code`, que contiene los controladores de los pasos de antes y de después.

> **Note**
>
> Nuestro código de consulta estará en un archivo que se transfiere como argumento:

```
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;
}
```

Un comando de ejemplo puede tener este aspecto:

```
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}
```

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

```
{
    "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**
>
> Antes de usar el CDK, le recomendamos que consulte la documentación oficial de este, junto con la referencia del CDK de AWS AppSync.
> Los pasos que se indican a continuación solo mostrarán un ejemplo general del fragmento de código utilizado para añadir un recurso concreto. No se pretende que esta sea una solución funcional en su código de producción. También, se presupone que ya tiene una aplicación en funcionamiento.

Una aplicación básica necesitará lo siguiente:

1. Directivas de importación de servicios
2. Código de esquema
3. Generador de orígenes de datos
4. Código de función
5. Código de solucionador

En las secciones Diseñar el esquema y Asociar un origen de datos, sabemos que el archivo de pila incluirá las directivas de importación del siguiente formato:

```
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**
>
> En las secciones anteriores, solo hemos explicado cómo importar constructos de AWS AppSync. En código real, deberá importar más servicios simplemente para ejecutar la aplicación. En nuestro ejemplo, si quisiéramos crear una aplicación CDK muy sencilla, importaríamos al menos el servicio de AWS AppSync junto con nuestro origen de datos, que era una tabla de DynamoDB. También necesitaríamos importar otros constructos para implementar la aplicación:
>
> ```
> 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';
```

Resumamos cada uno de estos elementos:

- `import * as cdk from 'aws-cdk-lib';`: permite definir la aplicación del CDK y sus constructos, como la pila. También contiene funciones de utilidad útiles para nuestra aplicación, como la manipulación de metadatos. Si conoce esta directiva de importación, pero se pregunta por qué la biblioteca principal de cdk no se utiliza aquí, consulte la página Migración.
- `import * as appsync from 'aws-cdk-lib/aws-appsync';`: importa el servicio de AWS AppSync.
- `import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';`: importa el servicio de DynamoDB.
- `import { Construct } from 'constructs';`: lo necesitamos para definir el constructo raíz.

El tipo de importación depende de los servicios que llame. Te recomendamos consultar la documentación del CDK para ver ejemplos. El esquema de la parte superior de la página será un archivo independiente en su aplicación de CDK, en forma de archivo `.graphql`. En el archivo de pila, podemos asociarlo a un nuevo GraphQL mediante el siguiente formulario:

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

Note

En el ámbito `add_api`, añadiremos una nueva API de GraphQL con la palabra clave `new` seguida de `appsync.GraphqlApi(scope: Construct, id: string , props: GraphqlApiProps)`. Nuestro ámbito es `this`, el identificador de CFN es `graphQL-example`, y nuestros accesorios son `my-first-api` (nombre de la API en la consola) y `schema.graphql` (la ruta absoluta al archivo de esquema).

Para añadir un origen de datos, añádala primero a la pila. Luego, asóciela a la API de GraphQL mediante el método específico del origen. La asociación se producirá cuando ponga en funcionamiento su solucionador. Mientras tanto, usemos un ejemplo para crear la tabla de DynamoDB con `dynamodb.Table`:

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

> **Note**
>
> Si usáramos esto en nuestro ejemplo, añadiríamos una nueva tabla de DynamoDB con el identificador de CFN de `posts-table` y una clave de partición de `id (S)`.

A continuación, debemos implementar nuestro solucionador en el archivo de pila. A continuación, se muestra un ejemplo de una consulta sencilla que busca todos los elementos de una tabla de 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**
>
> En primer lugar, creamos una función llamada `add_func`. Este orden de creación puede parecer un poco contradictorio, pero hay que crear las funciones en el solucionador de canalizaciones antes de crear el solucionador propiamente dicho. Una función tiene el siguiente formato:
>
> ```
> AppsyncFunction(scope: Construct, id: string, props: AppsyncFunctionProps)
> ```
>
> Nuestro alcance era `this`, nuestro identificador de CFN era `func-get-posts` y nuestros accesorios contenían los detalles reales de la función. Dentro de los accesorios, hemos incluido:
>
> - El `name` de la función que estará presente en la consola de AWS AppSync (`get_posts_func_1`).
> - La API de GraphQL que hemos creado antes (`add_api`).
> - El origen de datos; este es el punto en el que vinculamos el origen de datos al valor de la API GraphQL y, a continuación, la asociamos a la función. Tomamos la tabla que hemos creado (`add_ddb_table`) y la asociamos a la API GraphQL (`add_api`) mediante uno de los métodos de `GraphqlApi` (`addDynamoDbDataSource`). El valor del identificador (`table-for-posts`) es el nombre del origen de datos de la consola de AWS AppSync. Para obtener una lista de métodos específicos del origen, consulte las páginas siguientes:
>   - DynamoDbDataSource

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

- El código contiene los controladores de solicitudes y respuestas de nuestra función, que son fáciles de escanear y devolver.
- El tiempo de ejecución especifica que queremos usar la versión 1.0.0 del tiempo de ejecución APPSYNC_JS. Tenga en cuenta que actualmente esta es la única versión disponible para APPSYNC_JS.

Luego, debemos asociar la función al solucionador de canalización. Hemos creado nuestro solucionador usando este formulario:

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

Nuestro alcance era `this`, nuestro identificador de CFN era `pipeline-resolver-get-posts` y nuestros accesorios contenían los detalles reales de la función. Dentro de los accesorios, hemos incluido:

- La API de GraphQL que hemos creado antes (`add_api`).
- El nombre del tipo de objeto especial; se trata de una operación de consulta, por lo que simplemente hemos añadido el valor `Query`.
- El nombre de campo (`getPost`) es el nombre del campo del esquema en el tipo `Query`.
- El código contiene sus controladores de antes y después. Nuestro ejemplo simplemente devuelve los resultados que estaban en el contexto después de que la función realizara su operación.
- El tiempo de ejecución especifica que queremos usar la versión 1.0.0 del tiempo de ejecución APPSYNC_JS. Tenga en cuenta que actualmente esta es la única versión disponible para APPSYNC_JS.

- La configuración de la canalización contiene la referencia a la función que hemos creado (`add_func`).

Como resumen de lo que ha ocurrido en este ejemplo, has visto una función de AWS AppSync que ha implementado un controlador de solicitudes y respuestas. La función se ha encargado de interactuar con el origen de datos. El controlador de solicitudes ha enviado una operación de `Scan` a AWS AppSync que le indicaba qué operación debía realizar para su origen de datos de DynamoDB. El controlador de respuestas ha devuelto la lista de elementos (`ctx.result.items`). A continuación, la lista de elementos se ha asignado automáticamente al tipo GraphQL de `Post`.

Creación de solucionadores de mutaciones básicos

En esta sección se mostrará cómo crear un solucionador de mutaciones básico.

Console

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   a. En el panel de API, seleccione su API de GraphQL.
   b. En la barra lateral, seleccione Esquema.
2. En la sección Solucionadores y el tipo Mutación, seleccione Asociar junto al campo.

   > **Note**
   >
   > En nuestro ejemplo, vamos a asociar un solucionador para `createPost`, que añade un objeto `Post` a nuestra table. Supongamos que utilizamos la misma tabla de DynamoDB de la última sección. Su clave de partición está establecida en `id` y está vacía.

3. En la página Asociar solucionador, en Tipo de solucionador, elija `pipeline resolvers`. Le recordamos que puede encontrar más información sobre los solucionadores aquí. En Tiempo de ejecución del solucionador, seleccione `APPSYNC_JS` para habilitar el tiempo de ejecución de JavaScript.
4. Puede habilitar el almacenamiento en caché para esta API. También recomendamos desactivar esta característica por el momento. Seleccione Crear.
5. Seleccione Añadir función y, a continuación, Crear nueva función. Como alternativa, puede que vea el botón Crear función para seleccionarlo.

a. Elija el origen su origen de datos. Debería ser el origen cuyos datos va a manipular con la mutación.

b. Introduzca un `Function name`.

c. En Código de función, deberá implementar el comportamiento de la función. Se trata de una mutación, por lo que lo ideal es que la solicitud realice alguna operación de cambio de estado en el origen de datos invocado. La función de respuesta procesará el resultado.

> **Note**
>
> `createPost` va a añadir o "poner" una nueva `Post` en la tabla con nuestros parámetros como datos. Podríamos añadir algo parecido a esto:
>
> ```
> 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;
> }
> ```
>
> En este paso, hemos añadido también las funciones `request` y `response`:
>
> - `request`: el controlador de solicitudes acepta el contexto como argumento. La instrucción return del controlador de solicitudes ejecuta un comando `PutItem`, que es una operación de DynamoDB integrada (consulte aquí o aquí para ver ejemplos). El comando `PutItem` añade un objeto de

Post a nuestra tabla de DynamoDB tomando el valor key de la partición (generado automáticamente por util.autoid()) y los attributes de la entrada del argumento de contexto (son los valores que transferiremos en nuestra solicitud). La key es el id y attributes son los argumentos de los campos date y title. Ambos se preformatean mediante el elemento auxiliar util.dynamodb.toMapValues para que funcionen con la tabla de DynamoDB.

- response: la respuesta acepta el contexto actualizado y devuelve el resultado del controlador de solicitudes.

d. Cuando haya terminado, elija Crear.

6. De vuelta a la pantalla de solucionador, en Funciones, seleccione el menú desplegable Añadir función y añada la función a su lista de funciones.
7. Seleccione Guardar para actualizar el solucionador.

CLI

Para añadir su función

- Cree una función para su solucionador de canalizaciones mediante el comando create-function.

  Para este comando en particular, deberá introducir varios parámetros:

  1. El api-id de su API.
  2. El name de la función de la consola de AWS AppSync.
  3. El data-source-name o el nombre del origen de datos que la función utilizará. Ya debe estar creado y vinculado a su API de GraphQL en el servicio de AWS AppSync.
  4. El runtime o el entorno y el idioma de la función. Para JavaScript, el nombre debe ser APPSYNC_JS y el tiempo de ejecución, 1.0.0.
  5. El code o los controladores de solicitud y respuesta de su función. Si bien puede escribirlo manualmente, es mucho más fácil añadirlo a un archivo.txt (o un formato similar) y luego transferirlo como argumento.

> **Note**
>
> Nuestro código de consulta estará en un archivo que se transfiere como argumento:

```
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;
}
```

Un comando de ejemplo puede tener este aspecto:

```
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}
```

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

```
{
    "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**
>
> Asegúrese de grabar el `functionId` en algún lugar, ya que se utilizará para asociar la función al solucionador.

Para crear su primer solucionador

- Cree una función de canalización para `Mutation` ejecutando el comando `create-resolver`.

  Para este comando en particular, deberá introducir varios parámetros:

  1. El `api-id` de su API.
  2. El `type-name` o el tipo de objeto especial de su esquema (consulta, mutación, suscripción).
  3. El `field-name` o la operación de campo de dentro del tipo de objeto especial al que desee asociar el solucionador.
  4. El `kind`, que especifica un solucionador unitario o de canalización. Configúrelo en `PIPELINE` para habilitar las funciones de canalización.
  5. La `pipeline-config` o las funciones que se van a asociar al solucionador. Asegúrese de conocer los valores de `functionId` de sus funciones. El orden de la lista es importante.
  6. El `runtime`, que era `APPSYNC_JS` (JavaScript). La `runtimeVersion` actualmente es `1.0.0`.

7. El `code`, que contiene los pasos de antes y de después.

> **Note**
>
> Nuestro código de consulta estará en un archivo que se transfiere como argumento:

```
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;
}
```

Un comando de ejemplo puede tener este aspecto:

```
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}
```

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

```
{
    "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**
>
> Antes de usar el CDK, le recomendamos que consulte la documentación oficial de este, junto con la referencia del CDK de AWS AppSync.
> Los pasos que se indican a continuación solo mostrarán un ejemplo general del fragmento de código utilizado para añadir un recurso concreto. No se pretende que esta sea una solución funcional en su código de producción. También, se presupone que ya tiene una aplicación en funcionamiento.

- Para realizar una mutación, si usted se encuentra en el mismo proyecto, puede añadirla al archivo de pila como la consulta. Esta es una función modificada y un solucionador para una mutación que añade una nueva `Post` a la tabla:

```
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

Como esta mutación y la consulta tienen una estructura similar, nos limitaremos a explicar los cambios que hemos hecho para realizar la mutación.
En la función, hemos cambiado el identificador de CFN por `func-add-post` y el nombre por `add_posts_func_1` a para reflejar el hecho de que estamos añadiendo `Posts` a la tabla. En el origen de datos, hemos hecho una nueva asociación con

nuestra tabla (`add_ddb_table`) en la consola de AWS AppSync como `table-for-posts-2` porque el método `addDynamoDbDataSource` así lo requiere.
Tenga en cuenta que esta nueva asociación sigue utilizando la misma tabla que creamos antes, pero ahora tenemos dos conexiones hacia ella en la consola de AWS AppSync: una para la consulta como `table-for-posts` y otra para la mutación como `table-for-posts-2`. El código se ha modificado para añadir una `Post` mediante la generación automática de su valor de `id` y la aceptación de la entrada de un cliente para el resto de los campos.
En el solucionador, hemos cambiado el valor del identificador por `pipeline-resolver-create-posts` para reflejar el hecho de que estamos añadiendo `Posts` a la tabla. Para reflejar la mutación en el esquema, se cambiado el nombre del tipo por `Mutation`, y el nombre, `createPost`. La configuración de la canalización se ha establecido en nuestra nueva función de mutación `add_func_2`.

Para resumir lo que ocurre en este ejemplo, AWS AppSync convierte automáticamente los argumentos definidos en el campo `createPost` del esquema de GraphQL en operaciones de DynamoDB. El ejemplo almacena los registros en DynamoDB mediante una clave de `id`, que se crea automáticamente con nuestro texto auxiliar `util.autoId()`. Todos los demás campos que pase a los argumentos de contexto (`ctx.args.input`) a partir de las solicitudes realizadas en la consola de AWS AppSync o de otro modo se almacenarán como los atributos de la tabla. Tanto la clave como los atributos se asignan automáticamente a un formato de DynamoDB compatible mediante el texto auxiliar `util.dynamodb.toMapValues(values)`.

AWS AppSync también es compatible con los flujos de trabajo de prueba y depuración para editar los solucionadores. Puede usar un objeto `context` simulado para ver el valor transformado de la plantilla antes de invocarlo. De forma opcional, puede ver la solicitud completa en un origen de datos de forma interactiva cuando ejecute una consulta. Para obtener más información, consulte Solucionadores de pruebas y depuración (JavaScript) y Monitoreo y registro.

Solucionadores avanzados

Si sigue la sección de paginación opcional de Diseño del esquema, deberá añadir el solucionador a la solicitud para poder utilizar la paginación. En nuestro ejemplo, se ha utilizado una paginación de consultas llamada `getPosts` para devolver solo una parte de los elementos solicitados a la vez. Nuestro código de solucionador en ese campo puede ser como este:

```
/**
```

```
 * 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 };
}
```

En la solicitud, transferimos el contexto de esta. Nuestro `limit` es *20*, lo que significa que devolvemos hasta 20 `Posts` en la primera consulta. Nuestro cursor `nextToken` está fijo en la primera entrada de `Post` en el origen de datos. Se transfieren a los argumentos. A continuación, la solicitud escanea desde la primera `Post` hasta el número límite de escaneo. El origen de datos almacena el resultado en el contexto, que se transfiere a la respuesta. La respuesta devuelve las `Posts` recuperadas y, a continuación, establece el `nextToken` en la entrada de la `Post` que se encuentra justo después del límite. La siguiente solicitud se envía para hacer exactamente lo mismo, pero empezando por el desplazamiento justo después de la primera consulta. Tenga en cuenta que este tipo de solicitudes se realizan de forma secuencial y no en paralelo.

Solucionadores de pruebas y depuración (JavaScript)

AWS AppSync ejecuta solucionadores en un campo de GraphQL con respecto a un origen de datos. Cuando se trabaja con solucionadores de canalizaciones, las funciones interactúan con los orígenes de datos. Tal y como se explica en JavaScript resolvers overview, las funciones se comunican con los orígenes de datos mediante controladores de solicitudes y respuestas escritos en JavaScript y que se ejecutan en el tiempo de ejecución `APPSYNC_JS`. Esto le permite proporcionar lógica y condiciones personalizadas antes y después de comunicarse con el origen de datos.

A fin de ayudar a los desarrolladores a escribir, probar y depurar estos solucionadores, la consola de AWS AppSync también proporciona herramientas para crear una solicitud y una respuesta de GraphQL con datos simulados, incluso hasta llegar al solucionador de cada campo individual. Además, puede realizar consultas, mutaciones y suscripciones en la consola de AWS AppSync y ver un flujo de registro detallado de toda la solicitud desde Amazon CloudWatch. Esto incluye los resultados del origen de datos.

Prueba con datos simulados

Cuando se invoca a un solucionador de GraphQL, este contiene un objeto `context` que incluye información útil sobre la solicitud. Por ejemplo, contiene los argumentos de un cliente, información de identidad y datos del campo principal de GraphQL. También almacena los resultados del origen de datos, que puede usar en el controlador de respuestas. Si desea más información acerca de esta estructura y las utilidades auxiliares disponibles para programar, consulte Resolver context object reference.

Al escribir o editar una función de solucionador, puede pasar un objeto simulado o de contexto de prueba al editor de la consola. Esto le permite ver cómo se evalúan los controladores de solicitudes y de respuestas sin que se utilice en realidad ningún origen de datos. Por ejemplo, puede transferir un argumento `firstname: Shaggy` de prueba y ver cómo se evalúa cuando utiliza `ctx.args.firstname` en el código de la plantilla. También puede probar la evaluación de cualquier utilidad auxiliar, como `util.autoId()` o `util.time.nowISO8601()`.

Prueba de solucionadores

En este ejemplo, se utilizará la consola de AWS AppSync para probar los solucionadores.

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   a. En el panel de API, seleccione su API de GraphQL.
   b. En la barra lateral, seleccione Funciones.
2. Seleccione una función existente.
3. En la parte superior de la página Actualizar función, seleccione Seleccionar contexto de prueba y, a continuación, Crear nuevo contexto.
4. Seleccione un objeto de contexto de ejemplo o rellene el JSON manualmente en la ventana Configurar contexto de prueba que aparece a continuación.
5. Introduzca un Nombre de contexto de texto.
6. Seleccione el botón Save (Guardar).
7. Para evaluar el solucionador con el objeto de contexto simulado, elija Run Test (Ejecutar prueba).

Veamos un ejemplo más práctico. Imagine que tiene una aplicación que almacena un tipo de GraphQL llamado `Dog`, que genera automáticamente un identificador para los objetos y los almacena en Amazon DynamoDB. También desea escribir algunos valores tomándolos de los argumentos

de una mutación de GraphQL y permitir que solo determinados usuarios vean la respuesta. En el siguiente fragmento de código se muestra el aspecto que podría tener el esquema:

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

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

Puede escribir una función de AWS AppSync y añadirla a su solucionador de `addDog` para gestionar la mutación. Para probar la función de AWS AppSync, puede rellenar un objeto de contexto como en el siguiente ejemplo. Se incluyen argumentos del cliente como `name` y `age`, así como un `username` rellenado en el objeto `identity`:

```
{
    "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"
    }
}
```

Puede probar la función de AWS AppSync mediante el siguiente código:

```
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();
}
```

El controlador de solicitudes y respuestas evaluado tiene los datos de su objeto de contexto de prueba y el valor generado de `util.autoId()`. Además, si cambia el `username` por un valor distinto de `Nadia`, no se devolverán resultados, porque la comprobación de autorización daría error. Para obtener más información acerca del control de acceso preciso, consulte Casos de uso de autorizaciones.

Prueba de los controladores de solicitudes y respuestas con las API de AWS AppSync

Puede usar el comando de API `EvaluateCode` para probar el código de forma remota con datos simulados. Para empezar a usar el comando, asegúrese de haber añadido el permiso `appsync:evaluateMappingCode` a su política. Por ejemplo:

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

Puede utilizar el comando mediante la AWS CLI o los SDK de AWS. Tomemos como ejemplo el esquema Dog y sus controladores de solicitudes y respuestas de la función AWS AppSync de la sección anterior. Con la CLI de la estación local, guarde el código en un archivo denominado

`code.js` y, a continuación, guarde el objeto `context` en un archivo denominado `context.json`. Ejecute el siguiente comando desde el intérprete de comandos:

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

La respuesta contiene un `evaluationResult` que incluye la carga útil devuelta por el controlador. También contiene un objeto `logs` que incluye la lista de registros generados por el controlador durante la evaluación. Esto facilita la depuración de la ejecución de código y la consulta de información sobre la evaluación como ayuda para solucionar problemas. Por ejemplo:

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

El `evaluationResult` se puede analizar como JSON, lo que da como resultado:

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

Cuando se utiliza el SDK, puede incorporar fácilmente pruebas de su conjunto de pruebas favorito para validar el comportamiento de los controladores. Recomendamos crear pruebas con el marco de pruebas Jest, pero cualquier conjunto de pruebas funciona. En el siguiente fragmento de código se muestra una ejecución de validación hipotética. Tenga en cuenta que esperamos que la respuesta de la evaluación sea un JSON válido, por lo que utilizamos `JSON.parse` para recuperar el JSON de la respuesta de cadena:

```
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)
})
```

Esto produce el siguiente resultado:

```
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
```

Depuración de una consulta en tiempo real

No hay nada que pueda reemplazar a una prueba de extremo a extremo y el registro para depurar una aplicación de producción. AWS AppSync permite registrar los errores y todos los detalles de cada solicitud mediante Amazon CloudWatch. Además, puede utilizar la consola de AWS AppSync para probar las consultas, mutaciones y suscripciones de GraphQL y enviar los datos de registro de cada solicitud al editor de consultas para depurarlas en tiempo real. Para las suscripciones, los registros muestran la información del tiempo de conexión.

Para ello, debe haber habilitado previamente los registros de Amazon CloudWatch como se describe en Monitoreo y registro. A continuación, en la consola de AWS AppSync, elija la pestaña Consultas y escriba en una consulta de GraphQL válida. En la sección inferior derecha, haga clic y arrastre la ventana Registros para abrir la vista de registros. Utilice el icono de flecha de reproducción de la parte superior de la página para ejecutar la consulta de GraphQL. Al cabo de un momento, los registros completos de la solicitud y la respuesta de la operación se enviarán a esta sección y podrá verlos en la consola.

Solucionadores de canalización (JavaScript)

AWS AppSync ejecuta solucionadores en un campo de GraphQL. En algunos casos, las aplicaciones requieren la ejecución de varias operaciones para resolver un único campo de GraphQL. Con los solucionadores de canalización, los desarrolladores ahora pueden componer operaciones llamadas Funciones y ejecutarlas de forma secuencial. Los solucionadores de canalización son útiles para las aplicaciones que, por ejemplo, requieren realizar una comprobación de autorización antes de recuperar datos para un campo.

Para obtener más información sobre la arquitectura de un solucionador de canalización de JavaScript, consulte la Descripción general de los solucionadores de JavaScript.

Crear un solucionador de canalización

En la consola de AWS AppSync, vaya a la página Esquema.

Guarde el siguiente esquema:

```
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
}
```

Vamos a conectar un solucionador de canalización al campo signUp en el tipo Mutation (Mutación). En el tipo Mutación en el lado derecho, elija Asociar junto al campo de mutación `signUp`. Configura el solucionador en `pipeline resolver` y el tiempo de ejecución en `APPSYNC_JS` y, a continuación, crea el solucionador.

Nuestro solucionador de canalización inicia sesión a un usuario. Para ello, primero valida la entrada de dirección de correo electrónico y, a continuación, guarda al usuario en el sistema. Vamos a encapsular la validación de correo electrónico dentro de una función validateEmail y el guardado del usuario dentro de una función saveUser. La función validateEmail se ejecuta en primer lugar y, si el correo electrónico es válido, se ejecuta la función saveUser.

El flujo de ejecución será como se indica a continuación:

1. Controlador de solicitudes de solucionador Mutation.signUp
2. Función validateEmail
3. Función saveUser
4. Controlador de respuestas de solucionador Mutation.signup

Dado que probablemente reutilizaremos la función validateEmail en otros solucionadores de nuestra API, queremos para evitar el acceso a `ctx.args`, ya que estos cambiarán de un campo GraphQL a otro. En su lugar, podemos utilizar `ctx.stash` para almacenar el atributo de correo electrónico desde el argumento de campo de entrada `signUp(input: Signup)`.

Actualice su código de solucionador sustituyendo sus funciones de solicitud y respuesta:

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

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

Seleccione Crear o Guardar para actualizar el solucionador.

Crear una función

En la página del solucionador de canalización, en la sección Funciones, haga clic en Agregar función y luego en Crear la función nueva. También es posible crear funciones sin tener que recurrir a la

página del solucionador; para ello, en la consola de AWS AppSync, vaya a la página Funciones. Elija el botón Create a function (Crear una función). Vamos a crear una función que comprueba si un mensaje de correo electrónico es válido y proviene de un dominio específico. Si el correo electrónico no es válido, la función genera un error. De lo contrario, reenvía cualquier entrada.

Asegúrese de haber creado un origen de datos del tipo NINGUNO. Elija este origen de datos en la lista de Nombre del origen de datos. Para el nombre de a función, escriba `validateEmail`. En el área de código de la función, sobrescriba todo con este fragmento:

```
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;
}
```

Revise la información indicada y, a continuación, seleccione Crear. Acabamos de crear nuestra función validateEmail. Repita estos pasos para crear la función SaveUser con el siguiente código (para simplificar, utilizamos un origen de datos NINGUNO y simulamos que el usuario se ha guardado en el sistema después de ejecutar la función):

```
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;
```

```
}
```

Acabamos de crear la función saveUser.

Adición de una función a un solucionador de canalización

Nuestras funciones deberían haberse agregado automáticamente al solucionador de canalización que acabamos de crear. Si este no es el caso, o si ha creado las funciones a través de la página Funciones, puede volver a hacer clic en Agregar función en la página `signUp` del solucionador para asociarlas. Agregue las funciones validateEmail y saveUser al solucionador. La función validateEmail se debe colocar antes de la función saveUser. A medida que agrega más funciones, puede utilizar las opciones mover hacia arriba y mover hacia abajo para reorganizar el orden de ejecución de las funciones. Revise sus cambios y, a continuación, elija Guardar.

Ejecutar una consulta

En la consola de AWS AppSync, vaya a la página Consultas. En el explorador, asegúrese de utilizar la mutación. Si no es así, seleccione `Mutation` en la lista desplegable y, a continuación, elija +. Escriba la siguiente consulta:

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

Debería devolver algo parecido a esto:

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

Hemos registrado correctamente nuestro usuario y validado el correo electrónico de entrada utilizando un solucionador de canalización.

Configuración de solucionadores (VTL)

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

Los solucionadores de GraphQL conectan los campos de un esquema de tipo a un origen de datos. Los solucionadores son el mecanismo mediante el cual se atienden las solicitudes. AWS AppSync puede crear y conectar solucionadores automáticamente desde un esquema o crear un esquema y conectar solucionadores desde una tabla existente sin que tenga que escribir código.

Los solucionadores de AWS AppSync utilizan JavaScript para convertir una expresión de GraphQL en un formato que pueda utilizar el origen de datos. Como alternativa, las plantillas de asignación se pueden escribir en Apache Velocity Template Language (VTL) para convertir una expresión de GraphQL en un formato que el origen de datos pueda utilizar.

En esta sección se mostrará cómo configurar los solucionadores mediante VTL. Puede consultar una guía de programación a modo de tutorial para escribir solucionadores en Guía de programación de plantillas de asignación de solucionadores y herramientas de ayuda para la programación en Referencia de contexto de las plantillas de asignación de solucionadores. AWS AppSync también tiene flujos de prueba y depuración integrados que se pueden usar al editar o crear desde cero. Para obtener más información, consulte la sección sobre la prueba y depuración de solucionadores.

Recomendamos seguir esta guía antes de intentar utilizar alguno de los tutoriales mencionados anteriormente.

En esta sección, veremos cómo crear un solucionador, añadir un solucionador para mutaciones y usar configuraciones avanzadas.

Cree su primer solucionador

Siguiendo los ejemplos de las secciones anteriores, el primer paso es crear un solucionador para su tipo de `Query`.

Console

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.

a. En el panel de API, seleccione su API de GraphQL.

b. En la barra lateral, seleccione Esquema.

2. En la parte derecha de la página, hay una ventana llamada Solucionadores. Este cuadro contiene una lista de los tipos y campos definidos en la ventana Esquema del lado izquierdo de la página. Puede asociar solucionadores a los campos. Por ejemplo, en el tipo de Consulta, seleccione Asociar junto al campo `getTodos`.

3. En la página Crear un solucionador, elija el origen de datos que creó en la guía Asociar un origen de datos. En la ventana Configurar plantillas de asignación, puede elegir las plantillas de asignación genéricas de solicitud y respuesta utilizando la lista desplegable de la derecha o escribir las suyas propias.

> **Note**
>
> El emparejamiento de una plantilla de asignación de solicitudes con una plantilla de asignación de respuestas se denomina solucionador unitario. Los solucionadores unitarios suelen estar diseñados para realizar operaciones rutinarias; recomendamos usarlos solo para operaciones singulares con un número reducido de orígenes de datos. Para operaciones más complejas, recomendamos utilizar solucionadores de canalización, que pueden ejecutar varias operaciones con diversos orígenes de datos de forma secuencial.
> Para obtener más información acerca de la diferencia entre las plantillas de asignación de solicitudes y respuestas, consulte Solucionadores unitarios.
> Para obtener más información sobre el uso de solucionadores de canalización, consulte Solucionadores de canalización.

4. Para los casos de uso comunes, la consola de AWS AppSync tiene plantillas integradas que puede utilizar para obtener elementos de orígenes de datos (por ejemplo, todas las consultas de elementos, búsquedas individuales, etc.). Por ejemplo en la versión sencilla del esquema que se da en la sección Diseño del esquema en la que `getTodos` no tenía paginación, la plantilla de asignación de solicitudes para enumerar elementos es la siguiente:

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

5. Siempre se necesita una plantilla de asignación de respuestas para complementar la solicitud. La consola ofrece un valor predeterminado con el siguiente valor de acceso directo para las listas:

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

   En este ejemplo, el objeto `context` (con el alias `$ctx`) para las listas de elementos tiene la forma `$context.result.items`. Si su operación de GraphQL devuelve un solo elemento, este sería `$context.result`. AWS AppSync proporciona funciones auxiliares para operaciones comunes, como la función `$util.toJson` indicada anteriormente, con el fin de aplicar un formato correcto a las respuestas. Para ver una lista completa de funciones, consulte Referencia de utilidad de la plantilla de asignación de solucionadores.

6. Seleccione Guardar solucionador.

API

1. Cree un objeto de resolución llamando a la API de `CreateResolver`.
2. Puede modificar los campos del solucionador llamando a la API de `UpdateResolver`.

CLI

1. Cree un solucionador ejecutando el comando `create-resolver`.

   Deberá escribir 6 parámetros para este comando concreto:

   1. El `api-id` de su API.
   2. El `type-name` del tipo que quiere modificar en su esquema. En el ejemplo de la consola, esto era `Query`.
   3. El `field-name` del campo que quiere modificar en su tipo. En el ejemplo de la consola, esto era `getTodos`.
   4. El `data-source-name` del origen de datos que creó en la guía Asociar un origen de datos.
   5. La `request-mapping-template`, que es el cuerpo de la solicitud. En el ejemplo de la consola, esto era:

```
{
```

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

6. La `response-mapping-template`, que es el cuerpo de la respuesta. En el ejemplo de la consola, esto era:

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

Un comando de ejemplo puede tener este aspecto:

```
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)"
```

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

```
{
    "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. Para modificar los campos y/o las plantillas de asignación de un solucionador, ejecute el comando `update-resolver`.

   Con la excepción del parámetro `api-id`, los parámetros utilizados en el comando `create-resolver` se sobrescribirán con los nuevos valores del comando `update-resolver`.

Adición de un solucionador para mutaciones

El siguiente paso es crear un solucionador para su tipo de `Mutation`.

Console

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.

   a. En el panel de API, seleccione su API de GraphQL.

   b. En la barra lateral, seleccione Esquema.

2. En el tipo de Mutación, seleccione Asociar junto al campo `addTodo`.

3. En la página Crear un solucionador, elija el origen de datos que creó en la guía Asociar un origen de datos.

4. En la ventana Configurar plantillas de asignación, modifique la plantilla de solicitud, ya que se trata de una mutación en la que se añade un elemento nuevo a DynamoDB. Use la siguiente plantilla de asignación de solicitud.

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

5. AWS AppSync transforma automáticamente los argumentos definidos en el campo `addTodo` desde su esquema de GraphQL en operaciones de DynamoDB. En el ejemplo anterior se almacenan registros en DynamoDB mediante una clave de `id`, que se transfiere desde el argumento de mutación como `$ctx.args.id`. Todos los demás campos que transfiera se mapearán automáticamente en atributos de DynamoDB con `$util.dynamodb.toMapValuesJson($ctx.args)`.

   Para este solucionador, use la siguiente plantilla de mapeo de respuesta:

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

   AWS AppSync también es compatible con los flujos de trabajo de prueba y depuración para editar los solucionadores. Puede usar un objeto `context` simulado para ver el valor transformado de la plantilla antes de la invocación. De forma opcional, puede ver la ejecución de la solicitud completa en un origen de datos de forma interactiva cuando ejecute una

consulta. Para obtener más información, consulte las secciones sobre prueba y depuración de solucionadores y sobre monitorización y registro.

6. Seleccione Guardar solucionador.

API

También puede hacer esto con las API mediante los comandos de la sección Cree su primer solucionador y los detalles de los parámetros de esta sección.

CLI

También puede hacer esto en la CLI utilizando los comandos de la sección Cree su primer solucionador y los detalles de los parámetros de esta sección.

En este momento, si no usa los solucionadores avanzados, puede empezar a utilizar su API de GraphQL tal como se indica en Uso de la API.

Solucionadores avanzados

Si está siguiendo la sección avanzada y crea un esquema de ejemplo en Diseño del esquema para realizar un análisis paginado, utilice la siguiente plantilla de solicitud para el campo `getTodos`:

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "limit": $util.defaultIfNull(${ctx.args.limit}, 20),
    "nextToken": $util.toJson($util.defaultIfNullOrBlank($ctx.args.nextToken, null))
}
```

En este caso de uso de paginación el mapeo de la respuesta es algo más que un acceso directo, ya que debe contener tanto el cursor (de modo que el cliente sepa en qué página comenzar a continuación) como el conjunto de resultados. La plantilla de mapeo es la siguiente:

```
{
    "todos": $util.toJson($context.result.items),
    "nextToken": $util.toJson($context.result.nextToken)
}
```

Los campos de la plantilla de mapeo de respuesta anterior tienen que coincidir con los campos definidos en su tipo `TodoConnection`.

En el caso de las relaciones en las que tiene una tabla `Comments` y usted está solucionando el campo de comentarios del tipo `Todo` (que devuelve un tipo de `[Comment]`), puede utilizar una plantilla de asignación que ejecute una consulta en la segunda tabla. Para ello, ya debe haber creado un origen de datos para la tabla `Comments` tal como se indica en Asociar un origen de datos.

> **Note**
>
> Usamos una operación de consulta en otra tabla solo con fines ilustrativos. También podría usar otra operación realizada en DynamoDB en su lugar. Además, podría extraer los datos de otro origen de datos, como, por ejemplo AWS Lambda o Amazon OpenSearch Service, ya que el esquema de GraphQL controla la relación.

Console

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   a. En el panel de API, seleccione su API de GraphQL.
   b. En la barra lateral, seleccione Esquema.
2. En el tipo Todo, seleccione Asociar junto al campo `comments`.
3. En la página Crear solucionador, elija el origen de datos de la tabla Comentarios.
   El nombre predeterminado de la tabla Comentarios en las guías de inicio rápido es `AppSyncCommentTable`, pero puede variar en función del nombre que le haya dado.
4. Añada el siguiente fragmento de código a su plantilla de asignación de solicitud:

```
{
    "version": "2017-02-28",
    "operation": "Query",
    "index": "todoid-index",
    "query": {
        "expression": "todoid = :todoid",
        "expressionValues": {
            ":todoid": {
                "S": $util.toJson($context.source.id)
            }
        }
    }
}
```

5. `context.source` hace referencia al objeto principal del campo actual que se está resolviendo. En este ejemplo, `source.id` hace referencia al objeto `Todo` individual, que se usa a continuación para la expresión de consulta.

   Puede utilizar la plantilla de mapeo de respuesta de acceso directo de la siguiente manera:

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

6. Seleccione Guardar solucionador.
7. Por último, en la página Esquema, asocie un solucionador al campo `addComment` y especifique el origen de datos para la tabla `Comments`. La plantilla de mapeo de solicitud en este caso es un elemento `PutItem` sencillo con el elemento `todoid` específico en el que se realiza el comentario como argumento, pero usa la utilidad `$utils.autoId()` para crear una clave de ordenación única para el comentario de la siguiente manera:

   ```
   {
       "version": "2017-02-28",
       "operation": "PutItem",
       "key": {
           "todoid": { "S": $util.toJson($context.arguments.todoid) },
           "commentid": { "S": "$util.autoId()" }
       },
       "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
   }
   ```

   Utilice una plantilla de respuesta de acceso directo de la siguiente manera:

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

API

También puede hacer esto con las API mediante los comandos de la sección Cree su primer solucionador y los detalles de los parámetros de esta sección.

CLI

También puede hacer esto en la CLI utilizando los comandos de la sección Cree su primer solucionador y los detalles de los parámetros de esta sección.

Solucionadores de Lambda directos (VTL)

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

Con los solucionadores de Lambda directos, puede evitar usar plantillas de asignación de VTL al utilizar los orígenes de datos de AWS Lambda. AWS AppSync puede proporcionar una carga predeterminada a la función de Lambda, así como una traducción predeterminada de la respuesta de una función de Lambda a un tipo de GraphQL. Puede optar por proporcionar una plantilla de solicitud, una plantilla de respuesta o ninguna de las dos, y AWS AppSync se ocupará de su gestión en consecuencia.

Para obtener más información sobre la carga de solicitudes y la traducción de respuestas predeterminadas que AWS AppSync proporciona, consulte la referencia sobre solucionadores de Lambda directos. Para obtener más información sobre la configuración de un origen de datos de AWS Lambda y de una política de confianza de IAM, consulte Asociar un origen de datos.

Configurar solucionadores de Lambda directos

En las secciones siguientes, se muestra cómo asociar orígenes de datos de Lambda y cómo añadir solucionadores de Lambda a los campos.

Añadir un origen de datos de Lambda

Para poder activar los solucionadores de Lambda directos, debe añadir un origen de datos de Lambda.

Console

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   a. En el panel de API, seleccione su API de GraphQL.
   b. En la barra lateral, seleccione Origen de datos.
2. Elija Crear origen de datos.
   a. En Nombre de origen de datos, introduzca un nombre para el origen de datos (por ejemplo, **`myFunction`**).

b. En Tipo de origen de datos, elija AWS LambdaNinguno.

c. En Región, elija la región apropiada.

d. En ARN de la función, elija la función de Lambda en la lista desplegable. Puede buscar el nombre de la función o introducir manualmente el ARN de la función que desee utilizar.

e. Cree un nuevo rol de IAM (recomendado) o elija un rol existente que tenga el permiso de IAM `lambda:invokeFunction`. Los roles existentes necesitan una política de confianza, tal y como se explica en la sección Asociar un origen de datos.

   El siguiente es un ejemplo de política de IAM que tiene los permisos necesarios para llevar a cabo operaciones en el recurso:

```
{
     "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. Pulse el botón Crear.

CLI

1. Cree un origen de datos ejecutando el comando `create-data-source`.

   Deberá escribir 4 parámetros para este comando concreto:

   1. El `api-id` de su API.
   2. El `name` de origen de datos. En el ejemplo de la consola, este es el nombre del origen de datos.

3. El `type` de origen de datos. En el ejemplo de la consola, se trata de una función AWS Lambda.
4. El `lambda-config`, que es el ARN de la función el ejemplo de la consola.

> **Note**
>
> Hay otros parámetros, como `Region`, que deben configurarse pero que normalmente se utilizarán de forma predeterminada en los valores de configuración de la CLI.

Un comando de ejemplo puede tener este aspecto:

```
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
```

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

```
{
    "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. Para modificar los atributos de un origen de datos, ejecute el comando `update-data-source`.

   Con la excepción del parámetro `api-id`, los parámetros utilizados en el comando `create-data-source` se sobrescribirán con los nuevos valores del comando `update-data-source`.

Activar los solucionadores de Lambda directos

Tras crear un origen de datos de Lambda y configurar el rol de IAM adecuada para permitir que AWS AppSync invoque la función, puede vincularlo a una función de solucionador o de canalización.

Console

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   a. En el panel de API, seleccione su API de GraphQL.
   b. En la barra lateral, seleccione Esquema.
2. En la ventana Solucionadores, elija un campo u operación y, a continuación, seleccione el botón Asociar.
3. En la página Crear nuevo solucionador, elija la función de Lambda en la lista desplegable.
4. Para aprovechar los solucionadores de Lambda directos, confirme que las plantillas de asignación de solicitud y respuesta estén deshabilitadas en la sección Configurar plantillas de asignación.
5. Pulse el botón Guardar solucionador.

CLI

- Cree un solucionador ejecutando el comando `create-resolver`.

  Deberá escribir 6 parámetros para este comando concreto:

  1. El `api-id` de su API.
  2. El `type-name` del tipo de su esquema.
  3. El `field-name` del campo de su esquema.
  4. El `data-source-name` o el nombre de su función de Lambda.
  5. La `request-mapping-template`, que es el cuerpo de la solicitud. En el ejemplo de la consola, esto estaba deshabilitado:

     ```
     " "
     ```

  6. La `response-mapping-template`, que es el cuerpo de la respuesta. En el ejemplo de la consola, esto también estaba deshabilitado:

```
" "
```

Un comando de ejemplo puede tener este aspecto:

```
aws appsync create-resolver --api-id abcdefghijklmnopqrstuvwxyz --type-name
 Subscription --field-name onCreateTodo --data-source-name LambdaTest --request-
mapping-template " " --response-mapping-template " "
```

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo:

```
{
    "resolver": {
        "resolverArn": "arn:aws:appsync:us-west-2:102847592837:apis/
abcdefghijklmnopqrstuvwxyz/types/Subscription/resolvers/onCreateTodo",
        "typeName": "Subscription",
        "kind": "UNIT",
        "fieldName": "onCreateTodo",
        "dataSourceName": "LambdaTest"
    }
}
```

Al deshabilitar las plantillas de asignación, en AWS AppSync se producen varios comportamientos adicionales:

- Al deshabilitar una plantilla de asignación, indica a AWS AppSync que acepta las traducciones de datos predeterminadas especificadas en la referencia sobre solucionadores de Lambda directos.
- Al deshabilitar la plantilla de asignación de solicitudes, el origen de datos de Lambda recibirá una carga compuesta por todo el objeto Context.
- Al deshabilitar la plantilla de asignación de respuestas, el resultado de la invocación de Lambda se traducirá en función de la versión de la plantilla de asignación de solicitudes o de si la plantilla de asignación de solicitudes también está deshabilitada.

Solucionadores de pruebas y depuración (VTL)

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSync ejecuta solucionadores en un campo de GraphQL con respecto a un origen de datos. Como se describe en Resolver mapping template overview, los solucionadores se comunican con los orígenes de datos mediante un lenguaje de plantillas. Esto permite personalizar el comportamiento y aplicar lógica y condiciones antes y después de comunicarse con el origen de datos. Encontrará una guía de programación introductoria similar a un tutorial para escribir solucionadores en Resolver mapping template programming guide.

A fin de ayudar a los desarrolladores a escribir, probar y depurar estos solucionadores, la consola de AWS AppSync también proporciona herramientas para crear una solicitud y una respuesta de GraphQL con datos simulados, incluso hasta llegar al solucionador de cada campo individual. Además, puede realizar consultas, mutaciones y suscripciones en la consola de AWS AppSync y ver un flujo de registro detallado de toda la solicitud desde Amazon CloudWatch. Esto incluye los resultados del origen de datos.

Prueba con datos simulados

Cuando se invoca a un solucionador de GraphQL, este contiene un objeto `context` que incluye información sobre la solicitud. Por ejemplo, contiene los argumentos de un cliente, información de identidad y datos del campo principal de GraphQL. También contiene los resultados del origen de datos, que puede usar en la plantilla de respuesta. Si desea más información acerca de esta estructura y las utilidades auxiliares disponibles para programar, consulte la Referencia del contexto de las plantillas de mapeo de solucionador.

Al escribir o editar un solucionador, puede pasar un objeto simulado o de contexto de prueba al editor de la consola. Esto le permite ver cómo se evalúan la plantillas de solicitud y de respuesta sin que se utilice en realidad ningún origen de datos. Por ejemplo, puede transferir un argumento `firstname: Shaggy` de prueba y ver cómo se evalúa cuando utiliza `$ctx.args.firstname` en el código de la plantilla. También puede probar la evaluación de cualquier utilidad auxiliar, como `$util.autoId()` o `util.time.nowISO8601()`.

Prueba de solucionadores

En este ejemplo, se utilizará la consola de AWS AppSync para probar los solucionadores.

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   a. En el panel de API, seleccione su API de GraphQL.
   b. En la barra lateral, seleccione Esquema.
2. Si aún no lo ha hecho, en el tipo y junto al campo, seleccione Asociar para añadir su solucionador.

   Para obtener más información acerca de cómo crear un solucionador completo, consulte Configuring resolvers.

   De lo contrario, seleccione el solucionador que ya esté en el campo.
3. En la parte superior de la página Editar el solucionador, seleccione Seleccionar contexto de prueba y, a continuación, Crear nuevo contexto.
4. Seleccione un objeto de contexto de ejemplo o rellene el JSON manualmente en la ventana Contexto de ejecución que aparece a continuación.
5. Introduzca un Nombre de contexto de texto.
6. Seleccione el botón Save (Guardar).
7. En la parte superior de la página Editar solucionador, seleccione Ejecutar prueba.

Veamos un ejemplo más práctico. Imagine que tiene una aplicación que almacena un tipo de GraphQL llamado Dog, que genera automáticamente un identificador para los objetos y los almacena en Amazon DynamoDB. También desea escribir algunos valores tomándolos de los argumentos de una mutación de GraphQL y permitir que solo determinados usuarios vean la respuesta. El esquema podría tener el siguiente aspecto:

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

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

Al añadir un solucionador para la mutación `addDog`, puede rellenar un objeto de contexto como el que aparece en el ejemplo a continuación. Se incluyen argumentos del cliente como `name` y `age`, así como un `username` rellenado en el objeto `identity`:

```
{
    "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"
    }
}
```

Puede probarlo utilizando las siguientes plantillas de mapeo de solicitud y de respuesta:

Plantilla de solicitud

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

Plantilla de respuesta

```
#if ($context.identity.username == "Nadia")
  $util.toJson($ctx.result)
#else
```

```
  $util.unauthorized()
#end
```

La plantilla evaluada tiene los datos del objeto de contexto de prueba y el valor generado por `$util.autoId()`. Además, si cambia el `username` por un valor distinto de `Nadia`, no se devolverán resultados, porque la comprobación de autorización daría error. Para obtener más información acerca del control de acceso preciso, consulte Casos de uso de autorizaciones.

Prueba de plantillas de mapeo con las API de AWS AppSync

Puede usar el comando de API `EvaluateMappingTemplate` para probar las plantillas de mapeo de forma remota con datos simulados. Para empezar a usar el comando, asegúrese de haber añadido el permiso `appsync:evaluateMappingTemplate` a su política. Por ejemplo:

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

Puede utilizar el comando mediante la AWS CLI o los SDK de AWS. Tomemos como ejemplo el esquema Dog y sus plantillas de mapeo de solicitudes/respuestas de la sección anterior. Con la CLI de la estación local, guarde la plantilla de solicitudes en un archivo denominado `request.vtl` y, a continuación, guarde el objeto `context` en un archivo denominado `context.json`. Ejecute el siguiente comando desde el intérprete de comandos:

```
aws appsync evaluate-mapping-template --template file://request.vtl --context file://
context.json
```

El comando devuelve la siguiente respuesta:

```
{
  "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`Contiene los resultados de la prueba de la plantilla proporcionada con el `context` proporcionado. También puede probar sus plantillas con los SDK de AWS. A continuación, se muestra un ejemplo del uso del SDK de AWS para 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))
```

Cuando se utiliza el SDK, puede incorporar fácilmente pruebas de su conjunto de pruebas favorito para validar el comportamiento de la plantilla. Recomendamos crear pruebas con el marco de pruebas Jest, pero cualquier conjunto de pruebas funciona. En el siguiente fragmento de código se muestra una ejecución de validación hipotética. Tenga en cuenta que esperamos que la respuesta de la evaluación sea un JSON válido, por lo que utilizamos `JSON.parse` para recuperar el JSON de la respuesta de cadena:

```
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)
})
```

Esto produce el siguiente resultado:

```
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
```

Depuración de una consulta en tiempo real

No hay nada que pueda reemplazar a una prueba de extremo a extremo y el registro para depurar una aplicación de producción. AWS AppSync permite registrar los errores y todos los detalles de cada solicitud mediante Amazon CloudWatch. Además, puede utilizar la consola de AWS AppSync para probar las consultas, mutaciones y suscripciones de GraphQL y enviar los datos de registro de cada solicitud al editor de consultas para depurarlas en tiempo real. Para las suscripciones, los registros muestran la información del tiempo de conexión.

Para ello, debe haber habilitado previamente los registros de Amazon CloudWatch como se describe en Monitoreo y registro. A continuación, en la consola de AWS AppSync, elija la pestaña Consultas y escriba en una consulta de GraphQL válida. En la sección inferior derecha, haga clic y arrastre la ventana Registros para abrir la vista de registros. Utilice el icono de flecha de reproducción de la parte superior de la página para ejecutar la consulta de GraphQL. Al cabo de unos momentos, los registros completos de la solicitud y la respuesta de la operación se enviarán a esta sección y podrá verlos en la consola.

Solucionadores de canalización (VTL)

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSync ejecuta solucionadores en un campo de GraphQL. En algunos casos, las aplicaciones requieren la ejecución de varias operaciones para resolver un único campo de GraphQL. Con los

solucionadores de canalización, los desarrolladores ahora pueden componer operaciones llamadas Funciones y ejecutarlas de forma secuencial. Los solucionadores de canalización son útiles para las aplicaciones que, por ejemplo, requieren realizar una comprobación de autorización antes de recuperar datos para un campo.

Un solucionador de canalización se compone de una plantilla de mapeo Antes, una plantilla de mapeo Después y una lista de funciones. Cada función tiene una plantilla de mapeo de solicitudes y respuestas que ejecuta con un origen de datos. Puesto que un solucionador de canalización delega la ejecución a una lista de funciones, no está vinculado a ningún origen de datos. Las funciones y los solucionadores de unidad son primitivos que ejecutan operaciones frente a los orígenes de datos. Para obtener más información, consulte Información general sobre las plantillas de mapeo de solucionador.

Crear un solucionador de canalización

En la consola de AWS AppSync, vaya a la página Esquema.

Guarde el siguiente esquema:

```
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
}
```

Vamos a conectar un solucionador de canalización al campo signUp en el tipo Mutation (Mutación). En el tipo Mutación en el lado derecho, elija Asociar junto al campo de mutación `signUp`. En la página de creación de solucionadores, haga clic en Acciones y, a continuación, en Actualizar el tiempo de ejecución. Elija `Pipeline Resolver`, luego VTL y, a continuación, Actualizar. Ahora, la página ahora debería mostrar tres secciones: un área de texto Plantilla de mapeo Antes, una sección Funciones y un área de texto Plantilla de mapeo Después.

Nuestro solucionador de canalización inicia sesión a un usuario. Para ello, primero valida la entrada de dirección de correo electrónico y, a continuación, guarda al usuario en el sistema. Vamos a encapsular la validación de correo electrónico dentro de una función validateEmail y el guardado del usuario dentro de una función saveUser. La función validateEmail se ejecuta en primer lugar y, si el correo electrónico es válido, se ejecuta la función saveUser.

El flujo de ejecución será como se indica a continuación:

1. Plantilla de mapeo de solicitud de solucionador Mutation.signUp
2. Función validateEmail
3. Función saveUser
4. Plantilla de mapeo de respuesta de solucionador Mutation.signUp

Dado que probablemente reutilizaremos la función validateEmail en otros solucionadores de nuestra API, queremos para evitar el acceso a `$ctx.args`, ya que estos cambiarán de un campo GraphQL a otro. En su lugar, podemos utilizar `$ctx.stash` para almacenar el atributo de correo electrónico desde el argumento de campo de entrada `signUp(input: Signup)`.

Plantilla de mapeo ANTES:

```
## store email input field into a generic email key
$util.qr($ctx.stash.put("email", $ctx.args.input.email))
{}
```

La consola ofrece un acceso directo predeterminado de plantilla de mapeo DESPUÉS que vamos a utilizar:

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

Seleccione Crear o Guardar para actualizar el solucionador.

Crear una función

En la página del solucionador de canalización, en la sección Funciones, haga clic en Agregar función y luego en Crear la función nueva. También es posible crear funciones sin tener que recurrir a la página del solucionador; para ello, en la consola de AWS AppSync, vaya a la página Funciones. Elija el botón Crear una función. Vamos a crear una función que comprueba si un mensaje de correo electrónico es válido y proviene de un dominio específico. Si el correo electrónico no es válido, la función genera un error. De lo contrario, reenvía cualquier entrada.

En la nueva página de funciones, seleccione Acciones y, a continuación, Actualizar tiempo de ejecución. Elija VTL y, a continuación, Actualizar. Asegúrese de haber creado un origen de datos del tipo NINGUNO. Elija este origen de datos en la lista de Nombre del origen de datos. Para el nombre de función, introduzca `validateEmail`. En el área de código de la función, sobrescriba todo con este fragmento:

```
#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}) }
}
```

Péguelo en la plantilla de mapeo de respuestas:

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

Revise sus modificaciones y, a continuación, elija Crear. Acabamos de crear nuestra función validateEmail. Repita estos pasos para crear la función SaveUser con las siguientes plantillas de mapeo de solicitudes y respuestas (para simplificar, utilizamos un origen de datos NINGUNO y simulamos que el usuario se ha guardado en el sistema después de ejecutar la función):

Plantilla de mapeo de solicitudes:

```
## $ctx.prev.result contains the signup input values. We could have also
## used $ctx.args.input.
{
    "payload": $util.toJson($ctx.prev.result)
}
```

Plantilla de mapeo de respuestas:

```
## 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)
```

Acabamos de crear la función saveUser.

Adición de una función a un solucionador de canalización

Nuestras funciones deberían haberse agregado automáticamente al solucionador de canalización que acabamos de crear. Si este no es el caso, o si ha creado las funciones a través de la página Funciones, puede volver a hacer clic en Agregar función en la página del solucionador para asociarlas. Agregue las funciones validateEmail y saveUser al solucionador. La función validateEmail se debe colocar antes de la función saveUser. A medida que agrega más funciones, puede utilizar las opciones mover hacia arriba y mover hacia abajo para reorganizar el orden de ejecución de las funciones. Revise sus cambios y, a continuación, elija Guardar.

Ejecutar una consulta

En la consola de AWS AppSync, vaya a la página Consultas. En el explorador, asegúrese de utilizar la mutación. Si no es así, seleccione `Mutation` en la lista desplegable y, a continuación, elija +. Escriba la siguiente consulta:

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

Debería devolver algo parecido a esto:

```
{
  "data": {
    "signUp": {
      "id": "256b6cc2-4694-46f4-a55e-8cb14cc5d7fc",
      "email": "nadia@myvaliddomain.com"
    }
```

```
  }
}
```

Hemos registrado correctamente nuestro usuario y validado el correo electrónico de entrada utilizando un solucionador de canalización. Para seguir un tutorial más completo centrado en los solucionadores de canalización, puede ir al tutorial de solucionadores de canalización

Paso 4: Uso de una API: ejemplo de CDK

> **Tip**
>
> Antes de usar el CDK, le recomendamos que consulte la documentación oficial de este, junto con la referencia del CDK de AWS AppSync.
> También recomendamos asegurarse de que las instalaciones de la CLI de AWS y NPM funcionan en el sistema.

En esta sección, vamos a crear una aplicación de CDK sencilla que pueda añadir y recuperar elementos de una tabla de DynamoDB. Se pretende que sea un ejemplo rápido en el que se utiliza parte del código de las secciones Diseño del esquema, Asociar un origen de datos y Configuración de solucionadores (JavaScript).

Configuración de un proyecto de CDK

> **Warning**
>
> Es posible que estos pasos no sean completamente precisos en función del entorno. Se da por hecho que su sistema tiene instaladas las utilidades necesarias, dispone de una forma de interactuar con los servicios de AWS y tiene las configuraciones adecuadas.

El primer paso es instalar el CDK de AWS. En su CLI, puede introducir el comando siguiente:

```
npm install -g aws-cdk
```

A continuación, debe crear un directorio del proyecto y, luego, navegar hasta él. Un ejemplo de un conjunto de comandos para crear un directorio y navegar a él es:

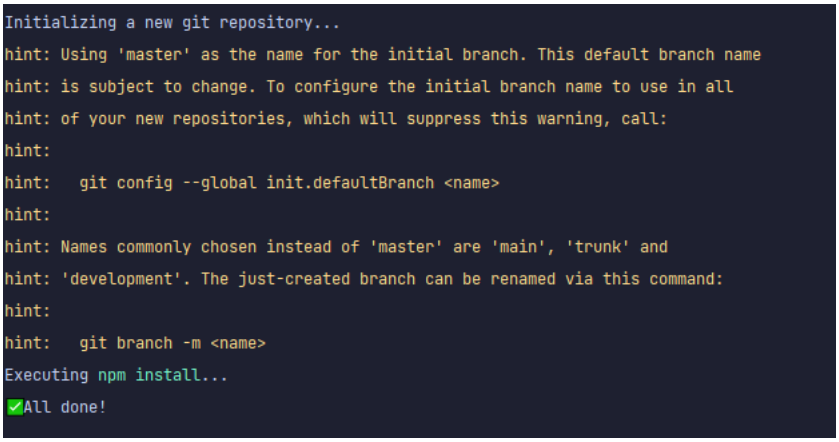
```
mkdir example-cdk-app
```

```
cd example-cdk-app
```

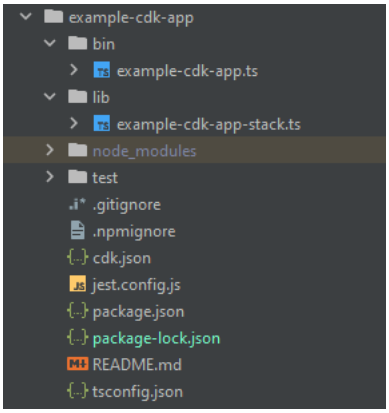
A continuación, debe crear una aplicación. Nuestro servicio utiliza principalmente TypeScript. Ejecute el siguiente comando en el directorio de su proyecto:

```
cdk init app --language typescript
```

Al hacerlo, se instalará una aplicación CDK junto con sus archivos de inicialización:



La estructura de proyecto puede tener un aspecto similar al siguiente:



Se dará cuenta de que tenemos varios directorios importantes:

- `bin`: el archivo bin inicial creará la aplicación. No trataremos este tema en esta guía.
- `lib`: el directorio lib contiene los archivos de pila. Los archivos de pila se podrían comparar a unidades de ejecución individuales. Las construcciones estarán dentro de nuestros archivos de pila. Básicamente, se trata de recursos para un servicio que se pondrá en marcha en AWS CloudFormation cuando se implemente la aplicación. Aquí es donde se realizará la mayor parte de nuestra codificación.

- `node_modules`: este directorio lo creó NPM y contiene todas las dependencias de paquetes que instaló mediante el comando `npm`.

Nuestro archivo de pila inicial puede contener algo como esto:

```
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)
    // });
  }
}
```

Este es el código reutilizable para crear una pila en nuestra aplicación. La mayor parte del código de este ejemplo se incluirá en el ámbito de esta clase.

Para comprobar que el archivo de pila está en la aplicación, en el directorio de la aplicación, ejecute el siguiente comando en el terminal:

```
cdk ls
```

Debería aparecer una lista de sus pilas. Si no es así, es posible que tenga que volver a realizar los pasos o consultar la documentación oficial para obtener ayuda.

Si quiere compilar los cambios de código antes de implementarlos, siempre puede ejecutar el siguiente comando en el terminal:

```
npm run build
```

Y, para ver los cambios antes de la implementación:

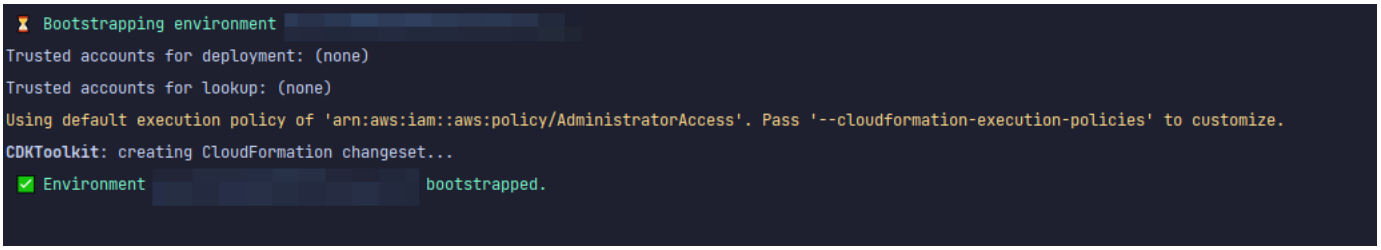
```
cdk diff
```

Antes de añadir nuestro código al archivo de pila, vamos a realizar un arranque. El arranque nos permite aprovisionar recursos para el CDK antes de que se implemente la aplicación. Puede encontrar más información sobre este proceso aquí. El comando para crear un arranque es:

```
cdk bootstrap aws://ACCOUNT-NUMBER/REGION
```

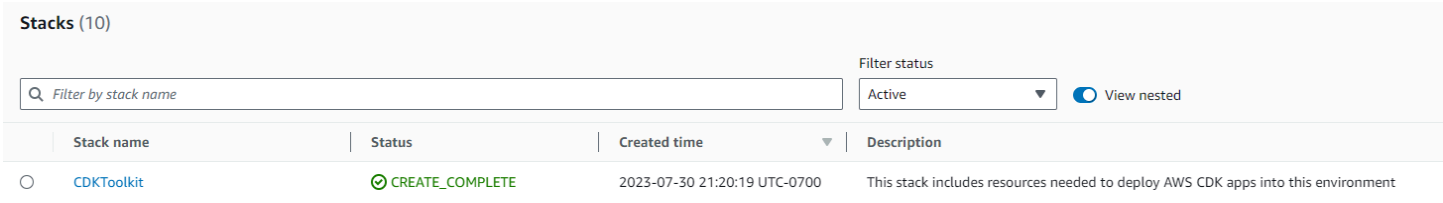
> **Tip**
>
> Este paso requiere varios permisos de IAM en su cuenta. Se denegará el arranque si no los tiene. Si esto ocurre, es posible que tenga que eliminar los recursos incompletos que ha causado el arranque, como el bucket de S3 que genera.

El arranque generará varios recursos. El mensaje final tendrá el siguiente aspecto:

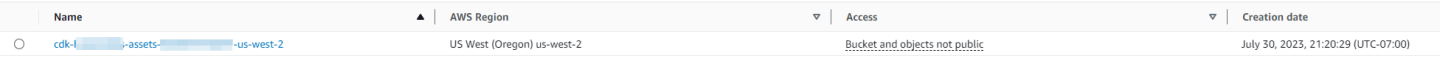


Esto se hace una vez por cuenta y región, por lo que no tendrá que hacerlo con frecuencia. Los recursos principales del arranque son la pila de AWS CloudFormation y el bucket de Amazon S3.

El bucket de Amazon S3 se utiliza para almacenar archivos y roles de IAM que conceden los permisos necesarios para realizar las implementaciones. Los recursos necesarios se definen en una pila de AWS CloudFormation, denominada pila de arranque, que suele tener el nombre `CDKToolkit`. Como cualquier pila de AWS CloudFormation, aparece en la consola de AWS CloudFormation una vez implementada:



Lo mismo puede decirse del bucket:



Para importar los servicios que necesitamos en nuestro archivo de pila, podemos usar el siguiente comando:

```
npm install aws-cdk-lib # V2 command
```

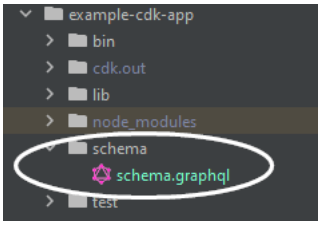
> **Tip**
>
> Si tiene problemas con la V2, puede instalar las bibliotecas individuales mediante los comandos de la V1:
>
> ```
> npm install @aws-cdk/aws-appsync @aws-cdk/aws-dynamodb
> ```
>
> No recomendamos esta opción porque la V1 está en desuso.

Implementación de un proyecto de CDK: esquema

Ahora podemos empezar a implementar nuestro código. En primer lugar, debemos crear nuestro esquema. Simplemente puede crear un archivo `.graphql` en su aplicación:

```
mkdir schema
touch schema.graphql
```

En nuestro ejemplo, incluimos un directorio de nivel superior llamado `schema` que contiene nuestro `schema.graphql`:



Dentro de nuestro esquema, vamos a incluir un ejemplo sencillo:

```
input CreatePostInput {
    title: String
    content: String
}
```

```
type Post {
    id: ID!
    title: String
    content: String
}

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

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

De vuelta a nuestro archivo de pila, debemos asegurarnos de que estén definidas las siguientes directivas de importación:

```
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';
```

Dentro de la clase, añadiremos código para crear nuestra API de GraphQL y conectarla a nuestro archivo `schema.graphql`:

```
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'),
    });
  }
}
```

También añadiremos código para imprimir la URL de GraphQL, la clave de API y la región:

```
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
    });
  }
}
```

En este punto, volveremos a implementar nuestra aplicación:

```
cdk deploy
```

El resultado es el siguiente:

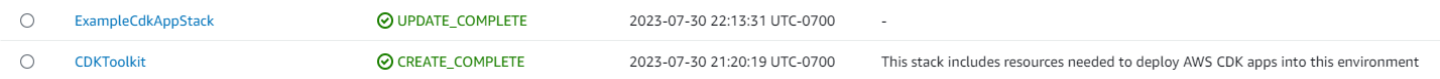
Parece que nuestro ejemplo se ha ejecutado correctamente, pero revisemos la consola de AWS AppSync para confirmarlo:

Parece que nuestra API se ha creado. Ahora, comprobaremos el esquema asociado a la API:

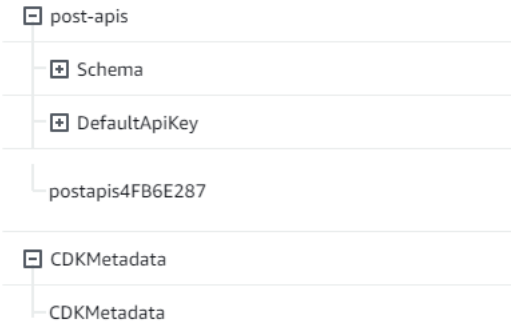
Schema

```
input CreatePostInput {
  title: String
  date: AWSDateTime
}

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

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

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

Esto parece coincidir con nuestro código de esquema, por lo que se ha realizado correctamente. Otra forma de confirmarlo desde el punto de vista de los metadatos es observar la pila de AWS CloudFormation:



Cuando implementamos nuestra aplicación de CDK, pasa por AWS CloudFormation para activar recursos, como el arranque. Cada pila de nuestra aplicación se asigna con una correspondencia de 1:1 a una pila de AWS CloudFormation. Si vuelve al código de la pila, verá que el nombre de la pila se ha tomado del nombre de la clase`ExampleCdkAppStack`. Puede ver los recursos que creó, que también se ajustan a nuestras convenciones de nomenclatura, en nuestra construcción de la API de GraphQL:



Implementación de un proyecto de CDK: origen de datos

A continuación, debemos añadir nuestro origen de datos. En nuestro ejemplo, se utilizará una tabla de DynamoDB. Dentro de la clase de pila, añadiremos código para crear una tabla nueva:

```
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 || ''
    });
```

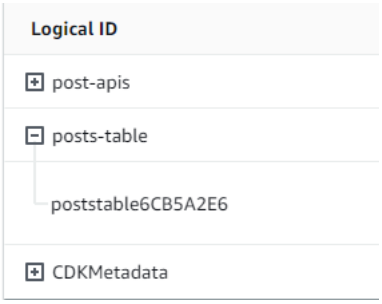
```
    // Prints out the stack region to the terminal
    new cdk.CfnOutput(this, "Stack Region", {
      value: this.region
    });
  }
}
```

Llegados a este punto, volvamos a implementarla:

```
cdk deploy
```

Deberíamos comprobar la consola de DynamoDB para ver nuestra nueva tabla:

El nombre de nuestra pila es correcto y el nombre de la tabla coincide con nuestro código. Si volvemos a comprobar nuestra pila de AWS CloudFormation, ahora veremos la nueva tabla:



Implementación de un proyecto de CDK: solucionador

En este ejemplo, se utilizarán dos solucionadores: uno para consultar la tabla y otro para añadirle elementos. Como utilizamos solucionadores de canalizaciones, tendremos que declarar dos solucionadores de canalizaciones con una función en cada uno. En la consulta, añadiremos el siguiente código:

```
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
```

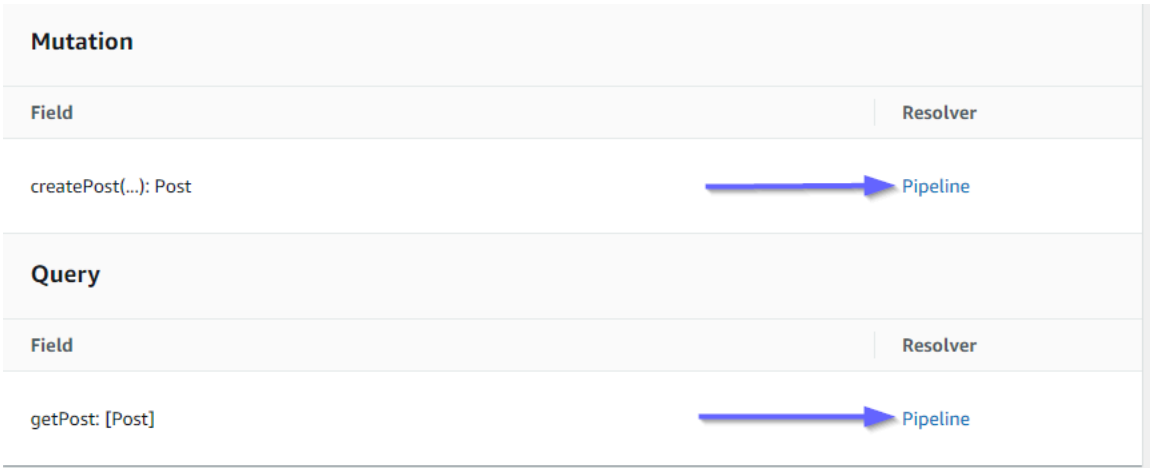
```
    new cdk.CfnOutput(this, "GraphQLAPIKey", {
      value: api.apiKey || ''
    });

    // Prints out the stack region to the terminal
    new cdk.CfnOutput(this, "Stack Region", {
      value: this.region
    });
  }
}
```

En este fragmento, añadimos un solucionador de canalizaciones llamado `pipeline-resolver-create-posts` con una función llamada `func-add-post` asociada. Este es el código que añadirá `Posts` a la tabla. El otro solucionador de canalizaciones se ha denominado `pipeline-resolver-get-posts`, con una función llamada `func-get-post` que recupera `Posts` añadidos a la tabla.

Implementaremos esto para añadirlo al servicio de AWS AppSync:

```
cdk deploy
```

Comprobemos la consola de AWS AppSync para ver si estaban asociados a nuestra API de GraphQL:



Parece correcto. En el código, estos dos solucionadores estaban conectados a la API de GraphQL que creamos (indicada por el valor de accesorios `api` presente tanto en los solucionadores como en las funciones). En la API de GraphQL, los campos a los que asociamos nuestros solucionadores también estaban especificados en los accesorios (definidos por `typename` y los accesorios `fieldname` de cada solucionador).

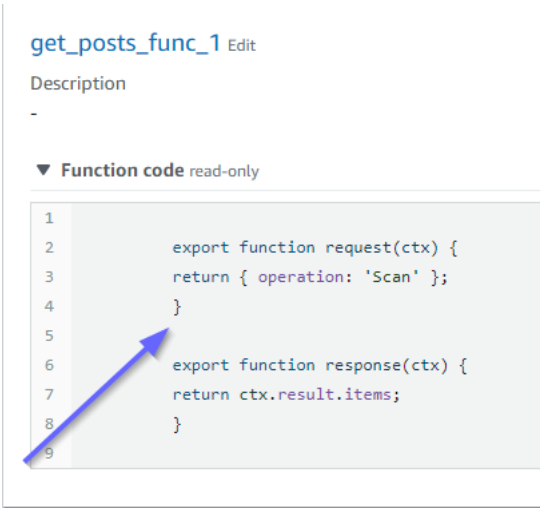
Veamos si el contenido de los solucionadores es correcto empezando por `pipeline-resolver-get-posts`:

Los controladores de antes y después coinciden con nuestro valor de accesorios `code`. También podemos ver una función llamada `add_posts_func_1`, que coincide con el nombre de la función que asociamos en el solucionador.

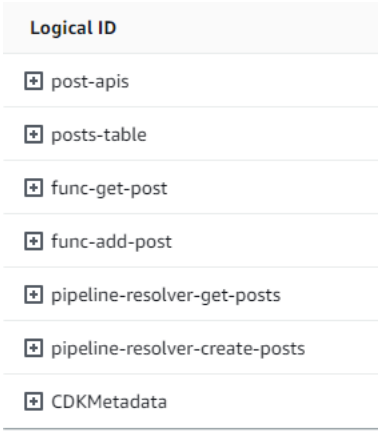
Veamos el contenido del código de esa función:

Esto coincide con los accesorios `code` de la función `add_posts_func_1`. Nuestra consulta se ha cargado correctamente, así que revisemos la consulta:

También coinciden con el código. Si nos fijamos en `get_posts_func_1`:



Todo parece estar en su sitio. Para confirmarlo desde el punto de vista de los metadatos, podemos volver a revisar nuestra pila en AWS CloudFormation:



Ahora, necesitamos probar este código realizando algunas solicitudes.

Implementación de un proyecto de CDK: solicitudes

Para probar nuestra aplicación en la consola de AWS AppSync, creamos una consulta y una mutación:

```
query MyQuery {
  getPost {
    id
    date
    title
  }
}

mutation MyMutation {
  createPost(input: {date: "1970-01-01T12:30:00.000Z", title: "first post"}) {
    date
    id
    title
  }
}
```

`MyMutation` contiene una operación `createPost` con los argumentos `1970-01-01T12:30:00.000Z` y `first post`. Devuelve los `date` y `title` que hemos introducido, así como el valor `id` generado automáticamente. Al ejecutar la mutación, se obtiene el resultado:

```
{
  "data": {
    "createPost": {
      "date": "1970-01-01T12:30:00.000Z",
      "id": "4dc1c2dd-0aa3-4055-9eca-7c140062ada2",
      "title": "first post"
    }
  }
}
```

Si comprobamos rápidamente la tabla de DynamoDB, podemos ver nuestra entrada en la tabla cuando la escaneamos:

id *(String)*	date	title
9f62c4dd-49d5-48d5-b835-143284c72fe0	1970-01-01T12:30:00.000Z	first post

De vuelta en la consola de AWS AppSync, si ejecutamos la consulta para recuperar este `Post`, obtenemos el siguiente resultado:

```
{
  "data": {
    "getPost": [
      {
```

```
        "id": "9f62c4dd-49d5-48d5-b835-143284c72fe0",
        "date": "1970-01-01T12:30:00.000Z",
        "title": "first post"
      }
    ]
  }
}
```

Datos en tiempo real

AWS AppSync le permite utilizar las suscripciones para implementar actualizaciones de aplicaciones en tiempo real, notificaciones push, etc. Cuando los clientes invocan las operaciones de suscripción de GraphQL, AWS AppSync establece y mantiene automáticamente una conexión WebSocket segura. De este modo, las aplicaciones pueden distribuir los datos en tiempo real desde un origen de datos a los suscriptores, mientras que AWS AppSync gestiona continuamente los requisitos de conexión y escalado de la aplicación. En las siguientes secciones se explica el funcionamiento de las suscripciones en AWS AppSync.

Directivas de suscripción del esquema de GraphQL

Las suscripciones en AWS AppSync se invocan como respuesta a una mutación. Esto significa que puede hacer que cualquier origen de datos de AWS AppSync envíe datos en tiempo real especificando una directiva de esquema de GraphQL para una mutación.

Las bibliotecas de cliente de AWS Amplify gestionan automáticamente la administración de las conexiones de suscripción. Las bibliotecas utilizan WebSockets puros como protocolo de red entre el cliente y el servicio.

> **Note**
>
> Para controlar la autorización en el momento de conectarse a una suscripción, puede usar AWS Identity and Access Management (IAM), AWS Lambda, los grupos de identidades de Amazon Cognito o los grupos de usuarios de Amazon Cognito para la autorización de nivel de campo. Para controles de acceso a las suscripciones más precisos, puede asociar solucionadores a los campos de la suscripción y aplicar una lógica basada en la identidad del intermediario y los orígenes de datos de AWS AppSync. Para obtener más información, consulte Autorización y autenticación.

Las suscripciones se activan a partir de mutaciones y el conjunto de mutaciones seleccionado se envía a los suscriptores.

En el siguiente ejemplo se muestra cómo usar las suscripciones de GraphQL. No especifica un origen de datos porque este puede ser Lambda, Amazon DynamoDB o Amazon OpenSearch Service.

Para comenzar a utilizar las suscripciones, debe agregar un punto de entrada de la suscripción a su esquema de este modo:

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

Imagine que dispone de un sitio donde se publican blogs y que desea suscribirse a nuevos blogs y a los cambios en los blogs existentes. Para ello, puede añadir la siguiente definición `Subscription` a su esquema:

```
type Subscription {
    addedPost: Post
    updatedPost: Post
    deletedPost: Post
}
```

Imagine también que tiene las siguientes mutaciones:

```
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!
}
```

Puede convertir estos campos a tiempo real añadiendo una directiva `@aws_subscribe(mutations: ["mutation_field_1", "mutation_field_2"])` para cada una de las suscripciones de las que desea recibir notificaciones, tal y como se indica a continuación:

```
type Subscription {
```

```
    addedPost: Post
    @aws_subscribe(mutations: ["addPost"])
    updatedPost: Post
    @aws_subscribe(mutations: ["updatePost"])
    deletedPost: Post
    @aws_subscribe(mutations: ["deletePost"])
}
```

Dado que la mutación `@aws_subscribe(mutations: ["",..,""])` tiene una matriz de entradas de mutaciones, puede especificar varias mutaciones que inician una suscripción. Si se suscribe desde un cliente, su consulta de GraphQL podría tener el siguiente aspecto:

```
subscription NewPostSub {
    addedPost {
        __typename
        version
        title
        content
        author
        url
    }
}
```

Esta consulta de suscripción es necesaria para las conexiones de cliente y para las herramientas.

Si utiliza el cliente WebSockets puro, el filtrado del conjunto de selecciones se realiza por cliente, ya que cada cliente puede definir su propio conjunto de este tipo. En este caso, el conjunto de selecciones de la suscripción debe ser un subconjunto del conjunto de selecciones de la mutación. Por ejemplo, una suscripción `addedPost{author title}` vinculada a la mutación `addPost(...){id author title url version}` solo recibe el autor y el título de la publicación. No recibe el resto de campos. Sin embargo, si la mutación no incluye el autor en su conjunto de selecciones, el suscriptor obtiene un valor `null` para el campo de autor (o bien un error, en caso de que el campo de autor se defina como obligatorio o no nulo en el esquema).

El conjunto de selección de suscripciones es esencial cuando se utiliza WebSockets puro. Si un campo no está definido explícitamente en la suscripción, AWS AppSync no lo devuelve.

En el ejemplo anterior, las suscripciones no tenía argumentos. Imaginemos que su esquema es similar al siguiente:

```
type Subscription {
```

```
    updatedPost(id:ID! author:String): Post
    @aws_subscribe(mutations: ["updatePost"])
}
```

En este caso, el cliente define una suscripción de este modo:

```
subscription UpdatedPostSub {
    updatedPost(id:"XYZ", author:"ABC") {
        title
        content
    }
}
```

El tipo que devuelve el campo `subscription` del esquema debe coincidir con el tipo devuelto para el campo de mutación correspondiente. En el ejemplo anterior, tanto `addPost` como `addedPost` se devolvían con el tipo `Post`.

Para configurar las suscripciones en el cliente, consulte Creación de una aplicación cliente.

Uso de argumentos de suscripción

Un aspecto importante del uso de las suscripciones de GraphQL es comprender cuándo y cómo se deben usar los argumentos. Puede hacer cambios sutiles para modificar la forma y el momento de informar a los clientes acerca de las mutaciones que se han producido. Para ello, consulte el esquema de ejemplo del capítulo de inicio rápido, en el que se crean “Todos”. Para este esquema de ejemplo se definen las mutaciones siguientes:

```
type Mutation {
    createTodo(input: CreateTodoInput!): Todo
    updateTodo(input: UpdateTodoInput!): Todo
    deleteTodo(input: DeleteTodoInput!): Todo
}
```

En el ejemplo de muestra, los clientes pueden suscribirse a las actualizaciones de cualquier `Todo` utilizando la opción `onUpdateTodo` `subscription` sin argumentos:

```
subscription OnUpdateTodo {
  onUpdateTodo {
    description
```

```
    id
    name
    when
  }
}
```

Puede filtrar su `subscription` utilizando sus argumentos. Por ejemplo, para activar solo una `subscription` cuando se actualiza un `todo` con un `ID` específico, especifique el valor `ID`:

```
subscription OnUpdateTodo {
  onUpdateTodo(id: "a-todo-id") {
    description
    id
    name
    when
  }
}
```

También se pueden transferir varios argumentos. Por ejemplo, la siguiente `subscription` muestra cómo recibir notificaciones de cualquier actualización de `Todo` en un lugar y hora específicos:

```
subscription todosAtHome {
  onUpdateTodo(when: "tomorrow", where: "at home") {
    description
    id
    name
    when
    where
  }
}
```

Tenga en cuenta que todos los argumentos son opcionales. Si no especifica ningún argumento en su `subscription`, se suscribirá a todas las actualizaciones de `Todo` que se produzcan en su aplicación. Sin embargo, puede actualizar la definición de campo de su `subscription` para requerir el argumento `ID`. De este modo, forzaría la respuesta de un `todo` específico en lugar de la de todos los `todo`:

```
onUpdateTodo(
  id: ID!,
  name: String,
```

```
  when: String,
  where: String,
  description: String
): Todo
```

El valor nulo del argumento tiene un significado

Al hacer una consulta de suscripción en AWS AppSync, un valor de argumento `null` filtrará los resultados de una forma diferente sin omitir el argumento por completo.

Volvamos al ejemplo de la API de todos, donde podríamos crear todos. Consulte el ejemplo de esquema del capítulo de inicio rápido.

Vamos a modificar nuestro esquema para incluir un nuevo campo `owner`, del tipo `Todo`, que describa quién es el propietario. El campo `owner` no es obligatorio y solo se puede configurar en `UpdateTodoInput`. Consulte la siguiente versión simplificada del esquema:

```
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"])
}
```

La siguiente suscripción devuelve todas las actualizaciones de `Todo`:

```
subscription MySubscription {
  onUpdateTodo {
    description
    id
    name
    when
    where
  }
}
```

Si modifica la suscripción anterior para agregar el argumento del campo `owner: null`, ahora está planteando una pregunta diferente. Ahora, esta suscripción registra al cliente para recibir notificaciones de todas las actualizaciones de `Todo` que no hayan indicado un propietario.

```
subscription MySubscription {
  onUpdateTodo(owner: null) {
    description
    id
    name
    when
    where
  }
}
```

> **Note**
>
> A partir del 1 de enero de 2022, MQTT en WebSockets ya no estará disponible como protocolo para las suscripciones de GraphQL en las API de AWS AppSync. WebSockets puro es el único protocolo compatible con AWS AppSync.

Los clientes basados que usan el SDK de AWS AppSync o las bibliotecas de Amplify, publicadas después de noviembre de 2019, utilizan automáticamente WebSockets puro de forma predeterminada. Al actualizar los clientes a la última versión, les permite utilizar el motor de WebSockets puro de AWS AppSync.
WebSockets puro incorpora un tamaño de carga mayor (240 KB), una variedad más amplia de opciones de cliente y métricas de CloudWatch mejoradas. Para obtener más información sobre el uso de clientes WebSocket puros, consulte Creación de un cliente WebSocket en tiempo real.

Creación de API pub/sub genéricas con tecnología de WebSockets sin servidor

Algunas aplicaciones solo requieren API de WebSocket sencillas en las que los clientes escuchan un canal o un tema específicos. Los datos JSON genéricos sin un formato específico ni requisitos estrictamente definidos se pueden distribuir a los clientes que escuchan algunos de estos canales en un patrón de publicación suscripción (pub/sub) puro y simple.

Utilice AWS AppSync para implementar API de WebSocket pub/sub sencillas con poco o ningún conocimiento de GraphQL en cuestión de minutos mediante la generación automática de código de GraphQL tanto en el backend de la API como en el lado del cliente.

Creación y configuración de API pub/sub

Para empezar, siga estos pasos:

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   - En el Dashboard (Panel), elija Create API (Crear API).
2. En la siguiente pantalla, seleccione Crear una API en tiempo real y, a continuación, seleccione Siguiente.
3. Escriba un nombre fácil de recordar para su API pub/sub.
4. Puede habilitar las características de API privada, pero le recomendamos que mantenga esta opción desactivada por ahora. Elija Next (Siguiente).
5. Puede optar por generar automáticamente una API pub/sub que funcione mediante WebSockets. También recomendamos mantener esta característica desactivada por el momento. Elija Next (Siguiente).

6. Seleccione Crear API y, a continuación, espere un par de minutos. Se creará una nueva API pub/sub de AWS AppSync preconfigurada en su cuenta de AWS.

La API utiliza los solucionadores locales integrados de AWS AppSync (para obtener más información sobre el uso de los solucionadores locales, consulte el Tutorial: Solucionadores locales en la Guía para desarrolladores de AWS AppSync) para gestionar varios canales pub-sub temporales y conexiones de WebSocket, que entregan y filtran automáticamente los datos a los clientes suscritos basándose únicamente en el nombre del canal. Las llamadas a la API se autorizan con una clave de API.

Una vez implementada la API, verá un par de pasos adicionales para generar el código de cliente e integrarlo con su aplicación cliente. Para ver un ejemplo de cómo integrar rápidamente un cliente, en esta guía se utilizará una sencilla aplicación web de React.

1. Para empezar cree una aplicación de React reutilizable usando NPM en su máquina local:

```
$ npx create-react-app mypubsub-app
$ cd mypubsub-app
```

> **Note**
>
> En este ejemplo, se utilizan las bibliotecas Amplify para conectar los clientes a la API de backend. Sin embargo, no es necesario crear un proyecto de CLI de Amplify a nivel local. Si bien React es el cliente preferido para este ejemplo, las bibliotecas de Amplify también son compatibles con los clientes de iOS, Android y Flutter, y proporcionan las mismas capacidades en estos tiempos de ejecución diferentes. Los clientes de Amplify compatibles proporcionan abstracciones sencillas para interactuar con los backends de la API de GraphQL de AWS AppSync con apenas unas líneas de código, que incluyen funcionalidad WebSocket integrada totalmente compatible con el protocolo WebSocket en tiempo real de AWS AppSync:
>
> ```
> $ npm install @aws-amplify/api
> ```

2. En la consola de AWS AppSync, selecciona JavaScript y, a continuación, Descargar para descargar un único archivo con los detalles de configuración de la API y el código de operaciones de GraphQL generado.
3. Copie el archivo descargado en la carpeta `/src` de su proyecto de React.

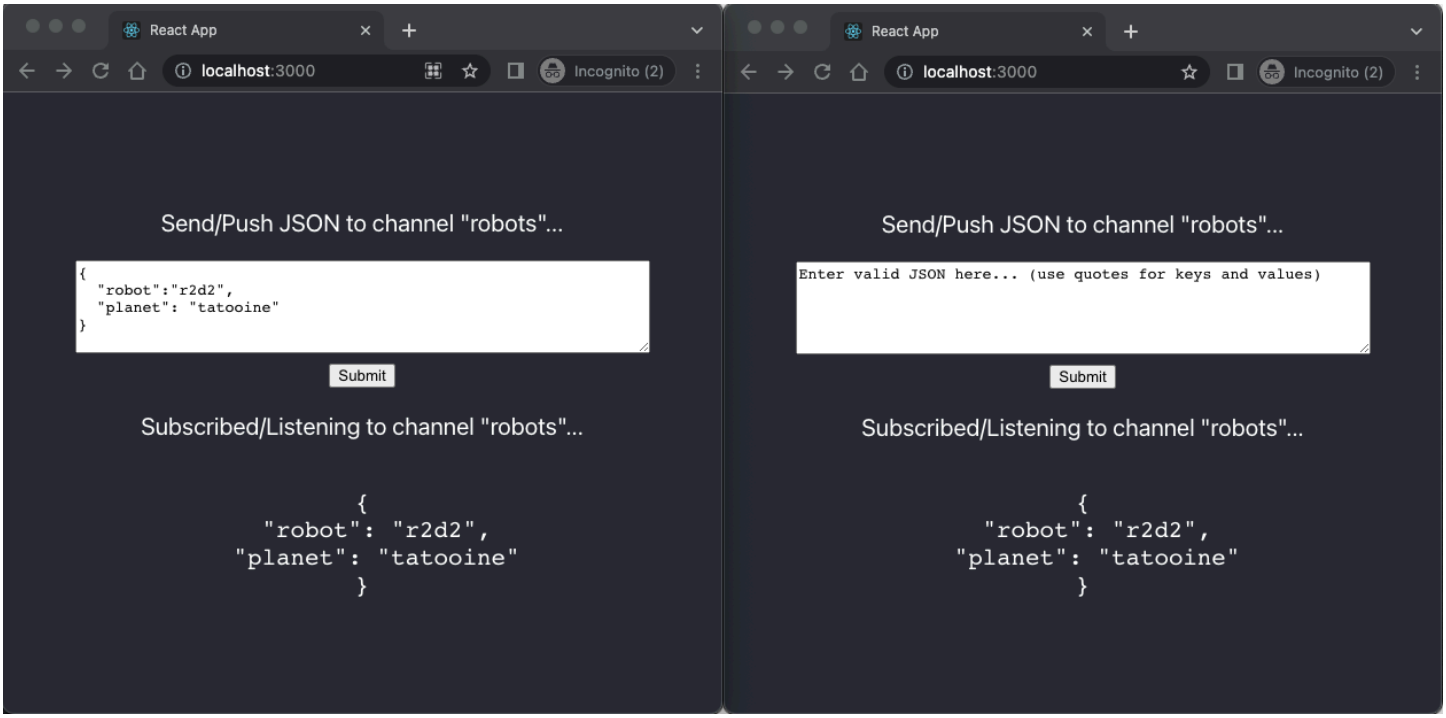
4. A continuación, reemplace el contenido del archivo `src/App.js` reutilizable existente por el código de cliente de muestra disponible en la consola.
5. Utilice el comando siguiente para compilar y ejecutar la aplicación a nivel local:

```
$ npm start
```

6. Para probar el envío y la recepción de datos en tiempo real, abra dos ventanas del navegador y acceda a *`localhost:3000`*. La aplicación de muestra está configurada para enviar datos JSON genéricos a un canal con codificación rígida denominado *`robots`*.
7. En una de las ventanas del navegador, introduzca el siguiente blob JSON en el cuadro de texto y, a continuación, haz clic en Enviar:

```
{
  "robot":"r2d2",
  "planet": "tatooine"
}
```

Ambas instancias del navegador están suscritas al canal de *`robots`* y reciben los datos publicados en tiempo real, que se muestran en la parte inferior de la aplicación web:



Todo el código de la API de GraphQL necesario, incluido el esquema, los solucionadores y las operaciones, se generan automáticamente para habilitar un caso de uso genérico de pub/sub. En el

backend, los datos se publican en el punto de conexión de AWS AppSync en tiempo real con una mutación de GraphQL como la siguiente:

```
mutation PublishData {
    publish(data: "{\"msg\": \"hello world!\"}", name: "channel") {
        data
        name
    }
}
```

Los suscriptores acceden a los datos publicados que se envían al canal temporal específico con una suscripción de GraphQL relacionada:

```
subscription SubscribeToData {
    subscribe(name:"channel") {
        name
        data
    }
}
```

Implementación de las API pub/sub en las aplicaciones existentes

Si solo necesita implementar una característica en tiempo real en una aplicación existente, esta configuración genérica de API pub/sub se puede integrar fácilmente en cualquier aplicación o tecnología de API. Si bien utilizar un único punto de conexión de API para acceder, modificar y combinar de forma segura los datos de uno o más orígenes de datos en una sola llamada a la red con GraphQL, no es necesario convertir o rediseñar una aplicación existente basada en REST desde cero para aprovechar las capacidades en tiempo real de AWS AppSync. Por ejemplo, podría tener una carga de trabajo CRUD existente en un punto de conexión de API independiente en el que los clientes envíen y reciban mensajes o eventos desde la aplicación existente a la API pub/sub genérica solo para fines de pub/sub y en tiempo real.

Filtrado de suscripciones mejorado

En AWS AppSync, puede definir y habilitar la lógica empresarial para el filtrado de datos en el backend directamente en los solucionadores de suscripciones de la API de GraphQL mediante filtros que admitan operadores lógicos adicionales. A diferencia de los argumentos de suscripción que se definen en la consulta de suscripción en el cliente, en este caso sí puede configurar estos filtros. Para obtener más información acerca de los argumentos de suscripción, consulte Uso de

argumentos de suscripción. Para ver una lista de operadores, consulte Referencia de utilidad de la plantilla de mapeo de solucionador.

Para los fines de este documento, dividimos el filtrado de datos en tiempo real en las siguientes categorías:

- Filtrado básico: filtrado basado en los argumentos definidos por el cliente en la consulta de suscripción.
- Filtrado mejorado: filtrado basado en una lógica definida de forma centralizada en el backend del servicio de AWS AppSync.

En las siguientes secciones se explica cómo configurar los filtros de suscripción mejorados y se muestran sus aplicaciones prácticas.

Definición de suscripciones en su esquema de GraphQL

Para usar filtros de suscripción mejorados, defina la suscripción en el esquema de GraphQL y, a continuación, defina el filtro mejorado mediante una extensión de filtrado. Para ilustrar cómo funciona el filtrado de suscripciones mejorado en AWS AppSync, utilice el siguiente esquema de GraphQL, que define la API de un sistema de gestión de tickets, como ejemplo:

```
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
```

Supongamos que crea un origen de datos NONE para su API y, a continuación, adjunta un solucionador a la mutación `createTicket` utilizando este origen de datos. Los controladores pueden tener un aspecto similar al siguiente:

```
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**
>
> Los filtros mejorados están habilitados en el controlador del solucionador de GraphQL en una suscripción determinada. Para obtener más información, consulte Referencia al solucionador.

Para implementar el comportamiento del filtro mejorado, debe usar la función `extensions.setSubscriptionFilter()` para definir una expresión de filtro evaluada en función de los datos publicados de una mutación de GraphQL que pueda interesar a los clientes suscritos. Para obtener más información acerca de las extensiones de filtrado, consulte Extensiones.

En la siguiente sección, se explica cómo utilizar extensiones de filtrado para implementar filtros mejorados.

Creación de filtros de suscripciones mejorados mediante extensiones de filtrado

Los filtros mejorados están escritos en JSON en el controlador de respuestas de los solucionadores de la suscripción. Los filtros se pueden agrupar en una lista llamada `filterGroup`. Los filtros se definen mediante al menos una regla, cada una con campos, operadores y valores. Definamos un nuevo solucionador para `onSpecialTicketCreated` que configure un filtro mejorado. Puede configurar varias reglas en un filtro, que se evalúan mediante la lógica AND. Por su parte, varios filtros de un grupo de filtros se evalúan mediante la lógica 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;
}
```

Según los filtros definidos en el ejemplo anterior, los tickets importantes se distribuyen automáticamente a los clientes de la API suscritos si se crea un ticket con:

- nivel de `priority` `high` o `medium`

  Y

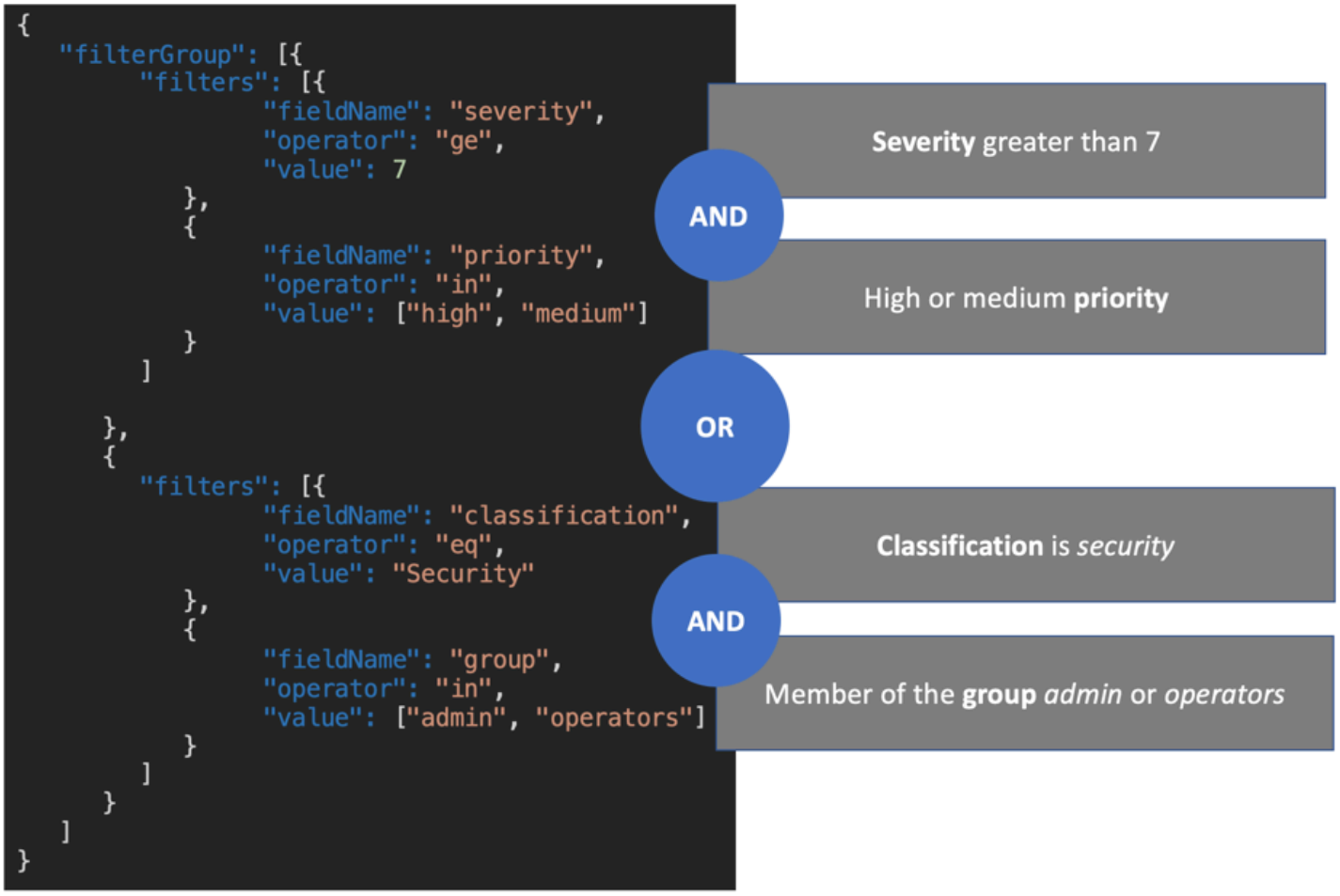
- nivel de `severity` mayor o igual que 7 (`ge`)

O BIEN

- ticket de `classification` con la definición de `Security`

  Y

- asignación de `group` establecida en `admin` o `operators`



Los filtros definidos en el solucionador de suscripciones (filtrado mejorado) tienen prioridad sobre el filtrado basado únicamente en los argumentos de suscripción (filtrado básico). Para obtener más información acerca de los argumentos de suscripción, consulte Uso de argumentos de suscripción.

Si un argumento está definido en el esquema de GraphQL de la suscripción y es obligatorio, el filtrado basado en el argumento en cuestión solo se lleva a cabo si el argumento está definido como una regla en el método `extensions.setSubscriptionFilter()` del solucionador. Sin embargo, si no hay métodos de filtrado de `extensions` en el solucionador de suscripciones, los argumentos definidos en el cliente se utilizan únicamente para el filtrado básico. No puede utilizar el filtrado básico y el filtrado mejorado a la vez.

Puede usar la variable `context` de la lógica de extensión del filtro de la suscripción para acceder a la información contextual sobre la solicitud. Por ejemplo, si utiliza autorizadores personalizados de grupos de usuarios de Amazon Cognito, OIDC o Lambda para la autorización, puede recuperar información sobre sus usuarios en `context.identity` cuando se establezca la suscripción. Puede usar esa información para establecer filtros basados en la identidad de sus usuarios.

Supongamos ahora que desea implementar el comportamiento de filtro mejorado para `onGroupTicketCreated`. La suscripción `onGroupTicketCreated` requiere un nombre `group` obligatorio como argumento. Cuando se crean, a los tickets se les asigna automáticamente un estado `pending`. Puedes configurar un filtro de suscripción para recibir solo los tickets recién creados que pertenezcan al grupo proporcionado:

```
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;
}
```

Cuando los datos se publican mediante una mutación, como en el siguiente ejemplo:

```
mutation CreateTicket {
  createTicket(input: {priority: medium, severity: 2, group: "aws"}) {
    id
    priority
    severity
    status
    group
    createdAt
  }
}
```

Los clientes suscritos escuchan los datos que se distribuirán automáticamente a través de WebSockets en cuanto se cree un ticket con la mutación `createTicket`:

```
subscription OnGroup {
  onGroupTicketCreated(group: "aws") {
    category
    status
    severity
```

```
    priority
    id
    group
    createdAt
    content
  }
}
```

Los clientes se pueden suscribir sin argumentos porque el servicio AWS AppSync tiene implementada una lógica de filtrado con filtrado mejorado, lo que simplifica el código del cliente. Los clientes reciben datos solo si se cumplen los criterios de filtro definidos.

Definición de filtros mejorados para campos de esquema anidados

Puede utilizar el filtrado de suscripciones mejorado para filtrar los campos de esquema anidados. Supongamos que modificamos el esquema de la sección anterior para incluir los tipos de ubicación y dirección:

```
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
```

Con este esquema, puede usar un separador `.` para representar el anidamiento. En el siguiente ejemplo, se agrega una regla de filtrado para un campo de esquema anidado en `location.address.country`. La suscripción se activará si la dirección del ticket se define como USA:

```
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;
}
```

En el ejemplo anterior, `location` representa el nivel de anidamiento uno, `address` representa el nivel de anidamiento dos y `country` representa el nivel de anidamiento tres, todos los cuales están separados por el separador `..`

Puede probar esta suscripción mediante la mutación `createTicket`:

```
mutation CreateTicketInUSA {
  createTicket(input: {location: "{\"address\":{\"country\":\"USA\"}}"}) {
    category
    content
    createdAt
    group
    id
    location {
      address {
        country
      }
    }
    priority
    severity
    status
  }
}
```

Definición de filtros mejorados desde el cliente

Puede usar el filtrado básico en GraphQL con argumentos de suscripciones. El cliente que realiza la llamada en la consulta de suscripción define los valores de los argumentos. Cuando están habilitados los filtros mejorados en un solucionador de suscripciones de AWS AppSync con el filtrado de `extensions`, los filtros secundarios definidos en la resolución son preferentes y prioritarios.

Configure filtros mejorados dinámicos definidos por el cliente en la suscripción usando un argumento `filter`. Al configurar estos filtros, debe actualizar el esquema de GraphQL para que refleje el nuevo argumento:

```
...
type Subscription {
```

```
    onSpecialTicketCreated(filter: String): Ticket
        @aws_subscribe(mutations: ["createTicket"])
}
...
```

A continuación, el cliente puede enviar una consulta de suscripción como en el siguiente ejemplo:

```
subscription onSpecialTicketCreated($filter: String) {
     onSpecialTicketCreated(filter: $filter) {
        id
        group
        description
        priority
        severity
     }
 }
```

Puede configurar la variable de consulta de la siguiente manera:

```
{"filter" : "{\"severity\":{\"le\":2}}"}
```

La utilidad de solucionador `util.transform.toSubscriptionFilter()` se puede implementar en la plantilla de mapeo de respuestas de suscripción para aplicar el filtro definido en el argumento de suscripción a cada cliente:

```
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;
}
```

Con esta estrategia, los clientes pueden definir sus propios filtros que utilizan una lógica de filtrado mejorada y operadores adicionales. Los filtros se asignan cuando un cliente determinado invoca la

consulta de suscripción en una conexión WebSocket segura. Para obtener más información sobre la utilidad de transformación para el filtrado mejorado, incluido el formato de la carga de la variable de consulta `filter`, consulte Descripción general de los solucionadores de JavaScript.

Restricciones adicionales al filtrado mejorado

A continuación, se muestran varios casos de uso en los que se imponen restricciones adicionales a los filtros mejorados:

- Los filtros mejorados no admiten el filtrado de las listas de objetos de nivel superior. En este caso de uso, los datos publicados de la mutación se ignoran en las suscripciones mejoradas.
- AWS AppSync admite hasta cinco niveles de anidamiento. Los filtros más allá del nivel cinco de los campos de esquema de anidamiento se ignoran. Observe la respuesta de GraphQL que aparece a continuación. El campo `continent` en `venue.address.country.metadata.continent` está permitido porque es un nido de nivel cinco. Sin embargo, `financial` en `venue.address.country.metadata.capital.financial` es un nido de nivel seis, por lo que el filtro no funcionará:

```
{
    "data": {
        "onCreateFilterEvent": {
            "venue": {
                "address": {
                    "country": {
                        "metadata": {
                            "capital": {
                                "financial": "New York"
                            },
                            "continent" : "North America"
                        }
                    },
                    "state": "WA"
                },
                "builtYear": 2023
            },
            "private": false,
        }
    }
}
```

Cancelación de la suscripción de las conexiones de WebSocket mediante filtros

En AWS AppSync, puede forzar la cancelación de la suscripción y cerrar (invalidar) una conexión WebSocket desde un cliente conectado en función de una lógica de filtrado específica. Esto resulta útil en contextos de autorización, por ejemplo, cuando se elimina un usuario de un grupo.

La invalidación de la suscripción se produce en respuesta a una carga útil definida en una mutación. Le recomendamos que considere las mutaciones utilizadas para invalidar las conexiones a una suscripción como operaciones administrativas de su API y que ajuste el ámbito de los permisos en consecuencia limitando su uso a un usuario administrador, un grupo o un servicio de backend. Por ejemplo, si utiliza directivas de autorización de esquemas como `@aws_auth(cognito_groups: ["Administrators"])` o `@aws_iam`. Para obtener más información, consulte Uso de modos de autorización adicionales.

Los filtros de invalidación utilizan la misma sintaxis y lógica que los filtros de suscripción mejorados. Defina estos filtros mediante las siguientes utilidades:

- `extensions.invalidateSubscriptions()`: definido en el controlador de respuestas del solucionador de GraphQL para una mutación.
- `extensions.setSubscriptionInvalidationFilter()`: definido en el controlador de respuestas del solucionador de GraphQL de las suscripciones vinculadas a la mutación.

Para obtener más información sobre las extensiones de filtrado de invalidación, consulte Descripción general de los solucionadores de JavaScript.

Uso de la invalidación de suscripciones

Para ver cómo funciona la invalidación de suscripciones en AWS AppSync, use el siguiente esquema de GraphQL:

```
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
}
```

Defina un filtro de invalidación en el código del solucionador de mutaciones `removeUserFromGroup`:

```
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 };
}
```

Cuando se invoca la mutación, los datos definidos en el objeto `payload` se utilizan para cancelar la suscripción definida en `subscriptionField`. También se define un filtro de invalidación en la plantilla de mapeo de respuestas de la suscripción `onGroupMessageCreated`.

Si la carga útil `extensions.invalidateSubscriptions()` contiene un ID que coincide con los ID del cliente suscrito tal como se definen en el filtro, se cancela la suscripción correspondiente. Además, se cierra la conexión WebSocket. Defina el código del solucionador de suscripciones para la suscripción `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;
}
```

Tenga en cuenta que el controlador de respuestas de suscripción puede tener definidos filtros de suscripción y filtros de invalidación al mismo tiempo.

Por ejemplo, supongamos que el cliente A suscribe a un nuevo usuario con el ID *`user-1`* en el grupo con el ID *`group-1`* mediante la siguiente solicitud de suscripción:

```
onGroupMessageCreated(userId : "user-1", groupId: :"group-1"){...}
```

AWS AppSync ejecuta el solucionador de suscripciones, que genera filtros de suscripción e invalidación tal como se definió en la plantilla de mapeo de respuestas `onGroupMessageCreated` anterior. Para el cliente A, los filtros de suscripción solo permiten enviar datos a *`group-1`*, y los filtros de invalidación están definidos tanto para *`user-1`* como para *`group-1`*.

Ahora, supongamos que el cliente B suscribe a un usuario con el ID *user-2* en el grupo con el ID *group-2* mediante la siguiente solicitud de suscripción:

```
onGroupMessageCreated(userId : "user-2", groupId: :"group-2"){...}
```

AWS AppSync ejecuta el solucionador de suscripciones, que genera filtros de suscripción y de invalidación. Para el cliente B, los filtros de suscripción solo permiten enviar datos a *group-2*, y los filtros de invalidación están definidos tanto para *user-2* como para *group-2*.

A continuación, supongamos que se crea un nuevo mensaje de grupo con el ID *message-1* mediante una solicitud de mutación, como en el ejemplo siguiente:

```
createGroupMessage(id: "message-1", groupId :
      "group-1", message: "test message"){...}
```

Los clientes suscritos que coincidan con los filtros definidos reciben automáticamente la siguiente carga útil de datos a través de WebSockets:

```
{
  "data": {
    "onGroupMessageCreated": {
      "id": "message-1",
      "groupId": "group-1",
      "message": "test message",
    }
  }
}
```

El cliente A recibe el mensaje porque los criterios de filtrado coinciden con el filtro de suscripción definido. Sin embargo, el cliente B no recibe el mensaje porque el usuario no forma pertenece a *group-1*. Además, la solicitud no coincide con el filtro de suscripción definido en el solucionador de suscripciones.

Por último, supongamos que *user-1* se ha eliminado de *group-1* con la siguiente solicitud de mutación:

```
removeUserFromGroup(userId: "user-1", groupId : "group-1"){...}
```

La mutación inicia una invalidación de la suscripción, tal y como se define en el código del controlador de respuestas del solucionador `extensions.invalidateSubscriptions()`. Luego,

AWS AppSync cancela la suscripción del cliente A y cierra su conexión WebSocket. El cliente B no se ve afectado, ya que la carga de invalidación definida en la mutación no coincide con su usuario o grupo.

Cuando AWS AppSync invalida una conexión, el cliente recibe un mensaje para confirmarle que se ha dado de baja:

```
{
  "message": "Subscription complete."
}
```

Uso de variables de contexto en los filtros de invalidación de suscripciones

Al igual que con los filtros de suscripción mejorados, puede utilizar la variable `context` de la extensión del filtro de invalidación de suscripciones para acceder a determinados datos.

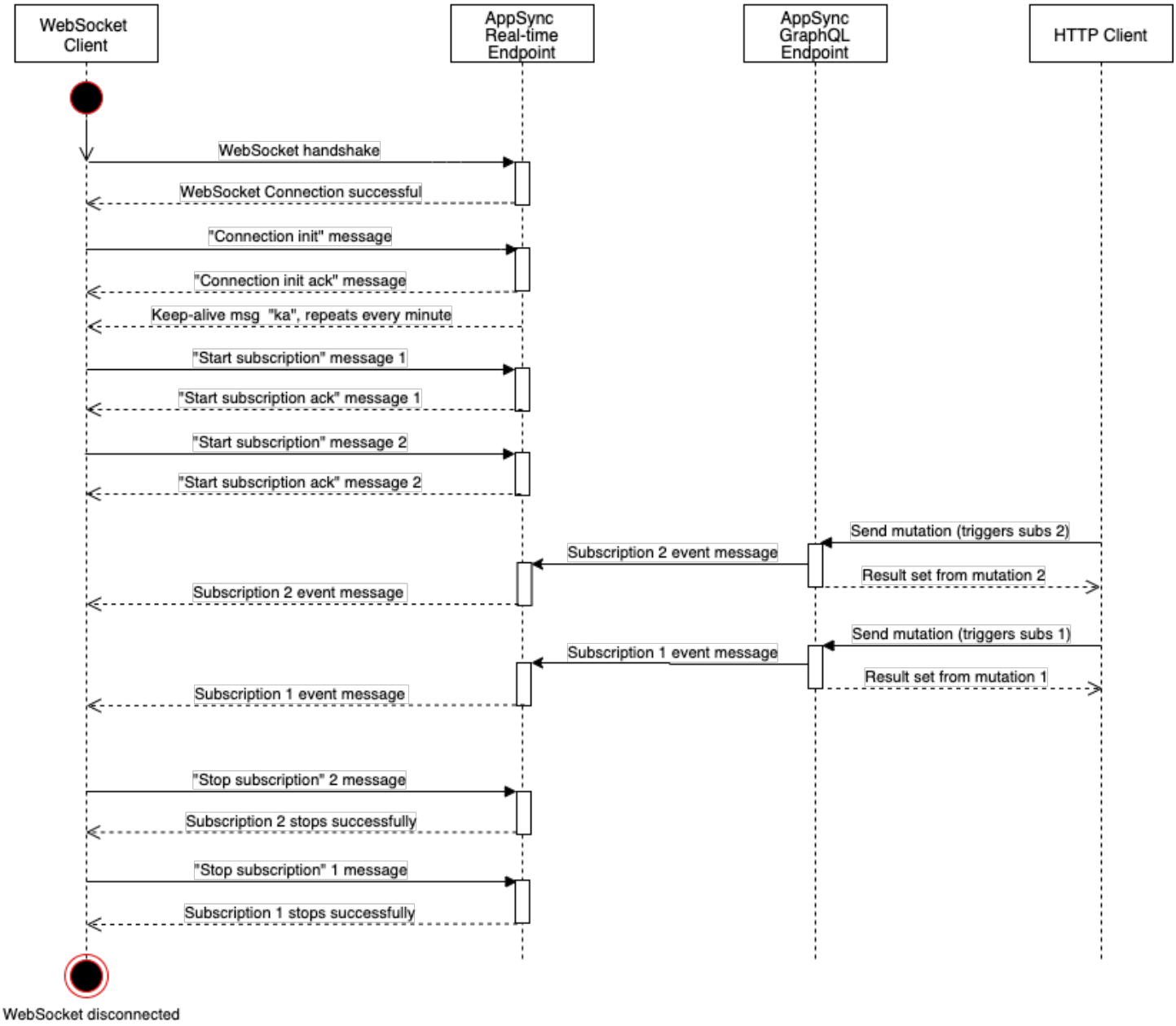
Por ejemplo, puede configurar una dirección de correo electrónico como carga útil de invalidación en la mutación y, a continuación, vincularle una correspondencia con el atributo de correo electrónico o la solicitud de un usuario suscrito autorizado con los grupos de usuarios de Amazon Cognito o con OpenID Connect. El filtro de invalidación definido en el invalidador de suscripciones `extensions.setSubscriptionInvalidationFilter()` comprueba si la dirección de correo electrónico establecida por la carga útil `extensions.invalidateSubscriptions()` de la mutación coincide con la dirección de correo electrónico recuperada del token JWT del usuario en `context.identity.claims.email`, iniciándose así la invalidación.

Creación de un cliente WebSocket en tiempo real

En las siguientes secciones, se explica la arquitectura en la que se basan las capacidades en tiempo real de AWS AppSync.

Implementación de clientes WebSocket en tiempo real para suscripciones a GraphQL

Los siguientes pasos y el diagrama de secuencias muestran el flujo de trabajo de suscripciones en tiempo real entre el cliente WebSocket, el cliente HTTP y AWS AppSync.



1. El cliente establece una conexión WebSocket con el punto de conexión en tiempo real de AWS AppSync. Si hay un error de red, el cliente debe hacer un retardo exponencial con fluctuaciones. Para obtener más información, consulte Retroceso exponencial y fluctuación en el blog de arquitectura de AWS.
2. Después de establecer correctamente la conexión WebSocket, el cliente envía un mensaje `connection_init`.
3. El cliente espera el mensaje `connection_ack` de AWS AppSync. Este mensaje incluye un parámetro `connectionTimeoutMs`, que es el tiempo de espera máximo en milisegundos para un mensaje `"ka"` (keep-alive).

4. AWS AppSync envía mensajes `"ka"` periódicamente. El cliente realiza un seguimiento de la hora en que recibió cada mensaje `"ka"`. Si el cliente no recibe un mensaje `"ka"` en cuestión de milisegundos `connectionTimeoutMs`, debe cerrar la conexión.
5. El cliente registra la suscripción enviando un mensaje de suscripción `start`. Una sola conexión WebSocket admite varias suscripciones incluso si están en distintos modos de autorización.
6. El cliente espera a que AWS AppSync envíe mensajes `start_ack` para confirmar las suscripciones correctas. Si hay un error, AWS AppSync devuelve un mensaje `"type": "error"`.
7. El cliente escucha los eventos de suscripción, que se envían después de llamar a la mutación correspondiente. Las consultas y mutaciones generalmente se envían a través de `https://` al punto de conexión de GraphQL de AWS AppSync. Las suscripciones fluyen a través del punto de conexión de AWS AppSync en tiempo real utilizando el WebSocket seguro (`wss://`).
8. El cliente anula el registro de la suscripción enviando un mensaje de suscripción `stop`.
9. Después de anular el registro de todas las suscripciones y verificar que no haya mensajes que se transfieran a través del WebSocket, el cliente puede desconectarse de la conexión WebSocket.

Detalles del protocolo de enlace para establecer la conexión WebSocket

Para conectarse e iniciar un protocolo de enlace con AWS AppSync, un cliente WebSocket necesita lo siguiente:

- El punto de conexión de AWS AppSync en tiempo real
- Una cadena de consulta que contiene los parámetros `header` y `payload`:
  - `header`: contiene información relevante para la autorización y el punto de conexión de AWS AppSync. Esta es una cadena cifrada en base64 a partir de un objeto JSON representado en forma de cadena. El contenido del objeto JSON varía en función del modo de autorización.
  - `payload`: cadena cifrada en Base64 de `payload`.

Con estos requisitos, un cliente WebSocket puede conectarse a la URL, que contiene el punto de conexión en tiempo real con la cadena de consulta, utilizando `graphql-ws` como protocolo WebSocket.

Descubrir el punto de enlace de en tiempo real desde el punto de enlace de GraphQL

El punto de conexión de GraphQL de AWS AppSync y el punto de conexión en tiempo real de AWS AppSync son ligeramente diferentes en el protocolo y el dominio. Puede recuperar el punto de

conexión de GraphQL mediante el comando `aws appsync get-graphql-api` de AWS Command Line Interface (AWS CLI).

Punto de conexión de GraphQL de AWS AppSync:

```
https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql
```

Punto de conexión de AWS AppSync en tiempo real:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Las aplicaciones pueden conectarse al punto de conexión de GraphQL de AWS AppSync (`https://`) utilizando cualquier cliente HTTP para consultas y mutaciones. Las aplicaciones pueden conectarse al punto de conexión de AWS AppSync en tiempo real (`wss://`) utilizando cualquier cliente WebSocket para suscripciones.

Con los nombres de dominio personalizados, puede interactuar con ambos puntos de conexión mediante un único dominio. Por ejemplo, si configura `api.example.com` como su dominio personalizado, puede interactuar con sus puntos de conexión de GraphQL y en tiempo real mediante estas URL:

Punto de conexión de GraphQL del dominio personalizado de AWS AppSync:

```
https://api.example.com/graphql
```

Punto de conexión en tiempo real del dominio personalizado de AWS AppSync:

```
wss://api.example.com/graphql/realtime
```

Formato de parámetro de encabezado basado en el modo de autorización de la API de AWS AppSync

El formato del objeto `header` utilizado en la cadena de consulta de conexión varía según el modo de autorización de la API de AWS AppSync. El campo `host` del objeto hace referencia al punto de conexión de GraphQL de AWS AppSync, que se utiliza para validar la conexión incluso si la llamada `wss://` se realiza referida al punto de conexión en tiempo real. Para iniciar el protocolo de enlace y establecer la conexión autorizada, `payload` debe ser un objeto JSON vacío.

Clave de API

Encabezado de clave API

Contenido del encabezado

- `"host": <string>`: el host del punto de conexión de GraphQL de AWS AppSync o su nombre de dominio personalizado.
- `"x-api-key": <string>`: la clave de API configurada para la API de AWS AppSync.

Ejemplo

```
{
    "host":"example1234567890000.appsync-api.us-east-1.amazonaws.com",
    "x-api-key":"da2-12345678901234567890123456"
}
```

Contenido de la carga útil

```
{}
```

URL de la solicitud

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJob3N0IjoiZXhhbXBsZTEyMzQ1Njc4OTAwMDAuYXBwc3luYy1hcGkudXMtZWFzdC0xLmFtYXpvbmF3cy5jb20i
```

Grupos de usuarios de Amazon Cognito y OpenID Connect (OIDC)

Amazon Cognito y OIDCHeader

Contenido del encabezado:

- `"Authorization": <string>`: un token de ID de JWT. El encabezado puede usar un esquema Bearer.
- `"host": <string>`: el host del punto de conexión de GraphQL de AWS AppSync o su nombre de dominio personalizado.

Ejemplo:

```
{

 "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"
}
```

Contenido de la carga útil:

```
{}
```

URL de la solicitud:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJBdXRob3JpemF0aW9uIjoiZXlKcmFXUWlPaUpqYkc1eGIzQTVlVzVNSzA5UVlYSXJNVEpIV0VGTFNYQmllVTVX
```

IAM

Encabezado de IAM

Contenido del encabezado

- `"accept": "application/json, text/javascript"`: un parámetro `<string>` constante.
- `"content-encoding": "amz-1.0"`: un parámetro `<string>` constante.
- `"content-type": "application/json; charset=UTF-8"`: un parámetro `<string>` constante.
- `"host": <string>`: este es el host para el punto de conexión de GraphQL de AWS AppSync.
  - `"x-amz-date": <string>`: la marca de tiempo debe estar en UTC y en el formato ISO 8601 siguiente: AAAAMMDD'T'HHMMSS'Z'. Por ejemplo, 20150830T123600Z es una marca de tiempo válida. No incluya milisegundos en la marca de tiempo. Para obtener más información, consulte Control de fechas en Signature Version 4 en la Referencia general de AWS.
  - `"X-Amz-Security-Token": <string>`: el token de sesión de AWS, que se requiere cuando se utilizan credenciales de seguridad temporales. Para obtener más información, consulte Uso de credenciales temporales con AWS en la Guía del usuario de IAM.

- `"Authorization": <string>`: información de firma de Signature Version 4 (SigV4) para el punto de conexión de AWS AppSync. Para obtener más información sobre el proceso de firma, consulte Tarea 4: Añadir la firma a la solicitud HTTP en la Referencia general de AWS.

La solicitud HTTP de firma SigV4 incluye una URL canónica, que es el punto de conexión de GraphQL de AWS AppSync con `/connect` anexado. La Región de AWS del punto de conexión del servicio es la misma Región donde está utilizando la API de AWS AppSync y el nombre de servicio es “appsync”. La solicitud HTTP para firmar es la siguiente:

```
{
  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",
  }
}
```

Ejemplo

```
{
  "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=XXXXXXXXXXXXXXXXXX/20200401/
us-east-1/appsync/aws4_request, SignedHeaders=accept;content-
encoding;content-type;host;x-amz-date;x-amz-security-token,
 Signature=83EXAMPLEbcc1fe3ee69f75cd5ebbf4cb4f150e4f99cec869f149c5EXAMPLEdc"
}
```

Contenido de la carga útil

```
{}
```

URL de la solicitud

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyEXAMPLEHQiOiJhcHBsaWNhdGlvbi9qc29uLCB0ZXh0L2phdmFEXAMPLEQiLCJjb250ZW50LWVuY29kaW5nIjoE
```

Para firmar la solicitud con un dominio personalizado:

```
{
  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",
  }
}
```

Ejemplo

```
{
  "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=XXXXXXXXXXXXXXXXXXX/20200401/
us-east-1/appsync/aws4_request, SignedHeaders=accept;content-
encoding;content-type;host;x-amz-date;x-amz-security-token,
 Signature=83EXAMPLEbcc1fe3ee69f75cd5ebbf4cb4f150e4f99cec869f149c5EXAMPLEdc"
}
```

Contenido de la carga útil

```
{}
```

URL de la solicitud

```
wss://api.example.com/graphql?
header=eyEXAMPLEHQiOiJhcHBsaWNhdGlvbi9qc29uLCB0ZXh0L2phdmFEXAMPLEQiLCJjb250ZW50LWVuY29kaW5nIjoE
```

> **Note**
>
> Una conexión WebSocket puede tener varias suscripciones (incluso con distintos modos de autenticación). Una forma de implementar esto consiste en crear una conexión WebSocket para la primera suscripción y después cerrarla cuando la última suscripción no esté registrada. Podría optimizar esto esperando unos segundos antes de cerrar la conexión

WebSocket, en caso de que la aplicación se suscriba inmediatamente después de que no se registre la última suscripción. Para un ejemplo de aplicación móvil, al cambiar de una pantalla a otra, al desmontar un evento, se detiene una suscripción y al montar un evento, se inicia una suscripción distinta.

Autorización Lambda

Encabezado de autorización Lambda

Contenido del encabezado

- `"Authorization": <string>`: el valor por el que se pasa como `authorizationToken`.
- `"host": <string>`: el host del punto de conexión de GraphQL de AWS AppSync o su nombre de dominio personalizado.

Ejemplo

```
{

 "Authorization":"M0UzQzM1MkQtMkI0Ni00OTZCLUI1NkQtMUM0MTQ0QjVBRTczCkI1REEzRTIxLTk5NzItNDJENi1BQ
    "host":"example1234567890000.appsync-api.us-east-1.amazonaws.com"
}
```

Contenido de la carga útil

```
{}
```

URL de la solicitud

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJBdXRob3JpemF0aW9uIjoiZXlKcmFXUWlPaUpqYkc1eGIzQTVlVzVNSzA5UVlYSXJNVEpIV0VGTFNYQmllVTVX
```

Operación WebSocket en tiempo real

Después de iniciar un protocolo de enlace WebSocket correcto con AWS AppSync, el cliente debe enviar un mensaje subsiguiente para completar el paso de conexión a AWS AppSync para distintas operaciones. Estos mensajes requieren los siguientes datos:

- `type`: el tipo de operación.

- `id`: un identificador único para la suscripción. Recomendamos usar un UUID para este fin.
- `payload`: la carga útil asociada según el tipo de operación.

El campo `type` es el único campo obligatorio; los campos `id` y `payload` son opcionales.

Secuencia de eventos

Para iniciar, establecer, registrar y procesar correctamente la solicitud de suscripción, el cliente debe pasar por la siguiente secuencia:

1. Inicializar conexión (`connection_init`)
2. Reconocimiento de conexión (`connection_ack`)
3. Registro de suscripción (`start`)
4. Reconocimiento de suscripción (`start_ack`)
5. Procesamiento de suscripción (`data`)
6. Anulación del registro de suscripción (`stop`)

Mensaje connection init

Después de un protocolo de enlace correcto, el cliente debe enviar el mensaje `connection_init` para empezar a comunicarse con el punto de conexión de AWS AppSync en tiempo real. Sin este paso, se omitirán todos los demás mensajes. El mensaje es una cadena obtenida mediante la representación en forma de cadena del siguiente objeto JSON como se indica a continuación:

```
{ "type": "connection_init" }
```

Mensaje de confirmación de conexión

Después de enviar el mensaje `connection_init`, el cliente debe esperar el mensaje `connection_ack`. Todos los mensajes enviados antes de recibir `connection_ack` se omitirán. El mensaje debería decir lo siguiente:

```
{
  "type": "connection_ack",
  "payload": {
    // Time in milliseconds waiting for ka message before the client should terminate
 the WebSocket connection
    "connectionTimeoutMs": 300000
```

```
  }
}
```

Mensaje Keep-alive

Además del mensaje de confirmación de conexión, el cliente recibe periódicamente mensajes keep-alive. Si el cliente no recibe un mensaje keep-alive durante el periodo de espera, debe cerrar la conexión. AWS AppSync sigue enviando estos mensajes y prestando servicio a las suscripciones registradas hasta que cierra la conexión automáticamente (después de 24 horas). Los mensajes keep-alive son latidos y no necesitan reconocimiento por parte del cliente.

```
{ "type": "ka" }
```

Mensaje de registro de suscripción

Cuando el cliente recibe un mensaje `connection_ack`, puede enviar mensajes de registro de suscripción a AWS AppSync. Este tipo de mensaje es un objeto JSON representado en forma de cadena que contiene los siguientes campos:

- `"id": <string>`: el ID de la suscripción. Este ID debe ser único para cada suscripción; de lo contrario, el servidor devuelve un error que indica que el ID de suscripción está duplicado.
- `"type": "start"`: un parámetro `<string>` constante.
- `"payload": <0bject>`: un objeto que contiene la información relevante para la suscripción.
  - `"data": <string>`: un objeto JSON representado en forma de cadena que contiene una consulta de GraphQL y variables.
    - `"query": <string>`: operación de GraphQL.
    - `"variables": <0bject>`: un objeto que contiene las variables para la consulta.
  - `"extensions": <0bject>`: un objeto que contiene un objeto de autorización.
- `"authorization": <0bject>`: un objeto que contiene los campos necesarios para la autorización.

Objeto de autorización para el registro de suscripción

Las mismas reglas de la sección Formato de parámetro de encabezado basado en el modo de autorización de la API de AWS AppSync se aplican al objeto de autorización. La única excepción es para IAM, donde la información de la firma SigV4 es ligeramente diferente. Para obtener más información, consulte el ejemplo de IAM.

Ejemplo de uso de grupos de usuarios de 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"
}
```

Ejemplo de uso de 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"
}
```

Ejemplo de uso de un nombre de dominio personalizado:

```
{
  "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"
}
```

La firma SigV4 no necesita que se anexe `/connect` a la URL y la operación de GraphQL representada en forma de cadena JSON reemplaza a `data`. A continuación, se muestra un ejemplo de una solicitud de firma 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",
  }
}
```

Mensaje de confirmación de suscripción

Después de enviar el mensaje de inicio de la suscripción, el cliente debe esperar a que AWS AppSync envíe el mensaje `start_ack`. El mensaje `start_ack` indica que la suscripción se ha realizado correctamente.

Ejemplo de reconocimiento de suscripción:

```
{
  "type": "start_ack",
  "id": "eEXAMPLE-cf23-1234-5678-152EXAMPLE69"
}
```

Mensaje de error

Si se produce un error en el registro de suscripción o el inicio de la suscripción, o bien si se termina una suscripción desde el servidor, el servidor envía un mensaje de error al cliente:

- `"type": "error"`: un parámetro `<string>` constante.
- `"id": <string>`: el ID de la suscripción registrada correspondiente, si procede.
- `"payload" <Object>`: un objeto que contiene la información de error correspondiente.

Ejemplo:

```
{
  "type": "error",
  "payload": {
    "errors": [
      {
        "errorType": "LimitExceededError",
```

```
        "message": "Rate limit exceeded"
      }
    ]
  }
}
```

Procesamiento de mensajes de datos

Cuando un cliente envía una mutación, AWS AppSync identifica a todos los suscriptores interesados en ella y envía un mensaje `"type":"data"` a cada uno, utilizando el `id` de suscripción correspondiente utilizado en la operación de suscripción `"start"`. Se espera que el cliente realice un seguimiento del `id` de suscripción que envía para que cuando se reciba un mensaje de datos, el cliente pueda asociarlo con la suscripción correspondiente.

- `"type": "data"`: un parámetro `<string>` constante.
- `"id": <string>`: el ID de la suscripción registrada correspondiente.
- `"payload" <Object>`: un objeto que contiene la información relevante para la suscripción.

Ejemplo:

```
{
  "type": "data",
  "id": "ee849ef0-cf23-4cb8-9fcb-152ae4fd1e69",
  "payload": {
    "data": {
      "onCreateMessage": {
        "__typename": "Message",
        "message": "test"
      }
    }
  }
}
```

Mensaje de anulación de registro de suscripción

Cuando la aplicación quiere dejar de escuchar los eventos de suscripción, el cliente debe enviar un mensaje con el siguiente objeto JSON representado en forma de cadena:

- `"type": "stop"`: un parámetro `<string>` constante.
- `"id": <string>`: el ID de la suscripción cuyo registro se va a cancelar.

Ejemplo:

```
{
  "type":"stop",
  "id":"ee849ef0-cf23-4cb8-9fcb-152ae4fd1e69"
}
```

AWS AppSync devuelve un mensaje de confirmación con el siguiente objeto JSON representado en forma de cadena:

- `"type": "complete"`: un parámetro `<string>` constante.
- `"id": <string>`: el ID de la suscripción cuyo registro se anuló.

Una vez que el cliente reciba el mensaje de confirmación, no recibirá más mensajes para esta suscripción en particular.

Ejemplo:

```
{
  "type":"complete",
  "id":"eEXAMPLE-cf23-1234-5678-152EXAMPLE69"
}
```

Desconexión del WebSocket

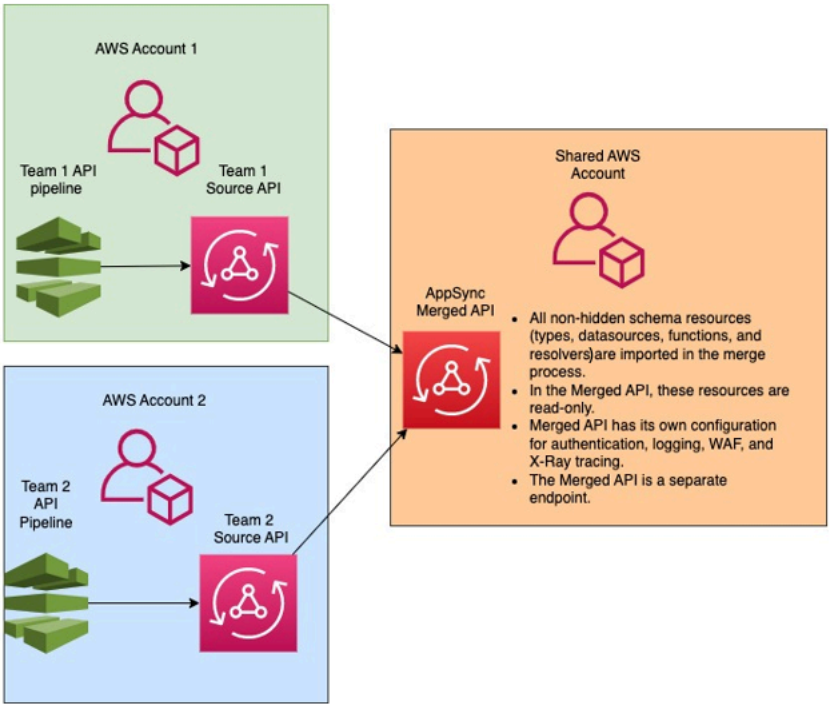
Antes de desconectar, para evitar una pérdida de datos, el cliente debe usar la lógica para verificar que no haya ninguna operación en marcha en ese momento a través de la conexión WebSocket. Se debe anular el registro de todas las suscripciones antes de desconectarse del WebSocket.

API fusionadas

A medida que se generaliza el uso de GraphQL en una organización, la facilidad de uso de las API puede afectar a su velocidad de desarrollo y viceversa. Por un lado, las organizaciones adoptan AWS AppSync y GraphQL para simplificar el desarrollo de aplicaciones, puesto que se ofrece a los desarrolladores una API flexible que les permite acceder a los datos de uno o más dominios de datos, así como manipularlos y combinarlos. Todo ello de forma segura y con una sola llamada a la red. Por otro lado, puede que los equipos de una organización que se encargan de los diferentes dominios de datos fusionados en un único punto de conexión de la API de GraphQL quieran tener la

capacidad para crear, administrar e implementar actualizaciones de API independientes entre sí a fin de acelerar su desarrollo.

Para resolver esta tensión, la característica de API fusionadas de AWS AppSync permite a los equipos de dominios de datos diferentes crear e implementar API de AWS AppSync de forma independiente (p. ej., esquemas, solucionadores, orígenes de datos y funciones de GraphQL), que luego se pueden combinar en una misma API fusionada. De este modo, las organizaciones pueden mantener una API multidominio fácil de usar y los diferentes equipos que colaboran en esa API pueden aplicar actualizaciones de la API de forma rápida e independiente.



Con las API fusionadas, las organizaciones pueden importar los recursos de varias API de AWS AppSync de orígenes independientes en un único punto de conexión de la API fusionada de AWS AppSync. Para ello, AWS AppSync le permite crear una lista de las API de origen de AWS AppSync y, a continuación, fusionar todos los metadatos asociados con las API de origen, incluidos el esquema, los tipos, las fuentes de datos, los solucionadores y las funciones, en una nueva API fusionada de AWS AppSync.

Durante las fusiones, pueden producirse conflictos de fusión debido a incoherencias en el contenido de los datos de la API de origen, como conflictos en la nomenclatura de los tipos cuando se combinan varios esquemas. En los casos de uso sencillos sin definiciones en el conflicto de las API de origen, no es necesario modificar los esquemas de las API de origen. La API fusionada

resultante simplemente importa todos los tipos, solucionadores, orígenes de datos y funciones de las API de origen de AWS AppSync originales. En los casos de uso complejos en los que surjan conflictos, los usuarios o los equipos deberán resolver los conflictos por diversos medios. AWS AppSync proporciona a los usuarios varias herramientas y ejemplos que pueden reducir los conflictos de fusión.

Las fusiones que se configuren en AWS AppSync posteriormente propagarán los cambios en las API de origen a la API fusionada asociada.

API fusionadas y federación

Existen muchas soluciones y patrones en la comunidad de GraphQL para combinar esquemas de GraphQL y permitir la colaboración en equipo a través de un gráfico compartido. AWS AppSync Las API fusionadas adoptan un enfoque de tiempo de compilación para la composición del esquema, en el que las API de origen se combinan en una API fusionada independiente. Otra opción consiste en colocar un enrutador en tiempo de ejecución en varias API de origen o gráficos secundarios. En este enfoque, el enrutador recibe una solicitud, hace referencia a un esquema combinado que mantiene como metadatos, crea un plan de solicitudes y, a continuación, distribuye los elementos de la solicitud entre sus servidores o gráficos secundarios subyacentes. La siguiente tabla compara el enfoque de tiempo de compilación de la API fusionada de AWS AppSync con los enfoques de tiempo de ejecución basados en enrutador para la composición del esquema de GraphQL:

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/deserialization, and reference resolvers in sub-graphs to perform joins.
Observability support	Built-in monitoring, logging, and tracing. A single, Merged API server means simplified debugging.	Build-your-own observability across router and all associated 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

* Las API fusionadas de AWS AppSync solo se pueden asociar a las API de origen de AWS AppSync. Si necesita asistencia para la composición de esquemas en gráficos secundarios tanto de AWS AppSync como ajenos a AWS AppSync, puede conectar una o más API fusionadas o de GraphQL de AWS AppSync a una solución basada en enrutadores. Por ejemplo, consulte el blog de referencia para agregar API de AWS AppSync como gráfico secundario mediante una arquitectura basada en enrutadores con Apollo Federation v2: Apollo GraphQL Federation with AWS AppSync.

Temas

- Resolución de conflictos de la API fusionada
- Configuración de esquemas
- Configuración de modos de autorización
- Configuración de roles de ejecución
- Configuración de API fusionadas entre cuentas mediante AWS RAM
- Fusión
- Asistencia adicional para las API fusionadas

- Limitaciones de las API fusionadas
- Creación de API fusionadas

Resolución de conflictos de la API fusionada

Si surge un conflicto de fusión, AWS AppSync proporciona a los usuarios varias herramientas y ejemplos para ayudar a solucionar los problemas.

Directivas de esquema de la API fusionada

AWS AppSync ha introducido varias directivas de GraphQL que se pueden utilizar para reducir o resolver conflictos entre las API de origen:

- @canonical: esta directiva establece la prioridad de los tipos o campos con nombres y datos similares. Si dos o más API de origen tienen el mismo tipo o campo de GraphQL, una de las API puede anotar su tipo o campo como canónico, y este se priorizará durante la fusión. Los tipos o campos en conflicto que no estén anotados en esta directiva en otras API de origen se ignoran al fusionarse.
- @hidden: esta directiva encapsula ciertos tipos o campos para eliminarlos del proceso de fusión. Los equipos tal vez deseen eliminar u ocultar tipos u operaciones específicos en la API de origen para que solo los clientes internos puedan acceder a datos de tipos específicos. Con esta directiva adjunta, los tipos o campos no se fusionan en la API fusionada.
- @renamed: esta directiva cambia los nombres de los tipos o campos para reducir los conflictos de nomenclatura. Hay situaciones en las que diferentes API tienen el mismo tipo o nombre de campo. Sin embargo, todas deben estar disponibles en el esquema fusionado. Una forma sencilla de incluirlos todos en la API fusionada consiste en cambiar el nombre del campo por uno similar, pero no idéntico.

Para mostrar el esquema de utilidades que proporcionan las directivas, observe el siguiente ejemplo:

En este ejemplo, supongamos que queremos fusionar dos API de origen. Tenemos dos esquemas que crean y recuperan publicaciones (p. ej., publicaciones en la sección de comentarios o en las redes sociales). Suponiendo que los tipos y los campos sean muy similares, la probabilidad de que surjan conflictos durante una operación de fusión es muy elevada. Los fragmentos siguientes muestran los tipos y campos de cada esquema.

El primer archivo, denominado Source1.graphql, es un esquema de GraphQL que permite al usuario crear `Posts` utilizando la mutación `putPost`. Cada `Post` contiene un título y un ID. El ID se utiliza para hacer referencia al `User`, o a la información del autor de la publicación (correo electrónico y dirección), y al `Message`, o la carga útil (contenido). El tipo de `User` se anota con la etiqueta @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
}
```

El segundo archivo, llamado Source2.graphql, es un esquema GraphQL con funciones muy similares a las de Source1.graphQL. Sin embargo, tenga en cuenta que los campos de cada tipo son diferentes. Al fusionar estos dos esquemas, se producirán conflictos de fusión debido a estas diferencias.

Observe también que Source2.graphql también contiene varias directivas para reducir estos conflictos. El tipo `Post` tiene anotada una etiqueta @hidden para ocultarse durante la operación

de fusión. El tipo `Message` tiene anotada la etiqueta @renamed para cambiar el nombre del tipo a `ChatMessage` en caso de conflicto de nomenclatura con otro tipo `Message`.

```
# 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")
}
```

Cuando se produzca la fusión, el resultado generará el archivo `MergedSchema.graphql`:

```
# 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
}
```

En la fusión ocurrieron varias cosas:

- Se priorizó el tipo `User` de Source1.graphql con respecto al `User` de Source2.graphql debido a la anotación @canonical.
- El tipo `Message` de Source1.graphql se incluyó en la fusión. Sin embargo, había un conflicto de nombres en el `Message` de Source2.graphql. Al tener la anotación @renamed, también se incluyó en la fusión, pero con el nombre alternativo `ChatMessage`.
- Se incluyó el tipo `Post` de Source1.graphql, pero no el tipo `Post` de Source2.graphql. Normalmente, habría un conflicto con este tipo, pero el tipo `Post` de Source2.graphql tenía una anotación @hidden, por lo que sus datos estaban ocultos y no se incluyeron en la fusión. Por tanto, no hubo conflicto.

- El tipo `Query` se actualizó para incluir el contenido de ambos archivos. Sin embargo, la directiva hizo que se cambiara el nombre de una consulta de `GetMessage` a `GetChatMessage`. Así se resolvió el conflicto de nomenclatura entre las dos consultas con el mismo nombre.

También puede ocurrir que no se agreguen directivas a un tipo con conflictos. En este caso, el tipo fusionado incluirá la unión de todos los campos de todas las definiciones de origen de ese tipo. Por ejemplo, observe el siguiente caso:

Este esquema, denominado Source1.graphql, permite crear y recuperar `Posts`. La configuración es similar a la del ejemplo anterior, pero con menos información.

```
# 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
}
```

Este esquema, denominado Source2.graphql, permite crear y recuperar `Reviews` (p. ej., valoraciones de películas o reseñas de restaurantes). Las `Reviews` están asociadas a la `Post` con el mismo valor de ID. En conjunto, contienen el título, el ID de la publicación y el mensaje de carga útil de la publicación de la reseña completa.

Al fusionarse, habrá un conflicto entre los dos tipos `Post`. Como no hay anotaciones para resolver este problema, se lleva a cabo, de manera predeterminada, una operación de unión de los tipos en conflicto.

```
# 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
}
```

Cuando se produzca la fusión, el resultado generará el archivo `MergedSchema.graphql`:

```
# 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
}
```

En la fusión ocurrieron varias cosas:

- Con el tipo `Mutation` no se produjo ningún conflicto y este se fusionó.
- Los campos del tipo `Post` se combinaron con una operación de unión. Observe que la unión entre los dos generó un único `id`, un `title` y una única `reviews`.
- Con el tipo `Review` no se produjo ningún conflicto y este se fusionó.
- Con el tipo `Query` no se produjo ningún conflicto y este se fusionó.

Administración de solucionadores en tipos compartidos

En el ejemplo anterior, consideremos el caso en el que Source1.graphql ha configurado un solucionador de unidades en `Query.getPost`, que utiliza un origen de datos de DynamoDB denominado `PostDatasource`. Este solucionador devolverá el `id` y el `title` de un tipo `Post`. Ahora, consideremos que Source2.graphql ha configurado un solucionador de canalización en `Post.reviews` que ejecuta dos funciones. `Function1` tiene un origen de datos `None` adjunto para realizar comprobaciones de autorización personalizadas. `Function2` tiene un origen de datos de DynamoDB adjunto para consultar la tabla `reviews`.

```
query GetPostQuery {
    getPost(id: "1") {
        id,
        title,
        reviews
    }
}
```

Cuando la consulta anterior se ejecuta mediante un cliente para el punto de conexión de la API fusionada, el servicio AWS AppSync ejecuta primero el solucionador de unidades para `Query.getPost` desde `Source1`, que llama a `PostDatasource` y devuelve los datos de DynamoDB. A continuación, ejecuta el solucionador de canalizaciones de `Post.reviews`, en el cual `Function1` ejecuta una lógica de autorización personalizada y `Function2` devuelve las revisiones en función del `id` encontrado en `$context.source`. El servicio procesa la solicitud como una sola ejecución de GraphQL, y esta solicitud sencilla requiere un único token de solicitud.

Gestión de conflictos de solucionadores en tipos compartidos

Veamos el caso siguiente, en el que también implementamos un solucionador en `Query.getPost` para proporcionar varios campos a la vez además del solucionador de campo en `Source2`. Source1.graphql sería parecido a esto:

```
# This snippet represents a file called Source1.graphql

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

type Query {
    getPost(id: ID!): Post
}
```

Source2.graphql sería parecido a esto:

```
# This snippet represents a file called Source2.graphql

type Post  {
  id: ID!
  content: String!
  contentHash: String!
  author: String!
}

type Query {
    getPost(id: ID!): Post
}
```

Si se intenta fusionar estos dos esquemas, se producirá un error de fusión, ya que las API fusionadas de AWS AppSync no permiten que haya varios solucionadores de origen asociados al mismo campo. Para resolver este conflicto, puede implementar un patrón de solucionador de campo que obligaría a Source2.graphql a agregar un tipo diferente que defina los campos del tipo `Post` que le pertenecen. En el siguiente ejemplo, agregamos un tipo denominado `PostInfo`, que contiene los campos de contenido y de autor que resolverá Source2.graphql. Source1.graphql implementará el solucionador asociado a `Query.getPost`, mientras que Source2.graphql asociará ahora un solucionador a `Post.postInfo` para garantizar que todos los datos se puedan recuperar correctamente:

```
type Post  {
  id: ID!
  postInfo: PostInfo
}
```

```
type PostInfo {
   content: String!
   contentHash: String!
   author: String!
}

type Query {
    getPost(id: ID!): Post
}
```

Si bien la resolución de este tipo de conflicto requiere que se reescriban los esquemas de las API de origen y, posiblemente, que los clientes cambien sus consultas, la ventaja de este enfoque es que la propiedad de los solucionadores fusionados queda clara entre los equipos de origen.

Configuración de esquemas

Hay dos partes responsables de configurar los esquemas para crear una API fusionada:

- Propietarios de las API fusionadas: los propietarios de las API fusionadas deben configurar la lógica de autorización de la API fusionada y los ajustes avanzados, como el registro, el seguimiento, el almacenamiento en caché y la compatibilidad con el WAF.
- Propietarios de las API de origen asociadas: los propietarios de las API asociadas deben configurar los esquemas, los solucionadores y los orígenes de datos que componen la API fusionada.

Como el esquema de la API fusionada se crea a partir de los esquemas de sus API de origen asociadas, este es de solo lectura. Esto significa que los cambios en el esquema deben iniciarse en las API de origen. En la consola AWS AppSync, puede cambiar entre su esquema fusionado y los esquemas individuales de las API de origen incluidas en su API fusionada. Para ello, use la lista desplegable situada encima de la ventana Esquema.

Configuración de modos de autorización

Hay varios modos de autorización disponibles para proteger su API fusionada. Para obtener más información sobre los modos de autorización en AWS AppSync, consulte Autorización y autenticación.

Los modos de autorización siguientes están disponibles para su uso con las API fusionadas:

- Clave de la API: la estrategia de autorización más sencilla. Todas las solicitudes deben incluir una clave de la API en el encabezado de la solicitud `x-api-key`. Las claves de la API vencidas se conservan durante 60 días después de la fecha de vencimiento.
- AWS Identity and Access Management (IAM): la estrategia de autorización de AWS IAM autoriza todas las solicitudes firmadas mediante sigv4.
- Grupos de usuarios de Amazon Cognito: autorice a sus usuarios a través de los grupos de usuarios de Amazon Cognito para lograr un control más detallado.
- Autorizadores de AWS Lambda: función sin servidor que permite autenticar y autorizar el acceso a la API de AWS AppSync mediante una lógica personalizada.
- OpenID Connect: este tipo de autorización aplica tókenes de OpenID Connect (OIDC) proporcionados por un servicio compatible con OIDC. Su aplicación puede aprovechar los usuarios y los privilegios definidos por su proveedor OIDC para controlar el acceso.

Los modos de autorización de una API fusionada los configura el propietario de la API fusionada. Al llevar a cabo una operación de fusión, la API fusionada debe incluir el modo de autorización principal configurado en una API de origen, ya sea como su propio modo de autorización principal o como modo de autorización secundario. De lo contrario, será incompatible, la operación de fusión producirá un error y se generará un conflicto. Cuando se utilizan directivas de autenticación múltiple en las API de origen, el proceso de fusión puede fusionar automáticamente estas directivas en el punto de conexión unificado. Si el modo de autorización principal de la API de origen no coincide con el modo de autorización principal de la API fusionada, este agregará automáticamente estas directivas de autorización para garantizar que el modo de autorización de los tipos de la API de origen sea coherente.

Configuración de roles de ejecución

Al crear una API fusionada, es necesario definir un rol de servicio. Un rol de servicio de AWS es un rol de AWS Identity and Access Management (IAM) que utilizan los servicios de AWS para realizar tareas en su nombre.

En este contexto, su API fusionada debe ejecutar solucionadores que accedan a los datos de orígenes de datos configurados en sus API de origen. El rol de servicio necesario para ello es `mergedApiExecutionRole`, y este debe tener acceso explícito para ejecutar solicitudes en las API de origen incluidas en la API fusionada mediante el permiso `appsync:SourceGraphQL` de IAM. Durante la ejecución de una solicitud de GraphQL, el servicio AWS AppSync asumirá este rol de servicio y lo autorizará a realizar la acción `appsync:SourceGraphQL`.

AWS AppSync admite la opción de permitir o denegar este permiso en campos específicos de nivel superior de la solicitud; por ejemplo, cómo actúa el modo de autorización de IAM para las API de IAM. Para los campos que no son de nivel superior, AWS AppSync le solicitará que defina el permiso en el propio ARN de la API de origen. Para restringir el acceso a campos específicos que no son de nivel superior en la API fusionada, le recomendamos implementar una lógica personalizada en su Lambda u ocultar los campos de la API de origen de la API fusionada mediante la directiva @hidden. Si quiere permitir que el rol realice todas las operaciones de datos dentro de una API de origen, puede agregar la política que se indica a continuación. Tenga en cuenta que la primera entrada de recursos permite el acceso a todos los campos de nivel superior y la segunda incluye los solucionadores secundarios que autorizan en el propio recurso de la API de origen:

```
{
    "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"]
    }]
}
```

Si quiere limitar el acceso únicamente a un campo de nivel superior específico, puede usar una política como esta:

```
{
    "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"]
    }]
}
```

También puede usar el asistente de creación de API de la consola de AWS AppSync para generar un rol de servicio que permita que la API fusionada acceda a los recursos configurados en las API de origen ubicadas en la misma cuenta que la API fusionada. En el caso de que las API de origen no

estén en la misma cuenta que la API fusionada, primero debe compartir los recursos mediante AWS Resource Access Manager (AWS RAM).

Configuración de API fusionadas entre cuentas mediante AWS RAM

Al crear una API fusionada, si lo desea, puede asociar las API de origen de otras cuentas que se hayan compartido mediante AWS Resource Access Manager (AWS RAM). AWS RAM le ayuda a compartir sus recursos de forma segura entre cuentas de AWS, dentro de su organización o en unidades organizativas (OU) y con roles y usuarios de IAM.

AWS AppSync se integra con AWS RAM para permitir la configuración y el acceso a las API de origen en varias cuentas desde una única API fusionada. AWS RAM permite crear un recurso compartido o un contenedor de recursos y los conjuntos de permisos que se compartirán para cada uno de ellos. Puede agregar API de AWS AppSync a un recurso compartido en AWS RAM. Dentro de un recurso compartido, AWS AppSync proporciona tres conjuntos de permisos diferentes que se pueden asociar a una API de AWS AppSync en RAM:

1. `AWSRAMPermissionAppSyncSourceApiOperationAccess`: el conjunto de permisos predeterminado que se agrega al compartir una API de AWS AppSync en AWS RAM si no se especifica ningún otro permiso. Este conjunto de permisos se utiliza para compartir una API de origen de AWS AppSync con el propietario de una API fusionada. Este conjunto de permisos incluye el permiso `appsync:AssociateMergedGraphqlApi` en la API de origen, así como el permiso `appsync:SourceGraphQL` necesario para acceder a los recursos de la API de origen en tiempo de ejecución.
2. `AWSRAMPermissionAppSyncMergedApiOperationAccess`: este conjunto de permisos debe configurarse al compartir una API fusionada con el propietario de una API de origen. Este conjunto de permisos permite a la API de origen configurar la API fusionada, incluida la posibilidad de asociar cualquier API de origen que sea propiedad de la entidad principal de destino con la API fusionada, así como de leer y actualizar las asociaciones con la API de origen de la API fusionada.
3. `AWSRAMPermissionAppSyncAllowSourceGraphQLAccess`: este conjunto de permisos permite utilizar el permiso `appsync:SourceGraphQL` con una API de AWS AppSync. La finalidad prevista de este permiso es la de compartir una API de origen con el propietario de una API fusionada. A diferencia del conjunto de permisos predeterminado para el acceso a las operaciones de la API de origen, este conjunto de permisos solo incluye el permiso de tiempo de ejecución `appsync:SourceGraphQL`. Si un usuario opta por compartir el acceso a la operación de la API fusionada con el propietario de la API de origen, también tendrá que compartir este

permiso de la API de origen con el propietario de la API fusionada para tener acceso en tiempo de ejecución a través del punto de conexión de la API fusionada.

AWS AppSync también admite permisos administrados por el cliente. Si uno de los permisos administrados por AWS facilitados no funciona, puede crear su propio permiso administrado por el cliente. Los permisos administrados por el cliente son permisos administrados que usted crea y mantiene especificando exactamente qué acciones se pueden llevar a cabo y en qué condiciones con los recursos que se comparten mediante AWS RAM. AWS AppSync le permite elegir entre las siguientes acciones al crear su propio permiso:

1. `appsync:AssociateSourceGraphqlApi`
2. `appsync:AssociateMergedGraphqlApi`
3. `appsync:GetSourceApiAssociation`
4. `appsync:UpdateSourceApiAssociation`
5. `appsync:StartSchemaMerge`
6. `appsync:ListTypesByAssociation`
7. `appsync:SourceGraphQL`

Cuando haya compartido adecuadamente una API de origen o una API fusionada AWS RAM y, en caso necesario, se haya aceptado la invitación para compartir recursos, podrá verse en la consola de AWS AppSync cuando cree o actualice las asociaciones de la API de origen en su API fusionada. También puede enumerar todas las API de AWS AppSync que se han compartido con su cuenta mediante AWS RAM, independientemente del permiso definido, llamando a la operación `ListGraphqlApis` proporcionada por AWS AppSync y utilizando el filtro del propietario `OTHER_ACCOUNTS`.

> **Note**
>
> Para compartir mediante AWS RAM, la persona que llama en AWS RAM debe tener permiso para realizar la acción `appsync:PutResourcePolicy` en cualquier API que se vaya a compartir.

Fusión

Administración de fusiones

Las API fusionadas están diseñadas para respaldar la colaboración en equipo en un punto de conexión de AWS AppSync unificado. Los equipos pueden desarrollar por su cuenta sus propias API de origen de GraphQL aisladas en el backend mientras el servicio AWS AppSync gestiona la integración de los recursos en el punto de conexión único de la API fusionada para reducir la fricción en la colaboración y reducir los plazos de desarrollo.

Fusiones automáticas

Las API de origen asociadas a su API fusionada de AWS AppSync se pueden configurar para que se fusionen automáticamente (fusión automática) en la API fusionada después de hacer cambios en la API de origen. Esto garantiza que los cambios en la API de origen siempre se propaguen al punto de conexión de la API fusionada en segundo plano. Cualquier cambio en el esquema de la API de origen se actualizará en la API fusionada siempre y cuando no genere un conflicto de fusión con una definición existente en la API fusionada. Si la actualización de la API de origen actualiza un solucionador, un origen de datos o una función, también se actualizará el recurso importado. Cuando se introduce un nuevo conflicto que no se puede resolver automáticamente (resolución automática), se rechaza la actualización del esquema de la API fusionada debido a un conflicto no admitido durante la operación de fusión. El mensaje de error está disponible en la consola para cada asociación de la API de origen cuyo estado sea `MERGE_FAILED`. También puede inspeccionar el mensaje de error llamando a la operación `GetSourceApiAssociation` de una asociación de API de origen determinada mediante el SDK de AWS o la CLI de AWS de la siguiente manera:

```
aws appsync get-source-api-association --merged-api-identifier <Merged API ARN> --
association-id <SourceApiAssociation id>
```

Esto generará un resultado con el formato siguiente:

```
{
    "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."
    }
}
```

Fusiones manuales

La configuración predeterminada de una API de origen es una fusión manual. Para fusionar cualquier cambio que se haya producido en las API de origen desde la última actualización de la API fusionada, el propietario de la API de origen puede invocar una fusión manual desde la consola de AWS AppSync o mediante la operación `StartSchemaMerge` disponible en el SDK de AWS y la CLI de AWS.

Asistencia adicional para las API fusionadas

Configuración de suscripciones

A diferencia de los enfoques basados en enrutadores para la composición de esquemas de GraphQL, las API fusionadas de AWS AppSync ofrecen asistencia integrada para las suscripciones de GraphQL. Todas las operaciones de suscripción definidas en sus API de origen asociadas se fusionarán automáticamente y funcionarán en la API combinada sin modificaciones. Para obtener más información sobre la compatibilidad de AWS AppSync con suscripciones a través de una conexión WebSockets sin servidor, consulte Datos en tiempo real.

Configuración de la observabilidad

Las API fusionadas de AWS AppSync proporcionan registro, supervisión y métricas integrados a través de Amazon CloudWatch. AWS AppSync también proporciona asistencia integrada para el rastreo mediante AWS X-Ray.

Configuración de dominios personalizados

Las API fusionadas de AWS AppSync proporcionan asistencia integrada para el uso de dominios personalizados con los puntos de conexión de GraphQL y en tiempo real de sus API fusionadas.

Configuración del almacenamiento en caché

Las API fusionadas de AWS AppSync ofrecen asistencia integrada para almacenar en caché, de forma opcional, las respuestas a nivel de solicitud o a nivel de solucionador, así como para la compresión de respuestas. Para obtener más información, consulte Almacenamiento en caché y compresión.

Configuración de API privadas

Las API fusionadas de AWS AppSync proporcionan asistencia integrada para las API privadas que limitan el acceso de los puntos de conexión de GraphQL y en tiempo real de sus API fusionadas al tráfico procedente de los puntos de conexión de la VPC que puede configurar.

Configuración de reglas de firewall

Las API fusionadas de AWS AppSync proporcionan asistencia integrada para AWS WAF, lo que le permite proteger sus API mediante la definición de reglas de firewall de aplicaciones web.

Configuración de registros de auditoría

Las API fusionadas de AWS AppSync proporcionan asistencia integrada para AWS CloudTrail, lo que le permite configurar y administrar los registros de auditoría.

Limitaciones de las API fusionadas

Tenga en cuenta las siguientes reglas cuando desarrolle API fusionadas:

1. Una API fusionada no puede ser una API de origen para otra API fusionada.
2. Una API de origen no se puede asociar a más de una API fusionada.
3. El límite de tamaño predeterminado para un documento de esquema de API fusionada es de 10 MB.
4. De manera predeterminada, el número de API de origen que se pueden asociar a una API fusionada es 10. No obstante, es posible solicitar un aumento de límite en caso de que sean necesarias más de 10 API de origen en su API fusionada.

Creación de API fusionadas

Para crear una API fusionada en la consola

1. Inicie sesión en AWS Management Console y abra la consola de AWS AppSync.

- En el Panel, elija Crear API.

2. Seleccione API fusionada y, a continuación, Siguiente.
3. En la página Especificar los detalles de la API, introduzca la información siguiente:

   a. En Detalles de API, escriba la información siguiente:

      i. Especifique el Nombre de API de su API fusionada. Este campo es una forma de etiquetar su API de GraphQL para distinguirla fácilmente de otras API de GraphQL.

      ii. Especifique los Datos de contacto. Este campo es opcional y adjunta un nombre o grupo a la API de GraphQL. No está vinculado a otros recursos ni es generado por ellos, y funciona de forma muy parecida al campo de nombre de la API.

   b. En Rol de servicio, debe asignar una función de ejecución de IAM a la API fusionada para que AWS AppSync pueda importar y utilizar sus recursos de forma segura en tiempo de ejecución. Puede optar por Crear y utilizar un nuevo rol de servicio, lo que le permitirá especificar las políticas y los recursos que utilizará AWS AppSync. También puede importar un rol de IAM existente seleccionando Usar un rol de servicio existente y, a continuación, seleccionando el rol en la lista desplegable.

   c. En Configuración de API privada, puede optar por habilitar las características de API privadas. Tenga en cuenta que esta opción no se puede cambiar después de crear la API fusionada. Para obtener más información acerca del uso de API privadas, consulte Uso de API privadas de AWS AppSync.

   Cuando haya terminado, elija Siguiente.

4. A continuación, debe agregar las API de GraphQL que se usarán como base para la API fusionada. En la página Seleccionar API de origen, introduzca la siguiente información:

   a. En la tabla API de su cuenta de AWS, elija Agregar API de origen. En la lista de API de GraphQL, cada entrada contiene los siguientes datos:

      i. Nombre: el campo de nombre de la API de la API de GraphQL.

      ii. ID de la API: el valor de ID único de la API de GraphQL.

      iii. Modo de autorización principal: el modo de autorización predeterminado para la API de GraphQL. Para obtener más información sobre los modos de autorización en AWS AppSync, consulte Autorización y autenticación.

      iv. Modos de autorización adicionales: los modos de autorización secundarios que se configuraron en la API de GraphQL.

v. Seleccione las API que usará en la API fusionada. Para ello, seleccione la casilla de verificación situada junto al campo Nombre de la API. A continuación, seleccione Agregar API de origen. Las API de GraphQL seleccionadas aparecerán en la tabla API de sus cuentas de AWS.

b. En la tabla API de sus cuentas de AWS, seleccione Agregar API de origen. Las API de GraphQL de esta lista provienen de otras cuentas que comparten sus recursos con la suya a través de AWS Resource Access Manager (AWS RAM). El proceso para seleccionar las API de GraphQL en esta tabla es el mismo que el proceso de la sección anterior. Para obtener más información sobre cómo compartir recursosAWS RAM, consulte ¿Qué es AWS Resource Access Manager? .

   Cuando haya terminado, elija Siguiente.

c. Agregue su modo de autenticación principal. Para obtener más información, consulte Autorización y autenticación. Seleccione Siguiente.

d. Revise la información indicada y, a continuación, seleccione Crear API.

Introspección de RDS

AWS AppSync facilita la creación de API a partir de bases de datos relacionales existentes. Su utilidad de introspección puede descubrir modelos a partir de tablas de bases de datos y proponer tipos de GraphQL. El asistente Creación de API de la consola de AWS AppSync puede generar una API al instante a partir de una base de datos Aurora MySQL o PostgreSQL. Crea automáticamente tipos y solucionadores de JavaScript para leer y escribir datos.

AWS AppSync ofrece una integración directa con las bases de datos de Amazon Aurora a través de la API de datos de Amazon RDS. En lugar de requerir una conexión de base de datos persistente, la API de datos de Amazon RDS ofrece un punto de conexión HTTP seguro al que AWS AppSync se conecta para ejecutar instrucciones SQL. Puede utilizarlo para crear una API de base de datos relacional para sus cargas de trabajo de MySQL y PostgreSQL en Aurora.

La creación de una API para su base de datos relacional con AWS AppSync tiene varias ventajas:

- Su base de datos no está expuesta directamente a los clientes, lo que desacopla el punto de acceso de la propia base de datos.
- Puede crear API diseñadas específicamente para que se adapten a las necesidades de las distintas aplicaciones, lo que elimina la necesidad de una lógica empresarial personalizada en los frontends. Esto se ajusta al patrón Backend-For-Frontend (BFF).

- La autorización y el control de acceso se pueden implementar en la capa de AWS AppSync mediante diversos modos de autorización para controlar el acceso. No se requieren recursos informáticos adicionales para conectarse a la base de datos, como alojar un servidor web o establecer conexiones mediante proxy.
- Las capacidades en tiempo real se pueden añadir mediante suscripciones, y las mutaciones de datos realizadas a través de AppSync se envían automáticamente a los clientes conectados.
- Los clientes pueden conectarse a la API a través de HTTPS mediante puertos comunes, como el 443.

AWS AppSync facilita la creación de API a partir de bases de datos relacionales existentes. Su utilidad de introspección puede descubrir modelos a partir de tablas de bases de datos y proponer tipos de GraphQL. El asistente Creación de API de la consola de AWS AppSync puede generar una API al instante a partir de una base de datos Aurora MySQL o PostgreSQL. Crea automáticamente tipos y solucionadores de JavaScript para leer y escribir datos.

AWS AppSync ofrece utilidades de JavaScript integradas para simplificar la escritura de instrucciones SQL en los solucionadores. Puede utilizar las plantillas de la etiqueta `sql` de AWS AppSync para instrucciones estáticas con valores dinámicos o las utilidades del módulo `rds` para crear instrucciones mediante programación. Para obtener más información, consulte los orígenes de datos Referencia a la función de solucionador para RDS y los módulos integrados.

Uso de la característica de introspección (consola)

Para ver un tutorial detallado y una guía de introducción, consulte Tutorial: Aurora PostgreSQL sin servidor con API de datos.

La consola AWS AppSync le permite crear una API de GraphQL de AWS AppSync a partir de su base de datos de Aurora existente configurada con la API de datos en solo unos minutos. Esto genera rápidamente un esquema operativo basado en la configuración de la base de datos. Puede usar la API tal cual o desarrollarla para añadir características.

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   - En el Panel, elija Crear API.
2. En Opciones de API, seleccione API de GraphQL, Comenzar con un clúster de Amazon Aurora y, a continuación, Siguiente.
   a. Escriba un Nombre de API. Se utilizará como identificador de la API en la consola.

b. Para obtener los detalles de contacto, puede introducir un punto de contacto para identificar al administrador de la API. Se trata de un campo opcional.

c. En Configuración de API privada, puede habilitar las características de API privadas. Solo se puede acceder a una API privada desde un punto de conexión de VPC (VPCE) configurado. Para obtener más información, consulte API privadas.

   No le recomendamos habilitar esta característica para este ejemplo. Seleccione Siguiente después de revisar sus entradas.

3. En la página Base de datos, elija Seleccionar base de datos.

   a. Es necesario elegir la base de datos de su clúster. El primer paso es elegir la región en la que se encuentra el clúster.

   b. Elija el clúster de Aurora en la lista desplegable. Tenga en cuenta que debe haber creado y habilitado la API de datos correspondiente antes de utilizar el recurso.

   c. A continuación, debe añadir las credenciales de la base de datos al servicio. Esto se hace principalmente con AWS Secrets Manager. Elija la región en la que se encuentra su secreto. Para obtener más información sobre cómo recuperar información del secreto, consulte Buscar secretos o Recuperar secretos.

   d. Añada su secreto en la lista desplegable. Tenga en cuenta que el usuario debe tener permisos de lectura para su base de datos.

4. Seleccione Importar.

   AWS AppSync empezará a realizar la introspección de la base de datos y descubrirá tablas, columnas, claves principales e índices. Comprueba que las tablas descubiertas se pueden admitir en una API de GraphQL. Tenga en cuenta que para permitir la creación de nuevas filas, las tablas necesitan una clave principal, que puede usar varias columnas. AWS AppSync asigna las columnas de la tabla a los campos de texto de la siguiente manera:

Tipo de datos	Tipo de campo
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. Una vez que se haya completado la detección de tablas, la sección Base de datos se rellenará con su información. En la nueva sección Tablas de base de datos, es posible que los datos de la tabla ya estén rellenados y convertidos a un tipo para su esquema. Si no ve algunos de los datos necesarios, puede comprobarlos. Para ello, seleccione Añadir tablas, haga clic en las casillas de verificación de esos tipos en el modal que aparece y, a continuación, seleccione Añadir.

   Para eliminar un tipo de la sección Tablas de bases de datos, haga clic en la casilla de verificación situada junto al tipo que desea eliminar y, a continuación, seleccione Eliminar. Los tipos eliminados se colocarán en el modal Añadir tablas si desea volver a añadirlos más adelante.

   Tenga en cuenta que AWS AppSync utiliza los nombres de las tablas como nombres de tipos, pero puede cambiarles el nombre; por ejemplo, cambiar el nombre de una tabla en plural, como `películas`, por el nombre de tipo `Película`. Para cambiar el nombre de un tipo en la sección Tablas de bases de datos, haga clic en la casilla de verificación del tipo al que desee cambiar el nombre y, a continuación, haga clic en el icono del lápiz en la columna Nombre del tipo.

   Para obtener una vista previa del contenido del esquema en función de sus selecciones, elija Vista previa del esquema. Tenga en cuenta que este esquema no puede estar vacío, por lo que tendrá al menos una tabla convertida en un tipo. Además, este esquema no puede tener un tamaño superior a 1 MB.

   - En Rol de servicio, elija si desea crear un nuevo rol de servicio específicamente para esta importación o utilizar un rol existente.

6. Elija Siguiente.

7. A continuación, elija si desea crear una API de solo lectura (solo consultas) o una API para leer y escribir datos (con consultas y mutaciones). Esta última también admite suscripciones en tiempo real activadas por mutaciones.
8. Elija Siguiente.
9. Revise sus opciones y, a continuación, seleccione Crear API. AWS AppSync creará la API y adjuntará los solucionadores a las consultas y mutaciones. La API generada es totalmente operativa y se puede ampliar según sea necesario.

Uso de la característica de introspección (API)

Puede usar la API de introspección `StartDataSourceIntrospection` para descubrir modelos en su base de datos mediante programación. Para obtener más información sobre el comando, consulte el uso de la API `StartDataSourceIntrospection`.

Para usar `StartDataSourceIntrospection`, proporcione el nombre de recurso de Amazon (ARN), el nombre de base de datos y el ARN del secreto AWS Secrets Manager de su clúster de Aurora. El comando inicia el proceso de introspección. Puede recuperar los resultados con el comando `GetDataSourceIntrospection`. Puede especificar si el comando debe devolver la cadena de lenguaje de definición de almacenamiento (SDL) de los modelos descubiertos. Esto resulta útil para generar una definición de esquema de SDL directamente a partir de los modelos descubiertos.

Por ejemplo, si tiene la siguiente instrucción de lenguaje de definición de datos (DDL) para una tabla `Todos` sencilla:

```
create table if not exists public.todos
(
id serial constraint todos_pk primary key,
description text,
due timestamp,
"createdAt" timestamp default now()
);
```

Comience la introspección con lo siguiente.

```
aws appsync start-data-source-introspection \
  --rds-data-api-config resourceArn=<cluster-arn>,secretArn=<secret-
arn>,databaseName=database
```

A continuación, utilice el comando `GetDataSourceIntrospection` para recuperar el resultado.

```
aws appsync get-data-source-introspection \
  --introspection-id a1234567-8910-abcd-efgh-identifier \
  --include-models-sdl
```

Esto devuelve el resultado siguiente.

```
{
    "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
    }
}
```

Creación de una aplicación cliente

Puedes conectarte a tu API de AWS AppSync GraphQL mediante cualquier cliente de GraphQL, pero te recomendamos encarecidamente el cliente Amplify. Amplify no solo genera automáticamente SDK de cliente fuertemente tipados para su API GraphQL, sino que también ofrece soporte para datos en tiempo real y capacidades de consulta de GraphQL mejoradas en aplicaciones cliente. Para aplicaciones web, Amplify puede crear un JavaScript cliente. Para aquellos que buscan entornos multiplataforma o móviles, Amplify es compatible con Android, iOS y React Native. Para profundizar en la generación de código de cliente para estas plataformas, consulte la documentación de Amplify. Aquí tienes una guía para empezar tu viaje con una aplicación de JavaScript React:

> **Note**
>
> Debe instalar y configurar npm y la CLI de Amazon antes de empezar. Si utilizas el cliente Amplify v6, sigue esta guía.

Primeros pasos:

1. En su máquina local, navegue hasta el directorio del proyecto. Instale la biblioteca Amplify mediante el comando siguiente:

   ```
   npm install aws-amplify
   ```

2. Descargue el archivo de configuración y colóquelo en la carpeta de su proyecto. El archivo de configuración suele contener una `config` variable con algunos ajustes (punto final, región, modo de autorización, etc.) definidos. Por ejemplo, podría tener este aspecto:

   ```
   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. En tu código, importa la biblioteca de Amplify y tu configuración para configurar Amplify:

```
import { Amplify } from 'aws-amplify';
import config from './aws-exports.js';

Amplify.configure(config);
```

   Como alternativa, usa el fragmento en la configuración de tu API para configurar Amplify directamente:

```
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. Con la cadena de herramientas de Amplify, tiene la opción de generar automáticamente las operaciones en función de su esquema, lo que le ahorra el esfuerzo de crear secuencias de comandos manuales. En el directorio raíz de la aplicación, utilice el siguiente comando CLI:

```
npx @aws-amplify/cli codegen add --apiId <id goes here> --region <region goes here>
```

   Esto descargará el esquema de la API y, de forma predeterminada, generará el código auxiliar del cliente en la `src/graphql` carpeta. Después de cada implementación de API, puedes volver a ejecutar el siguiente comando para generar sentencias y tipos de GraphQL actualizados:

```
npx @aws-amplify/cli codegen
```

5. Ahora puedes generar modelos para Android, Swift, Flutter y. JavaScript DataStore Usa el siguiente comando para descargar tu esquema:

```
aws appsync get-introspection-schema --api-id <id goes here> --region <region goes
 here> --format SDL schema.graphql
```

A continuación, ejecuta el siguiente comando desde el directorio raíz de la aplicación:

```
npx @aws-amplify/cli codegen models \
--model-schema schema.graphql \
--target [android|ios|flutter|javascript|typescript] \
--output-dir ./
```

Tutoriales de solucionadores (JavaScript)

Los orígenes de datos y los solucionadores son la forma en que AWS AppSync convierte las solicitudes de GraphQL y obtiene información de sus recursos de AWS. AWS AppSync admite el aprovisionamiento automático y las conexiones con determinados tipos de orígenes de datos. AWS AppSync admite AWS Lambda, Amazon DynamoDB, bases de datos relacionales (Amazon Aurora sin servidor), Amazon OpenSearch Service y puntos de conexión HTTP como orígenes de datos. Puede utilizar una API de GraphQL con los recursos de AWS que ya tiene o crear orígenes de datos y solucionadores. En esta sección recorrerá este proceso en una serie de tutoriales para comprender mejor cómo funcionan los detalles y las opciones de ajuste.

Temas

- Tutorial: Solucionadores de JavaScript de DynamoDB
- Tutorial: Solucionadores de Lambda
- Tutorial: Solucionadores locales
- Tutorial: Combinación de solucionadores de GraphQL
- Tutorial: Solucionadores de Amazon OpenSearch Service
- Tutorial: Solucionadores de transacciones de DynamoDB
- Tutorial: Solucionadores por lotes de DynamoDB
- Tutorial: Solucionadores de HTTP
- Tutorial: Aurora PostgreSQL con API de datos

Tutorial: Solucionadores de JavaScript de DynamoDB

En este tutorial, importará sus tablas de Amazon DynamoDB a AWS AppSync y las conectará para crear una API de GraphQL totalmente funcional mediante solucionadores de canalizaciones de JavaScript que podrá utilizar en su propia aplicación.

Utilizará la consola de AWS AppSync para aprovisionar sus recursos de Amazon DynamoDB, crear sus solucionadores y conectarlos a sus orígenes de datos. También podrá leer y escribir en la base de datos de Amazon DynamoDB a través de instrucciones de GraphQL y suscribirse a datos en tiempo real.

Existen pasos específicos que deben completarse para que las instrucciones de GraphQL se traduzcan a operaciones de Amazon DynamoDB y para que las respuestas se vuelvan a traducir a

GraphQL. En este tutorial se describe el proceso de configuración a través de varios escenarios y patrones de acceso a datos del mundo real.

Creación de la API de GraphQL

Para crear una API de GraphQL en AWS AppSync

1. Abra la consola de AppSync y elija Crear API.
2. Seleccione Diseñar desde cero y elija Siguiente.
3. Llame a su API `PostTutorialAPI` y, a continuación, seleccione Siguiente. Vaya a la página de revisión dejando el resto de las opciones con sus valores predeterminados y seleccione `Create`.

La consola de AWS AppSync crea automáticamente una nueva API de GraphQL. De forma predeterminada, utiliza el modo de autenticación con clave de API. Puede utilizar la consola para configurar el resto de la API de GraphQL y ejecutar consultas en ella durante el resto de este tutorial.

Definición de una API de publicación básica

Ahora que tiene una API de GraphQL, puede configurar un esquema básico que permita la creación, recuperación y eliminación básica de datos de publicaciones.

Para añadir datos a su esquema

1. En su API, seleccione la pestaña Esquema.
2. Crearemos un esquema que defina un tipo `Post` y una operación `addPost` para añadir y obtener objetos `Post`. En la página Esquema, sustituya el contenido por el código siguiente:

```
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. Elija Save Schema (Guardar esquema).

Configuración de la tabla de Amazon DynamoDB

La consola de AWS AppSync puede ayudarle a aprovisionar los recursos de AWS necesarios para almacenar sus propios recursos en una tabla de Amazon DynamoDB. En este paso, creará una tabla de Amazon DynamoDB para almacenar sus publicaciones. También configurará un índice secundario que utilizaremos más adelante.

Para crear una tabla de Amazon DynamoDB

1. En la página Esquema, seleccione Crear recursos.
2. Elija Usar tipo existente y, a continuación, elija el tipo `Post`.
3. En la sección Índices secundarios, elija Añadir índice.
4. Llame al índice `author-index`.
5. Establezca el `Primary key` en `author` y la clave `Sort` en `None`.
6. Deshabilite Generación automática de GraphQL. En este ejemplo, vamos a crear el solucionador nosotros mismos.
7. Seleccione Create (Crear).

Ahora tiene un nuevo origen de datos llamado `PostTable`, que puede ver en Orígenes de datos en la pestaña lateral. Este origen de datos se utiliza para vincular las consultas y mutaciones a la tabla de Amazon DynamoDB.

Configuración del solucionador addPost (PutItem de Amazon DynamoDB)

Una vez que AWS AppSync conoce la tabla de Amazon DynamoDB, puede vincularla a consultas y mutaciones individuales mediante la definición de solucionadores. El primer solucionador que va a crear es el solucionador de canalización `addPost` mediante JavaScript, que le permite crear una publicación en su tabla de Amazon DynamoDB. Un solucionador de canalización tiene los siguientes componentes:

- La ubicación en el esquema de GraphQL donde se asocia el solucionador. En este caso configuramos en el campo `createPost` un solucionador de tipo `Mutation`. El solucionador se invocará cuando el intermediario llame a la mutación `{ addPost(...){...} }`.
- El origen de datos que se va a utilizar para el solucionador. En este caso, queremos utilizar el origen de datos de DynamoDB definido anteriormente para poder añadir entradas en la tabla `post-table-for-tutorial` de DynamoDB.
- El controlador de solicitudes. El controlador de solicitudes es una función que gestiona la solicitud entrante del intermediario y la traduce en instrucciones para que AWS AppSync se ejecute en DynamoDB.
- El controlador de respuestas. El cometido del controlador de respuestas es gestionar la respuesta de DynamoDB y traducirla de nuevo a algo que GraphQL espera. Esto resulta útil si la forma de los datos en DynamoDB es diferente del tipo `Post` en GraphQL. Como este caso sí tienen la misma forma, solo hay que transmitir los datos.

Para configurar el solucionador

1. En su API, seleccione la pestaña Esquema.
2. En el panel Solucionadores, busque el campo `addPost` debajo del tipo `Mutation` y, a continuación, seleccione Asociar.
3. Elija el origen de datos y, a continuación, seleccione Crear.
4. En su editor de código, reemplace el código por este fragmento:

```
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. Seleccione Save.

> **Note**
>
> En este código, se emplean las utilidades del módulo de DynamoDB que permiten crear fácilmente solicitudes de DynamoDB.

AWS AppSync viene con una utilidad para la generación automática de identificadores que se denomina `util.autoId()` y que se utiliza para generar un identificador para la nueva publicación. Si no especifica un identificador, la utilidad lo generará automáticamente.

```
const key = { id: ctx.args.id ?? util.autoId() }
```

Para obtener más información sobre las utilidades disponibles para JavaScript, consulte JavaScript runtime features for resolvers and functions.

Llame a la API para añadir una publicación

Ahora que se ha configurado el solucionador, AWS AppSync puede convertir una mutación `addPost` entrante en una operación `PutItem` de Amazon DynamoDB. Ahora puede ejecutar una mutación para incluir datos en la tabla.

Para ejecutar la operación

1. En su API, seleccione la pestaña Consultas.
2. En el panel Consultas, pegue la mutación siguiente:

```
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. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `addPost`. Los resultados de la publicación que acaba de crear deben aparecer en el panel Resultados a la derecha del panel Consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

La siguiente explicación muestra lo que ocurrió:

1. AWS AppSync ha recibido una solicitud de mutación `addPost`.
2. AWS AppSync ejecuta el controlador de solicitudes del solucionador. La función `ddb.put` crea una solicitud `PutItem` similar a la siguiente:

```
{
  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 AppSync utiliza este valor para generar y ejecutar una solicitud `PutItem` de Amazon DynamoDB.
4. AWS AppSync ha tomado los resultados de la solicitud `PutItem` y los ha convertido de nuevo en tipos de 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. El controlador de respuestas devuelve el resultado inmediatamente (`return ctx.result`).
6. El resultado final es visible en la respuesta de GraphQL.

Configuración del solucionador getPost (GetItem de Amazon DynamoDB)

Ahora que puede añadir datos a la tabla de Amazon DynamoDB, tiene que configurar la consulta `getPost` para poder recuperar los datos de la tabla. Para ello, tiene que configurar otro solucionador.

Para añadir su solucionador

1. En su API, seleccione la pestaña Esquema.
2. En el panel Solucionadores a la derecha, busque el campo `getPost` en el tipo `Query` y, a continuación, seleccione Asociar.
3. Elija el origen de datos y, a continuación, seleccione Crear.
4. En el editor de código, sustituya el código por este fragmento:

```
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. Guarde el solucionador.

> **Note**
>
> En este solucionador, utilizamos una expresión de función de flecha para el controlador de respuestas.

Llame a la API para obtener una publicación

Ahora que está configurado el solucionador, AWS AppSync ya sabe cómo convertir una consulta `getPost` entrante en una operación `GetItem` de Amazon DynamoDB. Ahora puede ejecutar una consulta para recuperar la publicación que ha creado anteriormente.

Para ejecutar su consulta

1. En su API, seleccione la pestaña Consultas.
2. En el panel Consultas, añada el siguiente código y use el identificador que copió después de crear su publicación:

```
query getPost {
  getPost(id: "123") {
    id
    author
    title
```

```
    content
    url
    ups
    downs
    version
  }
}
```

3. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `getPost`. Los resultados de la publicación que acaba de crear deben aparecer en el panel Resultados a la derecha del panel Consultas.
4. Los resultados obtenidos de Amazon DynamoDB deben aparecer en el panel Resultados a la derecha del panel Consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

Como alternativa, vamos a tomar el siguiente ejemplo:

```
query getPost {
  getPost(id: "123") {
    id
    author
    title
  }
}
```

Si la consulta `getPost` solo necesita el `id`, `author` y `title`, puede cambiar la función de solicitud para utilizar expresiones de proyección que especifiquen únicamente los atributos que desee de la

tabla de DynamoDB y evitar la transferencia innecesaria de datos de DynamoDB a AWS AppSync. Por ejemplo, la función de solicitud puede tener un aspecto similar al siguiente fragmento:

```
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
```

También puede usar un selectionSetList con `getPost` para representar la `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
```

Creación de una mutación updatePost (UpdateItem de Amazon DynamoDB)

De momento, ya sabe crear y recuperar objetos `Post` en Amazon DynamoDB. A continuación, va a configurar una nueva mutación para actualizar objetos. En comparación con la mutación `addPost`, que requiere que se especifiquen todos los campos, esta mutación le permite especificar solo los campos que desea cambiar. También introdujo un nuevo argumento `expectedVersion` que permite especificar la versión que se quiere modificar. Va a configurar una condición que garantice que está modificando la versión más reciente del objeto. Para ello, utilizará la operación `UpdateItem` de Amazon DynamoDB.

Para actualizar su solucionador

1. En su API, seleccione la pestaña Esquema.

2. En el panel Schema (Esquema) modifique el tipo `Mutation` para agregar una nueva mutación `updatePost` de este modo:

```
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. Elija Save Schema (Guardar esquema).

4. En el panel Solucionadores de la derecha, busque el campo `updatePost` recién creado en el tipo `Mutation` y, a continuación, seleccione Asociar. Cree su nuevo solucionador con el siguiente fragmento:

```
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. Guarde los cambios que haya realizado.

Este solucionador utiliza `ddb.update` para crear una solicitud `UpdateItem` de Amazon DynamoDB. En lugar de escribir el elemento completo, solo pide a Amazon DynamoDB que actualice determinados atributos. Esto se consigue mediante expresiones de actualización de Amazon DynamoDB.

La función `ddb.update` toma como argumentos una clave y un objeto de actualización. A continuación, se comprueban los valores de los argumentos entrantes. Cuando un valor esté establecido en `null`, utilice la operación `remove` de DynamoDB para indicar que el valor debe eliminarse del elemento de DynamoDB.

También hay una nueva sección `condition`. Una expresión de condición le permite indicar a AWS AppSync y Amazon DynamoDB que la solicitud debe ejecutarse o no en función del estado del objeto que ya se encuentra en Amazon DynamoDB antes de efectuar la operación. En este caso, solo quiere que la solicitud `UpdateItem` se realice si el campo `version` del elemento que hay en Amazon DynamoDB coincide exactamente con el argumento `expectedVersion`. Cuando se actualice el elemento, queremos incrementar el valor de `version`. Esto es fácil de hacer con la función de operación `increment`.

Si desea más información sobre las expresiones de condición, consulte la documentación de expresiones de condición.

Para obtener más información sobre la solicitud `UpdateItem`, consulte la documentación de UpdateItem y la documentación del módulo de DynamoDB.

Para obtener más información sobre el modo de escribir expresiones de actualización, consulte la documentación de DynamoDB UpdateExpressions.

Llame a la API para actualizar una publicación

Vamos a intentar actualizar el objeto `Post` con el nuevo solucionador.

Para actualizar el objeto

1. En su API, seleccione la pestaña Consultas.
2. En el panel Consultas, pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente:

```
mutation updatePost {
  updatePost(
    id:123
    title: "An empty story"
    content: null
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `updatePost`.
4. La publicación actualizada en Amazon DynamoDB debería aparecer en el panel Resultados a la derecha del panel Consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
```

```
}
```

En esta solicitud, solo hemos pedido a AWS AppSync y Amazon DynamoDB que actualicen los campos `title` y `content`. Todos los demás campos se han quedado como estaban (salvo por el incremento del campo `version`). Hemos establecido un nuevo valor en el atributo `title` y hemos eliminado el atributo `content` de la publicación. Los campos `author`, `url`, `ups` y `downs` no se han modificado. Intente ejecutar la solicitud de mutación de nuevo, dejándola exactamente como está. Verá una respuesta parecida a la siguiente:

```
{
  "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)"
    }
  ]
}
```

La solicitud falla porque la expresión de condición se evalúa como `false`:

1. La primera vez que ejecutamos la solicitud, el valor del campo `version` de la publicación en Amazon DynamoDB era 1, que coincidía con el argumento `expectedVersion`. La solicitud se realizó correctamente, lo que significa que el campo `version` se incrementó en Amazon DynamoDB a 2.

2. La segunda vez que ejecutamos la solicitud, el valor del campo `version` de la publicación en Amazon DynamoDB era 2, que no coincide con el argumento `expectedVersion`.

Este método suele denominarse "bloqueo optimista".

Cree mutaciones de votación (UpdateItem de Amazon DynamoDB)

El tipo `Post` contiene los campos `ups` y `downs` para permitir el registro de votos a favor y en contra. Sin embargo, en este momento la API no nos permite hacer nada con ellos. Vamos a añadir una mutación para votar a favor o en contra de las publicaciones.

Para añadir su mutación

1. En su API, seleccione la pestaña Esquema.
2. En el panel Esquema, modifique el tipo `Mutation` y añada la enumeración `DIRECTION` para añadir nuevas mutaciones de votos:

```
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. Elija Save Schema (Guardar esquema).
4. En el panel Solucionadores de la derecha, busque el campo `vote` recién creado en el tipo `Mutation` y, a continuación, seleccione Asociar. Para crear un nuevo solucionador, cree y reemplace el código por el siguiente fragmento:

```
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. Guarde los cambios que haya realizado.

Llame a la API para votar a favor o en contra de una publicación

Ahora que se han configurado nuevos solucionadores, AWS AppSync ya sabe cómo convertir una mutación `upvotePost` o `downvote` entrante en una operación `UpdateItem` de Amazon DynamoDB. Ahora puede ejecutar mutaciones para votar a favor o en contra de la publicación que ha creado anteriormente.

Para ejecutar su mutación

1. En su API, seleccione la pestaña Consultas.
2. En el panel Consultas, pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente:

```
mutation votePost {
  vote(id:123, direction: UP) {
    id
    author
    title
```

```
    content
    url
    ups
    downs
    version
  }
}
```

3. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `votePost`.
4. La publicación actualizada en Amazon DynamoDB debería aparecer en el panel Resultados a la derecha del panel Consultas. Debería parecerse a lo que sigue:

```
{
  "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. Seleccione Ejecutar unas cuantas veces más. Debería ver cómo se incrementan en 1 los campos `ups` y `version` cada vez que ejecuta la consulta.
6. Cambie la consulta para llamarla con una `DIRECTION` diferente.

```
mutation votePost {
  vote(id:123, direction: DOWN) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
```

```
}
```

7. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `votePost`.

   En esta ocasión, debería ver cómo se incrementan en 1 los campos `downs` y `version` cada vez que ejecuta la consulta.

Configuración de un solucionador deletePost (DeleteItem de Amazon DynamoDB)

A continuación, hay que crear una mutación para eliminar una publicación. Para ello, utilizará la operación `DeleteItem` de Amazon DynamoDB.

Para añadir su mutación

1. En el esquema, seleccione la pestaña Esquema.
2. En el panel Esquema modifique el tipo `Mutation` para añadir una nueva mutación `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. Esta vez, ha hecho que el campo `expectedVersion` sea opcional. Elija Guardar esquema.

4. En el panel Solucionadores de la derecha, busque el campo `delete` recién creado en el tipo `Mutation` y, a continuación, seleccione Asociar. Cree un nuevo solucionador con el siguiente código:

```
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**
>
> El argumento `expectedVersion` es opcional. Si el intermediario ha establecido un argumento `expectedVersion` en la solicitud, entonces el controlador de solicitudes añade una condición que solo permitirá que la solicitud `DeleteItem` se atienda cuando el elemento se haya eliminado o cuando el atributo `version` de la publicación en Amazon DynamoDB coincida exactamente con `expectedVersion`. Si se omite, no se especificará ninguna expresión de condición para la solicitud `DeleteItem`. Se realizará correctamente independientemente del valor de `version` o de si el elemento existe o no en Amazon DynamoDB.

Aunque se trate de eliminar un elemento, puede devolver el elemento eliminado, siempre que no se haya eliminado previamente.

Para obtener más información acerca de la solicitud `DeleteItem`, consulte la documentación de DeleteItem.

Llame la API para eliminar una publicación

Ahora que está configurado el solucionador, AWS AppSync ya sabe cómo convertir una mutación `delete` entrante en una operación `DeleteItem` de Amazon DynamoDB. Ahora ya podemos ejecutar una mutación para eliminar datos de la tabla.

Para ejecutar su mutación

1. En su API, seleccione la pestaña Consultas.
2. En el panel Consultas, pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente:

```
mutation deletePost {
  deletePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `deletePost`.
4. La publicación se elimina de Amazon DynamoDB. Tenga en cuenta que AWS AppSync devuelve el valor del elemento que se eliminó de Amazon DynamoDB, que debería aparecer en el panel Resultados a la derecha del panel Consultas. Debería parecerse a lo que sigue:

```
{
  "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. El valor solo se devuelve si esta llamada a `deletePost` es la que realmente lo elimina de Amazon DynamoDB. Vuelva a seleccionar Ejecutar.
6. La llamada se sigue ejecutando con éxito, pero no se devuelve ningún valor:

```
{
  "data": {
    "deletePost": null
  }
}
```

7. Ahora vamos a probar a eliminar una publicación, pero esta vez especificando `expectedValue`. Primero tiene que crear una nueva, porque acaba de eliminar la publicación con la que ha estado trabajando hasta ahora.
8. En el panel Consultas, pegue la mutación siguiente:

```
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. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `addPost`.
10. Los resultados de la publicación que acaba de crear deben aparecer en el panel Resultados a la derecha del panel Consultas. Registre el `id` del objeto recién creado, pues lo necesitará en breve. Debería parecerse a lo que sigue:

```
{
  "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. Ahora, intentemos eliminar esa publicación con un valor ilegal para expectedVersion. En el panel Consultas, pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente:

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 9999
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

12. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `deletePost`. Se devuelve el siguiente resultado:

```
{
  "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. La solicitud ha fallado porque la expresión de condición se evalúa como `false`. El valor `version` de la publicación en Amazon DynamoDB no coincide con el `expectedValue` especificado en los argumentos. El valor actual del objeto se devuelve en el campo `data` de la sección `errors` de la respuesta de GraphQL. Vuelva a intentar la solicitud, pero corrija `expectedVersion`:

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 1
  ) {
    id
    author
    title
    content
```

```
    url
    ups
    downs
    version
  }
}
```

14. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `deletePost`.

    Esta vez la solicitud se realiza correctamente y se devuelve el valor eliminado de Amazon DynamoDB:

```
{
  "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. Vuelva a seleccionar Ejecutar. La llamada se sigue realizando correctamente, pero esta vez no se devuelve ningún valor, ya que la publicación ya se había eliminado en Amazon DynamoDB.

```
{ "data": { "deletePost": null } }
```

Configuración de un solucionador allPost (Scan de Amazon DynamoDB)

Hasta ahora, la API solo es útil si conoce el `id` de cada publicación que desea ver. Añada un nuevo solucionador que devuelva todas las publicaciones en la tabla.

Para añadir su mutación

1. En su API, seleccione la pestaña Esquema.

2. En el panel Schema (Esquema) modifique el tipo `Query` para agregar una nueva consulta `allPost`, de este modo:

```
type Query {
    allPost(limit: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
}
```

3. Añada un nuevo tipo `PaginationPosts`:

```
type PaginatedPosts {
    posts: [Post!]!
    nextToken: String
}
```

4. Elija Save Schema (Guardar esquema).
5. En el panel Solucionadores de la derecha, busque el campo `allPost` recién creado en el tipo `Query` y, a continuación, seleccione Asociar. Cree un nuevo solucionador con el siguiente código:

```
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 };
}
```

   El controlador de solicitudes de este solucionador espera dos argumentos opcionales:

   - `limit`: especifica el número máximo de elementos que se devolverán en una sola llamada.
   - `nextToken`: se usa para recuperar el siguiente conjunto de resultados (mostraremos de dónde proviene el valor de `nextToken` más adelante).

6. Guarde los cambios realizados en el solucionador.

Si desea más información sobre la solicitud `Scan`, consulte la documentación de referencia de Scan.

Llame a la API para escanear todas las publicaciones

Ahora que está configurado el solucionador, AWS AppSync ya sabe cómo convertir una consulta `allPost` entrante en una operación `Scan` de Amazon DynamoDB. Ahora puede recorrer la tabla para obtener todas las publicaciones. Sin embargo, antes de probarlo, debe rellenar la tabla datos, ya que hemos eliminado las publicaciones con las que hemos estado trabajado hasta ahora.

Para añadir y consultar datos

1. En su API, seleccione la pestaña Consultas.
2. En el panel Consultas, pegue la mutación siguiente:

```
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. Seleccione Ejecutar (el botón de reproducción naranja).
4. Ahora recorreremos la tabla obteniendo cinco resultados cada vez. En el panel Consultas, añada la siguiente consulta:

```
query allPost {
  allPost(limit: 5) {
    posts {
      id
      title
```

```
    }
    nextToken
  }
}
```

5. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `allPost`.

   Las cinco primeras publicaciones deben aparecer en el panel Resultados a la derecha del panel Consultas. Debería parecerse a lo que sigue:

```
{
  "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. Ha recibido cinco resultados y un `nextToken`, que permite obtener el siguiente conjunto de resultados. Actualice la consulta `allPost` para incluir el valor de `nextToken` del conjunto de resultados anterior:

```
query allPost {
  allPost(
    limit: 5
    nextToken: "<token>"
  ) {
    posts {
      id
      author
    }
    nextToken
  }
}
```

7. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `allPost`.

   Las cuatro publicaciones restantes deben aparecer en el panel Resultados a la derecha del panel Consultas. No hay ningún `nextToken` en este conjunto, porque ya ha recorrido las nueve publicaciones y no quedan más. Debería parecerse a lo que sigue:

```
{
  "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
    }
```

```
  }
}
```

Configuración de un solucionador allPostsByAuthor (Query de Amazon DynamoDB)

Además de escanear Amazon DynamoDB para obtener todas las publicaciones, también puede consultar Amazon DynamoDB para obtener las publicaciones creadas por un autor determinado. La tabla de Amazon DynamoDB que creó anteriormente ya tiene un `GlobalSecondaryIndex` llamado `author-index` que puede utilizar con una operación `Query` de Amazon DynamoDB para recuperar todas las publicaciones creadas por un autor específico.

Para añadir su consulta

1. En su API, seleccione la pestaña Esquema.
2. En el panel Schema (Esquema) modifique el tipo `Query` para agregar una nueva consulta `allPostsByAuthor`, de este modo:

   ```
   type Query {
       allPostsByAuthor(author: String!, limit: Int, nextToken: String): PaginatedPosts!
       allPost(limit: Int, nextToken: String): PaginatedPosts!
       getPost(id: ID): Post
   }
   ```

   Tenga en cuenta que aquí vuelve a usarse el tipo `PaginatedPosts` que empleamos con la consulta `allPost`.
3. Elija Save Schema (Guardar esquema).
4. En el panel Solucionadores de la derecha, busque el campo `allPostsByAuthor` recién creado en el tipo `Query` y, a continuación, seleccione Asociar. Cree un solucionador con el siguiente fragmento:

   ```
   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 };
}
```

Al igual que el solucionador `allPost`, este solucionador tiene dos argumentos opcionales:

- `limit`: especifica el número máximo de elementos que se devolverán en una sola llamada.
- `nextToken`: recupera el siguiente conjunto de resultados (el valor de `nextToken` puede obtenerse de una llamada anterior).

5. Guarde los cambios realizados en el solucionador.

Si desea más información sobre la solicitud `Query`, consulte la documentación de referencia de Query.

Llame a la API para consultar todas las publicaciones de un autor

Ahora que se ha configurado el solucionador, AWS AppSync ya sabe cómo convertir una mutación `allPostsByAuthor` entrante en una operación `Query` de DynamoDB referida al índice `author-index`. Ahora puede consultar la tabla para recuperar todas las publicaciones de un autor determinado.

Sin embargo, antes de hacerlo, debemos rellenar la tabla con algunas publicaciones más, ya que por el momento todas son del mismo autor.

Para añadir datos y consultas

1. En su API, seleccione la pestaña Consultas.
2. En el panel Consultas, pegue la mutación siguiente:

```
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. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `addPost`.
4. Ahora consultaremos la tabla para que devuelva todas las publicaciones creadas por `Nadia`. En el panel Consultas, añada la siguiente consulta:

```
query allPostsByAuthor {
  allPostsByAuthor(author: "Nadia") {
    posts {
      id
      title
    }
    nextToken
  }
}
```

5. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `allPostsByAuthor`. Todas las publicaciones creadas por `Nadia` deben aparecer en el panel Resultados a la derecha del panel Consultas. Debería parecerse a lo que sigue:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you know...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

6. La paginación funciona con `Query` de igual modo que con `Scan`. Por ejemplo, vamos a buscar todas las publicaciones de `AUTHORNAME` y obtener cinco cada vez.

7. En el panel Consultas, añada la siguiente consulta:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    limit: 5
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

8. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `allPostsByAuthor`. Todas las publicaciones creadas por `AUTHORNAME` deben aparecer en el panel Resultados a la derecha del panel Consultas. Debería parecerse a lo que sigue:

```
{
  "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. Actualice el argumento `nextToken` con el valor devuelto en la consulta anterior, de este modo:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    limit: 5
    nextToken: "<token>"
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

10. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `allPostsByAuthor`. Las demás publicaciones creadas por `AUTHORNAME` deben aparecer en el panel Resultados a la derecha del panel Consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

Uso de conjuntos

Hasta ahora, el tipo `Post` era un objeto clave/valor plano. El solucionador también permite modelar objetos complejos, como conjuntos, listas y mapas. Vamos a actualizar el tipo `Post` para incluir etiquetas. Una publicación puede tener cero o más etiquetas, que se almacenan en DynamoDB como un conjunto de cadenas. También configuraremos algunas mutaciones para añadir y eliminar etiquetas, y una nueva consulta para buscar las publicaciones con una etiqueta concreta.

Para configurar sus datos

1. En su API, seleccione la pestaña Esquema.
2. En el panel Schema (Esquema) modifique el tipo `Post` para agregar un nuevo campo `tags`, de este modo:

```
type Post {
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
  tags: [String!]
}
```

3. En el panel Schema (Esquema) modifique el tipo `Query` para agregar una nueva consulta `allPostsByTag`, de este modo:

```
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. En el panel Schema (Schema), modifique el tipo `Mutation` para agregar las nuevas mutaciones `addTag` y `removeTag`, de este modo:

```
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. Elija Save Schema (Guardar esquema).
6. En el panel Solucionadores de la derecha, busque el campo `allPostsByTag` recién creado en el tipo `Query` y, a continuación, seleccione Asociar. Cree su solucionador con el siguiente fragmento:

```
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. Guarde los cambios que haya realizado en el solucionador.
8. Ahora, haga lo mismo con el campo `addTag` de `Mutation` usando el siguiente fragmento:

> **Note**
>
> Aunque las utilidades de DynamoDB actualmente no admiten operaciones de conjuntos, puede interactuar con los conjuntos creando la solicitud usted mismo.

```
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. Guarde los cambios realizados en el solucionador.
10. Repita esta operación una vez más para el campo `removeTag` de `Mutation` utilizando el siguiente fragmento:

```
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. Guarde los cambios realizados en el solucionador.

Llame a la API para trabajar con etiquetas

Ahora que ha configurado los solucionadores, AWS AppSync sabe cómo convertir solicitudes `addTag`, `removeTag` y `allPostsByTag` entrantes en operaciones `Scan` y `UpdateItem` de DynamoDB. Para probarlo, vamos a seleccionar una de las publicaciones que ha creado anteriormente. Por ejemplo, tomemos una publicación creada por `Nadia`.

Para usar etiquetas

1. En su API, seleccione la pestaña Consultas.
2. En el panel Consultas, añada la siguiente consulta:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "Nadia"
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

3. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `allPostsByAuthor`.
4. Todas las publicaciones de Nadia deben aparecer en el panel Resultados a la derecha del panel Consultas. Debería parecerse a lo que sigue:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you known...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

5. Usemos el que tiene el título The cutest dog in the world. Registre su `id`, ya que vamos a utilizarlo más adelante. Ahora probemos a añadirle la etiqueta `dog`.
6. En el panel Consultas, pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente.

```
mutation addTag {
  addTag(id:10 tag: "dog") {
    id
    title
    tags
  }
}
```

7. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `addTag`. La publicación se actualiza con la nueva etiqueta:

```
{
  "data": {
```

```
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

8. Puede añadir más etiquetas. Actualice la mutación para cambiar el argumento `tag` a `puppy`:

```
mutation addTag {
  addTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

9. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `addTag`. La publicación se actualiza con la nueva etiqueta:

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog",
        "puppy"
      ]
    }
  }
}
```

10. También puede eliminar etiquetas. En el panel Consultas, pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente:

```
mutation removeTag {
  removeTag(id:10 tag: "puppy") {
    id
    title
```

```
    tags
  }
}
```

11. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `removeTag`. La publicación se actualiza y la etiqueta `puppy` se elimina.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

12. También puede buscar todas las publicaciones que tengan una etiqueta. En el panel Consultas, añada la siguiente consulta:

```
query allPostsByTag {
  allPostsByTag(tag: "dog") {
    posts {
      id
      title
      tags
    }
    nextToken
  }
}
```

13. Seleccione Ejecutar (el botón de reproducción naranja) y, a continuación, elija `allPostsByTag`. Se devolverán todas las publicaciones que tenga la etiqueta `dog`, de este modo:

```
{
  "data": {
    "allPostsByTag": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world",
```

```
          "tags": [
            "dog",
            "puppy"
          ]
        }
      ],
      "nextToken": null
    }
  }
}
```

Conclusión

En este tutorial ha creado una API que nos permite manipular objetos `Post` en DynamoDB mediante AWS AppSync y GraphQL.

Para limpiar, puede eliminar la API de GraphQL de AWS AppSync de la consola.

Para eliminar el rol asociado a la tabla de DynamoDB, seleccione el origen de datos en la tabla Orígenes de datos y haga clic en Editar. Anote el valor del rol en Crear o usar un rol existente. Vaya a la consola de IAM para eliminar el rol.

Para eliminar la tabla de DynamoDB, haga clic en el nombre de la tabla en la lista de orígenes de datos. Esto le llevará a la consola de DynamoDB, donde podrá eliminar la tabla.

Tutorial: Solucionadores de Lambda

Puede utilizar AWS Lambda con AWS AppSync para resolver cualquier campo de GraphQL. Por ejemplo, una consulta de GraphQL podría enviar una llamada a una instancia de Amazon Relational Database Service (Amazon RDS), y una mutación de GraphQL podría escribir en un flujo de Amazon Kinesis. En esta sección, veremos cómo puede escribir una función de lambda que ejecute la lógica de negocio en función de la invocación de una operación de campo de GraphQL.

Creación de una función de Lambda

En el siguiente ejemplo se muestra una función de lambda escrita en `Node.js` (tiempo de ejecución: Node.js 18.x) que realiza distintas operaciones en publicaciones de blogs como parte de una aplicación de publicaciones en blogs. Tenga en cuenta que el código debe guardarse en un nombre de archivo con la extensión .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)
  }
}
```

Esta función de Lambda recupera una publicación por identificador, añade una publicación, recupera una lista de publicaciones y recupera publicaciones relacionadas para una publicación determinada.

> **Note**
>
> La función de Lambda utiliza la instrucción `switch` en `event.field` para determinar qué campo se está resolviendo en ese momento.

Cree esta función de Lambda mediante la consola de administración de AWS.

Configure un origen de datos para Lambda

Una vez creada la función de Lambda, vaya a la API de GraphQL en la consola de AWS AppSync y elija la pestaña Orígenes de datos.

Elija Crear origen de datos, introduzca un Nombre de origen de datos fácil de recordar (por ejemplo, **`Lambda`**) y, a continuación, en Tipo de origen de datos, elija Función de AWS Lambda. En Región, elija la misma región que en su función. En ARN de función, elija el nombre de recurso de Amazon (ARN) para la función de Lambda.

Una vez seleccionada la función de Lambda, puede crear un nuevo rol de AWS Identity and Access Management (IAM) (al que AWS AppSync asignará los permisos adecuados) o bien elegir uno ya existente que tenga la política en línea siguiente:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
```

```
            ],
            "Resource": "arn:aws:lambda:REGION:ACCOUNTNUMBER:function/LAMBDA_FUNCTION"
        }
    ]
}
```

También debe configurar una relación de confianza con AWS AppSync para el rol de IAM, de este modo:

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

Cree un esquema de GraphQL

Ahora que el origen de datos está conectado a la función de Lambda, cree un esquema de GraphQL.

En el editor de esquemas de la consola de AWS AppSync, asegúrese de que su esquema coincida con el siguiente:

```
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]
}
```

Configure solucionadores

Ahora que ha registrado un origen de datos de Lambda y un esquema de GraphQL válido, puede conectar sus campos de GraphQL al origen de datos de Lambda utilizando solucionadores.

Creará un solucionador que utilice el tiempo de ejecución JavaScript (`APPSYNC_JS`) de AWS AppSync y que interactúe con las funciones de Lambda. Para obtener más información sobre cómo escribir solucionadores de AWS AppSync y funciones con JavaScript, consulte JavaScript runtime features for resolvers and functions.

Para obtener más información sobre las plantillas de mapeo de Lambda, consulte JavaScript resolver function reference for Lambda.

En este paso, debe asociar un solucionador a la función de Lambda para los siguientes campos: `getPost(id:ID!): Post`, `allPosts: [Post]`, `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!` y `Post.relatedPosts: [Post]`. En el editor Esquema de la consola de AWS AppSync, en el panel Solucionadores, elija Asociar junto al campo `getPost(id:ID!): Post`. Elija el origen de datos de Lambda. A continuación, introduzca el siguiente código:

```
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;
}
```

Este código de solucionador pasa el nombre del campo, la lista de argumentos y el contexto del objeto de origen a la función de Lambda cuando la invoca. Seleccione Save.

Acaba de asociar su primer solucionador. Repita esta operación para el resto de los campos.

Pruebe la API de GraphQL

Ahora que la función de Lambda está conectada a los solucionadores de GraphQL, puede ejecutar algunas mutaciones y consultas con la consola o una aplicación cliente.

A la izquierda de la consola de AWS AppSync, elija la pestaña Consultas y pegue el código siguiente:

Mutación addPost

```
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
    }
}
```

Consulta getPost

```
query GetPost {
    getPost(id: "2") {
        id
        author
```

```
        title
        content
        url
        ups
        downs
    }
}
```

Consulta allPosts

```
query AllPosts {
    allPosts {
        id
        author
        title
        content
        url
        ups
        downs
        relatedPosts {
            id
            title
        }
    }
}
```

Devolución de errores

Cualquier resolución de campo dada puede producir un error. Con AWS AppSync, puede generar errores de los orígenes siguientes:

- Controlador de respuestas del solucionador
- Lambda function

Desde el controlador de respuestas del solucionador

Para generar errores intencionados, puede utilizar el método de la utilidad `util.error`. Toma como argumento un `errorMessage`, un `errorType` y un valor opcional de `data`. El argumento `data` es útil para devolver datos adicionales al cliente cuando se produce un error. El objeto `data` se añade a `errors` en la respuesta final de GraphQL.

En el ejemplo siguiente se muestra cómo utilizarlo en el controlador de respuestas del solucionador de `Post.relatedPosts: [Post]`.

```
// the Post.relatedPosts response handler
export function response(ctx) {
    util.error("Failed to fetch relatedPosts", "LambdaFailure", ctx.result)
    return ctx.result;
}
```

Así se obtiene una respuesta de GraphQL similar a la siguiente:

```
{
    "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"
              }
            ]
        }
    ]
}
```

donde `allPosts[0].relatedPosts` es null debido al error y `errorMessage`, `errorType` y `data` se incluyen en el objeto `data.errors[0]`.

Desde la función de Lambda

AWS AppSync también entiende los errores que produce la función de Lambda. El modelo de programación de Lambda permite generar errores gestionados. Si la función de lambda produce un error, AWS AppSync no puede resolver el campo actual. La respuesta solo incluirá el mensaje de error que devuelva Lambda. Actualmente no es posible devolver datos adicionales al cliente generando un error desde la función de Lambda.

> **Note**
>
> Si su función de Lambda genera un error no gestionado, AWS AppSync utiliza el mensaje de error que Lambda estableció.

La siguiente función de Lambda genera un error:

```
export const handler = async (event) => {
  console.log('Received event {}', JSON.stringify(event, 3))
  throw new Error('I always fail.')
}
```

El error se recibe en el controlador de respuestas. Puede devolverlo en la respuesta de GraphQL añadiendo el error a la respuesta con `util.appendError`. Para ello, cambie el controlador de respuestas de la función de AWS AppSync por el siguiente:

```
// the lambdaInvoke response handler
export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type, result);
```

```
  }
  return result;
}
```

Así se obtiene una respuesta de GraphQL similar a la siguiente:

```
{
  "data": {
    "allPosts": null
  },
  "errors": [
    {
      "path": [
        "allPosts"
      ],
      "data": null,
      "errorType": "Lambda:Unhandled",
      "errorInfo": null,
      "locations": [
        {
          "line": 2,
          "column": 3,
          "sourceName": null
        }
      ],
      "message": "I fail. always"
    }
  ]
}
```

Caso de uso avanzado: agrupación en lotes

La función de lambda de este ejemplo tiene un campo `relatedPosts` que devuelve una lista de publicaciones relacionadas para una publicación determinada. En las consultas del ejemplo, la invocación al campo `allPosts` desde la función de lambda devuelve cinco publicaciones. Dado que hemos especificado que también queremos resolver `relatedPosts` para cada publicación obtenida, la operación del campo `relatedPosts` se invoca cinco veces.

```
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
        }
    }
}
```

Aunque no parezca mucho en este ejemplo concreto, esta sobrecarga compuesta puede perjudicar rápidamente a la aplicación.

Si quisiéramos obtener `relatedPosts` otra vez para todos los elementos de `Posts` en la misma consulta, el número de invocaciones aumentaría exponencialmente.

```
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
            }
        }
    }
}
```

En esta consulta relativamente sencilla, AWS AppSync invocaría la función de Lambda 1 + 5 + 25 = 31 veces.

Se trata de una situación bastante habitual que a menudo se denomina "problema N+1", (en nuestro caso, N = 5) y puede causar un aumento de la latencia y del costo de la aplicación.

Una forma de solucionarlo es agrupar por lotes las solicitudes de solucionador de campo similares. En este ejemplo, en lugar de hacer que la función de Lambda obtenga una lista de publicaciones relacionadas con una publicación individual determinada, hacemos que obtenga una lista de publicaciones relacionadas con un lote de publicaciones dado.

Para demostrarlo, actualicemos el solucionador para que `relatedPosts` gestione la agrupación en lotes.

```
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;
}
```

El código ahora cambia la operación de `Invoke` a `BatchInvoke` cuando el `fieldName` que se está resolviendo es `relatedPosts`. Ahora, habilite la agrupación en lotes en la función en la sección Configuración de la agrupación en lotes. Establezca el tamaño máximo de la agrupación en lotes en 5. Seleccione Save.

Con este cambio, al resolver `relatedPosts`, la función de Lambda recibe lo siguiente como entrada:

```
[
    {
        "field": "relatedPosts",
        "source": {
            "id": 1
```

```
        }
    },
    {
        "field": "relatedPosts",
        "source": {
            "id": 2
        }
    },
    ...
]
```

Cuando se especifica `BatchInvoke` en la solicitud, la función de Lambda recibe una lista de solicitudes y devuelve una lista de resultados.

En concreto, la lista de resultados debe coincidir en tamaño y orden con las entradas de la carga de la solicitud, por lo que AWS AppSync puede hacer coincidir los resultados como corresponda.

En este ejemplo de agrupación en lotes, la función de Lambda devuelve un lote de resultados de este modo:

```
[
    [{"id":"2","title":"Second book"}, {"id":"3","title":"Third book"}],   //
 relatedPosts for id=1
    [{"id":"3","title":"Third book"}]                                      //
 relatedPosts for id=2
]
```

Puede actualizar el código de Lambda para gestionar la agrupación en lotes para `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)
  }
```

```
}
```

Devolución de errores individuales

Los ejemplos anteriores muestran que es posible devolver un único error desde la función de Lambda o generar un error desde su controlador de respuestas. En las invocaciones en lotes, la generación de un error desde la función de Lambda marca como fallido todo el lote. Esto puede ser adecuado en situaciones concretas donde se haya producido un error irrecuperable, como, por ejemplo, un error de conexión a un almacén de datos. Sin embargo, en los casos en los que algunos elementos del lote se ejecutan correctamente y otros fallan, es posible devolver tanto los errores como los datos válidos. Puesto que AWS AppSync requiere que en la respuesta por lotes se enumeren los elementos que coinciden con el tamaño original del lote, debe definir una estructura de datos que pueda diferenciar los datos válidos de un error.

Por ejemplo, si se espera que la función de Lambda devuelva un lote de publicaciones relacionadas, podría optar por devolver una lista de objetos `Response` en la que cada objeto tenga campos opcionales data, errorMessage y errorType. Si el campo errorMessage está presente, significa que se ha producido un error.

El código siguiente muestra cómo podría actualizar la función de 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)
  }
}
```

Actualice el código del solucionador `relatedPosts`:

```
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
  }
}
```

El controlador de respuestas ahora comprueba los errores devueltos por la función de Lambda en las operaciones `Invoke`, comprueba los errores devueltos para elementos individuales de las operaciones `BatchInvoke` y, finalmente, comprueba el `fieldName`. Para `relatedPosts`, la función devuelve `result.data`. Para el resto de los campos, la función simplemente devuelve `result`. Por ejemplo, veamos la siguiente consulta:

```
query AllPosts {
  allPosts {
    id
    title
    content
    url
    ups
    downs
    relatedPosts {
      id
    }
```

```
    author
  }
}
```

Esta consulta devuelve una respuesta de GraphQL similar a la siguiente:

```
{
  "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"
    }
  ]
}
```

Configuración del tamaño máximo de agrupación en lotes

Para configurar el tamaño máximo de agrupación en lotes en un solucionador, utilice el siguiente comando en la AWS 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**
>
> Al proporcionar una plantilla de mapeo de solicitudes, debe usar la operación `BatchInvoke` para usar la agrupación en lotes.

Tutorial: Solucionadores locales

AWS AppSync le permite utilizar orígenes de datos compatibles (AWS Lambda, Amazon DynamoDB o Amazon OpenSearch Service) para realizar diversas operaciones. Sin embargo, en determinados casos, es posible que no sea necesario realizar una llamada a un origen de datos admitido.

Aquí es donde un solucionador local es útil. En lugar de llamar a un origen de datos remoto, el solucionador local simplemente reenvía los resultados del controlador de solicitudes al controlador de respuestas. La resolución de campo no sale de AWS AppSync.

Los solucionadores locales son útiles en una gran cantidad de situaciones. El caso de uso más habitual consiste en publicar notificaciones sin activar una llamada de origen de datos. Para demostrar este caso de uso, vamos a crear una aplicación pub/sub en la que los usuarios puedan publicar mensajes y suscribirse a ellos. Este ejemplo utiliza suscripciones, de modo que si no está familiarizado con las suscripciones, puede seguir el tutorial de datos en tiempo real.

Creación de la aplicación pub/sub

Primero, cree una API de GraphQL vacía con la opción Diseñar desde cero y configure los detalles opcionales al crear la API de GraphQL.

En nuestra aplicación pub/sub, los clientes pueden suscribirse y publicar mensajes. Cada mensaje publicado incluye un nombre y datos. Añada lo siguiente al esquema:

```
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"])
}
```

A continuación, asociaremos un solucionador al campo `Mutation.publish`. En el panel Solucionadores situado junto al panel Esquema, busque el tipo `Mutation`, después el campo `publish(...): Channel` y, a continuación, haga clic en Asociar.

Cree un origen de datos de tipo Ninguno y asígnele el nombre PageDataSource. Asócielo al solucionador.

Añada su implementación del solucionador mediante el siguiente fragmento de código:

```
export function request(ctx) {
  return { payload: ctx.args };
}

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

Asegúrese de crear el solucionador y guardar los cambios realizados.

Envíe y suscríbase a mensajes

Para que los clientes reciban mensajes, primero deben estar suscritos a una bandeja de entrada.

En el panel Consultas, ejecute la suscripción `SubscribeToData`:

```
subscription SubscribeToData {
    subscribe(name:"channel") {
        name
        data
    }
}
```

El suscriptor recibirá mensajes siempre que se invoque la mutación `publish`, pero solo cuando el mensaje se envíe a la suscripción `channel`. Probemos esto en el panel Consultas. Mientras la suscripción siga ejecutándose en la consola, abra otra consola y ejecute la siguiente solicitud en el panel Consultas:

> **Note**
>
> En este ejemplo, utilizamos cadenas JSON válidas.

```
mutation PublishData {
    publish(data: "{\"msg\": \"hello world!\"}", name: "channel") {
        data
        name
    }
}
```

El resultado tendrá este aspecto:

```
{
  "data": {
    "publish": {
      "data": "{\"msg\":\"hello world!\"}",
      "name": "channel"
    }
  }
}
```

Acabamos de demostrar el uso de solucionadores locales publicando un mensaje y recibiéndolo sin salir del servicio de AWS AppSync.

Tutorial: Combinación de solucionadores de GraphQL

Los solucionadores y los campos de un esquema de GraphQL mantienen una relación 1:1 con un alto grado de flexibilidad. Debido a que un origen de datos se configura en un solucionador independientemente de un esquema, usted tiene la capacidad de resolver o manipular sus tipos de GraphQL a través de diferentes orígenes de datos, lo que le permite mezclar y combinar un esquema para satisfacer mejor sus necesidades.

Los siguientes escenarios muestran cómo mezclar y hacer corresponder los orígenes de datos en su esquema. Antes de empezar, debe estar familiarizado con la configuración de orígenes de datos y solucionadores para AWS Lambda Amazon DynamoDB y Amazon OpenSearch Service.

Esquema de ejemplo

El siguiente esquema tiene un tipo `Post` con tres `Query` y operaciones `Mutation` cada una:

```
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
}
```

En este ejemplo, tendría un total de seis solucionadores y cada uno necesitaría un origen de datos. Una forma de resolver este problema sería conectarlos a una única tabla de Amazon DynamoDB, denominada `Posts`, en la que el campo `AllPost` ejecuta un análisis y el campo `searchPosts` ejecuta una consulta (consulte JavaScript resolver function reference for DynamoDB). Sin embargo, no está limitado a Amazon DynamoDB; existen diferentes orígenes de datos, como Lambda u OpenSearch Service, para cumplir con los requisitos de su empresa.

Modificación de datos mediante solucionadores

Es posible que tenga que devolver los resultados desde una base de datos de terceros que no sea compatible directamente con los orígenes de datos de AWS AppSync. También es posible que tenga que realizar modificaciones complejas en los datos antes de devolverlos a los clientes de la API. Esto podría deberse a un formato incorrecto de los tipos de datos, como diferencias de fecha y hora en los clientes, o a la gestión de problemas de compatibilidad con versiones anteriores. En este caso, la solución adecuada es conectar las funciones de AWS Lambda como un origen de datos a la API de AWS AppSync. A título ilustrativo, en el siguiente ejemplo, una función de AWS Lambda manipula datos obtenidos de un almacén de datos externo:

```
export const handler = (event, context, callback) => {
    // fetch data
    const result = fetcher()

    // apply complex business logic
    const data = transform(result)

    // return to AppSync
    return data
};
```

Se trata de una función de Lambda perfectamente válida y puede asociarse al campo `AllPost` del esquema de GraphQL para que cualquier consulta que devuelva todos los resultados obtenga números aleatorios para las subidas/bajadas.

DynamoDB y OpenSearch Service

En algunas aplicaciones, puede realizar mutaciones o consultas de búsqueda sencilla en DynamoDB y tener un proceso en segundo plano para transferir documentos a OpenSearch Service. Podría

simplemente asociar el solucionador `searchPosts` al origen de datos de OpenSearch Service y devolver los resultados de la búsqueda (a partir de los datos originados en DynamoDB) mediante una consulta GraphQL. Esto puede ser extremadamente útil al añadir operaciones de búsqueda avanzada a sus aplicaciones, como palabras clave, coincidencias parciales o incluso búsquedas geoespaciales. La transferencia de datos desde DynamoDB puede realizarse mediante un proceso ETL o, alternativamente, puede transmitir desde DynamoDB mediante Lambda.

Para empezar con estos orígenes de datos específicos, consulte nuestros tutoriales de DynamoDB y Lambda.

Por ejemplo, con el esquema de nuestro tutorial anterior, la siguiente mutación añade un elemento a 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
  }
}
```

Esto escribe datos en DynamoDB, que transmite mediante Lambda a Amazon OpenSearch Service y que luego se utilizan para buscar publicaciones por distintos campos. Por ejemplo, como los datos se encuentran en Amazon OpenSearch Service, puede buscar los campos de autor o contenido con texto libre, incluso con espacios, del siguiente modo:

```
query searchName{
    searchAuthor(name:"   Nadia   "){
        id
        title
```

```
        content
    }
}

---------- or ----------

query searchContent{
    searchContent(text:"test"){
        id
        title
        content
    }
}
```

Como los datos se escriben directamente en DynamoDB, aún puede realizar operaciones eficientes de búsqueda de lista o de elemento en la tabla con las consultas `allPost{...}` y `getPost{...}`. Esta pila utiliza el siguiente código de ejemplo para los flujos de DynamoDB:

> **Note**
>
> Este código de Python es un ejemplo y no está diseñado para usarse en código de producción.

```
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.'
```

Puede utilizar flujos de DynamoDB para asociarlo a una tabla de DynamoDB con la clave principal `id`, y cualquier cambio en el origen de DynamoDB se transmitirá al dominio de OpenSearch Service. Para obtener más información sobre cómo configurarlo, consulte la documentación de flujos de DynamoDB.

Tutorial: Solucionadores de Amazon OpenSearch Service

AWS AppSync admite el uso de Amazon OpenSearch Service desde dominios que ha aprovisionado en su cuenta de AWS, siempre que no se encuentren dentro de una VPC. Después de aprovisionar los dominios, puede conectarse a ellos con un origen de datos. En ese momento, puede configurar un solucionador en el esquema para que realice operaciones de GraphQL como consultas, mutaciones y suscripciones. Este tutorial le guiará a lo largo de algunos ejemplos comunes.

Para obtener más información, consulte JavaScript resolver function reference for OpenSearch.

Cree un nuevo dominio de OpenSearch Service

Para comenzar este tutorial, es necesario disponer de un dominio de OpenSearch Service. Si todavía no tiene uno, puede utilizar la siguiente muestra. Tenga en cuenta que crear un dominio de OpenSearch Service puede tardar hasta 15 minutos, por lo que deberá esperar para poder integrarlo con un origen de datos de AWS AppSync.

```
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
```

Como alternativa, puede iniciar la siguiente pila de AWS CloudFormation en la región EE. UU. Oeste 2 (Oregón) de su cuenta de AWS:

Configure un origen de datos para OpenSearch Service

Después de crear el dominio de OpenSearch Service, vaya a la API de GraphQL para AWS AppSync y elija la pestaña Orígenes de datos. Seleccione Crear origen de datos e introduzca un nombre fácil de recordar para el origen de datos, como "*oss*". A continuación, elija Dominio de Amazon OpenSearch en Tipo de origen de datos y elija la región que corresponda. En la lista debe aparecer el dominio de OpenSearch Service. Una vez seleccionado, puede crear un nuevo rol y AWS AppSync le asignará los permisos adecuados. También puede elegir un rol existente que tenga la siguiente política en línea:

```
{
    "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/*"
            ]
        }
    ]
}
```

También debe configurar una relación de confianza con AWS AppSync para ese rol:

```
{
```

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

Además, el dominio de OpenSearch Service tiene su propia política de acceso, que puede modificar en la consola de Amazon OpenSearch Service. Debe añadir una política similar a la que aparece a continuación, con las acciones y recursos adecuados para el dominio de OpenSearch Service. Tenga en cuenta que el Principal será el rol del origen de datos de AWS AppSync que, si deja que la consola lo cree, podrá encontrar en la consola de 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/*"
        }
    ]
}
```

Conexión de un solucionador

Ahora que el origen de datos está conectado a su dominio de OpenSearch Service, puede conectarlo también al esquema de GraphQL con un solucionador, como se muestra en el ejemplo siguiente:

```
 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
}
```

Observe que hay un tipo `Post` definido por el usuario con un campo `id`. En los siguientes ejemplos, supondremos que existe un proceso (que se puede automatizar) para incluir este tipo en el dominio de OpenSearch Service, lo que lo mapea a la ruta raíz de `/post/_doc`, donde `post` es el índice. A partir de esta ruta raíz, puede realizar búsquedas de documentos individuales, búsquedas de comodín con `/id/post*` o búsquedas en varios documentos con la ruta `/post/_search`. Por ejemplo, si tiene otro tipo llamado `User`, puede indexar documentos bajo un nuevo índice llamado `user`, y luego realizar búsquedas con una ruta de `/user/_search`.

En el editor Esquema de la consola de AWS AppSync, modifique el esquema `Posts` anterior para que incluya una consulta `searchPosts`:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  searchPosts: [Post]
}
```

Guarde el esquema. En el panel Solucionadores, busque `searchPosts` y seleccione Asociar. Elija el origen de datos de OpenSearch Service y guarde el solucionador. Actualice el código del solucionador mediante el siguiente fragmento:

```
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)
}
```

Esto supone que el esquema anterior contiene documentos que se han indexado en OpenSearch Service en el campo `post`. Si estructura los datos de manera diferente, tendrá que realizar una actualización como corresponda.

Modificación de las búsquedas

El controlador de solicitudes del solucionador anterior realiza una consulta sencilla para todos los registros. Supongamos que desea buscar por un autor específico. Además, supongamos que desea que ese autor sea un argumento definido en la consulta de GraphQL. En el editor Esquema de la consola de AWS AppSync, añada una consulta `allPostsByAuthor`:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  allPostsByAuthor(author: String!): [Post]
  searchPosts: [Post]
}
```

En el panel Solucionadores, busque `allPostsByAuthor` y seleccione Asociar. Elija el origen de datos de OpenSearch Service y utilice el código siguiente:

```
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)
}
```

Observe que body se llena con una consulta de término para el campo `author`, que se pasa desde el cliente como un argumento. Si lo desea, puede usar información previamente rellenada, como texto estándar.

Adición de datos a OpenSearch Service

Puede ser conveniente añadir datos al dominio de OpenSearch Service como resultado de una mutación de GraphQL. Se trata de un eficaz mecanismo para realizar búsquedas y para otros fines. Dado que puede utilizar suscripciones de GraphQL para que los datos se obtengan en tiempo real, esto puede servir como mecanismo para avisar a los clientes de actualizaciones de datos en el dominio de OpenSearch Service.

Vuelva a la página Esquema de la consola de AWS y seleccione Asociar para la mutación `addPost()`. Seleccione de nuevo el origen de datos de OpenSearch Service y utilice el código siguiente:

```
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
```

```
}
```

Como antes, este es un ejemplo de cómo pueden estar estructurados los datos. Si los nombres de campos o índices son distintos, debe actualizar la `path` y `body`. En este ejemplo, también se muestra cómo utilizar `context.arguments`, que también se puede escribir como `ctx.args`, en el controlador de solicitudes.

Recuperación de un solo documento

Por último, si desea utilizar la consulta `getPost(id:ID)` en su esquema para obtener un documento individual, encuentre esta consulta en el editor Esquema de la consola de AWS AppSync y seleccione Asociar. Seleccione de nuevo el origen de datos de OpenSearch Service y utilice el código siguiente:

```
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
}
```

Ejecución de consultas y mutaciones

Ahora puede realizar operaciones de GraphQL referidas al dominio de OpenSearch Service. Vaya a la pestaña Consultas de la consola de AWS y añada un nuevo registro:

```
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
       )
}
```

Verá el resultado de la mutación a la derecha. Del mismo modo, ahora puede ejecutar una consulta `searchPosts` en el dominio de OpenSearch Service:

```
query search {
    searchPosts {
        id
        title
        author
        content
    }
}
```

Prácticas recomendadas

- OpenSearch Service se debe utilizar para consultar datos, no como base de datos principal. Puede ser útil emplear OpenSearch Service junto con Amazon DynamoDB como se indica en Combinación de solucionadores de GraphQL.
- Solo debe conceder acceso al dominio permitiendo que el rol de servicio de AWS AppSync tenga acceso al clúster.
- Puede comenzar con un pequeño desarrollo, con el clúster de menor costo y, a continuación, pasar a un clúster de mayor tamaño con alta disponibilidad (HA) al pasar a producción.

Tutorial: Solucionadores de transacciones de DynamoDB

AWS AppSync admite el uso de operaciones de transacciones de Amazon DynamoDB en una o más tablas de una sola región. Las operaciones admitidas son `TransactGetItems` y `TransactWriteItems`. Estas características de AWS AppSync le permiten realizar tareas como las siguientes:

- Transferir una lista de claves en una sola consulta y devolver los resultados desde una tabla
- Leer registros desde una o varias tablas en una única consulta
- Escribir registros en transacciones en una o más tablas en régimen de todo o nada
- Ejecutar transacciones cuando se cumplan algunas condiciones

Permisos

Al igual que con otros solucionadores, debe crear un origen de datos en AWS AppSync y crear un rol o utilizar uno existente. Dado que las operaciones de transacciones requieren diferentes permisos para las tablas de DynamoDB, debe conceder los permisos de rol configurados para las acciones de lectura o escritura:

```
{
    "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**
>
> Los roles están vinculados a orígenes de datos de AWS AppSync y los solucionadores de los campos se invocan con referencia a un origen de datos. Los orígenes de datos configurados para recuperar información de DynamoDB solo tienen especificada una tabla para que la configuración siga siendo sencilla. Por lo tanto, al realizar una operación de transacciones en varias tablas con un único solucionador, que es una tarea más avanzada, debe conceder al rol de ese origen de datos acceso a cualquier tabla con la que el solucionador vaya a interactuar. Esto se hace en el campo Resource (Recurso) de la política de IAM anterior. La configuración de las tablas en las que se realizan llamadas de transacciones se lleva a cabo en el código del solucionador, que se describe a continuación.

Origen de datos

En aras de la simplicidad, vamos a utilizar el mismo origen de datos para todos los solucionadores que se utilizan en este tutorial.

Tendremos dos tablas denominadas savingAccounts y checkingAccounts, ambas con la clave de partición `accountNumber`, y una tabla transactionHistory con la clave de partición `transactionId`. Puede utilizar los siguientes comandos de la CLI para crear las tablas. Asegúrese de reemplazar `region` por la región.

Con la 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
```

En la consola de AWS AppSync, en pestaña Orígenes de datos, cree un nuevo origen de datos de DynamoDB y llámelo TransactTutorial. Seleccione savingAccounts en la tabla (aunque la tabla específica no importa cuando se utilizan transacciones). Elija crear un nuevo rol y el origen de datos. Puede revisar la configuración del origen de datos para ver el nombre del rol generado. En la consola de IAM, puede añadir una política en línea que permita que el origen de datos interactúe con todas las tablas.

Sustituya `region` y `accountID` por su región y su identificador de cuenta:

```
{
    "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/*"
            ]
        }
    ]
}
```

Transacciones

Para este ejemplo, el contexto es una transacción bancaria clásica, en la que usaremos `TransactWriteItems` para:

- Transferir dinero de cuentas de ahorro a cuentas corrientes
- Generar nuevos registros de transacciones para cada transacción

Y, a continuación, usaremos `TransactGetItems` para recuperar los detalles de las cuentas de ahorro y las cuentas corrientes.

Definimos nuestro esquema GraphQL de la siguiente manera:

```
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: rellenar cuentas

Para transferir dinero entre cuentas, tenemos que rellenar la tabla con los detalles. Para ello, utilizaremos la operación `Mutation.populateAccounts` de GraphQL.

En la sección Esquema, haga clic en Asociar junto a la operación `Mutation.populateAccounts`. Elija el origen de datos `TransactTutorial` y, a continuación, seleccione Crear.

Ahora utilice el siguiente código:

```
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 }
}
```

Guarde el solucionador y vaya a la sección Consultas de la consola de AppSync AWS para rellenar las cuentas.

Ejecute la mutación siguiente:

```
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
    }
  }
}
```

Hemos rellenado tres cuentas de ahorro y tres cuentas corrientes en una mutación.

Utilice la consola de DynamoDB para validar que los datos se muestren en las tablas savingAccounts y checkingAccounts.

TransactWriteItems: transferir dinero

Asocie un solucionador a la mutación `transferMoney` con el siguiente código. Para cada transferencia, necesitamos un modificador de éxito tanto en la cuenta corriente como en la de ahorros, y debemos hacer un seguimiento de la transferencia en las transacciones.

```
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 }
}
```

Ahora, vaya a la sección Consultas de la consola de AWS AppSync y ejecute la mutación transferMoney de la siguiente manera:

```
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
    }
  }
}
```

Hemos enviado tres transacciones bancarias en una mutación. Utilice la consola de DynamoDB para validar que los datos se muestren en las tablas savingAccounts, checkingAccounts y transactionHistory.

TransactGetItems: recuperar cuentas

Con el fin de recuperar los detalles de las cuentas de ahorro y corriente en una sola solicitud transaccional, vamos a asociar un solucionador a la operación `Query.getAccounts` de GraphQL en nuestro esquema. Seleccione Asociar y elija el origen de datos `TransactTutorial` creado al inicio del tutorial. Utilice el siguiente código:

```
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 }
}
```

Guarde el solucionador y vaya a las secciones Consultas de la consola de AWS AppSync. Para recuperar las cuentas de ahorro y corriente, ejecute la siguiente consulta:

```
query getAccounts {
  getAccounts(
    savingAccountNumbers: ["1", "2", "3"],
    checkingAccountNumbers: ["1", "2"]
  ) {
    savingAccounts {
      accountNumber
      username
      balance
    }
    checkingAccounts {
      accountNumber
      username
      balance
    }
  }
}
```

Hemos demostrado el uso de operaciones de transacciones de DynamoDB mediante AWS AppSync.

Tutorial: Solucionadores por lotes de DynamoDB

AWS AppSync admite el uso de operaciones por lotes de Amazon DynamoDB en una o más tablas de una sola región. Las operaciones admitidas son `BatchGetItem`, `BatchPutItem` y `BatchDeleteItem`. Estas características de AWS AppSync le permiten realizar tareas como las siguientes:

- Transferir una lista de claves en una sola consulta y devolver los resultados desde una tabla
- Leer registros desde una o varias tablas en una única consulta
- Escribir registros de forma masiva en una o varias tablas
- Escribir o eliminar condicionalmente registros en varias tablas que pueden tener una relación

Las operaciones por lotes en AWS AppSync tienen dos diferencias fundamentales con respecto a las operaciones que no se realizan por lotes:

- El rol del origen de datos debe tener permisos para todas las tablas a las que el solucionador obtiene acceso.
- La especificación de tabla de un solucionador forma parte del objeto de solicitud.

Lotes en una única tabla

Para empezar, vamos a crear una nueva API de GraphQL. En la consola de AWS AppSync, seleccione Crear API, API de GraphQL y Diseñar desde cero. Asigne a su API el nombre `BatchTutorial API`, elija Siguiente y, en el paso Especificar recursos de GraphQL, elija Crear recursos de GraphQL más adelante. Luego, haga clic en Siguiente. Revise sus detalles y cree la API. Vaya a la página Esquema y pegue el siguiente esquema, teniendo en cuenta que, para la consulta, pasaremos una lista de identificadores:

```
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]
}
```

Guarde el esquema y elija Crear recursos en la parte superior de la página. Elija Usar tipo existente y, a continuación, elija el tipo `Post`. Llame a su tabla `Posts`. Asegúrese de que Clave principal está establecido en `id`, desmarque Generar GraphQL automáticamente (usted proporcionará su propio código) y seleccione Crear. Para empezar, AWS AppSync crea una nueva tabla de DynamoDB y un origen de datos conectado a la tabla con los roles adecuados. Sin embargo, todavía hay un par de permisos que debe añadir al rol. Vaya a la página Orígenes de datos y elija el nuevo origen de datos. En Seleccionar un rol existente, verá que se ha creado automáticamente un rol para la tabla. Tome nota del rol (debería tener un aspecto similar a `appsync-ds-ddb-aaabbbcccddd-Posts`) y, a continuación, vaya a la consola de IAM (https://console.aws.amazon.com/iam/). En la consola de IAM, seleccione Roles y, a continuación, seleccione su rol en la tabla. En su rol, en Políticas de permisos, haga clic en el botón + situado junto a la política (debe tener un nombre similar al del rol). Seleccione Editar en la parte superior del menú desplegable cuando aparezca la política. Debe añadir permisos por lotes a su política, específicamente `dynamodb:BatchGetItem` y `dynamodb:BatchWriteItem`. Tendrá un aspecto similar al siguiente:

```
{
    "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:…"
            ]
        }
    ]
}
```

Seleccione Siguiente y, a continuación, Guardar cambios. Su política debería permitir ahora el procesamiento por lotes.

De vuelta a la consola de AWS AppSync, vaya a la página Esquema y seleccione Asociar junto al campo `Mutation.batchAdd`. Cree su solucionador utilizando la tabla `Posts` como origen de datos. En el editor de código, sustituya los controladores por el siguiente fragmento. Este fragmento toma automáticamente cada elemento con el tipo de `input PostInput` de GraphQL y crea un mapa, lo que es necesario para la operación `BatchPutItem`:

```
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;
}
```

Vaya a la página Consultas de la consola de AWS AppSync y ejecute la siguiente mutación `batchAdd`:

```
mutation add {
    batchAdd(posts:[{
            id: 1 title: "Running in the Park"},{
            id: 2 title: "Playing fetch"
        }]){
```

```
            id
            title
    }
}
```

Debería ver los resultados impresos en la pantalla; para validarlo, consulte la consola de DynamoDB para buscar los valores escritos en la tabla `Posts`.

A continuación, repita el proceso para asociar un solucionador, pero para el campo `Query.batchGet`, utilice la tabla `Posts` como origen de datos. Sustituya los controladores por el siguiente código. Esto toma automáticamente cada elemento del tipo de GraphQL `ids:[]` y crea un mapa que es necesario para la operación `BatchGetItem`:

```
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;
}
```

Ahora, vuelva a la página Consultas de la consola de AWS AppSync y ejecute la siguiente consulta `batchGet`:

```
query get {
    batchGet(ids:[1,2,3]){
        id
        title
    }
```

```
}
```

Esto debe devolver los resultados para los dos valores `id` que ha añadido anteriormente. Observe que se devuelve un valor `null` para el `id` con el valor `3`. Esto se debe a que aún no hay ningún registro en la tabla `Posts` con ese valor. Observe también que AWS AppSync devuelve los resultados en el mismo orden en que se pasaron las claves a la consulta, lo que es una característica automática adicional de AWS AppSync. Por lo tanto, si cambia a `batchGet(ids:[1,3,2])`, verá que el orden cambia. También sabrá por qué `id` devuelve un valor `null`.

Por último, asocie un solucionador más al campo `Mutation.batchDelete` utilizando la tabla `Posts` como origen de datos. Sustituya los controladores por el siguiente código. Esto toma automáticamente cada elemento del tipo de GraphQL `ids:[]` y crea un mapa que es necesario para la operación `BatchGetItem`:

```
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;
}
```

Ahora, vuelva a la página Consultas de la consola de AWS AppSync y ejecute la siguiente mutación `batchDelete`:

```
mutation delete {
    batchDelete(ids:[1,2]){ id }
}
```

Ahora se eliminarán los registros con `id` `1` y `2`. Si vuelve a ejecutar la consulta `batchGet()` anterior, devolverá `null`.

Lotes en varias tablas

AWS AppSync también permite realizar operaciones por lotes en varias tablas. Vamos a crear una aplicación más compleja. Imagine que queremos crear una aplicación de salud para mascotas, con sensores que comunican la ubicación y temperatura corporal de la mascota. Los sensores funcionan con pilas e intentan conectarse a la red cada pocos minutos. Cuando un sensor establece una conexión, envía sus lecturas a nuestra API de AWS AppSync. A continuación, los disparadores analizan los datos para presentar un panel al propietario de la mascota. Concentrémonos en representar las interacciones entre el sensor y el almacén de datos de backend.

En la consola de AWS AppSync, seleccione Crear API, API de GraphQL y Diseñar desde cero. Asigne a su API el nombre `MultiBatchTutorial API`, elija Siguiente y, en el paso Especificar recursos de GraphQL, elija Crear recursos de GraphQL más adelante. Luego, haga clic en Siguiente. Revise sus detalles y cree la API. Vaya a la página Esquema y pegue y guarde el siguiente esquema:

```
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
}
```

Necesitamos crear dos tablas de DynamoDB:

- `locationReadings` almacenará las lecturas de ubicación de los sensores.
- `temperatureReadings` almacenará las lecturas de temperatura de los sensores.

Ambas tablas compartirán la misma estructura de clave principal: `sensorId (String)` como clave de partición y `timestamp (String)` como clave de clasificación.

Elija Crear recursos en la parte superior de la página. Elija Usar tipo existente y, a continuación, elija el tipo `locationReadings`. Llame a su tabla `locationReadings`. Asegúrese de que Clave principal está configurada en `sensorId` y la clave de clasificación en`timestamp`. Desmarque Generar GraphQL automáticamente (proporcionará su propio código) y seleccione Crear. Repita este proceso para `temperatureReadings` utilizando `temperatureReadings` como tipo y nombre de la tabla. Utilice las mismas claves que antes.

Las nuevas tablas contendrán los roles generados automáticamente. Todavía hay que añadir un par de permisos a esos roles. Vaya a la página Orígenes de datos y elija `locationReadings`. En Seleccione un rol existente, puede ver el rol. Tome nota del rol (debería tener un aspecto similar a `appsync-ds-ddb-aaabbbcccddd-locationReadings`) y, a continuación, vaya a la consola de IAM (https://console.aws.amazon.com/iam/). En la consola de IAM, seleccione Roles y, a continuación, seleccione su rol en la tabla. En su rol, en Políticas de permisos, haga clic en el botón + situado junto a la política (debe tener un nombre similar al del rol). Seleccione Editar en la parte superior del menú desplegable cuando aparezca la política. Debe añadir permisos a esta política. Tendrá un aspecto similar al siguiente:

```
{
    "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/*"
            ]
        }
    ]
}
```

Seleccione Siguiente y, a continuación, Guardar cambios. Repita este proceso para la el origen de datos `temperatureReadings` utilizando el mismo fragmento de política anterior.

BatchPutItem: registrar lecturas del sensor

Nuestros sensores tiene que poder enviar sus lecturas cuando se conectan a Internet. El campo de GraphQL `Mutation.recordReadings` es la API que utilizarán para hacerlo. Necesitamos añadir un solucionador a este campo.

En la página Esquema de la consola de AWS AppSync, seleccione Asociar junto al campo `Mutation.recordReadings`. En la siguiente pantalla, cree su solucionador utilizando la tabla `locationReadings` como origen de datos.

Tras crear el solucionador, sustituya los controladores por el siguiente código en el editor. Esta operación `BatchPutItem` nos permite especificar varias tablas:

```
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
}
```

Con operaciones por lotes, la invocación puede devolver tanto errores como resultados. Por lo tanto, podemos realizar algún control de errores adicional.

> **Note**
>
> El uso de `utils.appendError()` es similar al de `util.error()`, con la gran diferencia de que no interrumpe la evaluación del controlador de solicitudes o respuestas. En su lugar, indica que hubo un error con el campo, pero permite al controlador evaluar la plantilla y, por tanto, devolver los datos al intermediario. Recomendamos que utilice `utils.appendError()` cuando la aplicación tenga que devolver resultados parciales.

Guarde el solucionador y vaya a la página Consultas de la consola de AWS AppSync. Ahora podemos enviar algunas lecturas de los sensores.

Ejecute la mutación siguiente:

```
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
    }
  }
}
```

Enviamos diez lecturas de sensores en una mutación con lecturas repartidas en dos tablas. Utilice la consola de DynamoDB para validar que los datos aparecen en las tablas `locationReadings` y `temperatureReadings`.

BatchDeleteItem: eliminar lecturas de los sensores

Del mismo modo, también es necesario poder eliminar lotes de lecturas de sensores. Vamos a utilizar el campo de GraphQL `Mutation.deleteReadings` para este fin. En la página Esquema de la consola de AWS AppSync, seleccione Asociar junto al campo `Mutation.deleteReadings`. En la siguiente pantalla, cree su solucionador utilizando la tabla `locationReadings` como origen de datos.

Tras crear el solucionador, sustituya los controladores del editor de código por el siguiente fragmento. En este solucionador, utilizamos un mapeador de funciones auxiliar que extrae `sensorId` y `timestamp` de las entradas proporcionadas.

```
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
}
```

Guarde el solucionador y vaya a la página Consultas de la consola de AWS AppSync. Ahora vamos a eliminar un par de lecturas de sensores.

Ejecute la mutación siguiente:

```
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**
>
> Al contrario que en la operación `DeleteItem`, el elemento completamente eliminado no se devuelve en la respuesta. Solo se devuelve la clave pasada. Para obtener más información, consulte BatchDeleteItem en la referencia de la función del solucionador de JavaScript para DynamoDB.

Valide a través de la consola de DynamoDB que estas dos lecturas se han eliminado de las tablas `locationReadings` y `temperatureReadings`.

BatchGetItem: recuperar lecturas

Otra operación habitual en nuestra aplicación es obtener las lecturas de un sensor en un momento determinado. Vamos a asociar un solucionador al campo de GraphQL `Query.getReadings` en nuestro esquema. En la página Esquema de la consola de AWS AppSync, seleccione Asociar junto al campo `Query.getReadings`. En la siguiente pantalla, cree su solucionador utilizando la tabla `locationReadings` como origen de datos.

Vamos a utilizar el siguiente código:

```
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' })),
 ]
}
```

Guarde el solucionador y vaya a la página Consultas de la consola de AWS AppSync. Ahora vamos a recuperar las lecturas de nuestro sensor.

Ejecute la siguiente consulta:

```
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
    }
  }
}
```

Hemos demostrado el uso de operaciones por lotes de DynamoDB mediante AWS AppSync.

Control de errores

En AWS AppSync, en ocasiones las operaciones de orígenes de datos pueden devolver resultados parciales. "Resultados parciales" es el término que utilizaremos para indicar que el resultado de una operación se compone de algunos datos y un error. Dado que la gestión de errores es inherentemente específica de la aplicación, AWS AppSync le da la oportunidad de gestionar los errores en el controlador de respuestas. El error de invocación del solucionador, si lo hay, está disponible en el contexto como `ctx.error`. Los errores de invocación siempre incluyen un mensaje y un tipo a los que se tiene acceso como propiedades `ctx.error.message` y `ctx.error.type`. En el controlador de respuestas, puede gestionar los resultados parciales de tres maneras:

1. Pasar por alto el error de la invocación y limitarse a devolver los datos.
2. Generar un error (con `util.error(...)`) y detener la evaluación del controlador, con lo que no se devuelven datos.
3. Agregar un error (con `util.appendError(...)`) y también devolver los datos.

Vamos a demostrar cada una de estas posibilidades con operaciones por lotes de DynamoDB.

Operaciones por lotes de DynamoDB

Con las operaciones por lotes de DynamoDB, es posible que un lote se complete parcialmente. Es decir, es posible que algunos de los elementos o claves solicitados se queden sin procesar. Si AWS AppSync no puede completar un lote, se establecen en el contexto los elementos que no se han procesado y un error de invocación.

Vamos a implementar la gestión de errores utilizando la configuración del campo `Query.getReadings` de la operación `BatchGetItem` de la sección anterior de este tutorial. Esta vez, supongamos que mientras se ejecutaba el campo `Query.getReadings`, la tabla de DynamoDB `temperatureReadings` agotó el desempeño aprovisionado. DynamoDB ha generado `ProvisionedThroughputExceededException` en el segundo intento de AWS AppSync de procesar los elementos restantes del lote.

El siguiente código JSON representa el contexto serializado después de la invocación por lotes de DynamoDB, pero antes de que se llamara al controlador de respuestas:

```
{
  "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": []
}
```

Cabe tener en cuenta algunos aspectos del contexto:

- AWS AppSync ha establecido el error de invocación en el contexto en `ctx.error` y el tipo de error se ha establecido en `DynamoDB:ProvisionedThroughputExceededException`.
- Los resultados se mapean para cada tabla en `ctx.result.data`, aunque haya un error.
- Las claves que quedaron sin procesar están disponibles en `ctx.result.data.unprocessedKeys`. Aquí, AWS AppSync no pudo recuperar el elemento con la clave (sensorId:1, timestamp:2018-02-01T17:21:05.000+08:00) debido a un rendimiento insuficiente de la tabla.

> **Note**
>
> Para `BatchPutItem`, es `ctx.result.data.unprocessedItems`. Para `BatchDeleteItem`, es `ctx.result.data.unprocessedKeys`.

Vamos a gestionar este error de tres formas diferentes.

1. Pasar por alto el error de invocación

Si se devuelven los datos sin gestionar el error de invocación se pasa por alto el error, lo que hace que el resultado para el campo de GraphQL indicado siempre tenga éxito.

El código que escribimos ya es conocido y solo se centra en los datos de resultados.

Controlador de respuestas

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

Respuesta de 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
      }
    ]
  }
}
```

No se añadirán errores a la respuesta, ya que solo se actúa en los datos.

2. Se genera un error para abortar la ejecución del controlador de respuestas

Cuando los errores parciales se deban tratar como errores completos desde la perspectiva del cliente, puede anular la ejecución del controlador de respuestas para evitar la devolución de datos. El método de utilidad `util.error(...)` consigue exactamente ese comportamiento.

Código de controlador de respuestas

```
export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type, null,
 ctx.result.data.unprocessedKeys);
  }
  return ctx.result.data;
}
```

Respuesta de 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. (...)"
    }
  ]
}
```

Aunque podrían haberse devuelto algunos resultados de la operación por lotes de DynamoDB, decidimos generar un error que hace que el campo de GraphQL `getReadings` sea nulo y se añada el error al bloque de errors de la respuesta de GraphQL.

3. Añadir el error para devolver tanto los datos como los errores

En algunos casos, para proporcionar una mejor experiencia al usuario, las aplicaciones pueden devolver resultados parciales e informar a sus clientes de los elementos que no se han procesado. Los clientes pueden decidir probar de nuevo o trasladar el error al usuario final. El método `util.appendError(...)` es la utilidad que permite este comportamiento al permitir que el creador de la aplicación agregue errores al contexto sin interferir con la evaluación del controlador de respuestas. Después de evaluar el controlador de respuestas, AWS AppSync procesará cualquier error de contexto agregándolo al bloque de errores de la respuesta de GraphQL.

Código de controlador de respuestas

```
export function response(ctx) {
  if (ctx.error) {
    util.appendError(ctx.error.message, ctx.error.type, null,
 ctx.result.data.unprocessedKeys);
  }
  return ctx.result.data;
}
```

Hemos reenviado el error de invocación y el elemento `unprocessedKeys` dentro del bloque de errores de la respuesta de GraphQL. El campo `getReadings` también devuelve datos parciales de la tabla `locationReadings`, como puede ver en la respuesta a continuación.

Respuesta de 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: Solucionadores de HTTP

AWS AppSync le permite utilizar orígenes de datos compatibles (es decir, AWS Lambda, Amazon DynamoDB, Amazon OpenSearch Service o Amazon Aurora) para realizar diversas operaciones, además de cualquier punto de conexión HTTP arbitrario para resolver campos de GraphQL. Una vez que los puntos de enlace HTTP están disponibles, puede conectar con ellos mediante un origen de datos. A continuación, puede configurar un solucionador en el esquema para realizar operaciones de GraphQL como consultas, mutaciones, y suscripciones. Este tutorial le guiará a lo largo de algunos ejemplos comunes.

En este tutorial, se utiliza una API de REST (creada mediante Amazon API Gateway y Lambda) con un punto de conexión de GraphQL de AWS AppSync.

Creación de una API de REST

Puede utilizar la siguiente plantilla de AWS CloudFormation para establecer un punto de enlace REST que funcione con este tutorial:

La pila de AWS CloudFormation realiza los siguientes pasos:

1. Configura una función Lambda que contiene la lógica de negocio del microservicio.
2. Configure una API de REST de puerta de enlace de API con la combinación punto de conexión/método/tipo de contenido siguiente:

Ruta de recurso de la API	Método HTTP	Tipo de contenido admitido
/v1/users	POST	application/json
/v1/users	GET	application/json
/v1/users/1	GET	application/json
/v1/users/1	PUT	application/json
/v1/users/1	DELETE	application/json

Creación de la API de GraphQL

Para crear la API de GraphQL en AWS AppSync:

1. Abra la consola de AppSync AWS y elija Crear API.
2. Elija API de GraphQL y, a continuación, seleccione Diseñar desde cero. Elija Next (Siguiente).
3. En el nombre de la API, introduzca `UserData`. Elija Next (Siguiente).
4. Elija `Create GraphQL resources later`. Elija Next (Siguiente).
5. Revise las entradas y seleccione Crear API.

La consola de AWS AppSync crea una nueva API de GraphQL automáticamente utilizando el modo de autenticación de clave de API. Puede utilizar la consola para configurar aún más la API de GraphQL y ejecutar solicitudes.

Creación de un esquema de GraphQL

Ahora que tiene una API de GraphQL, vamos a crear un esquema de GraphQL. En el editor Esquema de la consola de AWS AppSync, use el siguiente fragmento:

```
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
}
```

Configure el origen de datos HTTP

Para configurar el origen de datos HTTP, haga lo siguiente:

1. En la página Orígenes de datos de la API de GraphQL de AWS AppSync, seleccione Crear origen de datos.
2. Escriba un nombre para el origen de datos como `HTTP_Example`.
3. En Tipo de origen de datos, elija Punto de conexión HTTP.
4. Establezca el punto de conexión en el punto de conexión de la puerta de enlace de API que se creó al principio del tutorial. Para encontrar el punto de conexión generado por la pila, vaya a la consola de Lambda y busque su aplicación en Aplicaciones. Dentro de la configuración de la aplicación, debería ver un punto de conexión de API, que será su punto de conexión en AWS AppSync. Asegúrese de no incluir el nombre de etapa como parte del punto de conexión. Por ejemplo, si su punto de conexión fuera `https://aaabbbcccd.execute-api.us-east-1.amazonaws.com/v1`, escribiría `https://aaabbbcccd.execute-api.us-east-1.amazonaws.com`.

> **Note**
>
> En estos momentos, solo los puntos de conexión públicos son compatibles con AWS AppSync.
> Para obtener más información acerca de las entidades de certificación que reconoce el servicio de AWS AppSync, consulte Certificate Authorities (CA) Recognized by AWS AppSync for HTTPS Endpoints.

Configuración de los solucionadores

En este paso conectará el origen de datos HTTP a las consultas `getUser` y `addUser`.

Para configurar el solucionador de `getUser`:

1. En su API de GraphQL de AWS AppSync, seleccione la pestaña Esquema.
2. A la derecha del editor Esquema, en el panel Solucionadores y en la sección de tipo Consulta, busque el campo `getUser` y seleccione Asociar.
3. Mantenga el tipo de solucionador en `Unit` y el tiempo de ejecución en `APPSYNC_JS`.
4. En Nombre del origen de datos, elija el punto de conexión HTTP que creó anteriormente.
5. Seleccione Create (Crear).
6. En el editor de código Solucionador, añada el siguiente fragmento como controlador de solicitudes:

```
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. Añada el siguiente fragmento como controlador de respuestas:

```
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. Elija la pestaña Query (Consulta) y ejecute la consulta siguiente:

```
query GetUser{
    getUser(id:1){
        id
        username
    }
}
```

   Debe obtener la siguiente respuesta:

```
{
    "data": {
        "getUser": {
            "id": "1",
            "username": "nadia"
        }
    }
}
```

Para configurar el solucionador de `addUser`:

1. Elija la pestaña Schema (Esquema).
2. A la derecha del editor Esquema, en el panel Solucionadores y en la sección de tipo Consulta, busque el campo `addUser` y seleccione Asociar.
3. Mantenga el tipo de solucionador en `Unit` y el tiempo de ejecución en `APPSYNC_JS`.
4. En Nombre del origen de datos, elija el punto de conexión HTTP que creó anteriormente.

5. Seleccione Create (Crear).
6. En el editor de código Solucionador, añada el siguiente fragmento como controlador de solicitudes:

```
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. Añada el siguiente fragmento como controlador de respuestas:

```
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. Elija la pestaña Query (Consulta) y ejecute la consulta siguiente:

```
mutation addUser{
    addUser(userInput:{
        id:"2",
        username:"shaggy"
    }){
        id
        username
    }
}
```

Si vuelve a ejecutar la consulta `getUser`, debería devolver la siguiente respuesta:

```
{
    "data": {
        "getUser": {
        "id": "2",
        "username": "shaggy"
        }
    }
}
```

Invocación de servicios de AWS

Puede utilizar los solucionadores de HTTP para establecer una interfaz de API de GraphQL para los servicios de AWS. Las solicitudes HTTP a AWS deben firmarse con el proceso de Signature Version 4 para que AWS pueda identificar quién las envió. AWS AppSync calcula la firma en su nombre cuando asocia un rol de IAM al origen de datos HTTP.

Debe proporcionar dos componentes adicionales para invocar los servicios de AWS con solucionadores de HTTP:

- Un rol de IAM con permisos para realizar una llamada a las API del servicio de AWS.
- Configuración de la firma en el origen de datos

Por ejemplo, si desea realizar una llamada a la operación ListGraphqlApis con los solucionadores de HTTP, primero debe crear un rol de IAM que AWS AppSync asume con la siguiente política asociada:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "appsync:ListGraphqlApis"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
```

```
}
```

A continuación, cree el origen de datos HTTP para AWS AppSync. En este ejemplo, debe realizar una llamada a AWS AppSync en la región Oeste de EE. UU. (Oregón). Ajuste la siguiente configuración HTTP en un archivo denominado `http.json`, que incluye la región de la firma y el nombre del servicio:

```
{
    "endpoint": "https://appsync.us-west-2.amazonaws.com/",
    "authorizationConfig": {
        "authorizationType": "AWS_IAM",
        "awsIamConfig": {
            "signingRegion": "us-west-2",
            "signingServiceName": "appsync"
        }
    }
}
```

A continuación, utilice la AWS CLI para crear el origen de datos con un rol asociado de la siguiente forma:

```
aws appsync create-data-source --api-id <API-ID> \
                               --name AWSAppSync \
                               --type HTTP \
                               --http-config file:///http.json \
                               --service-role-arn <ROLE-ARN>
```

Cuando asocie un solucionador al campo del esquema, utilice la siguiente plantilla de mapeo de solicitudes para realizar una llamada a AWS AppSync:

```
{
    "version": "2018-05-29",
    "method": "GET",
    "resourcePath": "/v1/apis"
}
```

Cuando ejecuta una consulta de GraphQL para este origen de datos, AWS AppSync firma la solicitud mediante el rol que ha proporcionado e incluye la firma en la solicitud. La consulta devuelve una lista de API de GraphQL de AWS AppSync en su cuenta de dicha región de AWS.

Tutorial: Aurora PostgreSQL con API de datos

AWS AppSync proporciona un origen de datos para ejecutar instrucciones SQL con respecto a clústeres de Amazon Aurora que se han habilitado con una API de datos. Puede utilizar los solucionadores de AWS AppSync para ejecutar instrucciones SQL con respecto a la API de datos con consultas, mutaciones y suscripciones de GraphQL.

> **Note**
>
> En este tutorial se utiliza la región `US-EAST-1`.

Creación de clústeres

Antes de añadir un origen de datos de Amazon RDS a AWS AppSync, habilite primero una API de datos en un clúster de Aurora sin servidor. También debe configurar un secreto mediante AWS Secrets Manager. Para crear un clúster de Aurora sin servidor, puede usar la 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
```

Esto devolverá un ARN para el clúster. Puede comprobar el estado del clúster con el comando:

```
aws rds describe-db-clusters \
    --db-cluster-identifier appsync-tutorial \
    --query "DBClusters[0].Status"
```

Cree un secreto a través de la consola de AWS Secrets Manager o la AWS CLI con un archivo de entrada como el siguiente utilizando el `USERNAME` y la `COMPLEX_PASSWORD` del paso anterior:

```
{
    "username": "USERNAME",
    "password": "COMPLEX_PASSWORD"
}
```

Transfiera esto como parámetro a CLI:

```
aws secretsmanager create-secret \
    --name appsync-tutorial-rds-secret \
    --secret-string file://creds.json
```

Esto devolverá un ARN para el secreto. Anote el ARN de su clúster de Aurora sin servidor y el secreto para su uso posterior al crear un origen de datos en la consola de AWS AppSync.

Habilitación de la API de datos

Cuando el estado del clúster cambie a `available`, habilite la API de datos siguiendo la documentación de Amazon RDS. La API de datos debe estar habilitada antes de añadirse como un origen de datos de AWS AppSync. También puede habilitar la API de datos mediante la AWS CLI:

```
aws rds modify-db-cluster \
    --db-cluster-identifier appsync-tutorial \
    --enable-http-endpoint \
    --apply-immediately
```

Creación de la base de datos y la tabla

Tras habilitar la API de datos, valide que funciona mediante el comando `aws rds-data execute-statement` en la AWS CLI. Esto garantiza que el clúster de Aurora sin servidor esté configurado correctamente antes de añadirlo a la API de AWS AppSync. En primer lugar, cree una base de datos TESTDB con el parámetro `--sql`:

```
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\""
```

Si esto se ejecuta sin errores, añada dos tablas con el comando `create table`:

```
 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);'
```

Si todo se ejecuta sin problemas, ahora puede añadir el clúster como origen de datos en la API.

Creación de un esquema de GraphQL

Ahora que la API de datos de Aurora sin servidor se ejecuta con tablas configuradas, crearemos un esquema de GraphQL. Puede hacerlo manualmente, pero AWS AppSync le permite empezar rápidamente importando la configuración de la tabla desde una base de datos existente mediante el asistente de creación de la API.

Para empezar:

1. En la consola de AWS AppSync, elija Crear API y, a continuación, Comenzar con un clúster de Amazon Aurora.
2. Especifique los detalles de la API, como el nombre de la API, y, a continuación, seleccione su base de datos para generar la API.
3. Seleccione la base de datos. Si es necesario, actualice la región y, a continuación, elija el clúster de Aurora y la base de datos TESTDB.
4. Elija su secreto y, a continuación, seleccione Importar.
5. Una vez descubiertas las tablas, actualice los nombres de tipos. Cambie `Todos` a `Todo` y `Tasks` a `Task`.
6. Obtenga una vista previa del esquema generado seleccionando Vista previa del esquema. Sus esquema tendrá un aspecto similar al siguiente:

```
type Todo {
  id: Int!
  description: String!
```

```
  due: AWSDate!
  createdAt: String
}

type Task {
  id: Int!
  todoId: Int!
  description: String
}
```

7. Para el rol, puede permitir que AWS AppSync cree un rol nuevo o cree uno con una política similar a la siguiente:

```
{
    "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:*"
            ]
        }
    ]
}
```

Tenga en cuenta que hay dos instrucciones en esta política a las que está concediendo acceso de rol. El primer recurso es su clúster de Aurora y el segundo es su ARN de AWS Secrets Manager.

Elija Siguiente, revise los detalles de configuración y, a continuación, elija Crear API. Ahora puede disponer de una API totalmente operativa. Puede revisar todos los detalles de su API en la página Esquema.

Solucionadores para RDS

El flujo de creación de la API creó automáticamente los solucionadores para que interactuaran con nuestros tipos. Si consulta la página Esquema, encontrará los solucionadores necesarios para:

- Crear un `todo` mediante el campo `Mutation.createTodo`.
- Actualizar un `todo` mediante el campo `Mutation.updateTodo`.
- Eliminar un `todo` mediante el campo `Mutation.deleteTodo`.
- Obtener un `todo` individual mediante el campo `Query.getTodo`.
- Enumerar todos `todos` mediante el campo `Query.listTodos`.

Encontrará campos y solucionadores similares adjuntos para el tipo `Task`. Echemos un vistazo más de cerca a algunos de los solucionadores.

Mutation.createTodo

En el editor de esquemas de la consola de AWS AppSync, en el lado derecho, elija `testdb` junto a `createTodo(...): Todo`. El código de resolución utiliza la función `insert` del módulo `rds` para crear dinámicamente una instrucción de inserción que añade datos a la tabla `todos`. Puesto que estamos trabajando con Postgres, podemos aprovechar la instrucción `returning` para recuperar los datos insertados.

Actualicemos el solucionador para especificar correctamente el tipo `DATE` del campo `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]
}
```

Guarde el solucionador. La sugerencia de tipo marca el `due` correctamente en nuestro objeto de entrada como un tipo `DATE`. Esto permite que el motor Postgres interprete correctamente el valor. A continuación, actualice su esquema para eliminar el `id` de la entrada `CreateTodo`. Puesto que nuestra base de datos de Postgres puede devolver el ID generado, podemos confiar en él para crear y devolver el resultado como una única solicitud:

```
input CreateTodoInput {
    due: AWSDate!
    createdAt: String
    description: String!
}
```

Realice el cambio y actualice su esquema. Diríjase al editor de consultas para añadir un elemento a la base de datos:

```
mutation CreateTodo {
  createTodo(input: {description: "Hello World!", due: "2023-12-31"}) {
    id
    due
    description
    createdAt
  }
```

```
}
```

Obtiene el resultado:

```
{
  "data": {
    "createTodo": {
      "id": 1,
      "due": "2023-12-31",
      "description": "Hello World!",
      "createdAt": "2023-11-14 20:47:11.875428"
    }
  }
}
```

Query.listTodos

En el editor de esquemas de la consola, en el lado derecho, elija `testdb` junto a `listTodos(id: ID!): Todo`. El controlador de solicitudes utiliza la función de utilidad select para crear una solicitud de forma dinámica en tiempo de ejecución.

```
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)
}
```

Queremos filtrar `todos` en función de la fecha `due`. Actualicemos la resolución para convertir los valores `due` en `DATE`. Actualice la lista de importaciones y el controlador de solicitudes:

```
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 };
}
```

Vamos a probar la consulta. En el editor de consultas:

```
query LIST {
```

```
  listTodos(limit: 10, filter: {due: {between: ["2021-01-01", "2025-01-02"]}}) {
    items {
      id
      due
      description
    }
  }
}
```

Mutation.updateTodo

También puede `update` a `Todo`. En el editor de consultas, vamos a actualizar nuestro primer elemento `Todo` de `id` `1`.

```
mutation UPDATE {
  updateTodo(input: {id: 1, description: "edits"}) {
    description
    due
    id
  }
}
```

Tenga en cuenta que debe especificar el `id` del elemento que está actualizando. También puede especificar una condición para actualizar únicamente un elemento que cumpla condiciones específicas. Por ejemplo, es posible que solo queramos editar el elemento si la descripción comienza por `edits`:

```
mutation UPDATE {
  updateTodo(input: {id: 1, description: "edits: make a change"}, condition:
 {description: {beginsWith: "edits"}}) {
    description
    due
    id
  }
}
```

Al igual que gestionamos nuestras operaciones `create` y `list`, podemos actualizar nuestro solucionador para convertir el campo `due` en una `DATE`. Guarde estos cambios en `updateTodo`:

```
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];
}
```

Ahora intente realizar una actualización con una condición:

```
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
        }
}
```

Mutation.deleteTodo

Puede `delete` un `Todo` con la mutación `deleteTodo`. Funciona igual que la mutación `updateTodo`, y debe especificar el `id` del elemento que desea eliminar:

```
mutation DELETE {
  deleteTodo(input: {id: 1}) {
    description
    due
    id
  }
}
```

Escritura de consultas personalizadas

Hemos utilizado las utilidades del módulo `rds` para crear nuestras instrucciones SQL. También podemos escribir nuestra propia instrucción estática personalizada para interactuar con nuestra base de datos. En primer lugar, actualice el esquema para eliminar el campo `id` de la entrada `CreateTask`.

```
input CreateTaskInput {
    todoId: Int!
    description: String
}
```

A continuación, cree un par de tareas. Una tarea tiene una relación de clave externa con `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 }
```

```
}
```

Cree un campo nuevo de su tipo `Query` denominado `getTodoAndTasks`:

```
getTodoAndTasks(id: Int!): Todo
```

Añada un campo `tasks` al tipo `Todo`:

```
type Todo {
    due: AWSDate!
    id: Int!
    createdAt: String
    description: String!
    tasks:TaskConnection
}
```

Guarde el esquema. En el editor de esquemas de la consola, elija a la derecha Asociar solucionador para `getTodosAndTasks(id: Int!): Todo`. Elija el origen de datos de Amazon RDS. Actualice su solucionador con el siguiente código:

```
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;
}
```

En este código, utilizamos la plantilla de etiquetas de `sql` para escribir una instrucción SQL a la que podamos pasar un valor dinámico de forma segura en tiempo de ejecución. `createPgStatement` puede aceptar hasta dos solicitudes de SQL a la vez. La usamos para enviar una consulta para

nuestra `todo` y otra para nuestras `tasks`. Podría haberlo hecho con una instrucción `JOIN` o con cualquier otro método. La idea es poder escribir su propia instrucción SQL para implementar su lógica empresarial. Para usar la consulta en el editor de consultas, podemos probar lo siguiente:

```
query TodoAndTasks {
  getTodosAndTasks(id: 2) {
    id
    due
    description
    tasks {
      items {
        id
        description
      }
    }
  }
}
```

Eliminación de su clúster

> **Important**
>
> La eliminación de un clúster es permanente. Revise su proyecto detenidamente antes de llevar a cabo esta acción.

Para eliminar el clúster:

```
$ aws rds delete-db-cluster \
    --db-cluster-identifier appsync-tutorial \
    --skip-final-snapshot
```

Tutoriales de solucionadores (VTL)

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

Los orígenes de datos y los solucionadores son la forma en que AWS AppSync convierte las solicitudes de GraphQL y obtiene información de sus recursos de AWS. AWS AppSync admite el aprovisionamiento automático y las conexiones con determinados tipos de orígenes de datos. AWS AppSync admite AWS Lambda, Amazon DynamoDB, bases de datos relacionales (Amazon Aurora sin servidor), Amazon OpenSearch Service y puntos de conexión HTTP como orígenes de datos. Puede utilizar una API de GraphQL con los recursos de AWS que ya tiene o crear orígenes de datos y solucionadores. En esta sección recorrerá este proceso en una serie de tutoriales para comprender mejor cómo funcionan los detalles y las opciones de ajuste.

AWS AppSync usa plantillas de mapeo escritas en Apache Velocity Template Language (VTL) para solucionadores. Para obtener más información sobre el uso de plantillas de mapeo, consulte Resolver mapping template reference. Encontrará más información sobre cómo trabajar con VTL en la Resolver mapping template programming guide.

AWS AppSync permite el aprovisionamiento automático de tablas de DynamoDB desde un esquema de GraphQL, como se describe en las secciones sobre aprovisionamiento desde el esquema (opcional) y lanzamiento de un esquema de ejemplo. También puede importar desde una tabla de DynamoDB existente, lo que creará solucionadores de esquema y de conexión. Esto se detalla en la sección sobre importación desde Amazon DynamoDB (opcional).

Temas

- Tutorial: Solucionadores de DynamoDB
- Tutorial: Solucionadores de Lambda
- Tutorial: Solucionadores de Amazon OpenSearch Service
- Tutorial: Solucionadores locales
- Tutorial: Combinación de solucionadores de GraphQL
- Tutorial: Solucionadores por lotes de DynamoDB

- Tutorial: Solucionadores de transacciones de DynamoDB
- Tutorial: Solucionadores de HTTP
- Tutorial: Aurora sin servidor
- Tutorial: Solucionadores de canalizaciones
- Tutorial: Delta Sync

Tutorial: Solucionadores de DynamoDB

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

En este tutorial se muestra cómo puede aprovechar sus tablas de Amazon DynamoDB en AWS AppSync y conectarlas a una API de GraphQL.

Puede hacer que AWS AppSync aprovisione recursos de DynamoDB en su nombre. O bien, si lo prefiere, puede conectar las tablas existentes a un esquema de GraphQL creando un origen de datos y un solucionador. En cualquier caso, podrá leer y escribir en la base de datos de DynamoDB a través de instrucciones de GraphQL y suscribirse a datos en tiempo real.

Hay que realizar una serie de pasos de configuración específicos para que las instrucciones de GraphQL se traduzcan a operaciones de DynamoDB y que, a su vez, las respuestas se traduzcan de nuevo a GraphQL. En este tutorial se describe el proceso de configuración a través de varios escenarios y patrones de acceso a datos del mundo real.

Configuración de las tablas de DynamoDB

Para comenzar este tutorial, primero debe seguir los pasos que se indican a continuación para aprovisionar recursos de AWS.

1. Aprovisione recursos de AWS mediante la siguiente plantilla de AWS CloudFormation en la 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
```

Como alternativa, puede iniciar la siguiente pila de AWS CloudFormation en la región EE. UU. Oeste 2 (Oregón) de su cuenta de AWS.

Así se crea lo siguiente:

- Una tabla de DynamoDB llamada `AppSyncTutorial-Post` que contendrá datos de `Post`.
- Un rol de IAM y una política administrada de IAM asociada para que AWS AppSync pueda interactuar con la tabla `Post`.

2. Para ver más información acerca de la pila y de los recursos creados, ejecute el siguiente comando de la CLI:

```
aws cloudformation describe-stacks --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

3. Para eliminar los recursos más adelante, puede ejecutar lo siguiente:

```
aws cloudformation delete-stack --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

Creación de la API de GraphQL

Para crear la API de GraphQL en AWS AppSync:

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   - En el Panel de API, elija Crear API.
2. En la ventana Personalice su API o impórtela desde Amazon DynamoDB, seleccione Crear desde cero.
   - Seleccione Comenzar a la derecha de la misma ventana.
3. En el campo Nombre de la API, establezca el nombre de la API en `AWSAppSyncTutorial`.
4. Seleccione Create (Crear).

La consola de AWS AppSync crea una nueva API de GraphQL automáticamente utilizando el modo de autenticación de clave de API. Puede utilizar la consola para configurar el resto de la API de GraphQL y ejecutar consultas en ella durante el resto de este tutorial.

Definición de una API de publicación básica

Ahora que ha creado una API de GraphQL en AWS AppSync, puede configurar un esquema básico que permita la creación, recuperación y eliminación básica de datos de publicaciones.

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   - En el Panel de API, elija la API que ha creado.
2. En la barra lateral, seleccione Esquema.
   - En la página Esquema, sustituya el contenido por el código siguiente:

```
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. Seleccione Save.

Este esquema define un tipo `Post` y las operaciones para agregar y obtener objetos `Post`.

Configuración del origen de datos para las tablas de DynamoDB

A continuación, vincule las consultas y las mutaciones definidas en el esquema a la tabla `AppSyncTutorial-Post` de DynamoDB.

En primer lugar, AWS AppSync debe conocer las tablas. Para ello, configure un origen de datos en AWS AppSync:

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   a. En el panel de API, seleccione su API de GraphQL.
   b. En la barra lateral, seleccione Origen de datos.
2. Elija Create data source.
   a. En Nombre de origen de datos, escriba `PostDynamoDBTable`.
   b. En Tipo de origen de datos, elija Tabla de Amazon DynamoDB.
   c. En Región, elija US-WEST-2.
   d. En Nombre de tabla, elija la tabla de DynamoDB AppSyncTutorial-Post.
   e. Cree un nuevo rol de IAM (recomendado) o elija un rol existente que tenga el permiso de IAM `lambda:invokeFunction`. Los roles existentes necesitan una política de confianza, tal y como se explica en la sección Asociar un origen de datos.

      El siguiente es un ejemplo de política de IAM que tiene los permisos necesarios para llevar a cabo operaciones en el recurso:

```
{
     "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. Seleccione Create (Crear).

Configuración del solucionador addPost (PutItem de DynamoDB)

Una vez que AWS AppSync conoce la tabla de DynamoDB, puede vincularla a consultas y mutaciones individuales mediante la definición de solucionadores. El primer solucionador que va a crear es `addPost`, que le permite crear una publicación en la tabla `AppSyncTutorial-Post` de DynamoDB.

Un solucionador tiene los siguientes componentes:

- La ubicación en el esquema de GraphQL donde se asocia el solucionador. En este caso configuramos en el campo `addPost` un solucionador de tipo `Mutation`. El solucionador se invocará cuando el intermediario llame a `mutation { addPost(...){...} }`.
- El origen de datos que se va a utilizar para el solucionador. En este caso, queremos utilizar el origen de datos `PostDynamoDBTable` definido anteriormente para poder agregar entradas en la tabla `AppSyncTutorial-Post` de DynamoDB.
- La plantilla de mapeo de solicitudes. El objetivo de la plantilla de mapeo de solicitudes es tomar la solicitud entrante del intermediario y convertirla en instrucciones que AWS AppSync ejecuta en DynamoDB.
- La plantilla de mapeo de respuestas. El cometido de la plantilla de mapeo de respuesta es tomar la respuesta de DynamoDB y traducirla de nuevo a algo que GraphQL espera. Esto resulta útil si la forma de los datos en DynamoDB es diferente del tipo `Post` en GraphQL. Como este caso sí tienen la misma forma, solo hay que transmitir los datos.

Para configurar el solucionador:

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.

    a. En el panel de API, seleccione su API de GraphQL.

b. En la barra lateral, seleccione Origen de datos.

2. Elija Create data source.

a. En Nombre de origen de datos, escriba `PostDynamoDBTable`.

b. En Tipo de origen de datos, elija Tabla de Amazon DynamoDB.

c. En Región, elija US-WEST-2.

d. En Nombre de tabla, elija la tabla de DynamoDB AppSyncTutorial-Post.

e. Cree un nuevo rol de IAM (recomendado) o elija un rol existente que tenga el permiso de IAM `lambda:invokeFunction`. Los roles existentes necesitan una política de confianza, tal y como se explica en la sección Asociar un origen de datos.

El siguiente es un ejemplo de política de IAM que tiene los permisos necesarios para llevar a cabo operaciones en el recurso:

```
{
     "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. Seleccione Create (Crear).

4. Elija la pestaña Schema (Esquema).

5. En el panel Data types (Tipos de datos) de la derecha, busque el campo addPost en el tipo Mutation y, a continuación, seleccione Attach (Asociar).

6. En el menú Acción, seleccione Actualizar tiempo de ejecución y, a continuación, elija Solucionador de unidades (solo VTL).

7. En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.

8. En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "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 }
    }
}
```

Nota: se especifica un tipo para todas las claves y valores de atributos. Por ejemplo, puede establecer el campo `author` como `{ "S" : "${context.arguments.author}" }`. La parte `S` indica a AWS AppSync y a DynamoDB que el valor será una cadena. El valor real se rellena con el argumento `author`. Del mismo modo, el campo `version` es numérico, ya que utiliza `N` para el tipo. Por último, también inicializamos los campos `ups`, `downs` y `version`.

En este tutorial, especificó que el tipo `ID!` de GraphQL, que indexa el nuevo elemento que se inserta en DynamoDB, forma parte de los argumentos del cliente. AWS AppSync incluye una utilidad para la generación automática de identificadores denominada `$utils.autoId()` que también podría haber utilizado con el formato `"id" : { "S" : "${$utils.autoId()}" }`. Así bastaría con excluir `id: ID!` de la definición del esquema de `addPost()` y se insertaría de forma automática. No utilizará esta técnica en este tutorial, pero debe considerarla, ya que se trata de una práctica recomendada a la hora de escribir en tablas de DynamoDB.

Para obtener más información acerca de las plantillas de mapeo, consulte la sección Información general de plantillas de mapeo de solucionador. Para obtener más información sobre el mapeo de solicitudes GetItem, consulte la documentación de referencia de GetItem. Para obtener más información sobre los tipos, consulte la documentación de referencia del Sistema de tipos (mapeos de solicitud).

9. En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
$utils.toJson($context.result)
```

Nota: Debido a que la forma de los datos de la tabla `AppSyncTutorial-Post` coincide exactamente con la forma del tipo `Post` de GraphQL, la plantilla de mapeo de respuesta se limita a pasar directamente los resultados. Observe que todos los ejemplos de este tutorial utilizan la misma plantilla de mapeo de respuesta, de manera que solo tiene que crear un archivo.

10. Seleccione Save.

Llame a la API para añadir una publicación

Ahora que el solucionador está configurado, AWS AppSync puede convertir una mutación `addPost` entrante en una operación PutItem de DynamoDB. Ahora puede ejecutar una mutación para incluir datos en la tabla.

- Seleccione la pestaña Queries (Consultas).
- En el panel Queries (Consultas), pegue la mutación siguiente:

```
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
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).

- Los resultados de la publicación que acaba de crear deben aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

Esto es lo que ha ocurrido:

- AWS AppSync ha recibido una solicitud de mutación `addPost`.
- AWS AppSync ha aceptado la solicitud y la plantilla de mapeo de solicitudes, y ha generado un documento de mapeo de solicitudes. Esto habrá tenido el siguiente aspecto:

```
{
    "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 ha utilizado el documento de mapeo de solicitudes para generar y ejecutar una solicitud `PutItem` de DynamoDB.
- AWS AppSync ha tomado los resultados de la solicitud `PutItem` y los ha convertido de nuevo en tipos de 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
}
```

- Después los ha pasado a través del documento de mapeo de respuesta, que se ha limitado a transferirlos sin cambios.
- Por fin ha devuelto el objeto recién creado en la respuesta de GraphQL.

Configuración del solucionador getPost (GetItem de DynamoDB)

Ahora que puede añadir datos a la tabla de DynamoDB `AppSyncTutorial-Post`, tiene que configurar la consulta `getPost` para poder recuperar los datos de la tabla `AppSyncTutorial-Post`. Para ello, tiene que configurar otro solucionador.

- Elija la pestaña Schema (Esquema).
- En el panel Data types (Tipos de datos) de la derecha, busque el campo getPost en el tipo Query y, a continuación, seleccione Attach (Asociar).
- En el menú Acción, seleccione Actualizar tiempo de ejecución y, a continuación, elija Solucionador de unidades (solo VTL).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "version" : "2017-02-28",
```

```
    "operation" : "GetItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    }
}
```

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
$utils.toJson($context.result)
```

- Seleccione Save.

Llame a la API para obtener una publicación

Ahora que ya está configurado el solucionador, AWS AppSync ya sabe cómo convertir una consulta `getPost` entrante en una operación `GetItem` de DynamoDB. Ahora puede ejecutar una consulta para recuperar la publicación que ha creado anteriormente.

- Seleccione la pestaña Queries (Consultas).
- En el panel Queries (Consultas), pegue lo siguiente:

```
query getPost {
  getPost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- La publicación obtenida de DynamoDB debe aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

Esto es lo que ha ocurrido:

- AWS AppSync ha recibido una solicitud de consulta `getPost`.
- AWS AppSync ha aceptado la solicitud y la plantilla de mapeo de solicitudes, y ha generado un documento de mapeo de solicitudes. Esto habrá tenido el siguiente aspecto:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "id" : { "S" : "123" }
    }
}
```

- AWS AppSync ha utilizado el documento de mapeo de solicitudes para generar y ejecutar una solicitud GetItem de DynamoDB.
- AWS AppSync ha tomado los resultados de la solicitud `GetItem` y los ha convertido de nuevo en tipos de 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
```

```
}
```

- Después los ha pasado a través del documento de mapeo de respuesta, que se ha limitado a transferirlos sin cambios.
- Entonces ha devuelto el objeto recuperado en la respuesta.

Como alternativa, vamos a tomar el siguiente ejemplo:

```
query getPost {
  getPost(id:123) {
    id
    author
    title
  }
}
```

Si la consulta `getPost` solo necesita las letras `id`, `author` y `title`, puede cambiar la plantilla de mapeo de solicitudes para utilizar expresiones de proyección que especifiquen únicamente los atributos que desee de la tabla de DynamoDB y evitar la transferencia innecesaria de datos de DynamoDB a AWS AppSync. Por ejemplo, la plantilla de mapeo de solicitudes puede tener un aspecto similar al siguiente fragmento:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },
    "projection" : {
     "expression" : "#author, id, title",
     "expressionNames" : { "#author" : "author"}
    }
}
```

Cree una mutación updatePost (UpdateItem de DynamoDB)

De momento ya sabe crear y recuperar objetos `Post` en DynamoDB. A continuación va a configurar una nueva mutación que permite actualizar objetos. Para ello, utilizará la operación UpdateItem de DynamoDB.

- Elija la pestaña Schema (Esquema).
- En el panel Schema (Esquema) modifique el tipo `Mutation` para agregar una nueva mutación `updatePost` de este modo:

```
type Mutation {
    updatePost(
        id: ID!,
        author: String!,
        title: String!,
        content: String!,
        url: String!
    ): Post
    addPost(
        author: String!
        title: String!
        content: String!
        url: String!
    ): Post!
}
```

- Seleccione Save.
- En el panel Data types (Tipos de datos) de la derecha, busque el campo updatePost recién creado en el tipo Mutation y, a continuación, elija Attach (Asociar).
- En el menú Acción, seleccione Actualizar tiempo de ejecución y, a continuación, elija Solucionador de unidades (solo VTL).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "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 }
        }
    }
}
```

Nota: Este solucionador utiliza la operación UpdateItem de DynamoDB, que difiere bastante de la operación PutItem. En lugar de escribir el elemento completo, solo pide a DynamoDB que actualice determinados atributos. Esto se consigue mediante expresiones de actualización de DynamoDB. La expresión en sí se especifica en el campo `expression` de la sección `update`. Indica que se deben establecer los atributos `author`, `title`, `content` y URL, y a continuación incrementar el campo `version`. Los valores empleados no aparecen en la expresión en sí, que contiene marcadores de posición con nombres que comienzan con dos puntos y que luego se definen en el campo `expressionValues`. Por último, DynamoDB tiene palabras reservadas que no pueden aparecer en la `expression`. Por ejemplo, `url` es una palabra reservada, por lo que para actualizar el campo `url`, puede utilizar marcadores de posición de nombre y definirlos en el campo `expressionNames`.

Para obtener más información acerca del mapeo de la solicitud `UpdateItem`, consulte la documentación de referencia de UpdateItem. Para obtener más información sobre el modo de escribir expresiones de actualización, consulte la documentación de DynamoDB UpdateExpressions.

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
$utils.toJson($context.result)
```

Llame a la API para actualizar una publicación

Ahora que ya está configurado el solucionador, AWS AppSync ya sabe cómo convertir una mutación `update` entrante en una operación `Update` de DynamoDB. Ahora puede ejecutar una mutación para actualizar el elemento que escribió anteriormente.

- Seleccione la pestaña Queries (Consultas).
- En el panel Queries (Consultas), pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente.

```
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
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- La publicación actualizada en DynamoDB debería aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

En este ejemplo, los campos `downs` y `ups` no se han modificado, porque la plantilla de mapeo de solicitudes no ha pedido a AWS AppSync ni a DynamoDB que hagan nada con esos campos. Además, el campo `version` se ha incrementado en 1, ya que pedimos a AWS AppSync y DynamoDB que sumaran 1 al campo `version`.

Modificación del solucionador updatePost (UpdateItem en DynamoDB)

Se trata de un buen comienzo para la mutación `updatePost`, pero tiene dos inconvenientes:

- Aunque solo desee actualizar un campo, tiene que actualizar todos.
- Si dos personas modifican el objeto a la vez información podría perderse información.

Para solucionar estos problemas, va a modificar la mutación `updatePost` de manera que únicamente modifique los argumentos que se han especificado en la solicitud, y también va a agregar una condición a la operación `UpdateItem`.

1. Elija la pestaña Schema (Esquema).
2. En el panel Schema (Esquema), modifique el campo `updatePost` del tipo `Mutation` para eliminar los signos de admiración de los argumentos `author`, `title`, `content` y `url`, asegurándose de dejar sin cambios el campo `id`. Esto los convertirá en argumentos opcionales. Además, añada un nuevo argumento `expectedVersion` obligatorio.

```
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. Seleccione Save.

4. En el panel Data types (Tipos de datos) de la derecha, busque el campo updatePost en el tipo Mutation.
5. Elija PostDynamoDBTable para abrir el solucionador existente.
6. En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), modifique la plantilla de mapeo de solicitud de este modo:

```
{
    "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. Seleccione Save.

Esta plantilla es uno de los ejemplos más complejos. Demuestra la eficacia y flexibilidad de las plantillas de mapeo. Itera por todos los argumentos, saltando `id` y `expectedVersion`. Si el argumento tiene un valor, solicita a AWS AppSync y DynamoDB que actualice en el objeto en DynamoDB. Si el atributo tiene el valor nulo, solicita a AWS AppSync y DynamoDB que lo elimine del objeto publicación. Si no se ha especificado algún argumento, lo dejará como está. También incrementa el campo `version`.

También hay una nueva sección `condition`. Una expresión de condición le permite indicar a AWS AppSync y DynamoDB que la solicitud debe ejecutarse o no en función del estado del objeto que ya se encuentra en DynamoDB antes de efectuar la operación. En este caso, solo quiere que la solicitud `UpdateItem` se realice si el campo `version` del elemento que hay en DynamoDB coincide exactamente con el argumento `expectedVersion`.

Si desea más información sobre las expresiones de condición, consulte la documentación de referencia Expresiones de condición.

Llame a la API para actualizar una publicación

Vamos a intentar actualizar el objeto `Post` con el nuevo solucionador:

- Seleccione la pestaña Queries (Consultas).
- En el panel Queries (Consultas), pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente.

```
mutation updatePost {
  updatePost(
    id:123
    title: "An empty story"
    content: null
    expectedVersion: 2
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- La publicación actualizada en DynamoDB debería aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

En esta solicitud, ha pedido a AWS AppSync y DynamoDB que actualicen únicamente el campo `title` y `content`. Los demás campos se quedan como estaban (salvo por el incremento del campo

`version`). Hemos establecido un nuevo valor en el atributo `title` y hemos eliminado el atributo `content` de la publicación. Los campos `author`, `url`, `ups` y `downs` no se han modificado.

Intente ejecutar la solicitud de mutación de nuevo, dejándola exactamente como está. Verá una respuesta parecida a la siguiente:

```
{
  "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)"
    }
  ]
}
```

La solicitud falla porque la expresión de condición se evalúa como false:

- La primera vez que ejecutamos la solicitud, el valor del campo `version` de la publicación en DynamoDB era 2, que coincidía con el argumento `expectedVersion`. La solicitud se realizó correctamente, lo que significa que el campo `version` se incrementó en DynamoDB a 3.
- La segunda vez que ejecutamos la solicitud, el valor del campo `version` de la publicación en DynamoDB era 3, que no coincide con el argumento `expectedVersion`.

Este método suele denominarse "bloqueo optimista".

Una de las características del solucionador de AWS AppSync DynamoDB es que devuelve el valor actual del objeto de publicación en DynamoDB. Puede encontrarlo en el campo `data`, en la sección `errors` de la respuesta de GraphQL. La aplicación puede utilizar esta información para decidir cómo debe proceder. En este caso, podemos ver que el campo `version` del objeto en DynamoDB tiene el valor 3, por lo que solo tendríamos que actualizar el argumento `expectedVersion` a 3 para que la solicitud se ejecutase de nuevo.

Para obtener más información sobre cómo gestionar los casos en que no se cumpla la condición, consulte Expresiones de condición en la documentación de referencia de las plantillas de mapeo.

Cree mutaciones upvotePost y downvotePost (UpdateItem de DynamoDB)

El tipo `Post` tiene campos `ups` y `downs` para registrar los votos a favor y en contra, pero hasta el momento la API no nos deja hacer nada con ellos. Vamos a agregar algunas mutaciones para votar a favor o en contra de las publicaciones.

- Elija la pestaña Schema (Esquema).
- En el panel Schema (Schema), modifique el tipo `Mutation` para agregar las nuevas mutaciones `upvotePost` y `downvotePost`, de este modo:

```
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!
}
```

- Seleccione Save.
- En el panel Data types (Tipos de datos) de la derecha, busque el campo upvotePost recién creado en el tipo Mutation y, a continuación, elija Attach (Asociar).
- En el menú Acción, seleccione Actualizar tiempo de ejecución y, a continuación, elija Solucionador de unidades (solo VTL).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "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 }
        }
    }
}
```

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
$utils.toJson($context.result)
```

- Seleccione Save.
- En el panel Data types (Tipos de datos) de la derecha, busque el campo `downvotePost` recién creado en el tipo Mutation y, a continuación, elija Attach (Asociar).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.

- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "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 }
        }
    }
}
```

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
$utils.toJson($context.result)
```

- Seleccione Save.

Llame a la API para votar a favor o en contra de una publicación

Ahora que se han configurado nuevos solucionadores, AWS AppSync ya sabe cómo convertir una mutación `upvotePost` o `downvote` entrante en una operación UpdateItem de DynamoDB. Ahora puede ejecutar mutaciones para votar a favor o en contra de la publicación que ha creado anteriormente.

- Seleccione la pestaña Queries (Consultas).
- En el panel Queries (Consultas), pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente.

```
mutation votePost {
  upvotePost(id:123) {
    id
    author
    title
    content
```

```
    url
    ups
    downs
    version
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- La publicación se actualiza en DynamoDB y debe aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

- Elija Execute query (Ejecutar consulta) algunas veces más. Debería ver cómo se incrementan en 1 los campos `ups` y `version` cada vez que ejecuta la consulta.
- Modifique la consulta para llamar a la mutación `downvotePost` de este modo:

```
mutation votePost {
  downvotePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja). En esta ocasión, debería ver cómo se incrementan en 1 los campos `downs` y `version` cada vez que ejecuta la consulta.

```
{
  "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
    }
  }
}
```

Configuración de un solucionador deletePost (DeleteItem en DynamoDB)

La próxima mutación que desea configurar es la eliminación de una publicación. Para ello, utilizará la operación `DeleteItem` de DynamoDB.

- Elija la pestaña Schema (Esquema).
- En el panel Schema (Esquema) modifique el tipo `Mutation` para agregar una nueva mutación `deletePost` de este modo:

```
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!
}
```

Esta vez hemos declarado el campo `expectedVersion` como opcional, lo que se explica más adelante al añadir la plantilla de mapeo de solicitud.

- Seleccione Save.
- En el panel Data types (Tipos de datos) de la derecha, busque el campo delete recién creado en el tipo Mutation y, a continuación, elija Attach (Asociar).
- En el menú Acción, seleccione Actualizar tiempo de ejecución y, a continuación, elija Solucionador de unidades (solo VTL).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "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
}
```

Nota: el argumento `expectedVersion` es opcional. Si el intermediario ha establecido un argumento `expectedVersion` en la solicitud, entonces la plantilla añade una condición que

solo permitirá que la solicitud `DeleteItem` se atienda cuando el elemento se haya eliminado o cuando el atributo `version` de la publicación en DynamoDB coincida exactamente con `expectedVersion`. Si se omite, no se especificará ninguna expresión de condición para la solicitud `DeleteItem`. Se realizará correctamente independientemente del valor de `version` o de si el elemento existe o no en DynamoDB.

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
$utils.toJson($context.result)
```

  Note: Aunque se trate de eliminar un elemento, puede devolver el elemento eliminado, siempre que no se haya eliminado previamente.

- Seleccione Save.

Para obtener más información acerca del mapeo de la solicitud `DeleteItem`, consulte la documentación de referencia de DeleteItem.

Llame a la API para eliminar una publicación

Ahora que ya está configurado el solucionador, AWS AppSync ya sabe cómo convertir una mutación `delete` entrante en una operación `DeleteItem` de DynamoDB. Ahora ya podemos ejecutar una mutación para eliminar datos de la tabla.

- Seleccione la pestaña Queries (Consultas).
- En el panel Queries (Consultas), pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente.

```
mutation deletePost {
  deletePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
```

```
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- La publicación se elimina de DynamoDB. Observe que AWS AppSync devuelve el valor del elemento que se ha eliminado de DynamoDB, que debe aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

El valor solo se devuelve si esta llamada a `deletePost` es realmente la que lo elimina de DynamoDB.

- Elija Execute query (Ejecutar consulta) de nuevo.
- La llamada se sigue ejecutando con éxito, pero no se devuelve ningún valor.

```
{
  "data": {
    "deletePost": null
  }
}
```

Ahora vamos a probar a eliminar una publicación, pero esta vez especificando `expectedValue`. Primero tiene que crear una nueva, porque acaba de eliminar la publicación con la que ha estado trabajando hasta ahora.

- En el panel Queries (Consultas), pegue la mutación siguiente:

```
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
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- Los resultados de la publicación que acaba de crear deben aparecer en el panel de resultados a la derecha del panel de consultas. Anote el `id` del objeto recién creado, pues lo necesitaremos en breve. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

Ahora intentaremos eliminar esta publicación, pero especificaremos un valor erróneo en `expectedVersion`:

- En el panel Queries (Consultas), pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente.

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 9999
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).

```
{
  "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)"
    }
  ]
}
```

La solicitud falla, porque la expresión de condición se evalúa como false: el valor de `version` de la publicación en DynamoDB no coincide con el `expectedValue` especificado en los argumentos. El valor actual del objeto se devuelve en el campo `data` de la sección `errors` de la respuesta de GraphQL.

- Vuelva a intentar la solicitud, pero corrija `expectedVersion`:

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- Esta vez la solicitud se realiza correctamente y se devuelve el valor eliminado en DynamoDB:

```
{
  "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
    }
  }
}
```

- Elija Execute query (Ejecutar consulta) de nuevo.
- La llamada sigue realizándose, pero esta vez no se devuelve ningún valor, ya que la publicación ya se había eliminado en DynamoDB.

```
{
  "data": {
    "deletePost": null
  }
}
```

Configuración del solucionador allPost (Scan en DynamoDB)

Hasta ahora, la API solo es útil si conoce el `id` de cada publicación que desea ver. Añada un nuevo solucionador que devuelva todas las publicaciones en la tabla.

- Elija la pestaña Schema (Esquema).
- En el panel Schema (Esquema) modifique el tipo `Query` para agregar una nueva consulta `allPost`, de este modo:

```
type Query {
    allPost(count: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
}
```

- Añada un nuevo tipo `PaginationPosts`:

```
type PaginatedPosts {
```

```
    posts: [Post!]!
    nextToken: String
}
```

- Seleccione Save.
- En el panel Data types (Tipos de datos) de la derecha, busque el campo allPost recién creado en el tipo Query y, a continuación, elija Attach (Asociar).
- En el menú Acción, seleccione Actualizar tiempo de ejecución y, a continuación, elija Solucionador de unidades (solo VTL).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "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
}
```

  Este solucionador tiene dos argumentos opcionales: `count`, que especifica el número máximo de elementos que se devolverán en una sola llamada, y `nextToken`, que se puede utilizar para recuperar el siguiente conjunto de resultados (más adelante mostrará de dónde procede el valor de `nextToken`).

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
{
    "posts": $utils.toJson($context.result.items)
    #if( ${context.result.nextToken} )
        ,"nextToken": $util.toJson($context.result.nextToken)
    #end
}
```

Nota: esta plantilla de mapeo de respuesta es diferente de todas las que hemos visto hasta ahora. El resultado de la consulta `allPost` es del tipo `PaginatedPosts`, que contiene una lista de publicaciones y un token de paginación. La forma de este objeto es diferente a la que devuelve del solucionador para DynamoDB de AWS AppSync: la lista de publicaciones se llama `items` en los resultados del solucionador para DynamoDB de AWS AppSync, pero se llama `posts` en `PaginatedPosts`.

- Seleccione Save.

Si desea más información sobre el mapeo de solicitud `Scan`, consulte la documentación de referencia de Scan.

Llame a la API para escanear todas las publicaciones

Ahora que ya está configurado el solucionador, AWS AppSync ya sabe cómo convertir una consulta `allPost` entrante en una operación `Scan` de DynamoDB. Ahora puede recorrer la tabla para obtener todas las publicaciones.

Sin embargo, antes de probarlo, debe rellenar la tabla datos, ya que hemos eliminado las publicaciones con las que hemos estado trabajado hasta ahora.

- Seleccione la pestaña Queries (Consultas).
- En el panel Queries (Consultas), pegue la mutación siguiente:

```
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 }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).

Ahora recorreremos la tabla obteniendo cinco resultados cada vez.

- En el panel Queries (Consultas), pegue la siguiente consulta:

```
query allPost {
  allPost(count: 5) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- Las cinco primeras publicaciones deben aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

Hemos obtenido cinco resultados y el valor `nextToken`, que nos permite obtener el siguiente conjunto de resultados.

- Actualice la consulta `allPost` para incluir el valor de `nextToken` del conjunto de resultados anterior:

```
query allPost {
  allPost(
    count: 5
    nextToken:
 "eyJ2ZXJzaW9uIjoxLCJ0b2tlbiI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eThOSmRvcG9ad2RHYjI
  ) {
    posts {
      id
      author
    }
    nextToken
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- Las cuatro publicaciones restantes deben aparecer en el panel de resultados a la derecha del panel de consultas. No hay ningún `nextToken` en este conjunto, porque ya ha recorrido las nueve publicaciones y no quedan más. Debería parecerse a lo que sigue:

```
{
```

```
  "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
    }
  }
}
```

Configuración del solucionador allPostsByAuthor (Query de DynamoDB)

Además de escanear DynamoDB para obtener todas las publicaciones, también puede consultar DynamoDB para obtener las publicaciones creadas por un autor determinado. La tabla de DynamoDB que creó anteriormente ya tiene un `GlobalSecondaryIndex` llamado `author-index` que puede utilizar con una operación `Query` de DynamoDB para recuperar todas las publicaciones creadas por un autor determinado.

- Elija la pestaña Schema (Esquema).
- En el panel Schema (Esquema) modifique el tipo `Query` para agregar una nueva consulta `allPostsByAuthor`, de este modo:

```
type Query {
    allPostsByAuthor(author: String!, count: Int, nextToken: String): PaginatedPosts!
    allPost(count: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
```

```
}
```

Nota: aquí vuelve a usarse el tipo `PaginatedPosts` que empleamos en la consulta `allPost`.

- Seleccione Save.
- En el panel Data types (Tipos de datos) de la derecha, busque el campo allPostsByAuthor recién creado en el tipo Query y, a continuación, elija Attach (Asociar).
- En el menú Acción, seleccione Actualizar tiempo de ejecución y, a continuación, elija Solucionador de unidades (solo VTL).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "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
}
```

Al igual que el solucionador `allPost`, este solucionador tiene dos argumentos opcionales: `count`, que especifica el número máximo de elementos que se devolverá en una sola llamada, y `nextToken`, que se puede utilizar para recuperar el siguiente conjunto de resultados (el valor para `nextToken` se puede obtener en una llamada anterior).

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
{
```

```
    "posts": $utils.toJson($context.result.items)
    #if( ${context.result.nextToken} )
        ,"nextToken": $util.toJson($context.result.nextToken)
    #end
}
```

Nota: esta es la misma plantilla de mapeo de respuesta que usamos en el solucionador `allPost`.

- Seleccione Save.

Si desea más información sobre el mapeo de solicitud `Query`, consulte la documentación de referencia de Query.

Llame a la API para consultar todas las publicaciones de un autor

Ahora que se ha configurado el solucionador, AWS AppSync ya sabe cómo convertir una mutación `allPostsByAuthor` entrante en una operación `Query` de DynamoDB referida al índice `author-index`. Ahora puede consultar la tabla para recuperar todas las publicaciones de un autor determinado.

Sin embargo, antes de hacerlo, debemos rellenar la tabla con algunas publicaciones más, ya que por el momento todas son del mismo autor.

- Seleccione la pestaña Queries (Consultas).
- En el panel Queries (Consultas), pegue la mutación siguiente:

```
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 }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).

Ahora consultaremos la tabla para que devuelva todas las publicaciones creadas por `Nadia`.

- En el panel Queries (Consultas), pegue la siguiente consulta:

```
query allPostsByAuthor {
  allPostsByAuthor(author: "Nadia") {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- Todas las publicaciones creadas por `Nadia` deben aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you know...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

La paginación funciona con `Query` de igual modo que con `Scan`. Por ejemplo, vamos a buscar todas las publicaciones de `AUTHORNAME` y obtener cinco cada vez.

- En el panel Queries (Consultas), pegue la siguiente consulta:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
```

```
    count: 5
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- Todas las publicaciones creadas por `AUTHORNAME` deben aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
```

```
}
```

- Actualice el argumento `nextToken` con el valor devuelto en la consulta anterior, de este modo:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    count: 5
    nextToken:
 "eyJ2ZXJzaW9uIjoxLCJ0b2tlbiI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eThOSmRvcG9ad2RHYjI
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- Las demás publicaciones creadas por `AUTHORNAME` deben aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "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
    }
  }
}
```

Uso de conjuntos

Hasta ahora, el tipo `Post` era un objeto clave/valor plano. El solucionador AppSyncDynamoDB de AWS también permite modelar objetos complejos, como conjuntos, listas y mapas.

Vamos a actualizar el tipo `Post` para incluir etiquetas. Una publicación puede tener cero o más etiquetas, que se almacenan en DynamoDB como un conjunto de cadenas. También configuraremos algunas mutaciones para añadir y eliminar etiquetas, y una nueva consulta para buscar las publicaciones con una etiqueta concreta.

- Elija la pestaña Schema (Esquema).
- En el panel Schema (Esquema) modifique el tipo `Post` para agregar un nuevo campo `tags`, de este modo:

```
type Post {
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
  tags: [String!]
}
```

- En el panel Schema (Esquema) modifique el tipo `Query` para agregar una nueva consulta `allPostsByTag`, de este modo:

```
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
```

```
}
```

- En el panel Schema (Schema), modifique el tipo `Mutation` para agregar las nuevas mutaciones `addTag` y `removeTag`, de este modo:

```
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!
}
```

- Seleccione Save.
- En el panel Data types (Tipos de datos) de la derecha, busque el campo allPostsByTag recién creado en el tipo Query y, a continuación, elija Attach (Asociar).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "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
}
```

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
{
    "posts": $utils.toJson($context.result.items)
    #if( ${context.result.nextToken} )
        ,"nextToken": $util.toJson($context.result.nextToken)
    #end
}
```

- Seleccione Save.
- En el panel Data types (Tipos de datos) de la derecha, busque el campo addTag recién creado en el tipo Mutation y, a continuación, elija Attach (Asociar).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "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 }
        }
    }
}
```

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
$utils.toJson($context.result)
```

- Seleccione Save.
- En el panel Data types (Tipos de datos) de la derecha, busque el campo removeTag recién creado en el tipo Mutation y, a continuación, elija Attach (Asociar).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
    "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 }
        }
    }
}
```

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
$utils.toJson($context.result)
```

- Seleccione Save.

Llame a la API para trabajar con etiquetas

Ahora que ha configurado los solucionadores, AWS AppSync sabe cómo convertir solicitudes `addTag`, `removeTag` y `allPostsByTag` entrantes en operaciones `Scan` y `UpdateItem` de DynamoDB.

Para probarlo, vamos a seleccionar una de las publicaciones que ha creado anteriormente. Por ejemplo, tomemos una publicación creada por `Nadia`.

- Seleccione la pestaña Queries (Consultas).
- En el panel Queries (Consultas), pegue la siguiente consulta:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "Nadia"
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- Todas las publicaciones creadas por Nadia deben aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you known...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

- Usemos el que tiene el título `"The cutest dog in the world"`. Anote su `id`, ya que vamos a utilizarlo más adelante.

Ahora probemos a añadirle la etiqueta `dog`.

- En el panel Queries (Consultas), pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente.

```
mutation addTag {
  addTag(id:10 tag: "dog") {
    id
    title
    tags
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- La publicación se actualiza con la nueva etiqueta.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

Puede agregar más etiquetas, de este modo:

- Actualice la mutación para cambiar el argumento `tag` a `puppy`.

```
mutation addTag {
  addTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).

- La publicación se actualiza con la nueva etiqueta.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog",
        "puppy"
      ]
    }
  }
}
```

También puede eliminar etiquetas:

- En el panel Queries (Consultas), pegue la mutación siguiente. También tendrá que actualizar el argumento `id` con el valor que anotó anteriormente.

```
mutation removeTag {
  removeTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- La publicación se actualiza y la etiqueta puppy se elimina.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
```

```
}
```

También puede buscar todas las publicaciones que tengan una etiqueta:

- En el panel Queries (Consultas), pegue la siguiente consulta:

```
query allPostsByTag {
  allPostsByTag(tag: "dog") {
    posts {
      id
      title
      tags
    }
    nextToken
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- Se devolverán todas las publicaciones que tenga la etiqueta dog, de este modo:

```
{
  "data": {
    "allPostsByTag": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world",
          "tags": [
            "dog",
            "puppy"
          ]
        }
      ],
      "nextToken": null
    }
  }
}
```

Uso de listas y mapas

Además de usar conjuntos de DynamoDB, también puede usar las listas y mapas de DynamoDB para modelar datos complejos con un único objeto.

Vamos a añadir la capacidad de agregar comentarios a las publicaciones. Esto se modelará como una lista de objetos de mapa para el objeto `Post` en DynamoDB.

Nota: en una aplicación real, los comentarios se incluirían en su propia tabla. Para este tutorial, nos limitaremos a añadirlos a la tabla `Post`.

- Elija la pestaña Schema (Esquema).
- En el panel Schema (Esquema), agregue un nuevo tipo `Comment` de este modo:

```
type Comment {
    author: String!
    comment: String!
}
```

- En el panel Schema (Esquema) modifique el tipo `Post` para agregar un nuevo campo `comments`, de este modo:

```
type Post {
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
  tags: [String!]
  comments: [Comment!]
}
```

- En el panel Schema (Esquema) modifique el tipo `Mutation` para agregar una nueva mutación `addComment` de este modo:

```
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!
}
```

- Seleccione Save.
- En el panel Data types (Tipos de datos) de la derecha, busque el campo addComment recién creado en el tipo Mutation y, a continuación, elija Attach (Asociar).
- En Data source name (Nombre del tipo de datos), elija PostDynamoDBTable.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue lo siguiente:

```
{
  "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)
    }
  }
}
```

Esta expresión de actualización agregará una lista con nuestro nuevo comentario a la lista `comments` existente. Si la lista no existe todavía, se creará.

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue lo siguiente:

```
$utils.toJson($context.result)
```

- Seleccione Save.

Llame a la API para añadir un comentario

Ahora que ha configurado los solucionadores, AWS AppSync sabe cómo convertir solicitudes `addComment` entrantes en operaciones `UpdateItem` de DynamoDB.

Vamos a probarlo añadiendo un comentario a la misma publicación a la que añadimos las etiquetas.

- Seleccione la pestaña Queries (Consultas).
- En el panel Queries (Consultas), pegue la siguiente consulta:

```
mutation addComment {
  addComment(
    id:10
    author: "Steve"
    comment: "Such a cute dog."
  ) {
    id
    comments {
      author
      comment
    }
  }
}
```

- Elija Execute query (Ejecutar consulta) (el botón de reproducción naranja).
- Todas las publicaciones creadas por Nadia deben aparecer en el panel de resultados a la derecha del panel de consultas. Debería parecerse a lo que sigue:

```
{
  "data": {
    "addComment": {
      "id": "10",
      "comments": [
        {
          "author": "Steve",
          "comment": "Such a cute dog."
        }
      ]
    }
  }
}
```

Si ejecuta la solicitud varias veces, se agregarán varios comentarios a la lista.

Conclusión

En este tutorial ha creado una API que nos permite manipular objetos Post en DynamoDB mediante AWS AppSync y GraphQL. Para obtener más información, consulte la Referencia de plantillas de mapeo de solucionador para Elasticsearch.

Para limpiar, puede eliminar la API de GraphQL de AppSync de la consola.

Para eliminar la tabla de DynamoDB y el rol de IAM que ha creado para este tutorial, puede ejecutar lo siguiente para eliminar la pila `AWSAppSyncTutorialForAmazonDynamoDB` o ir a la consola de AWS CloudFormation y eliminar la pila:

```
aws cloudformation delete-stack \
    --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

Tutorial: Solucionadores de Lambda

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

Puede utilizar AWS Lambda con AWS AppSync para resolver cualquier campo de GraphQL. Por ejemplo, una consulta de GraphQL podría enviar una llamada a una instancia de Amazon Relational Database Service (Amazon RDS), y una mutación de GraphQL podría escribir en un flujo de Amazon Kinesis. En esta sección, veremos cómo puede escribir una función de lambda que ejecute la lógica de negocio en función de la invocación de una operación de campo de GraphQL.

Creación de una función de Lambda

En el siguiente ejemplo se muestra una función de Lambda escrita en `Node.js` que realiza distintas operaciones con publicaciones de blogs como parte de una aplicación de publicaciones de 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;
    }
};
```

Esta función de Lambda recupera una publicación por identificador, añade una publicación, recupera una lista de publicaciones y recupera publicaciones relacionadas para una publicación determinada.

Nota: La función de Lambda utiliza la instrucción `switch` en `event.field` para determinar qué campo se está resolviendo en ese momento.

Cree esta función de Lambda mediante la consola de administración de AWS o una pila de AWS CloudFormation. Para crear la función a partir de una pila de CloudFormation, puede usar el siguiente comando AWS Command Line Interface (AWS CLI):

```
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
```

Puede iniciar esta pila de AWS CloudFormation en la región de AWS Oeste de EE. UU. (Oregón) en su cuenta de AWS:

Configure un origen de datos para Lambda

Una vez creada la función de Lambda, vaya a la API de GraphQL en la consola de AWS AppSync y elija la pestaña Orígenes de datos.

Elija Crear origen de datos, introduzca un Nombre de origen de datos fácil de recordar (por ejemplo, **`Lambda`**) y, a continuación, en Tipo de origen de datos, elija Función de AWS Lambda. En Región, elija la misma región que en su función. (Si creó la función a partir de la pila de CloudFormation proporcionada, es probable que la función esté en US-WEST-2). En ARN de función, elija el nombre de recurso de Amazon (ARN) para la función de Lambda.

Una vez seleccionada la función de Lambda, puede crear un nuevo rol de AWS Identity and Access Management (IAM) (al que AWS AppSync asignará los permisos adecuados) o bien elegir uno ya existente que tenga la política en línea siguiente:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
```

```
            ],
            "Resource": "arn:aws:lambda:REGION:ACCOUNTNUMBER:function/LAMBDA_FUNCTION"
        }
    ]
}
```

También debe configurar una relación de confianza con AWS AppSync para el rol de IAM, de este modo:

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

Cree un esquema de GraphQL

Ahora que el origen de datos está conectado a la función de Lambda, cree un esquema de GraphQL.

En el editor de esquemas de la consola de AWS AppSync, asegúrese de que su esquema coincida con el siguiente:

```
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]
}
```

Configure solucionadores

Ahora que ha registrado un origen de datos de Lambda y un esquema de GraphQL válido, puede conectar sus campos de GraphQL al origen de datos de Lambda utilizando solucionadores.

Para crear un solucionador, necesitará plantillas de mapeo. Para obtener más información sobre las plantillas de mapeo, consulte Resolver Mapping Template Overview.

Para obtener más información sobre las plantillas de mapeo de Lambda, consulte Resolver mapping template reference for Lambda.

En este paso, debe asociar un solucionador a la función de Lambda para los siguientes campos: `getPost(id:ID!): Post`, `allPosts: [Post]`, `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!` y `Post.relatedPosts: [Post]`.

En el editor de esquemas de la consola de AWS AppSync, elija a la derecha Asociar solucionador para `getPost(id:ID!): Post`.

Luego, en el menú Acción, seleccione Actualizar tiempo de ejecución y, a continuación, elija Solucionador de unidades (solo VTL).

Después, elija el origen de datos de Lambda. En la sección de la plantilla de mapeo de solicitud, elija Invoke And Forward Argumentos (Invocar y reenviar argumentos).

Modifique el objeto `payload` para agregar el nombre de campo. La plantilla debe tener el aspecto siguiente:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments":  $utils.toJson($context.arguments)
    }
}
```

En la sección Plantilla de mapeo de respuestas, elija Devolver resultado Lambda.

En este caso utilice la plantilla base tal y como está. Debe parecerse a lo siguiente:

```
$utils.toJson($context.result)
```

Seleccione Save. Acaba de asociar su primer solucionador. Repita esta operación para el resto de los campos como se indica a continuación:

Para la plantilla de mapeo de solicitudes `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!`:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "addPost",
        "arguments":  $utils.toJson($context.arguments)
    }
}
```

Para la plantilla de mapeo de respuestas `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!`:

```
$utils.toJson($context.result)
```

Para la plantilla de mapeo de solicitudes `allPosts: [Post]`:

```
{
    "version": "2017-02-28",
```

```
    "operation": "Invoke",
    "payload": {
        "field": "allPosts"
    }
}
```

Para la plantilla de mapeo de respuestas `allPosts: [Post]`:

```
$utils.toJson($context.result)
```

Para la plantilla de mapeo de solicitudes `Post.relatedPosts: [Post]`:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "relatedPosts",
        "source":  $utils.toJson($context.source)
    }
}
```

Para la plantilla de mapeo de respuestas `Post.relatedPosts: [Post]`:

```
$utils.toJson($context.result)
```

Pruebe la API de GraphQL

Ahora que la función de Lambda está conectada a los solucionadores de GraphQL, puede ejecutar algunas mutaciones y consultas con la consola o una aplicación cliente.

A la izquierda de la consola de AWS AppSync, elija la pestaña Consultas y pegue el código siguiente:

Mutación addPost

```
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
    }
}
```

Consulta getPost

```
query getPost {
    getPost(id: "2") {
        id
        author
        title
        content
        url
        ups
        downs
    }
}
```

Consulta allPosts

```
query allPosts {
    allPosts {
        id
        author
        title
        content
        url
        ups
        downs
        relatedPosts {
            id
            title
        }
```

```
    }
}
```

Devolución de errores

Cualquier resolución de campo dada puede producir un error. Con AWS AppSync, puede generar errores de los orígenes siguientes:

- Plantilla de mapeo de solicitudes o de respuestas
- Lambda function

Desde la plantilla de mapeo

Para generar errores intencionados, puede utilizar el método de ayuda `$utils.error` de la plantilla de Velocity Template Language (VTL). Indique como argumentos un mensaje en `errorMessage`, un tipo en `errorType` y un valor opcional en `data`. El argumento `data` es útil para devolver datos adicionales al cliente cuando se produce un error. El objeto `data` se añade a `errors` en la respuesta final de GraphQL.

En el siguiente ejemplo se muestra cómo usarlo en la plantilla de mapeo de respuestas `Post.relatedPosts: [Post]`:

```
$utils.error("Failed to fetch relatedPosts", "LambdaFailure", $context.result)
```

Así se obtiene una respuesta de GraphQL similar a la siguiente:

```
{
    "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"
                }
            ]
        }
    ]
}
```

donde `allPosts[0].relatedPosts` es null debido al error y `errorMessage`, `errorType` y `data` se incluyen en el objeto `data.errors[0]`.

Desde la función de Lambda

AWS AppSync también entiende los errores que produce la función de Lambda. El modelo de programación de Lambda permite generar errores gestionados. Si la función de lambda produce un error, AWS AppSync no puede resolver el campo actual. La respuesta solo incluirá el mensaje de error que devuelva Lambda. Actualmente no es posible devolver datos adicionales al cliente generando un error desde la función de Lambda.

Nota: Si su función de Lambda genera un error no gestionado, AWS AppSync utiliza el mensaje de error que Lambda estableció.

La siguiente función de Lambda genera un error:

```
exports.handler = (event, context, callback) => {
    console.log("Received event {}", JSON.stringify(event, 3));
    callback("I fail. Always.");
};
```

Así se obtiene una respuesta de GraphQL similar a la siguiente:

```
{
    "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."
        }
    ]
}
```

Caso de uso avanzado: agrupación en lotes

La función de Lambda de este ejemplo tiene un campo `relatedPosts` que devuelve una lista de publicaciones relacionadas para una publicación determinada. En las consultas del ejemplo, la invocación al campo `allPosts` desde la función de Lambda devuelve cinco publicaciones. Dado

que hemos especificado que también queremos resolver `relatedPosts` para cada publicación obtenida, la operación del campo `relatedPosts` se invoca cinco veces.

```
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
        }
    }
}
```

Aunque no parezca mucho en este ejemplo concreto, esta sobrecarga compuesta puede perjudicar rápidamente a la aplicación.

Si quisiéramos obtener `relatedPosts` otra vez para todos los elementos de `Posts` en la misma consulta, el número de invocaciones aumentaría exponencialmente.

```
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
            }
```

```
        }
    }
}
```

En esta consulta relativamente sencilla, AWS AppSync invocaría la función de Lambda 1 + 5 + 25 = 31 veces.

Se trata de una situación bastante habitual que a menudo se denomina "problema N+1", (en nuestro caso, N = 5) y puede causar un aumento de la latencia y del costo de la aplicación.

Una forma de solucionarlo es agrupar por lotes las solicitudes de solucionador de campo similares. En este ejemplo, en lugar de hacer que la función de Lambda obtenga una lista de publicaciones relacionadas con una publicación individual determinada, hacemos que obtenga una lista de publicaciones relacionadas con un lote de publicaciones dado.

Para ilustrarlo modificaremos el solucionador `Post.relatedPosts: [Post]` de modo que utilice lotes.

A la derecha de la consola de AWS AppSync, elija el solucionador `Post.relatedPosts: [Post]` existente. Cambie la plantilla de mapeo de solicitud por lo siguiente:

```
{
    "version": "2017-02-28",
    "operation": "BatchInvoke",
    "payload": {
        "field": "relatedPosts",
        "source":  $utils.toJson($context.source)
    }
}
```

Observe que solo cambia el campo `operation` de `Invoke` a `BatchInvoke`. El campo de carga es ahora una matriz que contiene lo que especifique la plantilla. En este ejemplo, la función de Lambda recibe lo siguiente como entrada:

```
[
    {
        "field": "relatedPosts",
        "source": {
            "id": 1
        }
    },
```

```
    {
        "field": "relatedPosts",
        "source": {
            "id": 2
        }
    },
    ...
]
```

Cuando se especifica `BatchInvoke` en la plantilla de mapeo de solicitudes, la función de Lambda recibe una lista de solicitudes y devuelve una lista de resultados.

En concreto, la lista de resultados debe coincidir en tamaño y orden con las entradas de la carga de la solicitud, por lo que AWS AppSync puede hacer coincidir los resultados como corresponda.

En este ejemplo de agrupación en lotes, la función de Lambda devuelve un lote de resultados de este modo:

```
[
    [{"id":"2","title":"Second book"}, {"id":"3","title":"Third book"}],   //
 relatedPosts for id=1
    [{"id":"3","title":"Third book"}]
           // relatedPosts for id=2
]
```

La función de Lambda siguiente escrita en Node.js ilustra esta funcionalidad de agrupación en lotes para el campo `Post.relatedPosts`:

```
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;
    }
};
```

Devolución de errores individuales

Los ejemplos anteriores muestran cómo devolver un único error desde la función de Lambda o generar un error desde las plantillas de mapeo. En las invocaciones en lotes, la generación de un error desde la función de Lambda marca como fallido todo el lote. Esto puede ser adecuado en situaciones concretas donde se haya producido un error irrecuperable, como, por ejemplo, un error de conexión a un almacén de datos. Sin embargo, en los casos en los que algunos elementos

del lote se ejecutan correctamente y otros fallan, es posible devolver tanto los errores como los datos válidos. Puesto que AWS AppSync requiere que en la respuesta por lotes se enumeren los elementos que coinciden con el tamaño original del lote, debe definir una estructura de datos que pueda diferenciar los datos válidos de un error.

Por ejemplo, si se espera que la función de Lambda devuelva un lote de publicaciones relacionadas, podría optar por devolver una lista de objetos `Response` en la que cada objeto tenga campos opcionales data, errorMessage y errorType. Si el campo errorMessage está presente, significa que se ha producido un error.

El código siguiente muestra cómo podría actualizar la función de 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;
    }
};
```

En este ejemplo, la plantilla de mapeo de respuestas siguiente analiza cada elemento de la función de Lambda y genera los errores que se produzcan:

```
#if( $context.result && $context.result.errorMessage )
    $utils.error($context.result.errorMessage, $context.result.errorType,
 $context.result.data)
#else
    $utils.toJson($context.result.data)
#end
```

Este ejemplo devuelve una respuesta de GraphQL similar a la siguiente:

```
{
  "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"
    }
  ]
}
```

Configuración del tamaño máximo de agrupación en lotes

De forma predeterminada, cuando se usa `BatchInvoke`, AWS AppSync envía solicitudes a la función de Lambda en lotes de hasta cinco elementos. Puede configurar el tamaño máximo de lote de sus solucionadores de Lambda.

Para configurar el tamaño máximo de agrupación en lotes en un solucionador, utilice el siguiente comando en la 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**
>
> Al proporcionar una plantilla de mapeo de solicitudes, debe usar la operación `BatchInvoke` para usar la agrupación en lotes.

También puede utilizar el siguiente comando para habilitar y configurar la agrupación en lotes en solucionadores de Direct Lambda:

```
$ aws appsync create-resolver --api-id <api-id> --type-name Query --field-name
 relatedPosts \
 --data-source-name "<lambda-datasource>" \
 --max-batch-size X
```

Configuración del tamaño máximo de agrupación en lotes con plantillas de VTL

Para los solucionadores de Lambda que tienen plantillas en solicitudes de VTL, el tamaño máximo del lote no tendrá ningún efecto a menos que lo hayan especificado directamente como una operación `BatchInvoke` en VTL. Del mismo modo, si realiza una mutación de nivel superior, no se realiza la agrupación en lotes de las mutaciones porque la especificación GraphQL requiere que las mutaciones paralelas se ejecuten secuencialmente.

Por ejemplo, tomemos las mutaciones siguientes:

```
type Mutation {
    putItem(input: Item): Item
    putItems(inputs: [Item]): [Item]
}
```

Con la primera mutación, podemos crear 10 `Items`, como se muestra en el siguiente fragmento:

```
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
    }
}
```

En este ejemplo, los `Items` no se agruparán en un grupo de 10 aunque el tamaño máximo del lote esté establecido en 10 en el solucionador de Lambda. En su lugar, se ejecutarán secuencialmente de acuerdo con la especificación GraphQL.

Para realizar una mutación por lotes real, puede seguir el siguiente ejemplo utilizando la segunda mutación:

```
mutation MyMutation {
    putItems([$someItem1, $someItem2, $someItem3,$someItem4, $someItem5, $someItem6,
```

```
    $someItem7, $someItem8, $someItem9, $someItem10]) {
    id,
    name
    }
}
```

Para obtener más información sobre el uso de agrupaciones en lotes con solucionadores de Direct Lambda, consulte Solucionadores de Lambda directos.

Tutorial: Solucionadores de Amazon OpenSearch Service

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSync admite el uso de Amazon OpenSearch Service desde dominios que ha aprovisionado en su cuenta de AWS, siempre que no se encuentren dentro de una VPC. Después de aprovisionar los dominios, puede conectarse a ellos con un origen de datos. En ese momento, puede configurar un solucionador en el esquema para que realice operaciones de GraphQL como consultas, mutaciones y suscripciones. Este tutorial le guiará a lo largo de algunos ejemplos comunes.

Para obtener más información, consulte Resolver Mapping Template Reference for OpenSearch.

Configuración en un clic

Para configurar automáticamente un punto de conexión de GraphQL en AWS AppSync con Amazon OpenSearch Service configurado, puede utilizar esta plantilla de AWS CloudFormation:

Una vez completada la implementación de AWS CloudFormation, puede pasar directamente a ejecutar consultas y mutaciones de GraphQL.

Cree un nuevo dominio de OpenSearch Service

Para comenzar este tutorial, es necesario disponer de un dominio de OpenSearch Service. Si todavía no tiene uno, puede utilizar la siguiente muestra. Tenga en cuenta que crear un dominio de

OpenSearch Service puede tardar hasta 15 minutos, por lo que deberá esperar para poder integrarlo con un origen de datos de AWS AppSync.

```
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
```

Como alternativa, puede iniciar la siguiente pila de AWS CloudFormation en la región EE. UU. Oeste 2 (Oregón) de su cuenta de AWS:

Configure el origen de datos para OpenSearch Service

Después de crear el dominio de OpenSearch Service, vaya a la API de GraphQL para AWS AppSync y elija la pestaña Orígenes de datos. Elija Nuevo y escriba un nombre fácil de recordar para el origen de datos, por ejemplo “oss”. A continuación, elija Dominio de Amazon OpenSearch en Tipo de origen de datos y elija la región que corresponda. En la lista debe aparecer el dominio de OpenSearch Service. Una vez seleccionado, puede crear un nuevo rol y AWS AppSync le asignará los permisos adecuados. También puede elegir un rol existente que tenga la siguiente política en línea:

```
{
    "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/*"
            ]
        }
```

```
    ]
}
```

También debe configurar una relación de confianza con AWS AppSync para ese rol:

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

Además, el dominio de OpenSearch Service tiene su propia política de acceso, que puede modificar en la consola de Amazon OpenSearch Service. Tendrá que añadir una política similar a la que aparece a continuación, con las acciones y recursos adecuados para el dominio de OpenSearch Service. Tenga en cuenta que el Principal será el rol del origen de datos de AppSync, que si deja que la consola lo cree, podrá encontrar en la consola de 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/*"
        }
```

```
    ]
}
```

Conexión de un solucionador

Ahora que el origen de datos está conectado a su dominio de OpenSearch Service, puede conectarlo también al esquema de GraphQL con un solucionador, como se muestra en el ejemplo siguiente:

```
 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
}
...
```

Observe que hay un tipo `Post` definido por el usuario con un campo `id`. En los siguientes ejemplos, supondremos que existe un proceso (que se puede automatizar) para incluir este tipo en el dominio de OpenSearch Service, lo que lo mapea a la ruta raíz de `/post/_doc`, donde `post` es el índice. A partir de esta ruta raíz, puede realizar búsquedas de documentos individuales, búsquedas de comodín con `/id/post*` o búsquedas en varios documentos con la ruta `/post/_search`. Por ejemplo, si tiene otro tipo llamado `User`, puede indexar documentos bajo un nuevo índice llamado `user`, y luego realizar búsquedas con una ruta de `/user/_search`.

En el editor de esquemas de la consola de AWS AppSync, modifique el esquema `Posts` anterior para que incluya una consulta `searchPosts`:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  searchPosts: [Post]
}
```

Guarde el esquema. En la parte derecha, en `searchPosts`, elija Attach resolver (Asociar solucionador). En el menú Acción, seleccione Actualizar tiempo de ejecución y, a continuación, elija Solucionador de unidades (solo VTL). A continuación, elija el origen de datos de OpenSearch Service. En la sección de plantilla de mapeo de solicitud, seleccione el menú desplegable de Query posts (Consultar publicaciones) para obtener una plantilla base. Modifique la `path` para que sea `/post/_search`. Debe parecerse a lo siguiente:

```
{
    "version":"2017-02-28",
    "operation":"GET",
    "path":"/post/_search",
    "params":{
        "headers":{},
        "queryString":{},
        "body":{
            "from":0,
            "size":50
        }
    }
}
```

Esto supone que el esquema anterior contiene documentos que se han indexado en OpenSearch Service en el campo `post`. Si estructura los datos de manera diferente, tendrá que realizar una actualización como corresponda.

En la sección Plantilla de mapeo de respuestas, debe especificar el filtro `_source` adecuado para obtener los resultados de una consulta de OpenSearch Service y convertirlos a GraphQL. Utilice la plantilla siguiente:

```
[
    #foreach($entry in $context.result.hits.hits)
```

```
    #if( $velocityCount > 1 ) , #end
    $utils.toJson($entry.get("_source"))
    #end
]
```

Modificación de las búsquedas

La plantilla de mapeo de solicitud anterior realiza una consulta sencilla de todos los registros. Supongamos que desea buscar por un autor específico. Además, supongamos que desea que ese autor sea un argumento definido en la consulta de GraphQL. En el editor de esquemas de la consola de AWS AppSync, añada una consulta `allPostsByAuthor`:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  allPostsByAuthor(author: String!): [Post]
  searchPosts: [Post]
}
```

Ahora elija Asociar solucionador y seleccione el origen de datos de OpenSearch Service, pero usando el siguiente ejemplo como plantilla de mapeo de respuestas:

```
{
    "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)
                }
            }
        }
    }
}
```

Observe que body se llena con una consulta de término para el campo `author`, que se pasa desde el cliente como un argumento. Si lo desea, puede tener información previamente rellenada, como texto estándar, o incluso utilizar otras utilidades.

Si va a utilizar este solucionador, rellene la plantilla de mapeo de respuesta con la misma información que en el ejemplo anterior.

Adición de datos a OpenSearch Service

Puede ser conveniente añadir datos al dominio de OpenSearch Service como resultado de una mutación de GraphQL. Se trata de un eficaz mecanismo para realizar búsquedas y para otros fines. Dado que puede utilizar suscripciones de GraphQL para que los datos se obtengan en tiempo real, esto sirve como mecanismo para avisar a los clientes de actualizaciones de datos en el dominio de OpenSearch Service.

Vuelva a la página Esquema de la consola de AWS y seleccione Asociar solucionador para la mutación `addPost()`. Seleccione de nuevo el origen de datos de OpenSearch Service y use la siguiente plantilla de mapeo de respuestas para el esquema `Posts`:

```
{
    "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)
        }
    }
}
```

Como antes, este es un ejemplo de cómo pueden estar estructurados los datos. Si los nombres de campos o índices son distintos, debe actualizar la `path` y `body` como corresponda. Este

ejemplo también muestra cómo usar `$context.arguments` para rellenar la plantilla a partir de los argumentos de mutación de GraphQL.

Antes de continuar, utilice la siguiente plantilla de mapeo de respuestas, que devolverá el resultado de la operación de mutación o la información del error como salida:

```
#if($context.error)
    $util.toJson($ctx.error)
#else
    $util.toJson($context.result)
#end
```

Recuperación de un solo documento

Por último, si desea utilizar la consulta `getPost(id:ID)` en su esquema para obtener un documento individual, encuentre esta consulta en el editor de esquemas de la consola de AWS AppSync y seleccione Asociar solucionador. Seleccione de nuevo el origen de datos de OpenSearch Service y use la siguiente plantilla de mapeo:

```
{
    "version":"2017-02-28",
    "operation":"GET",
    "path": $util.toJson("post/_doc/$context.arguments.id"),
    "params":{
        "headers":{},
        "queryString":{},
        "body":{}
    }
}
```

Dado que el valor de `path` anterior utiliza el argumento `id` con un cuerpo vacío, esto devuelve el documento individual. Sin embargo, debe utilizar la siguiente plantilla de mapeo de respuesta, ya que ahora va a obtener un único elemento y no una lista:

```
$utils.toJson($context.result.get("_source"))
```

Ejecución de consultas y mutaciones

Ahora puede realizar operaciones de GraphQL referidas al dominio de OpenSearch Service. Vaya a la pestaña Consultas de la consola de AWS y añada un nuevo registro:

```
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
       )
}
```

Verá el resultado de la mutación a la derecha. Del mismo modo, ahora puede ejecutar una consulta `searchPosts` en el dominio de OpenSearch Service:

```
query searchPosts {
    searchPosts {
        id
        title
        author
        content
    }
}
```

Prácticas recomendadas

- OpenSearch Service se debe utilizar para consultar datos, no como base de datos principal. Puede ser útil emplear OpenSearch Service junto con Amazon DynamoDB como se indica en Combinación de solucionadores de GraphQL.
- Solo debe conceder acceso al dominio permitiendo que el rol de servicio de AWS AppSync tenga acceso al clúster.
- Puede comenzar con un pequeño desarrollo, con el clúster de menor costo y, a continuación, pasar a un clúster de mayor tamaño con alta disponibilidad (HA) al pasar a producción.

Tutorial: Solucionadores locales

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSync le permite utilizar orígenes de datos compatibles (AWS Lambda, Amazon DynamoDB o Amazon OpenSearch Service) para realizar diversas operaciones. Sin embargo, en determinados casos, es posible que no sea necesario realizar una llamada a un origen de datos admitido.

Aquí es donde un solucionador local es útil. En lugar de llamar a un origen de datos remoto, el solucionador local simplemente reenviará los resultados de la plantilla de mapeo de solicitudes a la plantilla de mapeo de respuestas. La resolución de campo no sale de AWS AppSync.

Los solucionadores locales son útiles para varios casos de uso. El caso de uso más habitual consiste en publicar notificaciones sin activar una llamada de origen de datos. Para demostrar este caso de uso vamos a crear una aplicación buscapersonas con la que los usuarios puedan enviarse notificaciones. Este ejemplo utiliza suscripciones, de modo que si no está familiarizado con las suscripciones, puede seguir el tutorial de datos en tiempo real.

Creación de la aplicación buscapersonas

En nuestra aplicación buscapersonas, los clientes pueden suscribirse a una bandeja de entrada y enviar notificaciones a otros clientes. Cada notificación contiene un mensaje. El esquema es el siguiente:

```
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
}
```

Asociemos un solucionador al campo `Mutation.page`. En el panel Schema (Esquema), haga clic en Attach Resolver (Asociar solucionador) junto a la definición de campo en el panel derecho. Cree un nuevo origen de datos de tipo None (Ninguno) y asígnele el nombre PageDataSource.

Para la plantilla de mapeo de solicitud, escriba:

```
{
  "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()"
  }
}
```

Y para la plantilla de mapeo de respuesta, seleccione la opción predeterminada Forward the result (Reenviar el resultado). Guarde el solucionador. Su aplicación ya está lista, ya puede ejecutarla.

Envío y suscripción a notificaciones

Para que los clientes reciban notificaciones, primero deben estar suscritos a una bandeja de entrada.

En el panel Queries (Consultas) vamos a ejecutar la suscripción `inbox`:

```
subscription Inbox {
```

```
    inbox(to: "Nadia") {
        body
        to
        from
        sentAt
    }
}
```

Nadia recibirá notificaciones siempre que se invoque la mutación `Mutation.page`. Vamos a invocar la mutación ejecutándola:

```
mutation Page {
    page(to: "Nadia", body: "Hello, World!") {
        body
        to
        from
        sentAt
    }
}
```

Acabamos de demostrar el uso de solucionadores locales enviando una notificación y recibiéndola sin salir de AWS AppSync.

Tutorial: Combinación de solucionadores de GraphQL

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

Los solucionadores y los campos de un esquema de GraphQL mantienen una relación 1:1 con un alto grado de flexibilidad. Puesto que se configura un origen de datos para cada solucionador independientemente de un esquema, tiene la posibilidad de resolver o manipular los tipos de GraphQL mediante orígenes de datos diferentes, combinándolos en el esquema que mejor se adapte a sus necesidades.

Los siguientes escenarios de ejemplo muestran cómo mezclar y hacer corresponder los orígenes de datos en su esquema. Antes de empezar, le recomendamos que esté familiarizado con la

configuración de orígenes de datos y solucionadores para AWS Lambda Amazon DynamoDB y Amazon OpenSearch Service, tal y como se describe en los tutoriales anteriores.

Esquema de ejemplo

El siguiente esquema tiene un tipo `Post` con 3 operaciones `Query` y 3 operaciones `Mutation` definidas:

```
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
}
```

En este ejemplo tendría que asociar un total de 6 solucionadores. Una opción sería hacer que todas procedieran de una tabla de Amazon DynamoDB, denominada `Posts`, donde `AllPosts` ejecuta un análisis y `searchPosts` ejecuta una consulta, tal y como se describe en DynamoDB Resolver Mapping Template Reference. No obstante, existen alternativas para satisfacer las necesidades de su empresa, como hacer que estas consultas de GraphQL se resuelvan desde Lambda u OpenSearch Service.

Modificación de datos mediante solucionadores

Puede que sea necesario enviar a los clientes resultados de una base de datos como DynamoDB (o Amazon Aurora) con algunos de los atributos modificados. Esto puede deberse al formato de los tipos de datos, por ejemplo diferencias de marcas de tiempo en los clientes, o para gestionar problemas de compatibilidad con versiones anteriores. A título ilustrativo, en el siguiente ejemplo mostramos una función AWS Lambda que manipula las votaciones a favor y en contra de las publicaciones del blog asignándoles números aleatorios cada vez que se invoca el solucionador de GraphQL:

```
'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);
    });
};
```

Se trata de una función de Lambda perfectamente válida y puede asociarse al campo `AllPosts` del esquema de GraphQL para que cualquier consulta que devuelva todos los resultados obtenga números aleatorios para los votos a favor y en contra.

DynamoDB y OpenSearch Service

En algunas aplicaciones, puede realizar mutaciones o consultas de búsqueda sencilla en DynamoDB y tener un proceso en segundo plano para transferir documentos a OpenSearch Service. Luego, podría simplemente asociar el solucionador `searchPosts` al origen de datos de OpenSearch Service y devolver los resultados de la búsqueda (a partir de los datos originados en DynamoDB) mediante una consulta de GraphQL. Esto puede ser extremadamente útil al añadir operaciones de búsqueda avanzada a sus aplicaciones, como palabras clave, coincidencias parciales o incluso búsquedas geoespaciales. La transferencia de datos desde DynamoDB puede realizarse mediante un proceso ETL o, alternativamente, puede transmitir desde DynamoDB mediante Lambda. Puede lanzar un ejemplo completo con la siguiente pila de AWS CloudFormation en la región EE. UU. Oeste 2 (Oregón) en su cuenta de AWS:

El esquema de este ejemplo le permite añadir publicaciones mediante un solucionador de DynamoDB, de este modo:

```
mutation add {
    putPost(author:"Nadia"
        title:"My first post"
        content:"This is some test content"
        url:"https://aws.amazon.com/appsync/"
    ){
        id
        title
    }
}
```

Esto escribe los datos en DynamoDB, que los transmite a través de Lambda a Amazon OpenSearch Service, donde puede buscar todas las publicaciones por diferentes campos. Por ejemplo, como los datos se encuentran en Amazon OpenSearch Service, puede buscar los campos de autor o contenido con texto libre, incluso con espacios, del siguiente modo:

```
query searchName{
```

```
    searchAuthor(name:"   Nadia   "){
        id
        title
        content
    }
}

query searchContent{
    searchContent(text:"test"){
        id
        title
        content
    }
}
```

Como los datos se escriben directamente en DynamoDB, aún puede realizar operaciones eficientes de búsqueda de lista o de elemento en la tabla con las consultas `allPosts{...}` y `singlePost{...}`. Esta pila utiliza el siguiente código de ejemplo para los flujos de DynamoDB:

Nota: este código es solo un ejemplo.

```
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);
};
```

Puede utilizar flujos de DynamoDB para asociarlo a una tabla de DynamoDB con la clave principal `id`, y cualquier cambio en el origen de DynamoDB se transmitirá al dominio de OpenSearch Service. Para obtener más información sobre cómo configurarlo, consulte la documentación de flujos de DynamoDB.

Tutorial: Solucionadores por lotes de DynamoDB

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSync admite el uso de operaciones por lotes de Amazon DynamoDB en una o más tablas de una sola región. Las operaciones admitidas son `BatchGetItem`, `BatchPutItem` y `BatchDeleteItem`. Estas características de AWS AppSync le permiten realizar tareas como las siguientes:

- Transferir una lista de claves en una sola consulta y obtener los resultados desde una tabla
- Leer registros de una o varias tablas en una única consulta
- Escribir registros de forma masiva en una o varias tablas
- Escribir o eliminar condicionalmente registros en varias tablas que pueden tener una relación

El uso de operaciones por lotes con DynamoDB en AWS AppSync es una técnica avanzada que exige reflexión y conocimientos adicionales de las operaciones de backend y las estructuras de las tablas. Además, las operaciones por lotes en AWS AppSync tienen dos diferencias fundamentales con respecto a las operaciones que no se realizan por lotes:

- El rol del origen de datos debe tener permisos para todas las tablas a las que el solucionador obtiene acceso.
- La especificación de tabla de un solucionador forma parte de la plantilla de mapeo.

Permisos

Al igual que con otros solucionadores, debe crear un origen de datos en AWS AppSync y crear un rol o utilizar uno existente. Dado que las operaciones por lotes requieren diferentes permisos para las tablas de DynamoDB, debe conceder los permisos de rol configurados para las acciones de lectura o escritura:

```
{
    "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/*"
            ]
        }
    ]
}
```

Nota: Los roles están vinculados a orígenes de datos de AWS AppSync y los solucionadores de los campos se invocan con referencia a un origen de datos. Los orígenes de datos configurados para recuperar información de DynamoDB solo tienen especificada una tabla a fin de que la configuración siga siendo sencilla. Por lo tanto, al realizar una operación por lotes en varias tablas con un único solucionador, que es una tarea más avanzada, debe conceder al rol de ese origen de datos acceso a cualquier tabla con la que el solucionador vaya a interactuar. Esto se hace en el campo Resource (Recurso) de la política de IAM anterior. La configuración de las tablas en las que se realizan llamadas por lotes se lleva a cabo en la plantilla de solucionador, que se describe a continuación.

Origen de datos

En aras de la simplicidad, vamos a utilizar el mismo origen de datos para todos los solucionadores que se utilizan en este tutorial. En la pestaña Orígenes de datos, cree un nuevo origen de datos de DynamoDB y llámelo BatchTutorial. El nombre de la tabla pueden ser uno cualquiera, ya que los nombre de las tablas se especifican como parte de la plantilla de mapeo de solicitud para las operaciones por lotes. Llamaremos a la tabla `empty`.

En este tutorial funcionará cualquier rol con la siguiente política insertada:

```
{
    "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/*"
            ]
        }
    ]
}
```

Lotes en una única tabla

En este ejemplo, supongamos que tiene una sola tabla denominada Posts en la que desea añadir y eliminar elementos mediante operaciones por lotes. Utilice el siguiente esquema teniendo en cuenta que, para la consulta, vamos a pasar una lista de 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
```

```
}
```

Asocie un solucionador al campo `batchAdd()` con la siguiente plantilla de mapeo de solicitud. Esto toma automáticamente cada elemento con el tipo de GraphQL `input PostInput` y crea un mapa, lo que es necesario para la operación `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)
    }
}
```

En este caso, la plantilla de mapeo de respuesta simplemente traslada la información, pero el nombre de la tabla se anexa como `..data.Posts` al objeto contexto de este modo:

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

Ahora vaya a la página Consultas de la consola de AWS AppSync y ejecute la siguiente mutación batchAdd:

```
mutation add {
    batchAdd(posts:[{
            id: 1 title: "Running in the Park"},{
            id: 2 title: "Playing fetch"
        }]){
            id
            title
    }
}
```

Los resultados deben aparecer en la pantalla y puede validarlos de manera independiente en la consola de DynamoDB que ambos valores han escrito en la tabla Posts.

A continuación, asocie un solucionador al campo `batchGet()` con la siguiente plantilla de mapeo de solicitud. Esto toma automáticamente cada elemento del tipo de GraphQL `ids:[]` y crea un mapa que es necesario para la operación `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"}
                }
        }
    }
}
```

De nuevo, la plantilla de mapeo de respuesta se limita a transferir la información, pero añadiendo otra vez el nombre de la tabla como `..data.Posts` al objeto contexto:

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

Ahora, vuelva a la página Consultas de la consola de AWS AppSync y ejecute la siguiente consulta batchGet:

```
query get {
    batchGet(ids:[1,2,3]){
        id
        title
    }
}
```

Esto debe devolver los resultados para los dos valores `id` que ha añadido anteriormente. Observe que se devuelve un valor `null` para el `id` con valor `3`. Esto se debe a que aún no hay ningún registro en la tabla Posts con ese valor. Observe también que AWS AppSync devuelve los resultados en el mismo orden en que se pasaron las claves a la consulta, lo que es una característica automática adicional de AWS AppSync. Por lo tanto, si cambia a `batchGet(ids:[1,3,2)`, verá que el orden cambia. También sabrá por qué `id` devuelve un valor `null`.

Por último, asocie un solucionador al campo `batchDelete()` con la siguiente plantilla de mapeo de solicitud. Esto toma automáticamente cada elemento del tipo de GraphQL `ids:[]` y crea un mapa que es necesario para la operación `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)
    }
}
```

De nuevo, la plantilla de mapeo de respuesta se limita a transferir la información, pero añadiendo otra vez el nombre de la tabla como `..data.Posts` al objeto contexto:

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

Ahora vuelva a la página Consultas de la consola de AWS AppSync y ejecute la siguiente mutación batchDelete:

```
mutation delete {
    batchDelete(ids:[1,2]){ id }
}
```

Ahora se eliminarán los registros con `id` `1` y `2`. Si vuelve a ejecutar la consulta `batchGet()` anterior, devolverá `null`.

Lotes en varias tablas

AWS AppSync también permite realizar operaciones por lotes en varias tablas. Vamos a crear una aplicación más compleja. Imagine que queremos crear una aplicación de salud de mascotas, con sensores que comunican la ubicación y temperatura corporal de la mascota. Los sensores funcionan con pilas e intentan conectarse a la red cada pocos minutos. Cuando un sensor establece una conexión, envía sus lecturas a nuestra API de AWS AppSync. A continuación, los disparadores analizan los datos para presentar un panel al propietario de la mascota. Concentrémonos en representar las interacciones entre el sensor y el almacén de datos de backend.

Como requisito previo, vamos a crear primero dos tablas de DynamoDB: locationReadings almacenará las lecturas de ubicación de los sensores y temperatureReadings almacenará las lecturas de temperatura de los sensores. Ambas tablas comparten la misma estructura de clave principal: `sensorId (String)` es la clave de partición y `timestamp (String)` la clave de ordenación.

Vamos a utilizar el siguiente esquema de GraphQL:

```
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: registrar lecturas del sensor

Nuestros sensores tiene que poder enviar sus lecturas cuando se conectan a Internet. El campo de GraphQL `Mutation.recordReadings` es la API que utilizarán para hacerlo. Vamos a asociar un solucionador para activar nuestra API.

Seleccione Attach (Asociar) junto al campo `Mutation.recordReadings`. En la pantalla siguiente, elija el origen de datos `BatchTutorial` creado al inicio del tutorial.

Vamos a añadir la siguiente plantilla de mapeo de solicitud:

Plantilla de mapeo de solicitud

```
## 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)
    }
}
```

Como puede ver, la operación `BatchPutItem` nos permite especificar varias tablas.

Vamos a usar la siguiente plantilla de mapeo de respuesta.

Plantilla de mapeo de respuesta

```
## 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)
```

Con operaciones por lotes, la invocación puede devolver tanto errores como resultados. Por lo tanto, podemos realizar algún control de errores adicional.

Nota: el uso de `$utils.appendError()` es similar al de `$util.error()`, siendo la principal diferencia que no interrumpe la evaluación de la plantilla de mapeo. En su lugar, indica que hubo un error con el campo, pero permite evaluar la plantilla y, por tanto, devolver los datos al intermediario. Recomendamos que utilice `$utils.appendError()` cuando la aplicación tenga que devolver resultados parciales.

Guarde el solucionador y vaya a la página Consultas de la consola de AWS AppSync. Enviemos ahora algunas lecturas de los sensores.

Ejecute la mutación siguiente:

```
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
    }
  }
}
```

Hemos enviado en una mutación 10 lecturas de sensor repartidas en dos tablas. Utilice la consola de DynamoDB para validar que los datos aparecen en las tablas locationReadings y temperatureReadings.

BatchDeleteItem: eliminar lecturas de los sensores

Del mismo modo, también es necesario eliminar lotes de lecturas de sensor. Vamos a utilizar el campo de GraphQL `Mutation.deleteReadings` para este fin. Seleccione Attach (Asociar) junto al campo `Mutation.recordReadings`. En la pantalla siguiente, elija el origen de datos `BatchTutorial` creado al inicio del tutorial.

Vamos a usar la siguiente plantilla de mapeo de solicitud.

Plantilla de mapeo de solicitud

```
## 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)
    }
}
```

La plantilla de mapeo de respuesta es la misma que la que hemos usado para `Mutation.recordReadings`.

Plantilla de mapeo de respuesta

```
## 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)
```

Guarde el solucionador y vaya a la página Consultas de la consola de AWS AppSync. Ahora vamos a eliminar un par de lecturas de sensores.

Ejecute la mutación siguiente:

```
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
    }
  }
}
```

Valide a través de la consola de DynamoDB que estas dos lecturas se han eliminado de las tablas locationReadings y temperatureReadings.

BatchGetItem: recuperar lecturas

Otra operación habitual en nuestra aplicación de salud de mascotas es obtener las lecturas de un sensor en un momento determinado. Vamos a asociar un solucionador al campo de GraphQL `Query.getReadings` en nuestro esquema. Seleccione Attach (Asociar) y, en la siguiente pantalla, elija el origen de datos `BatchTutorial` creado al inicio del tutorial.

Vamos a añadir la siguiente plantilla de mapeo de solicitud.

Plantilla de mapeo de solicitud

```
## 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
        }
    }
}
```

Observe que ahora usamos la operación BatchGetItem.

Nuestra plantilla de mapeo de respuesta será un poco diferente, ya que hemos decidido devolver una lista `SensorReading`. Demos ahora la forma deseada al resultado de la invocación.

Plantilla de mapeo de respuesta

```
## 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)
```

Guarde el solucionador y vaya a la página Consultas de la consola de AWS AppSync. Ahora vamos a recuperar lecturas de sensores.

Ejecute la siguiente consulta:

```
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
    }
  }
}
```

Hemos demostrado el uso de operaciones por lotes de DynamoDB mediante AWS AppSync.

Control de errores

En AWS AppSync, en ocasiones las operaciones de orígenes de datos pueden devolver resultados parciales. "Resultados parciales" es el término que utilizaremos para indicar que el resultado de una operación se compone de algunos datos y un error. Dado que la gestión de errores es inherentemente específica de la aplicación, AWS AppSync le da la oportunidad de gestionar los errores en la plantilla de mapeo de respuestas. El error de invocación del solucionador, si lo hay, está disponible en el contexto como `$ctx.error`. Los errores de invocación siempre incluyen un mensaje y un tipo a los que se tiene acceso como propiedades `$ctx.error.message` y `$ctx.error.type`. Durante la invocación de la plantilla de mapeo de respuestas, puede gestionar los resultados parciales de tres formas:

1. pasar por alto el error de la invocación y limitarse a devolver los datos
2. generar un error (con `$util.error(...)`) y detener la evaluación de la plantilla de mapeo de respuesta, con lo que no se devuelven datos.
3. agregar un error (con `$util.appendError(...)`) y devolver los datos

Vamos a demostrar cada una de estas posibilidades con operaciones por lotes de DynamoDB.

Operaciones por lotes de DynamoDB

Con las operaciones por lotes de DynamoDB, es posible que un lote se complete parcialmente. Es decir, es posible que algunos de los elementos o claves solicitados se queden sin procesar. Si AWS AppSync no puede completar un lote, se establecen en el contexto los elementos que no se han procesado y un error de invocación.

Vamos a implementar la gestión de errores utilizando la configuración del campo `Query.getReadings` de la operación `BatchGetItem` de la sección anterior de este tutorial. Esta vez, supongamos que mientras se ejecutaba el campo `Query.getReadings`, la tabla de DynamoDB `temperatureReadings` agotó el desempeño aprovisionado. DynamoDB ha generado una excepción ProvisionedThroughputExceededException en el segundo intento de AWS AppSync de procesar los elementos restantes del lote.

El siguiente código JSON representa el contexto serializado después de la invocación por lotes de DynamoDB, pero antes de la evaluación de la plantilla de mapeo de respuesta.

```
{
  "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": []
}
```

Cabe tener en cuenta algunos aspectos del contexto:

- AWS AppSync ha establecido el error de invocación en el contexto en `$ctx.error` y el tipo de error se ha establecido en Dynamodb:ProvisionedThroughputExceedException.
- los resultados se mapean para cada tabla en `$ctx.result.data`, aunque haya un error
- las claves que quedaron sin procesar están disponibles en `$ctx.result.data.unprocessedKeys`. Aquí, AWS AppSync no pudo recuperar el elemento con la clave (sensorId:1, timestamp:2018-02-01T17:21:05.000+08:00) debido a un rendimiento insuficiente de la tabla.

Nota: para `BatchPutItem`, es `$ctx.result.data.unprocessedItems`. Para `BatchDeleteItem`, es `$ctx.result.data.unprocessedKeys`.

Vamos a gestionar este error de tres formas diferentes.

1. Pasar por alto el error de invocación

Si se devuelven los datos sin gestionar el error de invocación se pasa por alto el error, lo que hace que el resultado para el campo de GraphQL indicado siempre tenga éxito.

La plantilla de mapeo de respuesta que escribimos ya es conocida y solo se centra en los datos de resultado.

Plantilla de mapeo de respuestas:

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

Respuesta de 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
      }
    ]
  }
}
```

No se añadirán errores a la respuesta, ya que solo se actúa en los datos.

2. Generar un error para anular la ejecución de la plantilla

Cuando los errores parciales se deban tratar como errores completos desde la perspectiva del cliente, puede anular la ejecución de la plantilla para evitar la devolución de datos. El método de utilidad `$util.error(...)` consigue exactamente ese comportamiento.

Plantilla de mapeo de respuestas:

```
## 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)
```

Respuesta de 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. (...)"
    }
  ]
}
```

Aunque podrían haberse devuelto algunos resultados de la operación por lotes de DynamoDB, decidimos generar un error que hace que el campo de GraphQL `getReadings` sea nulo y se añada el error al bloque de errors de la respuesta de GraphQL.

3. Añadir el error para devolver tanto los datos como los errores

En algunos casos, para proporcionar una mejor experiencia al usuario, las aplicaciones pueden devolver resultados parciales e informar a sus clientes de los elementos que no se han procesado. Los clientes pueden decidir probar de nuevo o trasladar el error al usuario final. El método `$util.appendError(...)` es la utilidad que permite este comportamiento al permitir que el creador de la aplicación agregue errores al contexto sin interferir con la evaluación de la plantilla. Después de evaluar la plantilla, AWS AppSync procesará cualquier error de contexto agregándolo al bloque de errores de la respuesta de GraphQL.

Plantilla de mapeo de respuestas:

```
#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)
```

Hemos reenviado el error de invocación y el elemento unprocessedKeys dentro del bloque de errores de la respuesta de GraphQL. El campo `getReadings` también devuelve datos parciales de la tabla locationReadings, como puede ver en la respuesta a continuación.

Respuesta de 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: Solucionadores de transacciones de DynamoDB

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSync admite el uso de operaciones de transacciones de Amazon DynamoDB en una o más tablas de una sola región. Las operaciones admitidas son `TransactGetItems` y `TransactWriteItems`. Estas características de AWS AppSync le permiten realizar tareas como las siguientes:

- Transferir una lista de claves en una sola consulta y obtener los resultados desde una tabla
- Leer registros de una o varias tablas en una única consulta
- Escribir registros en transacciones en una o más tablas en régimen de todo o nada

- Ejecutar transacciones cuando se cumplan algunas condiciones

Permisos

Al igual que con otros solucionadores, debe crear un origen de datos en AWS AppSync y crear un rol o utilizar uno existente. Dado que las operaciones de transacciones requieren diferentes permisos para las tablas de DynamoDB, debe conceder los permisos de rol configurados para las acciones de lectura o escritura:

```
{
    "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/*"
            ]
        }
    ]
}
```

Nota: Los roles están vinculados a orígenes de datos de AWS AppSync y los solucionadores de los campos se invocan con referencia a un origen de datos. Los orígenes de datos configurados para recuperar información de DynamoDB solo tienen especificada una tabla a fin de que la configuración siga siendo sencilla. Por lo tanto, al realizar una operación de transacciones en varias tablas con un único solucionador, que es una tarea más avanzada, debe conceder al rol de ese origen de datos acceso a cualquier tabla con la que el solucionador vaya a interactuar. Esto se hace en el campo Resource (Recurso) de la política de IAM anterior. La configuración de las tablas en las que se realizan llamadas de transacciones se lleva a cabo en la plantilla de solucionador, que se describe a continuación.

Origen de datos

En aras de la simplicidad, vamos a utilizar el mismo origen de datos para todos los solucionadores que se utilizan en este tutorial. En la pestaña Orígenes de datos, cree un nuevo origen de datos de DynamoDB y llámelo TransactTutorial. El nombre de la tabla pueden ser uno cualquiera, ya que los nombre de las tablas se especifican como parte de la plantilla de mapeo de solicitud para las operaciones de transacciones. Llamaremos a la tabla `empty`.

Tendremos dos tablas denominadas savingAccounts y checkingAccounts, ambas con la clave de partición `accountNumber`, y una tabla transactionHistory con la clave de partición `transactionId`.

En este tutorial funcionará cualquier rol con la siguiente política insertada. Sustituya `region` y `accountId` por su región y su identificador de cuenta.

```
{
    "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/*"
            ]
        }
    ]
}
```

Transacciones

Para este ejemplo, el contexto es una transacción bancaria clásica, en la que usaremos `TransactWriteItems` para:

- Transferir dinero de cuentas de ahorro a cuentas corrientes
- Generar nuevos registros de transacciones para cada transacción

Y, a continuación, usaremos `TransactGetItems` para recuperar los detalles de las cuentas de ahorro y las cuentas corrientes.

Definimos nuestro esquema GraphQL de la siguiente manera:

```
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: rellenar cuentas

Para transferir dinero entre cuentas, tenemos que rellenar la tabla con los detalles. Para ello, utilizaremos la operación `Mutation.populateAccounts` de GraphQL.

En la sección Esquema, haga clic en Asociar junto a la operación `Mutation.populateAccounts`. Vaya a Solucionadores de unidades VTL y, a continuación, elija el mismo origen de datos `TransactTutorial`.

Ahora, use la siguiente plantilla de mapeo de solicitud.

Plantilla de mapeo de solicitud

```
#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)
}
```

Y la siguiente plantilla de mapeo de respuesta:

Plantilla de mapeo de respuesta

```
#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)
```

Guarde el solucionador y vaya a la sección Consultas de la consola de AppSync AWS para rellenar las cuentas.

Ejecute la mutación siguiente:

```
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
    }
  }
}
```

Hemos rellenado 3 cuentas de ahorro y 3 cuentas corrientes en una mutación.

Utilice la consola de DynamoDB para validar que los datos se muestren en las tablas savingAccounts y checkingAccounts.

TransactWriteItems: transferir dinero

Asocie un solucionador a la mutación `transferMoney` con la siguiente plantilla de mapeo de solicitud. Tenga en cuenta que los valores de `amounts`, `savingAccountNumbers` y `checkingAccountNumbers` son los mismos.

```
#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)
}
```

Tendremos 3 transacciones bancarias en una sola operación de `TransactWriteItems`. Use la siguiente plantilla de mapeo de respuesta:

```
#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)
```

Ahora, vaya a la sección Consultas de la consola de AWS AppSync y ejecute la mutación transferMoney de la siguiente manera:

```
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
    }
  }
}
```

Hemos enviado 2 transacciones bancarias en una mutación. Utilice la consola de DynamoDB para validar que los datos se muestren en las tablas savingAccounts, checkingAccounts y transactionHistory.

TransactGetItems: recuperar cuentas

Con el fin de recuperar los detalles de las cuentas de ahorro y cuentas corrientes en una sola solicitud transaccional, vamos a asociar un solucionador a la operación `Query.getAccounts` de GraphQL en nuestro esquema. Seleccione Asociar, vaya a Solucionadores de unidades VTL y, en la siguiente pantalla, elija el mismo origen de datos de `TransactTutorial` creado al inicio del tutorial. Configure las plantillas de la siguiente manera:

Plantilla de mapeo de solicitud

```
#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)
}
```

Plantilla de mapeo de respuesta

```
#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)
```

Guarde el solucionador y vaya a las secciones Consultas de la consola de AWS AppSync. Para recuperar las cuentas de ahorro y las cuentas corrientes, ejecute la siguiente consulta:

```
query getAccounts {
  getAccounts(
    savingAccountNumbers: ["1", "2", "3"],
    checkingAccountNumbers: ["1", "2"]
  ) {
    savingAccounts {
      accountNumber
      username
      balance
    }
    checkingAccounts {
      accountNumber
      username
      balance
    }
  }
}
```

Hemos demostrado el uso de operaciones de transacciones de DynamoDB mediante AWS AppSync.

Tutorial: Solucionadores de HTTP

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSync le permite utilizar orígenes de datos compatibles (es decir, AWS Lambda, Amazon DynamoDB, Amazon OpenSearch Service o Amazon Aurora) para realizar diversas operaciones, además de cualquier punto de conexión HTTP arbitrario para resolver campos de GraphQL. Una vez que los puntos de enlace HTTP están disponibles, puede conectar con ellos mediante un origen de datos. A continuación, puede configurar un solucionador en el esquema para realizar operaciones de GraphQL como consultas, mutaciones, y suscripciones. Este tutorial le guiará a lo largo de algunos ejemplos comunes.

En este tutorial, se utiliza una API de REST (creada mediante Amazon API Gateway y Lambda) con un punto de conexión de GraphQL de AWS AppSync.

Configuración en un clic

Si desea crear automáticamente un punto de conexión de GraphQL en AWS AppSync con un punto de conexión HTTP configurado (mediante Amazon API Gateway y Lambda), puede utilizar la siguiente plantilla de AWS CloudFormation:

Creación de una API de REST

Puede utilizar la siguiente plantilla de AWS CloudFormation para establecer un punto de enlace REST que funcione con este tutorial:

La pila de AWS CloudFormation realiza los siguientes pasos:

1. Configura una función Lambda que contiene la lógica de negocio del microservicio.
2. Configure una API de REST de puerta de enlace de API con la combinación punto de conexión/método/tipo de contenido siguiente:

Ruta de recurso de la API	Método HTTP	Tipo de contenido admitido
/v1/users	POST	application/json
/v1/users	GET	application/json
/v1/users/1	GET	application/json
/v1/users/1	PUT	application/json
/v1/users/1	DELETE	application/json

Creación de la API de GraphQL

Para crear la API de GraphQL en AWS AppSync:

- Abra la consola de AppSync AWS y elija Crear API.
- En el nombre de la API, introduzca `UserData`.
- Elija Custom Schema (Esquema personalizado).
- Seleccione Create (Crear).

La consola de AWS AppSync crea una nueva API de GraphQL automáticamente utilizando el modo de autenticación de clave de API. Puede utilizar la consola para configurar el resto de la API de GraphQL y ejecutar consultas en ella durante el resto de este tutorial.

Creación de un esquema de GraphQL

Ahora que tiene una API de GraphQL, vamos a crear un esquema de GraphQL. En el editor de esquemas de la consola de AWS AppSync, asegúrese de que el esquema coincida con el siguiente:

```
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
}
```

Configure el origen de datos HTTP

Para configurar el origen de datos HTTP, haga lo siguiente:

- En la pestaña Data Sources (Orígenes de datos), elija New (Nuevo) y, a continuación, escriba un nombre fácil de recordar para el origen de datos (por ejemplo, HTTP).
- En Data source type (Tipo de origen de datos), elija HTTP.
- Establezca como punto de conexión el punto de conexión de la puerta de enlace de la API que se ha creado. Asegúrese de no incluir el nombre de etapa como parte del punto de enlace.

Nota: En estos momentos, solo los puntos de conexión públicos son compatibles con AWS AppSync.

Nota: Para obtener más información acerca de las entidades de certificación que reconoce el servicio de AWS AppSync, consulte Certificate Authorities (CA) Recognized by AWS AppSync for HTTPS Endpoints.

Configuración de solucionadores

En este paso conectará el origen de datos HTTP a la consulta getUser.

Para configurar el solucionador:

- Elija la pestaña Schema (Esquema).
- En el panel Data types (Tipos de datos) situado a la derecha, en el tipo Query (Consulta), busque el campo getUser y seleccione Attach (Asociar).
- En Data source name (Nombre del origen de datos), elija HTTP.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue el siguiente código:

```
{
    "version": "2018-05-29",
    "method": "GET",
    "params": {
        "headers": {
            "Content-Type": "application/json"
        }
    },
    "resourcePath": $util.toJson("/v1/users/${ctx.args.id}")
}
```

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue el siguiente código:

```
## 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
```

- Elija la pestaña Query (Consulta) y ejecute la consulta siguiente:

```
query GetUser{
    getUser(id:1){
        id
        username
    }
}
```

Debe obtener la siguiente respuesta:

```
{
    "data": {
        "getUser": {
            "id": "1",
            "username": "nadia"
```

```
        }
    }
}
```

- Elija la pestaña Schema (Esquema).
- En el panel Data types (Tipos de datos) situado en la parte derecha, en Mutation (Mutación), busque el campo addUser y seleccione Attach (Asociar).
- En Data source name (Nombre del origen de datos), elija HTTP.
- En Configure the request mapping template (Configurar la plantilla de mapeo de solicitud), pegue el siguiente código:

```
{
    "version": "2018-05-29",
    "method": "POST",
    "resourcePath": "/v1/users",
    "params":{
      "headers":{
        "Content-Type": "application/json",
      },
      "body": $util.toJson($ctx.args.userInput)
    }
}
```

- En Configure the response mapping template (Configurar la plantilla de mapeo de respuesta), pegue el siguiente código:

```
## 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
```

- Elija la pestaña Query (Consulta) y ejecute la consulta siguiente:

```
mutation addUser{
    addUser(userInput:{
        id:"2",
        username:"shaggy"
    }){
        id
        username
    }
}
```

Debe obtener la siguiente respuesta:

```
{
    "data": {
        "getUser": {
        "id": "2",
        "username": "shaggy"
        }
    }
}
```

Invocación de servicios de AWS

Puede utilizar los solucionadores de HTTP para establecer una interfaz de API de GraphQL para los servicios de AWS. Las solicitudes HTTP a AWS deben firmarse con el proceso de Signature Version 4 para que AWS pueda identificar quién las envió. AWS AppSync calcula la firma en su nombre cuando asocia un rol de IAM al origen de datos HTTP.

Debe proporcionar dos componentes adicionales para invocar los servicios de AWS con solucionadores de HTTP:

- Un rol de IAM con permisos para realizar una llamada a las API del servicio de AWS.
- Configuración de la firma en el origen de datos

Por ejemplo, si desea realizar una llamada a la operación ListGraphqlApis con los solucionadores de HTTP, primero debe crear un rol de IAM que AWS AppSync asume con la siguiente política asociada:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "appsync:ListGraphqlApis"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

A continuación, cree el origen de datos HTTP para AWS AppSync. En este ejemplo, debe realizar una llamada a AWS AppSync en la región Oeste de EE. UU. (Oregón). Ajuste la siguiente configuración HTTP en un archivo denominado `http.json`, que incluye la región de la firma y el nombre del servicio:

```
{
    "endpoint": "https://appsync.us-west-2.amazonaws.com/",
    "authorizationConfig": {
        "authorizationType": "AWS_IAM",
        "awsIamConfig": {
            "signingRegion": "us-west-2",
            "signingServiceName": "appsync"
        }
    }
}
```

A continuación, utilice la AWS CLI para crear el origen de datos con un rol asociado de la siguiente forma:

```
aws appsync create-data-source --api-id <API-ID> \
                               --name AWSAppSync \
                               --type HTTP \
                               --http-config file:///http.json \
                               --service-role-arn <ROLE-ARN>
```

Cuando asocie un solucionador al campo del esquema, utilice la siguiente plantilla de mapeo de solicitudes para realizar una llamada a AWS AppSync:

```
{
    "version": "2018-05-29",
    "method": "GET",
    "resourcePath": "/v1/apis"
}
```

Cuando ejecuta una consulta de GraphQL para este origen de datos, AWS AppSync firma la solicitud mediante el rol que ha proporcionado e incluye la firma en la solicitud. La consulta devuelve una lista de API de GraphQL de AWS AppSync en su cuenta de dicha región de AWS.

Tutorial: Aurora sin servidor

AWS AppSync proporciona un origen de datos para ejecutar comandos SQL con respecto a clústeres de Amazon Aurora sin servidor que se han habilitado con una API de datos. Puede utilizar los solucionadores de AppSync para ejecutar instrucciones de SQL con respecto a la API de datos con consultas, mutaciones y suscripciones de GraphQL.

Crear un clúster

Antes de añadir un origen de datos de RDS a AppSync primero debe habilitar una API de datos en un clúster de Aurora sin servidor y configurar un secreto con AWS Secrets Manager. Puede crear un clúster de Aurora sin servidor primero con 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
```

Esto devolverá un ARN para el clúster.

Cree un secreto a través de la consola de AWS Secrets Manager o también a través de la interfaz de línea de comandos (CLI) con un archivo de entrada como el siguiente utilizando los valores de USERNAME y COMPLEX_PASSWORD del paso anterior:

```
{
```

```
    "username": "USERNAME",
    "password": "COMPLEX_PASSWORD"
}
```

Transfiera esto como parámetro a la AWS CLI:

```
aws secretsmanager create-secret --name HttpRDSSecret --secret-string file://creds.json
 --region us-east-1
```

Esto devolverá un ARN para el secreto.

Anote el ARN de su clúster de Aurora Serverless y secreto para su uso posterior en la consola de AppSync a la hora de crear un origen de datos.

Habilitar la API de datos

Puede habilitar la API de datos en el clúster siguiendo las instrucciones de la documentación de RDS. La API de datos debe estar habilitada antes de añadirse como un origen de datos de AppSync.

Creación de una base de datos y tabla

Una vez que tenga habilitada su API de datos, puede asegurarse de que funciona con el comando `aws rds-data execute-statement` de la AWS CLI. Esto garantizará que el clúster de Aurora sin servidor esté configurado correctamente antes de añadirlo a la API de AppSync. En primer lugar, cree una base de datos denominada TESTDB con el parámetro `--sql` del siguiente modo:

```
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"
```

Si esto se ejecuta sin errores, añada una tabla con el comando de creación de tablas:

```
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"
```

Si todo se ha ejecutado sin problemas puede avanzar para añadir el clúster como origen de datos en la API de AppSync.

Esquema de GraphQL

Ahora que su API de datos de Aurora Serverless está en marcha con una tabla, vamos a crear un esquema GraphQL y asociar los solucionadores para realizar mutaciones y suscripciones. Cree una nueva API en la consola de AWS AppSync, vaya a la página Esquema y escriba lo siguiente:

```
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
}
```

Guarde su esquema y vaya a la página Data Sources (Orígenes de datos) y cree un nuevo origen de datos. Seleccione la Relational database (Base de datos relacional) para el tipo de origen de datos y proporcione un nombre fácil de recordar. Utilice el nombre de la base de datos que ha creado en el último paso, así como el Cluster ARN (ARN del clúster) en el que lo creó. Para el Role (Rol) puede permitir que AppSync cree un rol nuevo o cree uno con una política similar a la siguiente:

```
{
    "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:*"
            ]
        }
    ]
}
```

Tenga en cuenta que hay dos Statements (Instrucciones) en esta política a las que está concediendo acceso de rol. El primer Recurso es su clúster de Aurora sin servidor y el segundo es su ARN de AWS Secrets Manager. Tendrá que proporcionar AMBOS ARN en la configuración del origen de datos de AppSync antes de hacer clic en Create (Crear).

Configuración de solucionadores

Ahora que tenemos un esquema de GraphQL válido y un origen de datos de RDS, podemos asociar los solucionadores a los campos de GraphQL en el esquema. Nuestra API ofrecerá las siguientes capacidades:

1. crear una mascota a través del campo Mutation.createPet
2. actualizar una mascota a través del campo Mutation.updatePet
3. eliminar una mascota a través del campo Mutation.deletePet
4. obtener una mascota única a través del campo Query.getPet
5. enumerar todas las mascotas a través del campo Query.listPets
6. enumerar las mascotas en un rango de precios a través del campo Query.listPetsByPriceRange

Mutation.createPet

En el editor de esquemas de la consola de AWS AppSync, elija a la derecha Asociar solucionador para `createPet(input: CreatePetInput!): Pet`. Elija el origen de los datos de RDS. En la sección request mapping template (plantilla de mapeo de solicitudes), añada la siguiente plantilla:

```
#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)
    }
}
```

Las instrucciones SQL se ejecutarán de forma secuencial, en función del orden de la matriz statements (instrucciones). Los resultados volverán en el mismo orden. Dado que se trata de una mutación, ejecutamos una instrucción select (selección) después de insert (insertar) para recuperar los valores comprometidos para rellenar la plantilla de mapeo de respuesta de GraphQL.

En la sección Plantilla de mapeo de respuestas, añada la siguiente plantilla:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[1][0])
```

Debido a que las statements (instrucciones) tienen dos consultas SQL, necesitamos especificar el segundo resultado en la matriz que viene de la base de datos con: `$utils.rds.toJsonString($ctx.result))[1][0]`.

Mutation.updatePet

En el editor de esquemas de la consola de AWS AppSync, elija a la derecha Asociar solucionador para `updatePet(input: UpdatePetInput!): Pet`. Elija el origen de los datos de RDS. En la sección Plantilla de mapeo de solicitudes, añada la siguiente plantilla:

```
{
"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)
    }
}
```

En la sección Plantilla de mapeo de respuestas, añada la siguiente plantilla:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[1][0])
```

Mutation.deletePet

En el editor de esquemas de la consola de AWS AppSync, elija a la derecha Asociar solucionador para `deletePet(input: DeletePetInput!): Pet`. Elija el origen de los datos de RDS. En la sección Plantilla de mapeo de solicitudes, añada la siguiente plantilla:

```
{
"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"
    }
}
```

En la sección Plantilla de mapeo de respuestas, añada la siguiente plantilla:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0][0])
```

Query.getPet

Ahora que las mutaciones se han creado para su esquema, conectaremos las tres consultas para mostrar cómo obtener elementos individuales, listas y aplicar el filtrado de SQL. En el editor de esquemas de la consola de AWS AppSync, elija a la derecha Asociar solucionador para `getPet(id: ID!): Pet`. Elija el origen de los datos de RDS. En la sección Plantilla de mapeo de solicitudes, añada la siguiente plantilla:

```
{
"version": "2018-05-29",
        "statements": [
            $util.toJson("select * from Pets WHERE id=:ID")
    ],
    "variableMap": {
        ":ID": "$ctx.args.id"
    }
}
```

En la sección Plantilla de mapeo de respuestas, añada la siguiente plantilla:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0][0])
```

Query.listPets

En el editor de esquemas de la consola de AWS AppSync, elija a la derecha Asociar solucionador para `getPet(id: ID!): Pet`. Elija el origen de los datos de RDS. En la sección Plantilla de mapeo de solicitudes, añada la siguiente plantilla:

```
{
    "version": "2018-05-29",
    "statements": [
        "select * from Pets"
    ]
}
```

En la sección Plantilla de mapeo de respuestas, añada la siguiente plantilla:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0])
```

Query.listPetsByPriceRange

En el editor de esquemas de la consola de AWS AppSync, elija a la derecha Asociar solucionador para `getPet(id: ID!): Pet`. Elija el origen de los datos de RDS. En la sección Plantilla de mapeo de solicitudes, añada la siguiente plantilla:

```
{
    "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)
    }
}
```

En la sección Plantilla de mapeo de respuestas, añada la siguiente plantilla:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0])
```

Ejecutar mutaciones

Ahora que ha configurado todos sus solucionadores con instrucciones SQL y conectado API de GraphQL a su API de datos Aurora Serverless, puede comenzar a realizar mutaciones y consultas. En la consola de AWS AppSync, elija la pestaña Consultas e introduzca lo siguiente para crear una mascota:

```
mutation add {
    createPet(input : { type:fish, price:10.0 }){
        id
        type
        price
    }
}
```

La respuesta debe contener los valores de id, type (tipo) y price (precio) de la siguiente manera:

```
{
  "data": {
    "createPet": {
      "id": "c6fedbbe-57ad-4da3-860a-ffe8d039882a",
      "type": "fish",
      "price": "10.0"
    }
  }
}
```

Puede modificar este elemento mediante la ejecución de la mutación updatePet:

```
mutation update {
    updatePet(input : {
        id: ID_PLACEHOLDER,
        type:bird,
        price:50.0
    }){
        id
        type
        price
    }
```

```
}
```

Tenga en cuenta que hemos usado el id que ha devuelto la operación createPet anteriormente. Este será un valor único para su registro, ya que el solucionador obtuvo `$util.autoId()`. Podría eliminar un registro de un modo similar:

```
mutation delete {
    deletePet(input : {id:ID_PLACEHOLDER}){
        id
        type
        price
    }
}
```

Cree un par de registros con la primera mutación con valores diferentes de price (precio) y, a continuación, ejecute algunas consultas.

Ejecutar consultas

En la pestaña Queries (Consultas) de la consola, utilice la siguiente instrucción para obtener una lista de todos los registros que haya creado:

```
query allpets {
    listPets {
        id
        type
        price
    }
}
```

Esto está bien, pero vamos a aprovechar el predicado de SQL WHERE (DÓNDE) que tenía `where price > :MIN and price < :MAX` en nuestra plantilla de mapeo para Query.listPetsByPriceRange con la siguiente consulta GraphQL:

```
query petsByPriceRange {
    listPetsByPriceRange(min:1, max:11) {
        id
        type
        price
    }
}
```

Solo debe ver registros con un price (precio) superior a 1 USD o inferior a 10 USD. Por último, puede realizar consultas para obtener registros individuales tal y como se indica a continuación:

```
query onePet {
    getPet(id:ID_PLACEHOLDER){
        id
        type
        price
    }
}
```

Saneamiento de la entrada

Recomendamos a los desarrolladores que utilicen `variableMap` como protección contra los ataques de inyección de código SQL. Si no se utilizan mapas de variables, los desarrolladores son responsables de sanear los argumentos de sus operaciones de GraphQL. Una forma de hacerlo es proporcionando los pasos de validación específicos de entrada en la plantilla de mapeo de solicitudes antes de la ejecución de una instrucción SQL con respecto a la API de datos. Veamos cómo podemos modificar la plantilla de mapeo de solicitudes del ejemplo de `listPetsByPriceRange`. En lugar de confiar solamente en la entrada del usuario puede hacer lo siguiente:

```
#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)
    }
}
```

Otra forma de protegerse frente a la entrada no autorizada a la hora de ejecutar solucionadores con respecto a la API de datos es utilizar las instrucciones preparadas junto con el procedimiento almacenado y las entradas parametrizadas. Por ejemplo, en el solucionador para `listPets` defina el siguiente procedimiento que ejecuta select (seleccionar) como una instrucción preparada:

```
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
```

Esto se puede crear en su instancia Aurora Serverless utilizando el siguiente comando SQL de ejecución:

```
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"
```

El código de solucionador resultante para listPets se simplifica, ya que ahora simplemente llamamos al procedimiento almacenado: Como mínimo, toda entrada de cadena debe tener comillas simples con caracteres de escape.

```
#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("'", "''"))
    }
}
```

Escapar cadenas

Las comillas simples representan el inicio y el final de los literales de cadena en una instrucción SQL; por ejemplo, `'some string value'`. Para permitir el uso de valores de cadena con uno o más caracteres de comillas simples (`'`) dentro de una cadena, cada uno de ellos debe reemplazarse por dos comillas simples (`''`). Por ejemplo, si la cadena de entrada es `Nadia's dog`, para la instrucción SQL se aplicarían caracteres de escape de la siguiente forma:

```
update Pets set type='Nadia''s dog' WHERE id='1'
```

Tutorial: Solucionadores de canalizaciones

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSync proporciona una manera sencilla de conectar un campo de GraphQL a un único origen de datos a través de los solucionadores de unidad. Sin embargo, ejecutar una sola operación podría no ser suficiente. Los solucionadores de canalización ofrecen la posibilidad de ejecutar operaciones en serie con respecto a los orígenes de datos. Cree funciones en su API y asócielas a un solucionador de canalización. Cada resultado de ejecución de la función se canaliza a la siguiente hasta que no quede ninguna función que ejecutar. Con los solucionadores de canalización, ahora puede crear flujos de trabajo más complejos directamente en AWS AppSync. En este tutorial, se crea una aplicación sencilla de visualización de imágenes, en la que los usuarios pueden publicar y ver imágenes publicadas por sus amigos.

Configuración en un clic

Si desea configurar automáticamente el punto de conexión de GraphQL en AWS AppSync con todos los solucionadores configurados y los recursos necesarios de AWS, puede utilizar la siguiente plantilla de AWS CloudFormation:

Esta pila crea los siguientes recursos en su cuenta:

- Rol de IAM para que AWS AppSync obtenga acceso a los recursos de su cuenta
- 2 tablas de DynamoDB
- 1 grupo de usuarios de Amazon Cognito
- 2 grupos de grupos de usuarios de Amazon Cognito
- 3 usuarios de grupos de usuarios de Amazon Cognito
- 1 API de AWS AppSync

Al final del proceso de creación de la pila de AWS CloudFormation, el usuario recibirá un mensaje de correo electrónico para cada uno de los tres usuarios de Amazon Cognito que se hayan creado. Cada correo electrónico contiene una contraseña temporal que se utiliza para iniciar sesión como un usuario de Amazon Cognito en la consola de AWS AppSync. Guarde las contraseñas para el resto del tutorial.

Configuración manual

Si prefiere ir manualmente a través de un proceso paso a paso por la consola de AWS AppSync, siga el proceso de configuración que se indica más abajo.

Configuración de los recursos que no son AWS AppSync

La API se comunica con dos tablas de DynamoDB: una tabla de imágenes que almacena imágenes y una tabla de amigos que almacena las relaciones entre los usuarios. La API está configurada para utilizar el grupo de usuarios de Amazon Cognito como el tipo de autenticación. La siguiente pila de AWS CloudFormation configura estos recursos en la cuenta.

Al final del proceso de creación de la pila de AWS CloudFormation, el usuario recibirá un mensaje de correo electrónico para cada uno de los tres usuarios de Amazon Cognito que se hayan creado. Cada correo electrónico contiene una contraseña temporal que se utiliza para iniciar sesión como un usuario de Amazon Cognito en la consola de AWS AppSync. Guarde las contraseñas para el resto del tutorial.

Creación de la API de GraphQL

Para crear la API de GraphQL en AWS AppSync:

1. Abra la consola de AWS AppSync y elija Crear desde cero y Comenzar.
2. Establezca como nombre de la API `AppSyncTutorial-PicturesViewer`.
3. Seleccione Create (Crear).

La consola de AWS AppSync crea una nueva API de GraphQL automáticamente utilizando el modo de autenticación de clave de API. Puede utilizar la consola para configurar el resto de la API de GraphQL y ejecutar consultas en ella durante el resto de este tutorial.

Configuración de la API de GraphQL

Debe configurar la API de AWS AppSync con el grupo de usuarios de Amazon Cognito que acaba de crear.

1. Elija la pestaña Settings.
2. En la sección Authorization Type (Tipo de autorización), elija Amazon Cognito User Pool (Grupo de usuarios de Amazon Cognito).
3. En Configuración del grupo de usuarios, elija US-WEST-2 para la Región de AWS.
4. Elija el grupo de usuarios AppSyncTutorial-UserPool.
5. Elija DENY (DENEGAR) como Default Action (Acción predeterminada).
6. Deje el campo AppId client regex en blanco.
7. Seleccione Save.

La API ahora está configurada para utilizar el grupo de usuarios de Amazon Cognito como el tipo de autenticación.

Configuración de orígenes de datos para las tablas de DynamoDB

Una vez creadas las tablas de DynamoDB, vaya a la API de GraphQL para AWS AppSync en la consola y elija la pestaña Orígenes de datos. Ahora, va a crear un origen de datos en AWS AppSync para cada una de las tablas de DynamoDB que acaba de crear.

1. Elija la pestaña Data source (Origen de datos).

2. Elija New (Nuevo) para crear un nuevo origen de datos.
3. Escriba `PicturesDynamoDBTable` como nombre del origen de datos.
4. En Tipo de origen de datos, elija Tabla de Amazon DynamoDB.
5. Elija US-WEST-2 como región.
6. En la lista de tablas, elija la tabla AppSyncTutorial-PicturesDynamoDB.
7. En la sección Create or use an existing role (Crear o usar un rol existente), elija Existing role (Rol existente).
8. Elija el rol que se acaba de crear desde la plantilla de CloudFormation. Si no ha cambiado ResourceNamePrefix, el nombre del rol debe ser AppSyncTutorial-DynamoDBRole.
9. Seleccione Create (Crear).

Repita el mismo proceso para la tabla de amigos; el nombre de la tabla de DynamoDB debe ser AppSyncTutorial-Friends si no cambió el parámetro ResourceNamePrefix en el momento de crear la pila de CloudFormation.

Creación del esquema de GraphQL

Ahora que los orígenes de datos están conectados a las tablas de DynamoDB, vamos a crear un esquema de GraphQL. En el editor de esquemas de la consola de AWS AppSync, asegúrese de que su esquema coincida con el siguiente:

```
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!
}
```

Elija Save Schema (Guardar esquema) para guardar el esquema.

Algunos de los campos de esquema se han comentado con la directiva @aws_auth. Dado que la API de configuración de acción predeterminada está establecida en DENY (DENEGAR), la API rechaza a todos los usuarios que no son miembros de los grupos mencionados dentro de la directiva @aws_auth. Para obtener más información acerca de cómo proteger la API, puede leer la página de seguridad. En este caso, solo los usuarios administradores tienen acceso a los campos Mutation.createPicture y Mutation.createFriendship mientras que los usuarios que son miembros de los grupos Admins (Administradores) o Viewers (Espectadores) pueden obtener acceso al campo Query.getPicturesByOwner. Todos los demás usuarios no tienen acceso.

Configuración de solucionadores

Ahora que tiene un esquema GraphQL válido y dos orígenes de datos, puede asociar los solucionadores a los campos de GraphQL en el esquema. La API ofrece las siguientes capacidades:

- Crear una imagen a través del campo Mutation.createPicture
- Crear una amistad a través del campo Mutation.createFriendship
- Recuperar una imagen a través del campo Query.getPicture

Mutation.createPicture

En el editor de esquemas de la consola de AWS AppSync, elija a la derecha Asociar solucionador para `createPicture(input: CreatePictureInput!): Picture!`. Elija el origen de datos PicturesDynamoDBTableDynamoDB. En la sección Plantilla de mapeo de solicitudes, añada la siguiente plantilla:

```
#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)
}
```

En la sección Plantilla de mapeo de respuestas, añada la siguiente plantilla:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end
$util.toJson($ctx.result)
```

La funcionalidad de creación de imagen está lista. Va a guardar una imagen en la tabla Pictures (Imágenes) utilizando un UUID generado de forma aleatoria como id de la imagen y utilizando el nombre de usuario de Cognito como propietario de la imagen.

Mutation.createFriendship

En el editor de esquemas de la consola de AWS AppSync, elija a la derecha Asociar solucionador para `createFriendship(id: ID!, target: ID!): Boolean`. Elija el origen de datos DynamoDBFriendsDynamoDBTable. En la sección Plantilla de mapeo de solicitudes, añada la siguiente plantilla:

```
#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)
    }
}
```

Importante: En la plantilla de solicitud BatchPutItem, el nombre exacto de la tabla de DynamoDB debería estar presente. El valor predeterminado de la tabla es AppSyncTutorial-Friends. Si utiliza el nombre de tabla erróneo, obtiene un error cuando AppSync intenta asumir el rol proporcionado.

En aras de simplificar este tutorial, continúe como si la solicitud de amistad se hubiera aprobado y guarde la entrada de relación directamente en la tabla AppSyncTutorialFriends.

Efectivamente, está almacenando dos elementos para cada amistad, ya que la relación es bidireccional. Para obtener información más detallada sobre las prácticas recomendadas de Amazon DynamoDB para representar las relaciones de varios a varios, consulte las prácticas recomendadas de DynamoDB.

En la sección Plantilla de mapeo de respuestas, añada la siguiente plantilla:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end
true
```

Nota: Asegúrese de que su plantilla de solicitud contiene el nombre de la tabla correcto. El nombre predeterminado es AppSyncTutorial-Friends, pero su nombre de tabla que puede ser diferentes si ha cambiado el parámetro ResourceNamePrefix de CloudFormation.

Query.getPicturesByOwner

Ahora que tiene amistades e imágenes, debe proporcionar la capacidad a los usuarios de ver las imágenes de sus amigos. Para cumplir este requisito, debe comprobar primero que el solicitante es amigo con el propietario y, por último, consultar las imágenes.

Dado que esta funcionalidad requiere dos operaciones de origen de datos, va a crear dos funciones. La primera función, isFriend, comprueba si el solicitante y el propietario son amigos. La segunda función, getPicturesByOwner, recupera las imágenes solicitadas dado un ID de

propietario. Analicemos el flujo de ejecución por debajo del solucionador propuestos en el campo Query.getPicturesByOwner:

1. Plantilla de mapeo Antes: Prepare los argumentos de entrada de contexto y campo.
2. Función isFriend: Comprueba si el solicitante es el propietario de la imagen. En caso contrario, compruebe si los usuarios del solicitante y el propietario son amigos mediante una operación GetItem de DynamoDB en la tabla de amigos.
3. Función getPicturesByOwner: recupera las imágenes de la tabla Imágenes mediante una operación de consulta de DynamoDB en el índice secundario global owner-index.
4. Plantilla de mapeo Después: Mapea el resultado de la imagen para que los atributos de DynamoDB se mapeen correctamente en los campos de tipo GraphQL esperados.

Primero, vamos a crear las funciones.

Función isFriend

1. Elija la pestaña Functions (Funciones).
2. Elija Create Function (Crear función) para crear una función.
3. Escriba `FriendsDynamoDBTable` como nombre del origen de datos.
4. Para el nombre de la función, escriba isFriend.
5. Dentro del área de texto de la plantilla de mapeo de solicitudes, pegue la siguiente plantilla:

```
#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. Dentro del área de texto de la plantilla de mapeo de respuestas, pegue la siguiente plantilla:

```
#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. Elija Create Function (Crear función).

Resultado: Ha creado la función isFriend.

Función getPicturesByOwner

1. Elija la pestaña Functions (Funciones).
2. Elija Create Function (Crear función) para crear una función.
3. Escriba `PicturesDynamoDBTable` como nombre del origen de datos.
4. Para el nombre de la función, escriba `getPicturesByOwner`.
5. Dentro del área de texto de la plantilla de mapeo de solicitudes, pegue la siguiente plantilla:

```
{
    "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. Dentro del área de texto de la plantilla de mapeo de respuestas, pegue la siguiente plantilla:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end

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

7. Elija Create Function (Crear función).

Resultado: Ha creado la función getPicturesByOwner. Ahora que las funciones se han creado, adjunte un solucionador de canalización al campo Query.getPicturesByOwner.

En el editor de esquemas de la consola de AWS AppSync, elija a la derecha Asociar solucionador para `Query.getPicturesByOwner(id: ID!): [Picture]`. En la página siguiente, elija el enlace Convert to pipeline resolver (Convertir a solucionador de canalización) que aparece debajo de la lista desplegable del origen de datos. Use lo siguiente para la plantilla de mapeo de antes:

```
#set($result = { "owner": $ctx.args.id, "callerId": $ctx.identity.username })
$util.toJson($result)
```

En la sección after mapping template (plantilla de mapeo de después), use lo siguiente:

```
#foreach($picture in $ctx.result.items)
    ## prepend "src://" to picture.src property
    #set($picture['src'] = "src://${picture['src']}")
#end
$util.toJson($ctx.result.items)
```

Elija Create Resolver (Crear solucionador). Acaba de asociar su primer solucionador de canalización. En la misma página, añada las dos funciones que ha creado anteriormente. En sección de funciones, seleccione Add A Function (Añadir una función) y, a continuación, elija o escriba el nombre de la primera función, isFriend. Añada la segunda función siguiendo el mismo proceso para la función getPicturesByOwner. Asegúrese de que la función isFriend aparece primero en la lista seguido de la

función getPicturesByOwner. Puede utilizar las flechas hacia arriba y abajo para cambiar el orden de ejecución de las funciones de la canalización.

Ahora que el solucionador de canalización se ha creado y ha asociado las funciones, vamos a probar la nueva API de GraphQL.

Prueba de la API de GraphQL

En primer lugar, debe rellenar las imágenes y amistades ejecutando algunas mutaciones mediante el usuario administrador que ha creado. En el lateral izquierdo de la consola de AWS AppSync, elija la pestaña Consultas.

Mutación createPicture

1. En la consola de AWS AppSync, elija la pestaña Consultas.
2. Elija Login With User Pools (Inicio de sesión con grupos de usuarios).
3. En el modal, introduzca el ID de cliente de Cognito de muestra que creó la pila de CloudFormation (por ejemplo, 37solo6mmhh7k4v63cqdfgdg5d).
4. Escriba el nombre de usuario que ha pasado como parámetro a la pila de CloudFormation. El valor predeterminado es nadia.
5. Utilice la contraseña temporal que se envió al correo electrónico que proporcionó como parámetro a la pila de CloudFormation (por ejemplo, UserPoolUserEmail).
6. Seleccione Login (Iniciar sesión). Ahora debería poder ver el botón con el nuevo nombre Logout nadia (Cerrar sesión nadia) o cualquier nombre de usuario que haya elegido al crear la pila de CloudFormation (es decir, UserPoolUsername).

Vamos a enviar unas mutaciones de createPicture para rellenar la tabla de imágenes. Ejecute la siguiente consulta de GraphQL dentro de la consola:

```
mutation {
  createPicture(input:{
    owner: "nadia"
    src: "nadia.jpg"
  }) {
    id
    owner
    src
  }
```

```
}
```

La respuesta debe tener un aspecto similar al siguiente:

```
{
  "data": {
    "createPicture": {
      "id": "c6fedbbe-57ad-4da3-860a-ffe8d039882a",
      "owner": "nadia",
      "src": "nadia.jpg"
    }
  }
}
```

Agreguemos algunas imágenes más:

```
mutation {
  createPicture(input:{
    owner: "shaggy"
    src: "shaggy.jpg"
  }) {
    id
    owner
    src
  }
}
```

```
mutation {
  createPicture(input:{
    owner: "rex"
    src: "rex.jpg"
  }) {
    id
    owner
    src
  }
}
```

Ha agregado tres imágenes usando a nadia como usuario administrador.

Mutación createFriendship

Vamos a añadir una entrada de amistad. Ejecute las siguientes mutaciones en la consola.

Nota: Debe permanecer con la sesión iniciada como usuario administrador (el valor predeterminado es nadia).

```
mutation {
  createFriendship(id: "nadia", target: "shaggy")
}
```

La respuesta debe tener un aspecto similar al siguiente:

```
{
  "data": {
    "createFriendship": true
  }
}
```

nadia y shaggy son amigos. rex no es amigo de nadie.

Consulta getPicturesByOwner

En este paso, inicie sesión como el usuario nadia con los grupos de usuarios de Cognito, con las credenciales configuradas al principio de este tutorial. Como nadia, recupere las imágenes propiedad de shaggy.

```
query {
    getPicturesByOwner(id: "shaggy") {
        id
        owner
        src
    }
}
```

Dado que nadia y shaggy son amigos, la consulta debe devolver la imagen correspondiente.

```
{
  "data": {
    "getPicturesByOwner": [
```

```
      {
        "id": "05a16fba-cc29-41ee-a8d5-4e791f4f1079",
        "owner": "shaggy",
        "src": "src://shaggy.jpg"
      }
    ]
  }
}
```

Del mismo modo, si nadia intenta recuperar sus propias imágenes, también se ejecuta satisfactoriamente. El solucionador de canalización se ha optimizado para evitar ejecutar la operación isFriend de GetItem en ese caso. Pruebe la siguiente consulta:

```
query {
    getPicturesByOwner(id: "nadia") {
        id
        owner
        src
    }
}
```

Si habilita el registro en la API (en el panel Settings (Configuración)), configure el nivel de depuración a ALL (TODO) y ejecute la misma consulta de nuevo, devuelve los registros a la ejecución del campo. Observando los registros, puede determinar si la función isFriend se ha devuelto pronto en la etapa Request Mapping Template (Plantilla de mapeo de solicitud):

```
{
  "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
}
```

La clave earlyReturnedValue representa los datos que devolvió la directiva #return.

Por último, aunque rex es miembro del grupo Viewers (Espectadores) de Cognito UserPool, dado que rex no es amigo de nadie, no podrá acceder a ninguna de las imágenes propiedad de shaggy o nadia. Si iniciar sesión como rex en la consola y ejecuta la siguiente consulta:

```
query {
    getPicturesByOwner(id: "nadia") {
        id
        owner
        src
    }
}
```

Obtiene el siguiente error de no autorizado:

```
{
  "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"
    }
  ]
}
```

Ha implementado con éxito la autorización compleja con los solucionadores de canalización.

Tutorial: Delta Sync

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

Las aplicaciones cliente en AWS AppSync almacenan datos mediante el almacenamiento en caché de las respuestas de GraphQL localmente en el disco en una aplicación web/móvil. Los orígenes de datos con control de versiones y las operaciones `Sync` ofrecen a los clientes la posibilidad de realizar el proceso de sincronización mediante un único solucionador. Esto permite a los clientes incorporar a su memoria caché local los resultados de una consulta de base que podría tener una gran cantidad de registros y, a continuación, recibir solo los datos alterados desde la última consulta (las actualizaciones delta). Al permitir que los clientes realicen la incorporación de la caché con una consulta inicial y las actualizaciones incrementales con otra, puede mover el cálculo desde la aplicación cliente al backend. Esto es sustancialmente más eficiente para las aplicaciones cliente que cambian con frecuencia entre estados en línea y sin conexión.

Para implementar Delta Sync, la consulta `Sync` utiliza la operación `Sync` en un origen de datos con control de versiones. Cuando una mutación de AWS AppSync cambia un elemento en un origen de datos con control de versiones, también se almacena un registro de ese cambio en la tabla Delta. Puede optar por utilizar diferentes tablas Delta (por ejemplo, una por tipo y una por área de dominio) para otros orígenes de datos con control de versiones, o bien una sola tabla Delta para su API. AWS AppSync recomienda no usar una sola tabla Delta para varias API, con el fin de evitar colisiones entre claves principales.

Además, los clientes Delta Sync también pueden recibir una suscripción como argumento y, a continuación, la suscripción de coordenadas del cliente vuelve a conectar y escribe entre transiciones de sin conexión y online. Delta Sync realiza esto reanudando automáticamente las suscripciones (incluido el retardo exponencial y el reintento con fluctuación a través de diferentes escenarios de error de red) y almacenando los eventos en una cola. Entonces la consulta delta o base adecuada se ejecuta antes de fusionar cualquier evento de la cola y, por último, procesar las suscripciones como normales.

La documentación para las opciones de configuración del cliente, incluido Amplify DataStore, está disponible en el sitio web de Amplify Framework. En esta documentación se describe cómo configurar las operaciones `Sync` y los orígenes de datos de DynamoDB con control de versiones para que funcionen con el cliente de Delta Sync y obtener un acceso óptimo a los datos.

Configuración en un clic

Para configurar automáticamente el punto de conexión de GraphQL en AWS AppSync con todos los solucionadores configurados y los recursos de AWS necesarios, utilice la siguiente plantilla de AWS CloudFormation:

Esta pila crea los siguientes recursos en su cuenta:

- 2 tablas de DynamoDB (Base y Delta)
- 1 API de AWS AppSync con clave de API
- 1 rol de IAM con política para tablas de DynamoDB

Se utilizan dos tablas para particionar las consultas de Sync en una segunda tabla que actúa como un diario de los eventos no atendidos cuando los clientes estaban sin conexión. Para mantener las consultas eficientes en la tabla delta, se usan Amazon DynamoDB TTLs con el fin de mejorar automáticamente los eventos según sea necesario. El tiempo TTL se puede configurar según sus necesidades en el origen de datos (puede configurarlo en 1 hora, 1 día, etc.).

Esquema

Para demostrar Delta Sync, la aplicación de ejemplo crea un esquema de Publicaciones respaldado por una tabla Base y Delta en DynamoDB. AWS. AppSync escribe automáticamente las mutaciones

en ambas tablas. La consulta Sync extrae registros de la tabla Base o Delta según proceda y se define una sola suscripción para mostrar cómo los clientes pueden aprovechar esto en su lógica de reconexión.

```
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
}
```

El esquema GraphQL es estándar, pero vale la pena señalar un par de cosas antes de avanzar. Primero, todas las mutaciones se escriben automáticamente en la tabla Base y, luego, en la tabla Delta. La tabla de Base es el origen central de información de estado mientras que la tabla Delta es el diario. Si no se pasa el `lastSync: AWSTimestamp`, la consulta `syncPosts` se ejecuta con respecto a la tabla Base e hidrata la caché, además de ejecutarse de forma periódica como

un proceso de alcance global para casos límite en los que los clientes están sin conexión más del tiempo TTL configurado en la tabla Delta. Si pasa el `lastSync: AWSTimestamp`, la consulta `syncPosts` se ejecuta con respecto a la tabla Delta y los clientes la utilizan para recuperar los eventos cambiados desde la última vez que estuvieron sin conexión. Los clientes de Amplify pasan automáticamente el valor `lastSync: AWSTimestamp` y lo conservan en disco debidamente.

El campo _deleted de Post se utiliza para las operaciones DELETE. Cuando los clientes están sin conexión y se eliminan registros de la tabla Base, este atributo notifica a los clientes que están realizando la sincronización que expulsen los elementos de su caché local. En los casos en que los clientes están offline durante períodos más largos y el elemento se ha eliminado antes de que el cliente pueda recuperar este valor con una consulta Delta Sync, el evento de alcance global de consulta base (configurable en el cliente) se ejecuta y elimina el elemento de la caché. Este campo se marca como opcional porque solo devuelve un valor al ejecutar una consulta Sync que ha eliminado los elementos presentes.

Mutaciones

Para todas las mutaciones, AWS AppSync realiza una operación estándar de creación/actualización/eliminación en la tabla Base y, además, registra automáticamente el cambio en la tabla Delta . Puede reducir o ampliar el tiempo para conservar los registros modificando el valor `DeltaSyncTableTTL` en el origen de datos. Para las organizaciones con una alta velocidad de datos, se recomienda que sea breve. O bien, si sus clientes están sin conexión durante períodos más largos, puede ser prudente mantenerlo más tiempo.

Consultas Sync

La consulta base es una operación de DynamoDB Sync sin un valor de `lastSync` especificado. Para muchas organizaciones, esto funciona porque la consulta de base solo se ejecuta durante el inicio y de forma periódica a partir de entonces.

La consulta delta es una operación de DynamoDB Sync con un valor de `lastSync` especificado. La consulta delta se ejecuta cada vez que el cliente vuelva a estar online de un estado sin conexión (siempre que el tiempo periódico de consulta de base no se haya activado para ejecutarse). Los clientes realizan automáticamente el seguimiento de la última vez que ejecutaron de forma correcta una consulta para sincronizar datos.

Cuando se ejecuta una consulta delta, el solucionador de la consulta utiliza `ds_pk` y `ds_sk` para consultar únicamente los registros que han cambiado desde la última vez que el cliente realizó una sincronización. El cliente almacena la respuesta GraphQL adecuada.

Para obtener más información sobre cómo ejecutar consultas Sync, consulte la documentación de la operación de sincronización.

Ejemplo

Vamos a comenzar por llamar a una mutación `createPost` para crear un elemento:

```
mutation create {
  createPost(input: {author: "Nadia", title: "My First Post", content: "Hello World"})
 {
    id
    author
    title
    content
    _version
    _lastChangedAt
    _deleted
  }
}
```

El valor de retorno de esta mutación tendrá el siguiente aspecto:

```
{
  "data": {
    "createPost": {
      "id": "81d36bbb-1579-4efe-92b8-2e3f679f628b",
      "author": "Nadia",
      "title": "My First Post",
      "content": "Hello World",
      "_version": 1,
      "_lastChangedAt": 1574469356331,
      "_deleted": null
    }
  }
}
```

Si examina el contenido de la tabla Base, verá un registro similar a este:

```
{
  "_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"
  }
}
```

Si examina el contenido de la tabla Delta, verá un registro que tiene el siguiente aspecto:

```
{
  "_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"
  }
}
```

Ahora podemos simular una consulta Base que un cliente ejecutará para hidratar su almacén de datos local usando una consulta `syncPosts` como esta:

```
query baseQuery {
  syncPosts(limit: 100, lastSync: null, nextToken: null) {
    items {
      id
      author
      title
      content
      _version
      _lastChangedAt
    }
    startedAt
    nextToken
  }
}
```

El valor devuelto de esta consulta Base tiene este aspecto:

```
{
  "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
    }
  }
```

```
}
```

Más adelante, vamos a guardar el valor de `startedAt` para simular una consulta Delta, pero antes tenemos que hacer un cambio en nuestra tabla. Vamos a utilizar la mutación `updatePost` para modificar nuestro Post existente:

```
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
  }
}
```

El valor de retorno de esta mutación tendrá el siguiente aspecto:

```
{
  "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
    }
  }
}
```

Si examina el contenido de la tabla Base ahora, debería ver el elemento actualizado:

```
{
  "_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"
  }
}
```

Si examina el contenido de la tabla Delta ahora, debería ver dos registros:

1. Un registro de cuando se creó el elemento
2. Un registro de cuando se actualizó el elemento

El nuevo elemento tendrá este aspecto:

```
{
  "_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"
  }
}
```

Ahora podemos simular una consulta Delta para recuperar las modificaciones que ocurrieron mientras un cliente estaba sin conexión. Utilizaremos el valor `startedAt` devuelto por nuestra consulta Base para realizar la solicitud:

```
query delta {
  syncPosts(limit: 100, lastSync: 1574469602238, nextToken: null) {
    items {
      id
      author
      title
      content
      _version
    }
    startedAt
    nextToken
  }
}
```

El valor devuelto de esta consulta Delta tendrá este aspecto:

```
{
  "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
    }
  }
}
```

Configuración y ajustes

AWS AppSync le permite:

- Almacene en caché datos que se solicitan con frecuencia pero que es poco probable que cambien de una solicitud a otra. Esto puede reducir la carga de sus solucionadores. Para obtener más información, consulte the section called “Almacenamiento en caché y compresión”.
- Versionar objetos de GraphQL para gestionar y evitar conflictos entre varios clientes. Para obtener más información, consulte the section called “Detección de conflictos y sincronización”.
- Utilice los nombres de dominio personalizados para configurar un dominio único y fácil de recordar que funcione tanto para las API de GraphQL como para las de tiempo real. Para obtener más información, consulte Configuración de nombres de dominio personalizados.
- Permita el acceso a sus API de GraphQL a través de una VPC. Para obtener más información, consulte Uso de API privadas de AWS AppSync.
- Habilite la introspección y establezca la profundidad de las consultas y los límites de resolución por consulta. Para obtener más información, consulte Límites de configuración.

Además, AWS AppSync incluye las siguientes herramientas estándar de AWS para el registro, la monitorización y el seguimiento:

- Inicio de sesión en AWS CloudTrail
- Monitorización con Amazon CloudWatch
- Rastreo con AWS X-Ray

Almacenamiento en caché y compresión

Gracias a las capacidades de almacenamiento en caché de datos en el servidor de AWS AppSync, los datos están disponibles en una caché en memoria de alta velocidad, lo que mejora el rendimiento y reduce la latencia. Esto reduce la necesidad de acceder directamente a los orígenes de datos. El almacenamiento en caché está disponible tanto para los solucionadores de unidades como para los de canalización.

AWS AppSync también permite comprimir las respuestas de la API para que el contenido se cargue y descargue más rápido. Esto puede reducir la carga de trabajo de las aplicaciones y, al mismo

tiempo, reducir los cargos por transferencia de datos. El comportamiento de la compresión es configurable y se puede ajustar según criterio propios.

Consulte esta sección para obtener ayuda para definir el comportamiento deseado del almacenamiento en caché y la compresión en el servidor en su API de AWS AppSync.

Tipos de instancias

AWS AppSync aloja instancias de Amazon ElastiCache para Redis en la misma cuenta de AWS y región de AWS que la API de AWS AppSync.

Están disponibles los siguientes tipos de instancias de ElastiCache para Redis:

small

1 vCPU, 1,5 GiB de RAM, rendimiento de red de bajo a moderado

medium

2 vCPU, 3 GiB de RAM, rendimiento de red de bajo a moderado

large

2 vCPU, 12,3 GiB de RAM, rendimiento de red de hasta 10 Gigabits

xlarge

4 vCPU, 25,05 GiB de RAM, rendimiento de red de hasta 10 Gigabits

2xlarge

8 vCPU, 50,47 GiB de RAM, rendimiento de red de hasta 10 Gigabits

4xlarge

16 vCPU, 101,38 GiB de RAM, rendimiento de red de hasta 10 Gigabits

8xlarge

32 vCPU, 203,26 GiB de RAM, rendimiento de red de 10 Gigabits (no disponible en todas las regiones)

12xlarge

48 vCPU, 317,77 GiB de RAM, rendimiento de red de 10 Gigabits

> **Note**
>
> Históricamente, se indicaba un tipo de instancia específico (por ejemplo `t2.medium`). A partir de julio de 2020, estos tipos de instancias antiguas siguen estando disponibles, pero su uso está obsoleto y se desaconseja. Le recomendamos que utilice los tipos de instancia genéricos que se describen aquí.

Comportamiento del almacenamiento en caché

Los siguientes son los comportamientos relacionados con el almacenamiento en caché.

Ninguno

No hay almacenamiento en caché del lado del servidor.

Almacenamiento en caché completo de solicitudes

Si los datos no están en la caché, se recuperarán del origen de datos y rellenarán la caché hasta el vencimiento del tiempo de vida (TTL). Todas las solicitudes posteriores a su API se devuelven desde la caché. Esto significa que no se contacta directamente con los orígenes de datos a menos que el TTL venza. En esta configuración, como claves de almacenamiento en caché utilizamos el contenido de los mapas `context.arguments` y `context.identity`.

Almacenamiento en caché por solucionador

Con esta configuración, se debe dar de alta explícitamente en cada solucionador para que almacene las respuestas en caché. Puede especificar un TTL y claves de almacenamiento en caché en el solucionador. Las claves de almacenamiento en caché que puede especificar son los mapas `context.arguments`, `context.source` y `context.identity` de nivel superior y/o los campos de cadena de esos mapas. El valor de TTL es obligatorio, pero las claves de almacenamiento en caché son opcionales. Si no especifica ninguna clave de almacenamiento en caché, los valores predeterminados son el contenido de los mapas `context.arguments`, `context.source` y `context.identity`.

Por ejemplo, podría utilizar las combinaciones siguientes:

- context.arguments y context.source
- context.arguments y context.identity.sub
- context.arguments.id o context.arguments.InputType.id

- context.source.id y context.identity.sub
- context.identity.claims.username

Si especifica solo un TTL y ninguna clave de almacenamiento en caché, el comportamiento del solucionador es el mismo que el del almacenamiento en caché completo de las solicitudes.

Tiempo de vida en caché

Este ajuste define la cantidad de tiempo de almacenamiento de entradas en caché en la memoria. El TTL máximo es de 3.600 s (1 h), después de lo cual las entradas se eliminan automáticamente.

Cifrado de caché

El cifrado de caché tiene las dos versiones siguientes. Estos son similares a los ajustes permitidos por ElastiCache para Redis. La configuración de cifrado solo se puede activar al habilitar por primera vez el almacenamiento en caché para su API de AWS AppSync.

- Cifrado en tránsito: las solicitudes entre AWS AppSync, la caché y los orígenes de datos (excepto los orígenes de datos HTTP no seguros) se cifrarán en el nivel de red. Como se requiere cierto procesamiento para cifrar y descifrar los datos en los puntos de conexión, habilitar el cifrado en tránsito puede afectar al rendimiento.
- Cifrado en reposo: los datos guardados en el disco desde la memoria durante las operaciones de intercambio se cifran en la instancia de la caché. Esta configuración también afecta al rendimiento.

Para invalidar las entradas de la caché, puede realizar una llamada a la API de vaciado de la caché mediante la consola de AWS AppSync o con AWS Command Line Interface (AWS CLI).

Para obtener más información, consulte el tipo de datos ApiCache en la Referencia de la API de AWS AppSync.

Expulsión de caché

Al configurar el almacenamiento en caché en el servidor de AWS AppSync, puede configurar un TTL máximo. Este valor define la cantidad de tiempo que las entradas almacenadas en caché se almacenan en la memoria. En situaciones en las que deba eliminar entradas específicas de la memoria caché, puede utilizar la utilidad de extensiones `evictFromApiCache` de AWS AppSync en la solicitud o la respuesta del solucionador. (Por ejemplo, cuando los datos de los orígenes de datos han cambiado y la entrada de la caché ahora está obsoleta). Para expulsar un elemento de la

caché, debe conocer su clave. Por este motivo, si debe expulsar los elementos de forma dinámica, le recomendamos que utilice el almacenamiento en caché por solucionador y que defina de forma explícita una clave para añadir entradas a la caché.

Expulsar una entrada en caché

Para expulsar un elemento de la caché, utilice la utilidad de extensiones `evictFromApiCache`. Especifique el nombre del tipo y el nombre del campo y, a continuación, proporcione un objeto de elementos clave-valor para crear la clave de la entrada que desee expulsar. En el objeto, cada clave representa una entrada válida del objeto `context` que se utiliza en la lista `cachingKey` de solucionadores almacenada en caché. Cada valor es el valor real que se utiliza para crear el valor de la clave. Debe colocar los elementos del objeto en el mismo orden que las claves de almacenamiento en caché de la lista `cachingKey` del solucionador almacenado en caché.

Por ejemplo, vea el esquema siguiente:

```
type Note {
  id: ID!
  title: String
  content: String!
}

type Query {
  getNote(id: ID!): Note
}

type Mutation {
  updateNote(id: ID!, content: String!): Note
}
```

En este ejemplo, puede habilitar el almacenamiento en caché por solucionador y, a continuación, habilitarlo para la consulta `getNote`. A continuación, puede configurar la clave de almacenamiento en caché para que contenga `[context.arguments.id]`.

Cuando se intenta obtener una `Note`, para generar la clave de caché, AWS AppSync realiza una búsqueda en su caché del servidor utilizando el argumento `id` de la consulta `getNote`.

Al actualizar una `Note`, debe expulsar la entrada de la nota específica para asegurarse de que la siguiente solicitud la obtenga del origen de datos del backend. Para ello, debe crear un controlador de solicitudes.

En el siguiente ejemplo se muestra una forma de gestionar la expulsión mediante este método:

```
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;
```

Otra opción puede consistir en gestionar la expulsión en el controlador de respuestas.

Cuando se procesa la mutación `updateNote`, AWS AppSync intenta expulsar la entrada. Si una entrada se borra correctamente, la respuesta contiene un valor `apiCacheEntriesDeleted` en el objeto `extensions` que muestra cuántas entradas se han borrado:

```
"extensions": {  "apiCacheEntriesDeleted": 1}
```

Expulsar una entrada de caché en función de su identidad

Puede crear claves de almacenamiento en caché basadas en varios valores del objeto `context`.

Por ejemplo, tome el siguiente esquema que usa grupos de usuarios de Amazon Cognito como modo de autenticación predeterminado y está respaldado por un origen de datos de 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
}
```

Los tipos de objetos de `Note` se guardan en una tabla de DynamoDB. La tabla tiene una clave compuesta que utiliza el nombre de usuario de Amazon Cognito como clave principal y el `id` (un slug) de `Note` como clave de partición. Se trata de un sistema de varios inquilinos que permite que varios usuarios alojen y actualicen sus objetos privados de `Note`, que nunca se comparten.

Como se trata de un sistema de lectura intensiva, la consulta `getNote` se almacena en caché por solucionador, con la clave de almacenamiento en caché compuesta por `[context.identity.username, context.arguments.id]`. Cuando se actualiza una `Note`, puede desalojar la entrada correspondiente a esa `Note` específica. Debe añadir los componentes del objeto en el mismo orden en que se especifican en la lista `cachingKeys` del solucionador.

El ejemplo siguiente muestra esto:

```
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;
```

Un sistema de backend también puede actualizar la `Note` y expulsar la entrada. Tomemos por ejemplo esta mutación:

```
type Mutation {
  updateNoteFromBackend(id: ID!, content: String!, username: ID!): Note @aws_iam
}
```

Puede expulsar la entrada, pero añadir los componentes de la clave de almacenamiento en caché al objeto `cachingKeys`.

En el ejemplo siguiente, la expulsión se produce en la respuesta del solucionador:

```
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;
```

En los casos en que los datos de backend se hayan actualizado fuera de AWS AppSync, puede expulsar un elemento de la caché llamando a una mutación que utilice un origen de datos `NONE`.

Compresión de respuestas de API

AWS AppSync permite a los clientes solicitar cargas comprimidas. Si se solicita, las respuestas de la API se comprimen y se devuelven en respuesta a solicitudes que indican que se prefiere el contenido comprimido. Las respuestas de la API comprimida se cargan más rápido, el contenido se descarga más rápido y, además, es posible que también se reduzcan los cobros por transferencia de datos.

> **Note**
>
> La compresión está disponible en todas las API nuevas creadas después del 1 de junio de 2020.
> AWS AppSync comprime los objetos sobre la base del mejor esfuerzo. En raras ocasiones, AWS AppSync puede omitir la compresión en función de una serie de factores, como la capacidad actual.

AWS AppSync puede comprimir tamaños de carga útil de consultas de GraphQL entre 1000 y 10 000 000 bytes. Para habilitar la compresión, un cliente debe enviar el encabezado `Accept-Encoding` con el valor `gzip`. La compresión se puede verificar comprobando el valor del encabezado `Content-Encoding` en la respuesta (`gzip`).

El explorador de consultas de la consola de AWS AppSync establece automáticamente el valor del encabezado de la solicitud de forma predeterminada. Si ejecuta una consulta que tiene una respuesta lo suficientemente grande, la compresión se puede confirmar con las herramientas de desarrollo de su navegador.

Configuración de nombres de dominio personalizados

Con AWS AppSync, puede usar los nombres de dominio personalizados para configurar un dominio único y fácil de recordar que funcione tanto para las API de GraphQL como para las de tiempo real.

En otras palabras, puede utilizar direcciones URL de puntos de conexión sencillas y fáciles de recordar con los nombres de dominio que desee creando nombres de dominio personalizados que asocie a las API de AWS AppSync de su cuenta.

Al configurar una API de AWS AppSync, se aprovisionan dos puntos de conexión:

Punto de conexión de GraphQL de AWS AppSync:

```
https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql
```

Punto de conexión de AWS AppSync en tiempo real:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/
graphql
```

Con los nombres de dominio personalizados, puede interactuar con ambos puntos de conexión mediante un único dominio. Por ejemplo, si lo configura `api.example.com` como su dominio personalizado, puede interactuar con sus puntos de conexión de GraphQL y en tiempo real mediante estas URL:

Punto de conexión de GraphQL del dominio personalizado de AWS AppSync:

```
https://api.example.com/graphql
```

Punto de conexión en tiempo real del dominio personalizado de AWS AppSync:

```
wss://api.example.com/graphql/realtime
```

> **Note**
>
> Las API de AWS AppSync admiten TLS 1.2 y TLS 1.3 para nombres de dominio personalizados.

Registro y configuración de un nombre de dominio

Debe disponer de un nombre de dominio de Internet registrado para poder configurar nombres de dominio personalizados para sus API de AWS AppSync. Si es necesario, puede registrar un dominio de Internet utilizando Amazon Route 53 domain registration o un registrador de dominios de terceros de su elección. Para obtener más información, consulte ¿Qué es Amazon Route 53? en la Guía para desarrolladores de Amazon Route 53.

Un nombre de dominio personalizado de la API puede ser el nombre de un subdominio o el dominio raíz (lo que recibe el nombre de "vértice de zona") de un dominio de Internet registrado. Después de crear un nombre de dominio personalizado en AWS AppSync, debe crear o actualizar el registro de recursos del proveedor de DNS para asignarlo al punto de conexión de la API. Si no se realiza esta asignación, las solicitudes de API vinculadas al nombre de dominio personalizado no pueden llegar a AWS AppSync.

Creación de un nombre de dominio personalizado en AWS AppSync

Al crear un nombre de dominio personalizado para una API de AWS AppSync, se configura una distribución Amazon CloudFront. Pero debe configurar un registro de DNS para asignar el nombre de dominio personalizado al nombre de dominio de distribución de CloudFront. Esta asignación es obligatoria para dirigir solicitudes de API que estén enlazadas para el nombre de dominio personalizado de AWS AppSync a través de la distribución de CloudFront asignada. También debe proporcionar un certificado para el nombre de dominio personalizado.

Para configurar el nombre de dominio personalizado o actualizar su certificado, debe tener permiso para actualizar las distribuciones de CloudFront y describir el certificado de AWS Certificate Manager (ACM) que tenga pensado usar. Para conceder estos permisos, asocie la siguiente instrucción de política de AWS Identity and Access Management (IAM) a un usuario, grupo o rol de IAM en su cuenta:

```
{
  "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 admite nombres de dominio personalizados a través de la indicación de nombre de servidor (SNI) de la distribución de CloudFront. Para obtener más información acerca de cómo usar nombres de dominio personalizados en una distribución de CloudFront, incluido el formato de certificado necesario y la longitud máxima de clave de certificado, consulte Usar HTTPS con CloudFront en la Guía para desarrolladores de Amazon CloudFront.

Para configurar un nombre de dominio personalizado como el nombre de host de la API, el propietario de la API debe proporcionar un certificado SSL/TLS para el nombre de dominio personalizado. Para proporcionar un certificado, realice una de las operaciones siguientes:

- Solicite un certificado nuevo en ACP o importe uno emitido a ACM por una entidad de certificación externa en la región de AWS `us-east-1` (Este de EE. UU. [Norte de Virginia]). Para obtener más información acerca de ACM, consulte ¿Qué es AWS Certificate Manager? en la Guía del usuario de AWS Certificate Manager.
- Proporcione un certificado de servidor de IAM. Para obtener más información, consulte Administración de certificados de servidor en IAM en la Guía del usuario de IAM.

Nombres de dominio personalizados comodín en AWS AppSync

AWS AppSync también admite nombres de dominio personalizados comodín. Para configurar un nombre de dominio personalizado comodín, especifique un carácter comodín (*) como primer subdominio de un dominio personalizado. Esto representa todos los subdominios posibles del dominio raíz. Por ejemplo, el nombre de dominio personalizado comodín `*.example.com` produce subdominios como `a.example.com`, `b.example.com` y `c.example.com`. Todos estos subdominios se dirigen al mismo dominio.

Para utilizar un nombre de dominio personalizado comodín en AWS AppSync, debe proporcionar un certificado emitido por ACM que contenga un nombre comodín que pueda proteger varios sitios en el mismo dominio. Para obtener más información, consulte las características del certificado de ACM en la Guía del usuario de AWS Certificate Manager.

Detección de conflictos y sincronización

Orígenes de datos versionados

AWS AppSync actualmente admite el control de versiones en los orígenes de datos de DynamoDB. Las operaciones de detección de conflictos, resolución de conflictos y sincronización requieren un origen de datos de tipo `Versioned`. Cuando habilite el control de versiones en un origen de datos, AWS AppSync realizará automáticamente las acciones siguientes:

- Mejorar los elementos con metadatos de control de versiones de objetos.
- Registrar los cambios realizados en los elementos con mutaciones de AWS AppSync en una tabla Delta.
- Mantener los elementos eliminados en la tabla Base con una “lápida” durante un período de tiempo configurable.

Configuración del origen de datos con control de versiones

Al habilitar el control de versiones en un origen de datos de DynamoDB, especifique los siguientes campos:

`BaseTableTTL`

El número de minutos que se conservarán los elementos eliminados en la tabla Base con una “lápida”, un campo de metadatos que indica que el elemento se ha eliminado. Puede establecer este valor en 0 si desea que los elementos se supriman inmediatamente cuando se eliminen. Este campo es obligatorio.

`DeltaSyncTableName`

Nombre de la tabla en la que se almacenan los cambios realizados en los elementos con mutaciones de AWS AppSync. Este campo es obligatorio.

`DeltaSyncTableTTL`

Número de minutos que se deben conservar los elementos en la tabla Delta. Este campo es obligatorio.

Tabla de Delta Sync

AWS AppSync actualmente es compatible con Delta Sync Logging para mutaciones que utilizan las operaciones `PutItem`, `UpdateItem` y `DeleteItem` de DynamoDB.

Cuando una mutación de AWS AppSync cambia un elemento en un origen de datos con control de versiones, un registro de ese cambio se almacena en una tabla Delta optimizada para actualizaciones incrementales. Puede optar por utilizar diferentes tablas Delta (por ejemplo, una por tipo y una por área de dominio) para otros orígenes de datos con control de versiones, o bien una sola tabla Delta para su API.AWS AppSync recomienda no usar una sola tabla Delta para varias API, con el fin de evitar colisiones entre claves principales.

El esquema requerido para esta tabla es el siguiente:

`ds_pk`

Valor de cadena que se utiliza como clave de partición. Se construye concatenando el nombre del origen de datos Base y el formato ISO 8601 de la fecha en que se produjo el cambio (p. ej. `Comments:2019-01-01`).

Cuando la marca `customPartitionKey` de la plantilla de mapeo de VTL se establece como nombre de columna de la clave de partición (consulte Referencia de las plantillas de mapeo de solucionador para DynamoDB para AWS AppSync en la Guía para desarrolladores de AWS AppSync), el formato de `ds_pk` cambia y la cadena se construye añadiéndole el valor de la clave de partición en el nuevo registro de la tabla Base. Por ejemplo, si el registro de la tabla Base tiene un valor de clave de partición `1a` y un valor de clave de clasificación `2b`, el nuevo valor de la cadena será: `Comments:2019-01-01:1a`.

`ds_sk`

Valor de cadena que se utiliza como clave de ordenación. Se construye concatenando el formato ISO 8601 del momento en que se produjo el cambio, la clave principal del elemento y la versión del elemento. La combinación de estos campos garantiza la unicidad de cada entrada de la tabla Delta (por ejemplo, si la hora es `09:30:00`, el ID es `1a` y la versión es `2`, el valor sería `09:30:00:1a:2`).

Cuando la marca `customPartitionKey` de la plantilla de mapeo de VTL se establece como nombre de columna de la clave de partición (consulte Referencia de las plantillas de mapeo de solucionador para DynamoDB para AWS AppSync en la Guía para desarrolladores de AWS AppSync), el formato de `ds_sk` cambia y la cadena se construye sustituyendo el valor de la

clave de combinación con el valor de la clave de clasificación en la tabla Base. Con el ejemplo anterior, si el registro de la tabla Base tiene un valor de clave de partición `1a` y un valor de clave de clasificación `2b`, el nuevo valor de la cadena será: `09:30:00:2b:3`.

`_ttl`

Valor numérico que almacena la marca de tiempo, en segundos transcurridos desde la fecha de inicio, en cuyo momento se debe suprimir un elemento de la tabla Delta. Este valor se determina agregando el valor `DeltaSyncTableTTL` configurado en el origen de datos al momento en que se produjo el cambio. Este campo debe configurarse como el atributo TTL de DynamoDB.

El rol de IAM configurado para su uso con la tabla Base también debe contener permisos para operar con la tabla Delta. En este ejemplo, se muestra la política de permisos para una tabla Base denominada `Comments` y una tabla Delta denominada `ChangeLog`:

```
{
    "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/*"
            ]
        }
    ]
}
```

Metadatos de origen de datos con control de versiones

AWS AppSync administra automáticamente los campos de metadatos de los orígenes de datos `Versioned` en su nombre. Si modifica estos campos usted mismo, puede causar errores en la aplicación o pérdida de datos. Estos campos incluyen:

`_version`

Un contador con aumento monotónico que se actualiza cada vez que se produce un cambio en un elemento.

`_lastChangedAt`

Un valor numérico que almacena la marca de tiempo, en milisegundos transcurridos desde la fecha de inicio, en cuyo momento se modificó por última vez un elemento.

`_deleted`

Un valor booleano de “lápida” que indica que se ha eliminado un elemento. Pueden utilizarlo las aplicaciones para desalojar elementos eliminados de los almacenes de datos locales.

`_ttl`

Valor numérico que almacena la marca de tiempo, en segundos transcurridos desde la fecha de inicio, en cuyo momento se debe suprimir un elemento del origen de datos subyacente.

`ds_pk`

Un valor de cadena que se utiliza como clave de partición para las tablas Delta.

`ds_sk`

Un valor de cadena que se utiliza como clave de ordenación para las tablas Delta.

`gsi_ds_pk`

Un atributo de valor de cadena que se genera para soportar un índice secundario global como clave de partición. Solo se incluirá si las marcas `populateIndexFields` y `customPartitionKey` están habilitadas en la plantilla de mapeo de VTL (consulte la Referencia de las plantillas de mapeo de solucionador para DynamoDB en la Guía para desarrolladores de AWS AppSync). Si está habilitada, el valor se generará concatenando el nombre de el origen de datos Base y el formato ISO 8601 de la fecha en que se produjo el cambio (por ejemplo, si la tabla Base se denomina Comentarios, este registro se establecerá como `Comments:2019-01-01`).

`gsi_ds_sk`

Un atributo de valor de cadena que se genera para soportar un índice secundario global como clave de clasificación. Solo se incluirá si las marcas `populateIndexFields` y `customPartitionKey` están habilitadas en la plantilla de mapeo de VTL (consulte la Referencia de las plantillas de mapeo de solucionador para DynamoDB en la Guía para desarrolladores de AWS AppSync). Si están habilitadas, el valor se generará concatenando el formato ISO 8601 de la hora en que se produjo el cambio, la clave de partición del elemento de la tabla Base, la clave de clasificación del elemento de la tabla Base y la versión del elemento (por ejemplo, para la hora `09:30:00`, un valor de clave de partición `1a`, un valor de clave de clasificación `2b` y una versión `3`, sería `09:30:00:1a#2b:3`).

Estos campos de metadatos afectarán al tamaño total de los elementos en el origen de datos subyacente. AWS AppSync recomienda reservar un almacenamiento de 500 bytes más el tamaño máximo de la clave principal para los metadatos de los orígenes de datos versionados al diseñar su aplicación. Para utilizar estos metadatos en aplicaciones cliente, incluya los campos `_deleted`, `_version` y `_lastChangedAt` en los tipos de GraphQL y en el conjunto de selección de mutaciones.

Detección y resolución de conflictos

Cuando se producen escrituras simultáneas con AWS AppSync, puede configurar estrategias de detección y resolución de conflictos para gestionar las actualizaciones de forma adecuada. La detección de conflictos determina si la mutación está en conflicto con el elemento real escrito en el origen de datos. La detección de conflictos se habilita estableciendo en `VERSION` el valor del campo `conflictDetection` en SyncConfig.

La resolución de conflictos es la acción que se realiza en caso de que se detecte un conflicto. Esto se determina estableciendo el campo Conflict Handler (Controlador de conflictos) en SyncConfig. Existen tres estrategias de resolución de conflictos:

- OPTIMISTIC_CONCURRENCY (Simultaneidad optimista)
- AUTOMERGE (Combinar automáticamente)
- LAMBDA

A continuación se detallan cada una de estas estrategias de resolución de conflictos.

AppSync incrementa automáticamente las versiones durante las operaciones de escritura y los clientes no deben modificarlas ni tampoco deben modificarse fuera de un solucionador configurado con un origen de datos habilitado para versiones. Si se hace, se modificará el comportamiento de consistencia del sistema y podría dar lugar a la pérdida de datos.

Optimistic Concurrency (Simultaneidad optimista)

Optimistic Concurrency (Simultaneidad optimista) es una estrategia de resolución de conflictos que AWS AppSync proporciona para los orígenes de datos con control de versiones. Si el solucionador de conflictos se ha establecido en Optimistic Concurrency (Simultaneidad optimista) y se detecta que una mutación entrante tiene una versión distinta de la versión real del objeto, el controlador de conflictos simplemente rechazará la solicitud entrante. Dentro de la respuesta de GraphQL, se proporcionará el elemento existente en el servidor que tiene la última versión. A continuación, se espera que el cliente gestione este conflicto localmente y vuelva a intentar la mutación con la versión actualizada del elemento.

Automerge (Combinar automáticamente)

Automerge (Combinar automáticamente) proporciona a los desarrolladores una manera fácil de configurar una estrategia de resolución de conflictos sin tener que escribir la lógica del lado del cliente con el fin de combinar manualmente los conflictos que no se han podido gestionar mediante otras estrategias. Automerge (Combinar automáticamente) respeta un estricto conjunto de reglas al combinar los datos para resolver conflictos. Los principios de Automerge (Combinar automáticamente) se refieren al tipo de datos subyacente del campo de GraphQL. Son los siguientes:

- Conflicto en un campo escalar: escalar de GraphQL o cualquier campo que no sea una colección (es decir, List, Set, Map). Rechazar el valor entrante para el campo escalar y seleccionar el valor existente en el servidor.
- Conflicto en una lista: el tipo de GraphQL y el tipo de la base de datos son listas. Concatenar la lista entrante con la lista existente en el servidor. Los valores de lista de la mutación entrante se anexarán al final de la lista en el servidor. Se conservarán los valores duplicados.
- Conflicto en un conjunto: el tipo de GraphQL es una lista y el tipo de la base de datos es un conjunto. Aplicar una unión de conjuntos utilizando el conjunto entrante y el conjunto existente en el servidor. Se respetan las propiedades de un conjunto, lo que significa que no hay entradas duplicadas.
- Cuando una mutación entrante agrega un nuevo campo al elemento o se realiza en un campo con el valor `null`, se combinan con el elemento existente.

- Conflicto en un mapa: cuando el tipo de datos subyacente en la base de datos es un mapa (es decir, un documento de clave-valor), se aplican las reglas anteriores mientras se analiza y procesa cada propiedad del mapa.

Automerge (Combinar automáticamente) se ha diseñado para detectar, combinar y reintentar automáticamente las solicitudes con una versión actualizada, lo que evita al cliente la necesidad de combinar manualmente los datos en conflicto.

Para mostrar un ejemplo de cómo Automerge (Combinar automáticamente) gestiona un conflicto en un tipo escalar. Usaremos el siguiente registro como punto de partida.

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "_version" : 4
}
```

Ahora, una mutación entrante podría estar intentando actualizar el elemento, pero con una versión anterior, ya que el cliente aún no se ha sincronizado con el servidor. Su aspecto es el siguiente:

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 55,
  "_version" : 2
}
```

Observe la versión obsoleta de 2 en la solicitud entrante. Durante este flujo, Automerge (Combinar automáticamente) combinará los datos rechazando la actualización de campo 'jersey' a '55' y mantendrá el valor en '5', lo que dará lugar a la siguiente imagen del elemento que se guarda en el servidor.

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "_version" : 5 # version is incremented every time automerge performs a merge that is
 stored on the server.
```

```
}
```

Dado el estado del elemento mostrado anteriormente en la versión 5, ahora suponga que una mutación entrante intenta mutar el elemento con la siguiente imagen:

```
{
  "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
}
```

Hay tres puntos de interés en la mutación entrante. El nombre, un escalar, se ha cambiado, pero se han agregado dos nuevos campos “interests” (intereses), que es un conjunto, y “points” (puntos), que es una Lista. En este escenario, se detectará un conflicto debido a la discrepancia de versión. Automerge (Combinar automáticamente) respeta sus propiedades, rechaza el cambio de nombre debido a que es un escalar y agrega los campos que no presentan conflictos. Esto hace que el elemento que se guarda en el servidor aparezca de la siguiente manera.

```
{
  "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
}
```

Con la imagen actualizada del elemento con la versión 6, ahora suponga que una mutación entrante (con otra discrepancia de versión) intenta transformar el elemento a lo siguiente:

```
{
  "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
```

```
}
```

Aquí observamos que el campo entrante para “interests” (intereses) tiene un valor duplicado existente en el servidor y dos nuevos valores. En este caso, dado que el tipo de datos subyacente es un conjunto, Automerge (Combinar automáticamente) combinará los valores existentes en el servidor con los de la solicitud entrante y eliminará los duplicados. Del mismo modo, hay un conflicto en el campo “points” (puntos), donde hay un valor duplicado y un nuevo valor. Sin embargo, dado que el tipo de datos subyacente aquí es una lista, Automerge (Combinar automáticamente) simplemente agregará todos los valores de la solicitud entrante al final de los valores que ya existen en el servidor. La imagen combinada resultante que se almacenaría en el servidor tendría el siguiente aspecto:

```
{
  "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
}
```

Ahora, supongamos que el elemento almacenado en el servidor aparece de la siguiente manera en la versión 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
}
```

Sin embargo, una solicitud entrante intenta actualizar el elemento con la siguiente imagen, una vez más con una discrepancia de versión:

```
{
  "id" : 1,
  "name" : "Nadia",
  "stats": {
      "ppg": "25.7",
      "rpg": "6.9"
  }
  "_version" : 3
}
```

En este escenario, observamos que faltan los campos que ya existen en el servidor (interests, points, jersey). Además, se está editando el valor de “ppg” dentro del mapa “stats”, se añade un nuevo valor “rpg” y se omite “apg”. Automerge (Combinar automáticamente) conserva los campos que se han omitido (nota: si los campos están destinados a ser eliminados, entonces la solicitud se debe volver a intentar con la versión coincidente), por lo que no se perderán. También aplicará las mismas reglas a los campos dentro de los mapas y, por lo tanto, el cambio a “ppg” se rechazará, mientras que se conservará “apg” y se añadirá “rpg”, que es un nuevo campo. Ahora, el elemento resultante almacenado en el servidor tendrá este aspecto:

```
{
  "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

Opciones de resolución de conflictos:

- `RESOLVE`: sustituye el elemento existente por el nuevo elemento suministrado en la carga de respuesta. Solo puede volver a intentar la misma operación con un único elemento a la vez. Actualmente, es compatible con `PutItem` y `UpdateItem` de DynamoDB.

- `REJECT`: rechaza la mutación y devuelve un error con el elemento existente en la respuesta de GraphQL. Actualmente es compatible con `PutItem`, `UpdateItem` y `DeleteItem` de DynamoDB.
- `REMOVE`: elimina el elemento existente. Actualmente es compatible con `DeleteItem` de DynamoDB.

La solicitud de invocación Lambda

El solucionador de DynamoDB de AWS AppSync invoca la función de Lambda especificada en `LambdaConflictHandlerArn`. Se utiliza el mismo `service-role-arn` configurado en el origen de datos. La carga de la invocación tiene la siguiente estructura:

```
{
    "newItem": { ... },
    "existingItem": {... },
    "arguments": { ... },
    "resolver": { ... },
    "identity": { ... }
}
```

Los campos se definen de la siguiente manera:

`newItem`

El elemento de vista previa, si la mutación se ha realizado correctamente.

`existingItem`

El elemento residía actualmente en la tabla de DynamoDB.

`arguments`

Los argumentos de la mutación de GraphQL.

`resolver`

Información sobre el solucionador de AWS AppSync.

`identity`

Información sobre el intermediario. Este campo se establece en null si el acceso es con clave API.

Ejemplo de carga:

```
{
    "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"
    }
}
```

La respuesta de invocación Lambda

Para la resolución de conflictos de `PutItem` y `UpdateItem`

`RESOLVE` para la mutación. La respuesta debe tener el siguiente formato.

```
{
    "action": "RESOLVE",
    "item": { ... }
```

```
}
```

El campo `item` representa un objeto que se utilizará para sustituir el elemento existente en el origen de datos subyacente. La clave principal y los metadatos de sincronización se pasarán por alto si se han incluido en `item`.

`REJECT` para la mutación. La respuesta debe tener el siguiente formato.

```
{
    "action": "REJECT"
}
```

Para resolver los conflictos de `DeleteItem`

Se ejecuta `REMOVE` para eliminar el elemento. La respuesta debe tener el siguiente formato.

```
{
    "action": "REMOVE"
}
```

`REJECT` para la mutación. La respuesta debe tener el siguiente formato.

```
{
    "action": "REJECT"
}
```

El ejemplo de función de Lambda siguiente comprueba quién realiza la llamada y el nombre del solucionador. Si la realiza `jeffTheAdmin`, se ejecuta `REMOVE` para eliminar el objeto para el solucionador DeletePost o `RESOLVE` para resolver el conflicto con el nuevo elemento para los solucionadores Update/Put. Si no, la mutación es `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;
}
```

Errores

`ConflictUnhandled`

La detección de conflictos encuentra una disconformidad de versión y el controlador de conflictos rechaza la mutación.

Ejemplo: Resolución de conflictos con un controlador de conflictos Optimistic Concurrency (Simultaneidad optimista). O bien, se devuelve el controlador de conflictos de Lambda con `REJECT`.

`ConflictError`

Se produce un error interno al intentar resolver un conflicto.

Ejemplo: El controlador de conflictos de Lambda devolvió una respuesta mal formada. O bien, no se puede invocar el controlador de conflictos de Lambda porque el recurso suministrado `LambdaConflictHandlerArn` no se encuentra.

`MaxConflicts`

Se alcanzó el número máximo de reintentos para la resolución de conflictos.

Ejemplo: Demasiadas solicitudes simultáneas en el mismo objeto. Antes de resolver el conflicto, otro cliente actualiza el objeto a una nueva versión.

`BadRequest`

El cliente intenta actualizar los campos de metadatos (`_version`, `_ttl`, `_lastChangedAt`, `_deleted`).

Ejemplo: El cliente intenta actualizar el campo _version de un objeto con una mutación de actualización.

`DeltaSyncWriteError`

Error al escribir el registro de Delta Sync.

Ejemplo: La mutación se ha realizado correctamente, pero se ha producido un error interno al intentar escribir en la tabla de Delta Sync.

`InternalFailure`

Se ha producido un error interno.

Registros de CloudWatch

Si una API de AWS AppSync ha habilitado Registros de CloudWatch con la configuración de registro establecida en `enabled` para los registros del nivel de campo y el nivel de registro para los registros de nivel de campo establecido en `ALL`, AWS AppSync emitirá información de detección y resolución de conflictos al grupo de registros. Para obtener información sobre el formato de los mensajes de registro, consulte la documentación de detección de conflictos y registro de sincronización.

Operaciones de sincronización

Los orígenes de datos con control de versiones admiten operaciones de `Sync` que permiten recuperar todos los resultados de una tabla de DynamoDB y, a continuación, recibir tan solo los datos modificados desde la última consulta (las actualizaciones delta). Cuando AWS AppSync recibe una solicitud para una operación `Sync`, utiliza los campos especificados en la solicitud para determinar si se debe tener acceso a la tabla Base o a la tabla Delta.

- Si no se especifica el campo `lastSync`, se realiza una operación `Scan` en la tabla Base.
- Si se especifica el campo `lastSync`, pero el valor es anterior a `current moment - DeltaSyncTTL`, se realiza una operación `Scan` en la tabla Base.
- Si se especifica el campo `lastSync` y el valor corresponde al momento `current moment - DeltaSyncTTL` o a un momento posterior, se realiza una operación `Query` en la tabla Delta.

AWS AppSync devuelve el campo `startedAt` a la plantilla de asignación de respuesta para todas las operaciones `Sync`. El campo `startedAt` es el momento, en milisegundos transcurridos desde la fecha de inicio, en que se inició la operación `Sync` que puede almacenar localmente y usar en otra solicitud. Si se incluyó un token de paginación en la solicitud, este valor será el mismo que el devuelto por la solicitud para la primera página de resultados.

Para obtener información sobre el formato de las plantillas de mapeo de `Sync`, consulte la referencia de la plantilla de mapeo.

Supervisión y registro

Para monitorizar tu API de AWS AppSync GraphQL y ayudarte a depurar los problemas relacionados con las solicitudes, puedes activar el registro en Amazon Logs. CloudWatch

Ajustes y configuración

Para activar el registro automático en una API de GraphQL, usa la AWS AppSync consola.

1. Inicia sesión en la AppSyncconsola AWS Management Console y ábrela.
2. En la página API, elija el nombre de una API de GraphQL.
3. En el panel de navegación de la página de inicio de la API de, elija Ajustes.
4. En Logging (Registro), haga lo siguiente:

    a. Active Habilitar Logs.

    b. Para obtener un registro detallado a nivel de solicitud, seleccione la casilla de verificación en Incluir el contenido excesivamente detallado (opcional).

    c. En Nivel de registro del solucionador de campo, elija el nivel de registro de nivel de campo que prefiera (Ninguno, Error o Todo) (opcional).

    d. En Crear o usar un rol existente, selecciona Nuevo rol para crear uno nuevo AWS Identity and Access Management (IAM) que permita AWS AppSync escribir CloudWatch registros.

O bien, elija Rol existente para seleccionar el Nombre de recurso de Amazon (ARN) de un rol de IAM existente en su cuenta de AWS .

5. Seleccione Guardar.

Configuración manual del rol de IAM

Si decide utilizar una función de IAM existente, la función debe conceder AWS AppSync los permisos necesarios para escribir registros en ella. CloudWatch Para configurarlo manualmente, debe proporcionar un ARN de rol de servicio para que AWS AppSync pueda asumir el rol al escribir los registros.

En la consola de IAM, cree una nueva política con el nombre `AWSAppSyncPushToCloudWatchLogsPolicy` que tenga la siguiente definición:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

A continuación, cree una nueva función con ese nombre AWSAppSyncPushToCloudWatchLogsRoley adjunte la política recién creada a la función. Edite la relación de confianza para este rol de la siguiente manera:

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

```
        },
        "Action": "sts:AssumeRole"
        }
    ]
}
```

Copia el ARN del rol y utilízalo al configurar el registro para una API de GraphQL AWS AppSync .

CloudWatch métricas

Puedes usar CloudWatch las métricas para monitorear y enviar alertas sobre eventos específicos que pueden resultar en códigos de estado HTTP o en la latencia. Se emiten las siguientes métricas:

Lista de métricas

`4XXError`

Errores derivados de solicitudes no válidas a causa de una configuración de cliente incorrecta. Normalmente, estos errores suelen ser ajenos al procesamiento de GraphQL. Por ejemplo, estos errores pueden producirse cuando la solicitud incluye una carga JSON incorrecta o una consulta incorrecta, cuando el servicio está limitado o si la configuración de la autorización es errónea.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de estos errores.

`5XXError`

Son los errores encontrados durante la ejecución de una consulta GraphQL. Por ejemplo, esto puede ocurrir al invocar una consulta para un esquema vacío o incorrecto. También puede ocurrir cuando el ID o la AWS región del grupo de usuarios de Amazon Cognito no son válidos. Como alternativa, esto también podría ocurrir si AWS AppSync se produce un problema durante el procesamiento de una solicitud.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de estos errores.

`Latency`

El tiempo que transcurre entre el momento en que se AWS AppSync recibe una solicitud de un cliente y el momento en que devuelve una respuesta al cliente. Esto no incluye la latencia de red que se encuentra una respuesta para llegar a los dispositivos finales.

Unidad: milisegundos. Utilice la estadística Average para evaluar las latencias esperadas.

`Requests`

El número de solicitudes (consultas + mutaciones) que han procesado todas las API de su cuenta, por Región.

Unidad: recuento. El número total de solicitudes procesadas en una Región determinada.

`TokensConsumed`

Los tókenes se asignan en `Requests` función de la cantidad de recursos (tiempo de procesamiento y memoria utilizada) que consuma una `Request`. Por lo general, cada `Request` consume un token. Sin embargo, si una `Request` consume una mayor cantidad de recursos, se le asignan tókenes adicionales en función de las necesidades.

Unidad: recuento. El número total de tókenes asignados a solicitudes procesadas en una Región determinada.

`NetworkBandwidthOutAllowanceExceeded`

> **Note**
>
> En la AWS AppSync consola, en la página de configuración de la memoria caché, la opción Cache Health Metrics le permite habilitar esta métrica de estado relacionada con la memoria caché.

Los paquetes de red se descartaron porque el rendimiento superó el límite de ancho de banda agregado. Esto resulta útil para diagnosticar los cuellos de botella en una configuración de caché. Los datos de una API concreta se registran especificando los datos en la `API_Id` métrica. `appsyncCacheNetworkBandwidthOutAllowanceExceeded`

Unidad: recuento. El número de paquetes descartados tras superar el límite de ancho de banda de una API especificado por el ID.

`EngineCPUUtilization`

> **Note**
>
> En la AWS AppSync consola, en la página de configuración de la memoria caché, la opción Cache Health Metrics le permite habilitar esta métrica de estado relacionada con la memoria caché.

El uso de la CPU (porcentaje) asignado al proceso de Redis. Esto resulta útil para diagnosticar los cuellos de botella en una configuración de caché. Los datos de una API concreta se registran especificando los datos en la `API_Id` métrica. `appsyncCacheEngineCPUUtilization`

Unidad: porcentaje. El porcentaje de CPU que utiliza actualmente el proceso de Redis para una API especificada por el ID.

Suscripciones en tiempo real

Todas las métricas se emiten en una dimensión: GraphQLAPIId. Esto significa que todas las métricas están vinculadas a los ID de la API de GraphQL. Las siguientes métricas están relacionadas con las suscripciones de GraphQL en lugar de las puras: WebSockets

Lista de métricas

`ConnectRequests`

El número de solicitudes de WebSocket conexión realizadas a AWS AppSync, incluidos los intentos correctos y fallidos.

Unidad: recuento. Utilice la estadística Sum para obtener el número total de solicitudes de conexión.

`ConnectSuccess`

El número de WebSocket conexiones correctas a AWS AppSync. Se pueden tener conexiones sin suscripciones.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de conexiones correctas.

`ConnectClientError`

El número de WebSocket conexiones que se rechazaron AWS AppSync por errores del lado del cliente. Esto puede implicar que el servicio está limitado o que la configuración de autorización es incorrecta.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de errores de conexión del lado del cliente.

`ConnectServerError`

El número de errores que se originaron AWS AppSync al procesar las conexiones. Esto suele ocurrir cuando se produce un problema inesperado del lado del servidor.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de errores de conexión del lado del servidor.

`DisconnectSuccess`

El número de WebSocket desconexiones realizadas correctamente desde AWS AppSync.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de desconexiones correctas.

`DisconnectClientError`

El número de errores del cliente que se originaron AWS AppSync al desconectar las conexiones WebSocket.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de errores de desconexión.

`DisconnectServerError`

El número de errores del servidor que se originaron AWS AppSync al desconectar las conexiones WebSocket.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de errores de desconexión.

`SubscribeSuccess`

El número de suscripciones que se registraron correctamente AWS AppSync . WebSocket Se pueden tener conexiones sin suscripciones, pero no es posible tener suscripciones sin conexiones.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de suscripciones correctas.

`SubscribeClientError`

El número de suscripciones que se rechazaron AWS AppSync por errores del lado del cliente. Esto puede ocurrir cuando una carga JSON es incorrecta, cuando el servicio está limitado o si la configuración de autorización es errónea.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de errores de suscripción del lado del cliente.

`SubscribeServerError`

El número de errores que se originaron AWS AppSync al procesar las suscripciones. Esto suele ocurrir cuando se produce un problema inesperado del lado del servidor.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de errores de suscripción del lado del servidor.

`UnsubscribeSuccess`

El número de solicitudes de cancelación de suscripción que se procesaron correctamente.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de solicitudes de cancelación de suscripción correctas.

`UnsubscribeClientError`

El número de solicitudes de cancelación de suscripción que fueron rechazadas AWS AppSync por errores del cliente.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de errores de solicitud de cancelación de suscripción del lado del cliente.

`UnsubscribeServerError`

El número de errores que se originaron AWS AppSync al procesar las solicitudes de cancelación de suscripción. Esto suele ocurrir cuando se produce un problema inesperado del lado del servidor.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de errores de solicitud de cancelación de suscripción del lado del servidor.

`PublishDataMessageSuccess`

Número de mensajes de eventos de suscripción que se publicaron correctamente.

Unidad: recuento. Use la estadística Sum para obtener el total de mensajes de eventos de suscripción que se publicaron correctamente.

`PublishDataMessageClientError`

Número de mensajes de eventos de suscripción que no se pudieron publicar debido a errores del lado del cliente.

`Unit`: recuento. Use la estadística Sum para obtener el número total de apariciones de errores de publicación de eventos de suscripción del lado del cliente.

`PublishDataMessageServerError`

El número de errores que se originaron AWS AppSync al publicar los mensajes de eventos de suscripción. Esto suele ocurrir cuando se produce un problema inesperado del lado del servidor.

Unidad: recuento. Use la estadística Sum para obtener el número total de apariciones de errores de publicación de eventos de suscripción del lado del servidor.

`PublishDataMessageSize`

Tamaño de los mensajes de eventos de suscripción publicados.

Unidad: bytes.

`ActiveConnections`

El número de WebSocket conexiones simultáneas de los clientes AWS AppSync en 1 minuto.

Unidad: recuento. Use la estadística Sum para obtener el total de conexiones abiertas.

`ActiveSubscriptions`

Número de suscripciones simultáneas de clientes en 1 minuto.

Unidad: recuento. Use la estadística Sum para obtener el total de suscripciones activas.

`ConnectionDuration`

Cantidad de tiempo que la conexión permanece abierta.

Unidad: milisegundos. Use la estadística Average para evaluar la duración de la conexión.

`OutboundMessages`

El número de mensajes medidos publicados correctamente. Un mensaje medido equivale a 5 KB de datos entregados.

Unidad: recuento. Use la estadística Sum para obtener el número total de mensajes medidos que se han publicado correctamente.

`InboundMessageSuccess`

El número de mensajes entrantes que se han procesado correctamente. Cada tipo de suscripción invocado por una mutación genera un mensaje entrante.

Unidad: recuento. Use la estadística Sum para obtener el número total de mensajes entrantes que se han procesado correctamente.

`InboundMessageError`

El número de mensajes entrantes que no se procesaron debido a solicitudes de API no válidas, por ejemplo, por superar el límite de tamaño de carga útil de la suscripción de 240 KB.

Unidad: recuento. Use la estadística Sum para obtener el número total de mensajes entrantes con errores de procesamiento relacionados con la API.

`InboundMessageFailure`

El número de mensajes entrantes que no se procesaron debido a errores de. AWS AppSync

Unidad: recuento. Utilice la estadística Suma para obtener el número total de mensajes entrantes con errores de procesamiento AWS AppSync relacionados.

`InboundMessageDelayed`

El número de mensajes entrantes retrasados. Los mensajes entrantes se pueden retrasar si se supera la cuota de velocidad de mensajes entrantes o salientes.

Unidad: recuento. Utilice la estadística Suma para obtener el número total de mensajes entrantes que se retrasaron.

`InboundMessageDropped`

El número de mensajes entrantes descartados. Los mensajes entrantes se pueden eliminar cuando se supera la cuota de velocidad de mensajes entrantes o salientes.

Unidad: recuento. Utilice la estadística Suma para obtener el número total de mensajes entrantes que se descartaron.

`InvalidationSuccess`

El número de suscripciones invalidadas (canceladas) correctamente por una mutación con `$extensions.invalidateSubscriptions()`.

Unidad: recuento. Use la estadística Sum para recuperar el número total de suscripciones que se cancelaron correctamente.

`InvalidationRequestSuccess`

El número de solicitudes de invalidación que se procesaron correctamente.

Unidad: recuento. Use la estadística Sum para obtener el número total de solicitudes de invalidación que se han procesado correctamente.

`InvalidationRequestError`

El número de solicitudes de invalidación que no se procesaron debido a solicitudes de API no válidas.

Unidad: recuento. Use la estadística Sum para obtener el número total de solicitudes de invalidación con errores de procesamiento relacionados con la API.

`InvalidationRequestFailure`

El número de solicitudes de invalidación que no se procesaron debido a errores de. AWS AppSync

Unidad: recuento. Utilice la estadística Suma para obtener el número total de solicitudes de invalidación con errores de procesamiento AWS AppSync relacionados.

`InvalidationRequestDropped`

El número de solicitudes de invalidación descartadas cuando se ha superado la cuota de solicitudes de invalidación.

Unidad: recuento. Utilice la estadística Sum para obtener el número total de solicitudes de invalidación descartadas.

Comparación de los mensajes entrantes y salientes

Cuando se ejecuta una mutación, se invocan los campos de suscripción con la directiva @aws_subscribe para esa mutación. Cada invocación de suscripción genera un mensaje entrante. Por ejemplo, si dos campos de suscripción especifican la misma mutación en @aws_subscribe, se generan dos mensajes entrantes cuando se invoca esa mutación.

Un mensaje saliente equivale a 5 KB de datos entregados a los WebSocket clientes. Por ejemplo, el envío de 15 KB de datos a 10 clientes da como resultado 30 mensajes salientes (15 kB * 10 clientes/5 kB por mensaje = 30 mensajes).

Puede solicitar aumentos de cuota para los mensajes entrantes o salientes. Para obtener más información, consulte AWS AppSync los puntos finales y las cuotas en la guía de referencia AWS general y las instrucciones para solicitar un aumento de cuota en la Guía del usuario de Service Quotas.

Métricas mejoradas

Las métricas mejoradas emiten datos detallados sobre el uso y el rendimiento de la API, como el recuento de AWS AppSync solicitudes y errores, la latencia y los aciertos o errores de caché. Todos los datos de las métricas mejoradas se envían a tu CloudWatch cuenta y puedes configurar los tipos de datos que se enviarán.

> **Note**
>
> Se aplican cargos adicionales cuando se utilizan métricas mejoradas. Para obtener más información, consulta los niveles de precios de supervisión detallados en Amazon CloudWatch Pricing.

Estas métricas se encuentran en varias páginas de configuración de la AWS AppSync consola. En la página de configuración de la API, la sección Métricas mejoradas te permite activar o desactivar los siguientes elementos:

1. Comportamiento de las métricas de los resolutores: estas opciones controlan la forma en que se recopilan las métricas adicionales de los resolutores. Puede optar por habilitar las métricas de resolución de solicitudes completas (las métricas están habilitadas para todos los resolutores en las solicitudes) o las métricas por resolución (las métricas solo están habilitadas para los resolutores cuya configuración está configurada como habilitada). Están disponibles las siguientes opciones:

Métrica	Dimensión métrica	Nombre de métrica	Unidad	Descripción
Errores de GraphQL por resolución	API_ID, Resolver	Error de GraphQL	Recuento	El número de errores de GraphQL que se produjeron por resolución.
Solicitudes por resolución	API_ID, Resolver	Solicitud	Recuento	El número de invocaciones que se produjeron

				durante una solicitud. Esto se registra por resolución.
Latencia por resolución	API_ID, Resolver	Latencia	Milisegundo	El tiempo necesario para completar la invocación de un resolutor. La latencia se mide en milisegundos y se registra por resolución.
Caché las visitas por resolución	API_ID, Resolver	CacheHit	Recuento	El número de visitas a la caché durante una solicitud. Solo se emitirá si se utiliza una caché. Las visitas a la memoria caché se registran por resolución.

Fallos de caché por resolución	API_ID, Resolver	CacheMiss	Recuento	El número de cachés que faltan durante una solicitud . Esto solo se emitirá si se utiliza un caché. Los errores de caché se registran por resolución.

2. Comportamiento de las métricas de las fuentes de datos: estas opciones controlan la forma en que se recopilan las métricas adicionales de las fuentes de datos. Puede optar por habilitar las métricas de fuente de datos de solicitud completa (las métricas están habilitadas para todas las fuentes de datos en las solicitudes) o las métricas por fuente de datos (las métricas solo están habilitadas para las fuentes de datos en las que la configuración está configurada como habilitada). Están disponibles las siguientes opciones:

Métrica	Dimensión métrica	Nombre de métrica	Unidad	Descripción
Solicitudes por fuente de datos	API_ID, fuente de datos	Solicitud	Recuento	El número de invocacio nes que se produjeron durante una solicitud. Las solicitudes se registran por fuente de datos. Si las solicitudes completas están habilitadas, cada fuente de datos tendrá su

				propia entrada. CloudWatch
Latencia por fuente de datos	API_ID, fuente de datos	Latencia	Milisegundo	El tiempo necesario para completar la invocación de una fuente de datos. La latencia se registra por fuente de datos.
Errores por fuente de datos	API_ID, fuente de datos	Error de GraphQL	Recuento	El número de errores que se produjero n durante la invocación de una fuente de datos.

3. Métricas de operación: habilita métricas de nivel de operación de GraphQL.

Métrica	Dimensión métrica	Nombre de métrica	Unidad	Descripción
Solicitudes por operación	API_ID, operación	Solicitud	Recuento	El número de veces que se llamó a una operación de GraphQL específica.
Errores de GraphQL por operación	API_ID, operación	Error de GraphQL	Recuento	El número de errores de GraphQL que se produjero n durante una

operación de GraphQL específica.

CloudWatch registros

Puede configurar dos tipos de registros en cada API de GraphQL nueva o existente, tanto en el nivel del campo como de la solicitud.

Registros en el nivel de la solicitud

Cuando se configura el registro en el nivel de solicitud (Incluir el contenido excesivamente detallado), se registra la información siguiente:

- La cantidad de tókenes utilizados
- Los encabezados HTTP de la solicitud y la respuesta
- La consulta de GraphQL que se ejecuta en la solicitud
- El resumen general de la operación
- Las suscripciones a GraphQL nuevas y existentes que se están registrando

Registros en el nivel del campo

Cuando se configura el registro a nivel de campo, se registra la información siguiente:

- Mapeo de solicitudes generado con origen y argumentos para cada campo
- Mapeo de respuesta transformado para cada campo, que incluye los datos obtenidos de la resolución de dicho campo
- Información de seguimiento de cada campo

Si activas el registro, AWS AppSync administra los CloudWatch registros. El proceso incluye la creación de grupos y flujos de registros, y la notificación a los flujos de registro con dichos registros.

Cuando activas el registro en una API de GraphQL y realizas solicitudes, AWS AppSync crea un grupo de registros y registra flujos en el grupo de registros. Al grupo de registro se le asigna un nombre con el formato `/aws/appsync/apis/{graphql_api_id}`. Dentro de cada grupo, los

registros se dividen en flujos de registros. Estos se ordenan en función de la Last Event Time (Hora del último evento) según los datos registrados.

Cada evento de registro se etiqueta con el x-amzn- RequestId de esa solicitud. Esto le ayuda a filtrar los eventos de registro CloudWatch para obtener toda la información registrada sobre esa solicitud. Puedes obtenerlos RequestId de los encabezados de respuesta de cada solicitud de AWS AppSync GraphQL.

El registro en el nivel del campo se configura con los siguientes niveles de registro:

- Ninguno: no se capturan los registros en el nivel de campo.
- Error: se registra la siguiente información solo en los campos que tienen errores:
    - La sección de error en la respuesta del servidor
    - Los errores en el nivel del campo
    - Las funciones de solicitud/respuesta generadas resueltas para los campos de error
- Todo: se registra la siguiente información de todos los campos de la consulta:
    - Información de seguimiento en el nivel del campo
    - Las funciones de solicitud/respuesta generadas resueltas para cada campo

Ventajas de la supervisión

Puede utilizar el registro y las métricas para identificar, solucionar problemas y optimizar sus consultas de GraphQL. Por ejemplo, le pueden ayudar a depurar problemas de latencia mediante la información de seguimiento registrada para cada campo en la consulta. Para ver una demostración, suponga que utiliza uno o varios solucionadores anidados en una consulta de GraphQL. Un ejemplo de operación de campo en CloudWatch Logs podría tener un aspecto similar al siguiente:

```
{
    "path": [
        "singlePost",
        "authors",
        0,
        "name"
    ],
    "parentType": "Post",
    "returnType": "String!",
    "fieldName": "name",
    "startOffset": 416563350,
```

```
    "duration": 11247
}
```

Esto podría equivaler a un esquema de GraphQL, similar al siguiente:

```
type Post {
  id: ID!
  name: String!
  authors: [Author]
}

type Author {
  id: ID!
  name: String!
}

type Query {
  singlePost(id:ID!): Post
}
```

En los resultados del registro anterior, path muestra un solo elemento en los datos devueltos al ejecutar una consulta denominada `singlePost()`. En este ejemplo, representa el campo name en el primer índice (0). El valor de startOffset proporciona una demora desde el inicio de la operación de consulta de GraphQL. El valor de duration es el tiempo total necesario para resolver el campo. Estos valores pueden ser útiles para solucionar problemas relacionados con el motivo por el que un determinado origen de datos puede ejecutarse más lentamente de lo esperado, o si un campo específico está ralentizando toda la consulta. Por ejemplo, puede elegir aumentar el rendimiento aprovisionado de una tabla de Amazon DynamoDB o quitar un campo específico de una consulta que provoca que la ejecución vaya más lenta en general.

A partir del 8 de mayo de 2019, AWS AppSync genera eventos de registro en formato JSON completamente estructurado. Esto puede ayudarte a utilizar los servicios de análisis de CloudWatch registros, como Logs Insights y Amazon OpenSearch Service, para comprender el rendimiento de tus solicitudes de GraphQL y las características de uso de los campos de tu esquema. Por ejemplo, podrá identificar fácilmente solucionadores con latencias elevadas que podrían ser la causa de un problema de rendimiento. También podrá identificar los campos utilizados con mayor y menor frecuencia en su esquema y evaluar el impacto de dejar de usar campos de GraphQL.

Detección de conflictos y registro de sincronización

Si una AWS AppSync API tiene el registro de CloudWatch registros configurado con el nivel de registro de Field Resolver establecido en Todos, entonces AWS AppSync emite información de detección y resolución de conflictos al grupo de registros. Esto proporciona información detallada sobre cómo respondió la AWS AppSync API a un conflicto. Para ayudarle a interpretar la respuesta, en los registros se proporciona la información siguiente:

Lista de métricas

`conflictType`

Se detalla si el conflicto se ha producido debido a una discrepancia de versión o a la condición proporcionada por el cliente.

`conflictHandlerConfigured`

Se declara el controlador de conflictos configurado en el solucionador en el momento de la solicitud.

`message`

Se proporciona información sobre cómo se detectó y resolvió el conflicto.

`syncAttempt`

Número de intentos que el servidor intentó para sincronizar los datos antes de rechazar finalmente la solicitud.

`data`

Si el controlador de conflictos configurado era `Automerge`, este campo se rellenará para mostrar la decisión tomada por `Automerge` para cada campo. Las acciones proporcionadas pueden ser:

- RECHAZADO: cuando `Automerge` rechaza el valor del campo entrante a favor del valor en el servidor.
- AGREGADO: cuando `Automerge` agrega en el campo entrante debido a que no hay un valor preexistente en el servidor.
- ANEXADO: cuando `Automerge` agrega los valores entrantes a los valores de la lista que existe en el servidor.
- FUSIONADO: cuando `Automerge` combina los valores entrantes con los valores del conjunto que existe en el servidor.

Uso del recuento de tókenes para optimizar sus solicitudes

A las solicitudes que utilizan 1500 KB/s de memoria y tiempo de vCPU o menos se les asigna un token. Las solicitudes con un consumo de recursos superior a 1500 KB/s se les asignan más tókenes. Por ejemplo, si una solicitud consume 3.350 KB por segundo, AWS AppSync asigna tres fichas (redondeadas al siguiente valor entero) a la solicitud. De forma predeterminada, AWS AppSync asigna un máximo de 5000 o 10 000 tokens de solicitud por segundo a las API de tu cuenta, en función de la región en la AWS que se despliegue. Si cada una de tus API usa una media de dos tokens por segundo, tendrás un límite de 2500 o 5000 solicitudes por segundo, respectivamente. Si necesita más tókenes por segundo de lo que se le han asignado, puede enviar una solicitud para aumentar la cuota predeterminada de tókenes de solicitud. Para obtener más información, consulte AWS AppSync puntos finales y cuotas en la Referencia general de AWS guía y Solicitud de un aumento de cuota en la Guía del usuario de Service Quotas.

Un número elevado de tókenes por solicitud podría indicar que existe la posibilidad de optimizar tus solicitudes y mejorar el rendimiento de su API. Entre los factores que pueden aumentar el número de tókenes por solicitud se incluyen los siguientes:

- El tamaño y la complejidad de su esquema de GraphQL
- La complejidad de las plantillas de mapeo de solicitudes y respuestas
- El número de invocaciones del solucionador por solicitud
- La cantidad de datos devueltos por los solucionadores
- La latencia de los orígenes de datos posteriores
- Diseños de esquemas y consultas que requieren llamadas sucesivas al origen de datos (a diferencia de las llamadas en paralelo o por lotes)
- Configuración de registro, especialmente el contenido a nivel de campo y de registro excesivamente detallado.

> **Note**
>
> Además de las AWS AppSync métricas y los registros, los clientes pueden acceder a la cantidad de fichas consumidas en una solicitud a través del encabezado `x-amzn-appsync-TokensConsumed` de respuesta.

Referencia de tipos de registros

RequestSummary

- requestId: identificador único de la solicitud.
- graphQLAPIId: ID de la API de GraphQL que realiza la solicitud.
- statusCode: respuesta del código de estado HTTP.
- latencia: nd-to-end latencia E de la solicitud, en nanosegundos, como número entero.

```
{
     "logType": "RequestSummary",
     "requestId": "dbe87af3-c114-4b32-ae79-8af11f3f96f1",
     "graphQLAPIId": "pmo28inf75eepg63qxq4ekoeg4",
     "statusCode": 200,
     "latency": 242000000
}
```

ExecutionSummary

- requestId: identificador único de la solicitud.
- graphQLAPIId: ID de la API de GraphQL que realiza la solicitud.
- startTime: la marca temporal de inicio del procesamiento de GraphQL para la solicitud, en formato RFC 3339.
- endTime: la marca temporal de finalización del procesamiento de GraphQL para la solicitud, en formato RFC 3339.
- duration: el tiempo de ejecución de GraphQL total transcurrido en nanosegundos como número entero.
- versión: la versión de esquema de ExecutionSummary.
- parsing (análisis):
    - startOffset: la demora en el inicio del análisis, en nanosegundos, en relación con la invocación, como número entero.
    - duration (duración): el tiempo empleado en el análisis en nanosegundos como número entero.

- validation (validación):
    - startOffset: la demora en el inicio de la validación, en nanosegundos, en relación con la invocación, como número entero.
    - duration (duración): el tiempo empleado para realizar la validación en nanosegundos como número entero.

```
{
    "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"
}
```

Rastreo

- requestId: identificador único de la solicitud.
- graphQLAPIId: ID de la API de GraphQL que realiza la solicitud.
- startOffset: la demora en el inicio de la resolución de campo, en nanosegundos, en relación con la invocación, como número entero.
- duration (duración): el tiempo empleado en la resolución de campo en nanosegundos como número entero.
- fieldName: el nombre del campo que se está resolviendo.
- parentType: el tipo principal del campo que se está resolviendo.
- returnType: el tipo de retorno del campo que se está resolviendo.
- path (ruta): una lista de los segmentos de ruta, comenzando por el origen de la respuesta y finalizando con la resolución del campo.

- resolverArn: el ARN del solucionador utilizado para la resolución de campo. Podría no estar presente en los campos anidados.

```
{
    "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"
}
```

Analizar sus CloudWatch registros con Logs Insights

A continuación se muestran ejemplos de las consultas que puede ejecutar para obtener datos procesables sobre el rendimiento y el estado de las operaciones de GraphQL. Estos ejemplos están disponibles como consultas de muestra en la consola de CloudWatch Logs Insights. En la CloudWatchconsola, selecciona Logs Insights, selecciona el AWS AppSync grupo de registros para tu API de GraphQL y, a continuación, selecciona AWS AppSync consultas en Consultas de muestra.

La siguiente consulta devuelve las 10 solicitudes principales de GraphQL con que han usado el número máximo de tókenes:

```
filter @message like "Tokens Consumed"
| parse @message "* Tokens Consumed: *" as requestId, tokens
| sort tokens desc
| display requestId, tokens
| limit 10
```

La siguiente consulta devuelve los 10 solucionadores principales con latencia máxima:

```
  fields resolverArn, duration
```

```
| filter logType = "Tracing"
| limit 10
| sort duration desc
```

La siguiente consulta devuelve los solucionadores invocados con mayor frecuencia:

```
  fields ispresent(resolverArn) as isRes
| stats count() as invocationCount by resolverArn
| filter isRes and logType = "Tracing"
| limit 10
| sort invocationCount desc
```

La siguiente consulta devuelve los solucionadores con más errores en el mapeo de plantillas:

```
  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
```

La siguiente consulta devuelve las estadísticas de latencia del solucionador:

```
  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
```

La siguiente consulta devuelve las estadísticas de latencia del campo:

```
  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
```

Los resultados de las consultas de CloudWatch Logs Insights se pueden exportar a CloudWatch paneles de control.

Analice sus registros con Service OpenSearch

Puedes buscar, analizar y visualizar tus AWS AppSync registros con Amazon OpenSearch Service para identificar los cuellos de botella en el rendimiento y las causas fundamentales de los problemas operativos. Puede identificar los solucionadores con los errores y la latencia máxima. Además, puede usar los paneles de control para crear OpenSearch paneles con visualizaciones potentes. OpenSearch Dashboards es una herramienta de exploración y visualización de datos de código abierto disponible en Service. OpenSearch Con los OpenSearch paneles de control, puede supervisar de forma continua el rendimiento y el estado de sus operaciones de GraphQL. Por ejemplo, puede crear paneles que le permitan visualizar la latencia P90 de sus solicitudes de GraphQL y profundizar en las latencias P90 de cada solucionador.

Cuando utilices el OpenSearch Servicio, utiliza «cwl*» como patrón de filtro para buscar índices. OpenSearch OpenSearch El servicio indexa los registros transmitidos desde CloudWatch Logs con el prefijo «cwl-». Para diferenciar los registros de la AWS AppSync API de otros CloudWatch registros enviados al OpenSearch Servicio, te recomendamos añadir una expresión de `graphQLAPIID.keyword=`*`YourGraphQLAPIID`* filtro adicional a tu búsqueda.

Migración del formato de registro

Los eventos de registro que se AWS AppSync generen a partir del 8 de mayo de 2019 tienen el formato JSON completamente estructurado. Para analizar las solicitudes de GraphQL anteriores al 8 de mayo de 2019, puedes migrar los registros más antiguos a un JSON completamente estructurado mediante un script disponible en el GitHub ejemplo. Si necesita utilizar el formato del registro anterior al 8 de mayo de 2019, cree un tique de soporte técnico con la siguiente configuración: establezca Type (Tipo) en Account Management (Administración de cuentas) y, a continuación, establezca Category (Categoría) en General Account Question (Pregunta sobre la cuenta general).

También puedes usar filtros de métricas CloudWatch para convertir los datos de registro en CloudWatch métricas numéricas, de modo que puedas graficarlos o configurarlos como una alarma.

Rastreo con AWS X-Ray

Puede utilizar AWS X-Ray para rastrear las solicitudes a medida que se ejecutan en AWS AppSync. Puede utilizar X-Ray con AWS AppSync en todas las regiones de AWS donde X-Ray esté disponible. X-Ray le ofrece una visión general detallada de toda una solicitud de GraphQL. Esto le permite analizar latencias en sus API y sus solucionadores y orígenes de datos subyacentes. Puede utilizar

un mapa de servicio de X-Ray para ver la latencia de una solicitud, incluidos los servicios de AWS que estén integrados con X-Ray. También puede configurar reglas de muestreo para indicar a X-Ray qué solicitudes debe registrar y a qué velocidad de muestreo, de acuerdo con los criterios que especifique.

Para obtener más información sobre el muestreo en X-Ray, consulte Configuración de reglas de muestreo en la consola de AWS X-Ray.

Ajustes y configuración

Puede habilitar el rastreo de X-Ray para una API de GraphQL a través de la consola de AWS AppSync.

1. Inicie sesión en la consola de AWS AppSync.
2. Seleccione Settings (Configuración) en el panel de navegación.
3. En X-Ray, active Enable X-Ray (Habilitar X-Ray).
4. Seleccione Save. El rastreo de X-Ray ahora está habilitado para su API.

Si utiliza la AWS CLI o AWS CloudFormation, también puede habilitar el rastreo de X-Ray cuando cree una nueva API de AWS AppSync o cuando actualice una API de AWS AppSync existente estableciendo la propiedad `xrayEnabled` en `true`.

Cuando el rastreo de X-Ray está habilitado para una API de AWS AppSync, se crea automáticamente un rol vinculado al servicio de AWS Identity and Access Management en su cuenta con los permisos correspondientes. Esto permite a AWS AppSync enviar registros de rastreo a X-Ray de una manera segura.

Rastreo de su API con X-Ray

Muestreo

Al personalizar las reglas de muestreo, puede controlar la cantidad de datos que va a registrar en AWS AppSync y modificar el comportamiento de muestreo sobre la marcha sin modificar ni volver a implementar su código. Por ejemplo, esta regla realiza un muestreo de solicitudes a la API de GraphQL con el ID de API `3n572shhcpfokwhdnq1ogu59v6`.

- Nombre de la regla: `test-sample`

- Prioridad: `10`
- Tamaño del depósito: `10`
- Porcentaje fijo: `10`
- Nombre del servicio: *
- Tipo de servicio: `AWS::AppSync::GraphQLAPI`
- Método HTTP: *
- ARN del recurso: `arn:aws:appsync:us-west-2:123456789012:apis/3n572shhcpfokwhdnq1ogu59v6`
- Host: *

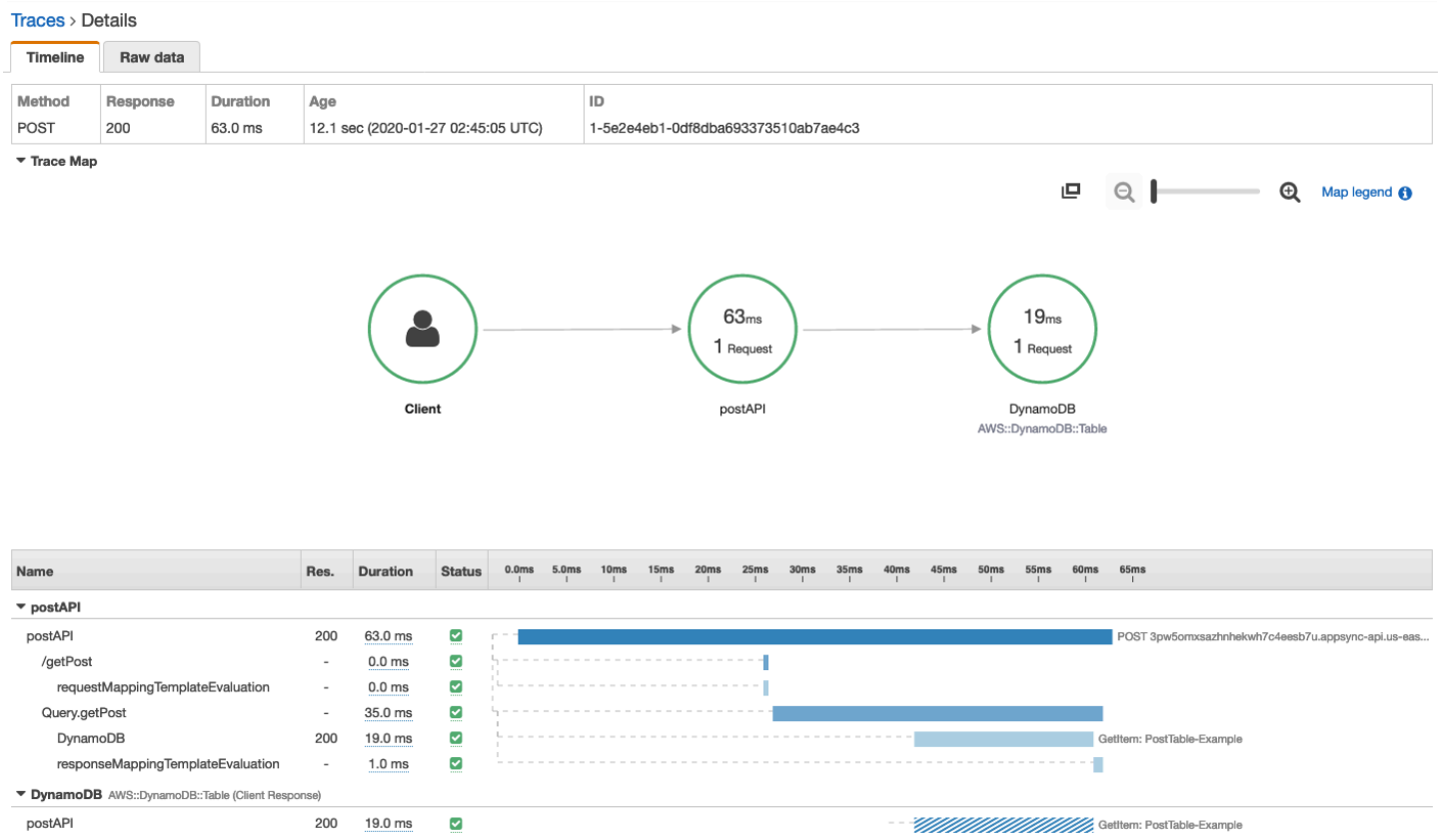
Descripción de los registros de seguimiento

Cuando habilita el rastreo de X-Ray para su API de GraphQL, puede utilizar la página de detalles de rastreo de X-Ray para obtener información detallada relativa a la latencia de las solicitudes realizadas a su API. En el ejemplo siguiente se muestra la vista de seguimiento junto con el mapa de servicio para esta solicitud específica. La solicitud se realizó a una llamada a la API `postAPI` con un tipo Post, cuyos datos están incluidos en una tabla de Amazon DynamoDB denominada `PostTable-Example`.

La siguiente imagen de seguimiento corresponde a la siguiente consulta de GraphQL:

```
query getPost {
    getPost(id: "1") {
      id
      title
    }
}
```

El solucionador de la consulta `getPost` utiliza el origen de datos de DynamoDB subyacente. La siguiente vista de rastreo muestra la llamada a DynamoDB, así como las latencias de varias partes de la ejecución de la consulta:



- En la imagen anterior, `/getPost` representa la ruta completa al elemento que se está resolviendo. En este caso, como `getPost` es un campo del tipo `Query` raíz, aparece directamente después de la raíz de la ruta.
- `requestMappingTemplateEvaluation` representa el tiempo empleado por AWS AppSync al evaluar la plantilla de mapeo de solicitudes para este elemento de la consulta.
- `Query.getPost` representa un tipo y un campo (con el formato `Type.field`). Puede contener varios subsegmentos, dependiendo de la estructura de la API y de la solicitud que se esté rastreando.
  - `DynamoDB` representa el origen de datos asociado a este solucionador. Contiene la latencia de la llamada de red a DynamoDB para resolver el campo.
  - `responseMappingTemplateEvaluation` representa el tiempo empleado por AWS AppSync al evaluar la plantilla de mapeo de respuestas para este elemento de la consulta.

Cuando consulta los registros de rastreo en X-Ray, puede obtener información contextual y de metadatos adicional sobre los subsegmentos del segmento de AWS AppSync eligiendo los subsegmentos y explorando la vista detallada.

Para determinadas consultas profundamente anidadas o complejas, tenga en cuenta que el segmento entregado a X-Ray por AWS AppSync puede ser mayor que el tamaño máximo permitido para los documentos de segmento, tal como se define en AWS X-Ray Segment Documents. X-Ray no muestra los segmentos que superen el límite.

Registro de llamadas a la API de AWS AppSync mediante AWS CloudTrail

AWS AppSync se integra con AWS CloudTrail, un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o un servicio de AWS en AWS AppSync. CloudTrail captura las llamadas a la API de AWS AppSync como eventos. Las llamadas capturadas incluyen las llamadas realizadas desde la consola de AWS AppSync y las llamadas de código a las API de AWS AppSync. La información recopilada por CloudTrail le permite determinar la solicitud que se realizó a AWS AppSync, la dirección IP del solicitante, quién hizo la solicitud, cuándo se hizo y otros detalles adicionales.

Puede crear un registro de seguimiento para habilitar la entrega continua de eventos de CloudTrail a un bucket de Amazon Simple Storage Service (Amazon S3), incluidos los eventos para AWS AppSync. Si no configura un registro de seguimiento, puede ver los eventos más recientes de la consola de CloudTrail en la consola de CloudTrail.

> **Important**
>
> No todas las acciones de GraphQL están registradas actualmente. AppSync no registra las acciones de consulta y mutación en CloudTrail.

Para obtener más información acerca de CloudTrail, consulte la AWS CloudTrailGuía del usuario de .

Información de AWS AppSync en CloudTrail

CloudTrail se habilita en su cuenta de AWS cuando la crea. En la consola de CloudTrail del Historial de eventos, puede ver, buscar y descargar los últimos eventos de la cuenta de AWS. Para obtener más información, consulte Ver eventos con el historial de eventos de CloudTrail en la Guía del usuario de AWS CloudTrail.

Para mantener un registro continuo de eventos en la cuenta de AWS, incluidos los eventos de AWS AppSync, cree un registro de seguimiento. De manera predeterminada, cuando se crea un

registro de seguimiento en la consola, el registro de seguimiento se aplica a todas las regiones de AWS. El registro de seguimiento registra los eventos de todas las regiones de la partición de AWS y envía los archivos de registro al bucket de Amazon S3 especificado. También es posible configurar otros servicios de AWS para analizar en profundidad y actuar en función de los datos de eventos recopilados en los registros de CloudTrail. Para obtener más información, consulte lo indicado en la Guía del usuario de AWS CloudTrail:

- Creación de un registro de seguimiento para su cuenta de AWS
- Integraciones de servicios de AWS con registros de CloudTrail
- Configuración de notificaciones de Amazon SNS para CloudTrail
- Recepción de archivos de registro de CloudTrail desde varias regiones
- Recepción de archivos de registro de CloudTrail desde varias cuentas

CloudTrail registra todas las operaciones de la API de AWS AppSync. Por ejemplo, las llamadas a las API `CreateGraphqlApi`, `CreateDataSource` y `ListResolvers` generan entradas en los archivos de registro de CloudTrail. Estas y otras operaciones se documentan en la referencia de la API de AWS AppSync.

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad le ayuda a determinar:

- Si la solicitud se realizó con credenciales de usuario AWS Identity and Access Management (IAM) o credenciales de usuario raíz.
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro servicio de AWS.

Para obtener más información, consulte Elemento userIdentity de CloudTrail en la Guía del usuario de AWS CloudTrail.

Descripción de las entradas de los archivos de registro de AWS AppSync

CloudTrail proporciona eventos como archivos de registro que contienen una o varias entradas de registro. Un evento representa una única solicitud de cualquier origen e incluye información sobre la operación solicitada, la fecha y la hora de la operación, los parámetros de la solicitud, etcétera. Estos

archivos de registro no rastrean el orden en la pila de las llamadas públicas a la API, por lo que estas no aparecen en ningún orden específico.

En el ejemplo que sigue se muestra una entrada de registro de CloudTrail que ilustra la operación `CreateApiKey`.


```
{
  "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"
    }
  ]
}
```


En el ejemplo que sigue se muestra una entrada de registro de CloudTrail que ilustra la operación `ListApiKeys`.

```
{
  "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"
    }
  ]
}
```


En el ejemplo que sigue se muestra una entrada de registro de CloudTrail que ilustra la operación `DeleteApiKey`.

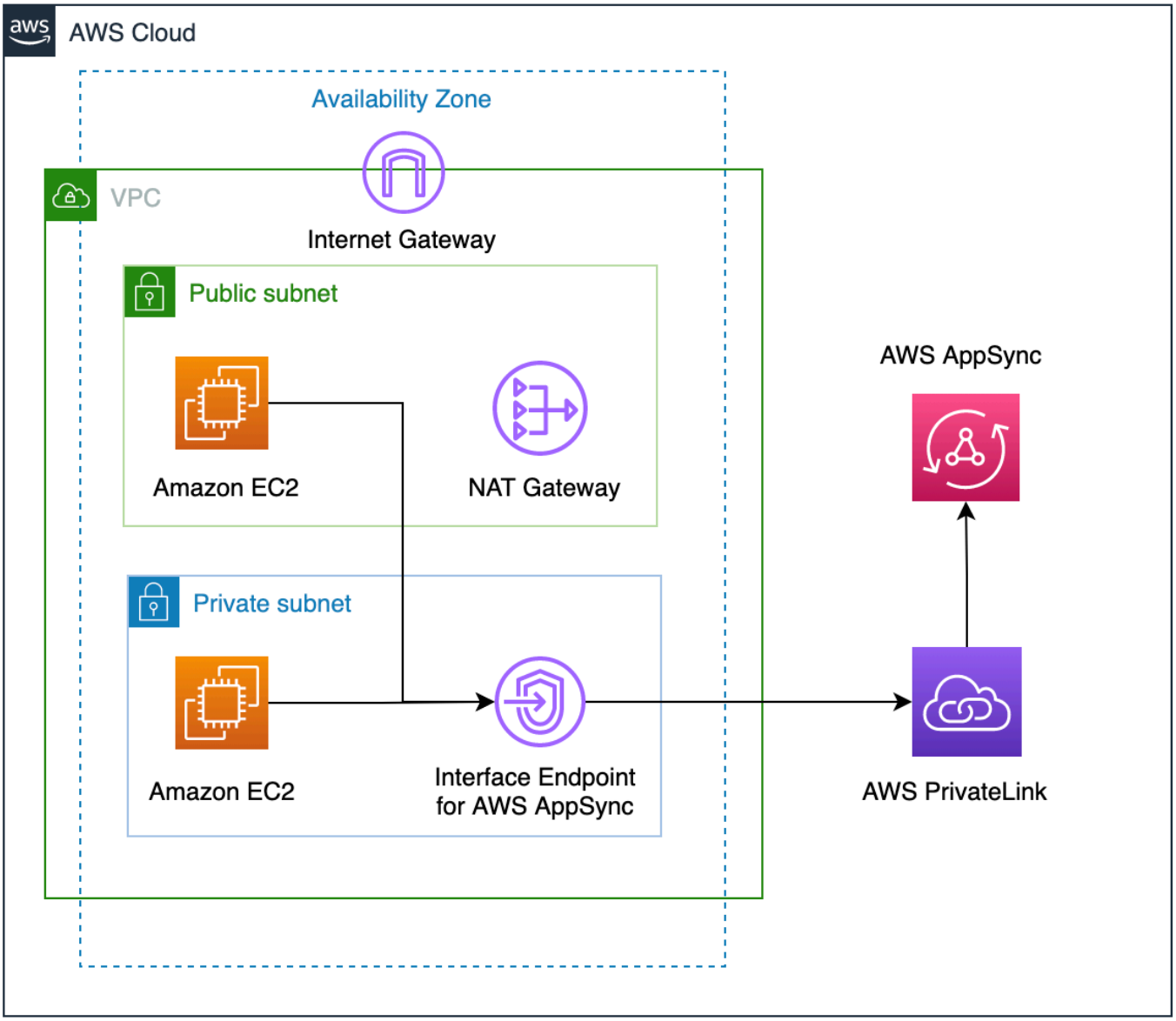
```
{
  "Records": [{
    "eventVersion": "1.05",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "A1B2C3D4E5F6G7EXAMPLE",
      "arn": "arn:aws:iam::111122223333:user/Alice",
      "accountId": "111122223333",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "userName": "Alice"
    },
    "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"
    }
  ]
}
```


Uso de API privadas de AWS AppSync

Si usa Amazon Virtual Private Cloud (Amazon VPC), puede crear API privadas de AWS AppSync, que son API a las que solo se puede acceder desde una VPC. Con una API privada, puede restringir el acceso de la API a sus aplicaciones internas y conectarse a sus puntos de conexión de GraphQL y Realtime sin exponer públicamente los datos.

Para establecer una conexión privada entre la VPC y el servicio AWS AppSync, cree un punto de conexión de VPC de interfaz. Los puntos de enlace de tipo interfaz cuentan con la tecnología de AWS PrivateLink, lo que les permite acceder de forma privada a las API de AWS AppSync sin utilizar

una gateway de Internet, un dispositivo NAT, una conexión de VPN o una conexión AWS Direct Connect. Las instancias de la VPC no necesitan direcciones IP públicas para comunicarse con las API de AWS AppSync. El tráfico entre la VPC y AWS AppSync no sale de la red de AWS.



Hay algunos factores adicionales que debe tener en cuenta antes de habilitar las características de la API privada:

- La configuración de los puntos de conexión de VPC de la interfaz para AWS AppSync con las características de DNS privadas habilitadas impedirá que los recursos de la VPC puedan invocar otras API públicas de AWS AppSync mediante la URL de la API generada de AWS AppSync. Esto se debe a que la solicitud a la API pública se enruta a través del punto de conexión de la interfaz,

lo que no está permitido en las API públicas. Para invocar las API públicas en este escenario, se recomienda configurar nombres de dominio personalizados en las API públicas, que luego los recursos de la VPC pueden usar para invocar la API pública.

- Sus API privadas de AWS AppSync solo estarán disponibles en su VPC. El editor de consultas de la consola de AWS AppSync solo podrá acceder a su API si la configuración de red de su navegador puede enrutar el tráfico a su VPC (por ejemplo, mediante una conexión a través de una VPN o a través de AWS Direct Connect).
- Con un punto de conexión de interfaz de VPC para AWS AppSync, puede acceder a cualquier API privada de la misma cuenta y Región de AWS. Para restringir aún más el acceso a las API privadas, cuenta con las opciones siguientes:
  - Garantizar que solo los administradores necesarios puedan crear interfaces de punto de conexión de VPC para AWS AppSync.
  - Usar políticas personalizadas de puntos de conexión de VPC para restringir las API que se pueden invocar desde los recursos de la VPC.
  - En el caso de los recursos de la VPC, le recomendamos que utilice la autorización de IAM para invocar las API de AWS AppSync, asegurándose de que los recursos tengan funciones específicas para las API.
- Al crear o utilizar políticas que restrinjan las entidades principales de IAM, debe establecer el `authorizationType` del método a `AWS_IAM` o `NONE`.

Creación de API privadas de AWS AppSync

Los siguientes pasos muestran cómo crear API privadas en el servicio de AWS AppSync.

> **Warning**
>
> Las características de la API privada solo se pueden habilitar durante la creación de la API. Esta configuración no se puede modificar en una API de AWS AppSync ni en una API privada de AWS AppSync una vez creadas.

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
   - En el Dashboard (Panel), elija Create API (Crear API).
2. Elija Diseñar una API desde cero y, a continuación, seleccione Siguiente.
3. En la sección API privada, selecciona Usar características de API privadas.

4. Configure el resto de las opciones, revise los datos de la API y, a continuación, seleccione Crear.

Antes de poder usar la API privada de AWS AppSync, debe configurar un punto de conexión de interfaz para AWS AppSync en su VPC. Tenga en cuenta que tanto la API privada como la VPC deben estar en la misma cuenta y Región de AWS.

Creación de un punto de conexión de interfaz para AWS AppSync

Puede crear un punto de enlace de interfaz para AWS AppSync mediante la consola de Amazon VPC o AWS Command Line Interface (AWS CLI). Para obtener más información, consulte Creación de un punto de conexión de interfaz en la Guía del usuario de Amazon VPC.

Console

1. Inicie sesión en AWS Management Console y abra la página Puntos de conexión de la consola de Amazon VPC.
2. Elija Crear punto de conexión.
   a. En el campo Categoría de servicio, asegúrese de que esté seleccionado Servicios de AWS.
   b. En la tabla Servicios, elija `com.amazonaws.{region}.appsync-api`. Compruebe que el valor de la columna Tipo es `Interface`.
   c. En el campo VPC, elija una VPC y sus subredes.
   d. Para habilitar características de DNS privado para el punto de conexión de a interfaz, seleccione la casilla de verificación Habilitar nombre de DNS.
   e. En Grupo de seguridad, elija uno o varios grupos de seguridad.
3. Elija Crear punto de conexión.

CLI

Utilice el comando `create-vpc-endpoint` y especifique el ID de la VPC, el tipo de punto de enlace de la VPC (interfaz), el nombre del servicio, las subredes que usarán el punto de enlace y los grupos de seguridad que se asociarán a las interfaces de red del punto de enlace. Por ejemplo:

```
$ 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
```

Para poder utilizar la opción de DNS privado, debe definir los valores `enableDnsHostnames` y `enableDnsSupportattributes` en su VPC. Para obtener más información, consulte Viewing and updating DNS support for your VPC (Visualización y actualización de la compatibilidad de DNS para su VPC) en la Guía del usuario de Amazon VPC. Si habilita características de DNS privado para el punto de conexión de la interfaz, puede hacer solicitudes a su punto de conexión de GraphQL y en tiempo real de la API de AWS AppSync usando sus puntos de conexión de DNS públicos predeterminados con el formato siguiente:

```
https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql
```

Para obtener más puntos de conexión de servicio, consulte Puntos de conexión y cuotas de servicio en la Referencia general de AWS.

Para obtener más información sobre las interacciones del servicio con los puntos de conexión de la interfaz, consulte Acceda a un Servicio de AWS mediante un punto de conexión de VPC de interfaz en la Guía del usuario de Amazon VPC.

Para obtener información sobre la creación y configuración de un punto de conexión mediante AWS, CloudFormation, consulte el recurso AWSAWS::EC2::VPCEndpoint en la AWSGuía del usuario de AWS CloudFormation.

Ejemplos avanzados

Si habilita características de DNS privado para el punto de conexión de la interfaz, puede hacer solicitudes a su punto de conexión de GraphQL y en tiempo real de la API de AWS AppSync usando sus puntos de conexión de DNS públicos predeterminados con el formato siguiente:

```
https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql
```

Si utiliza los nombres de host de DNS públicos de punto de conexión de VPC de interfaz, la URL base para invocar la API tendrá el formato siguiente:

```
https://{vpc_endpoint_id}-{endpoint_dns_identifier}.appsync-api.
{region}.vpce.amazonaws.com/graphql
```

También puede usar el nombre de host DNS específico de la AZ si ha implementado un punto de conexión en la AZ:

```
https://{vpc_endpoint_id}-{endpoint_dns_identifier}-{az_id}.appsync-api.
{region}.vpce.amazonaws.com/graphql.
```

Si utiliza el nombre de DNS público del punto de conexión de VPC, será necesario transferir el nombre de host del punto de conexión de la API de AWS AppSync como `Host` o como encabezado de `x-appsync-domain` a la solicitud. En estos ejemplos, se utiliza una `TodoAPI` creada en la guía Lanzar un esquema de ejemplo:

```
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"}}}'
```

En los siguientes ejemplos, utilizaremos la aplicación Todo que se genera en la guía Lanzar un esquema de ejemplo. Para probar la API Todo de ejemplo, utilizaremos el DNS privado para invocar la API. Puede usar su herramienta de línea de comandos preferida; en este ejemplo, se usa curl para enviar consultas y mutaciones y wscat para configurar las suscripciones. Para emular nuestro ejemplo, sustituya los valores entre paréntesis `{ }` en los comandos siguientes por los valores correspondientes de su cuenta de AWS.

Prueba de la operación de mutación: solicitud **`createTodo`**

```
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"}}}'
```

Operación de prueba de mutación: respuesta **`createTodo`**

```
{
    "data": {
        "createTodo": {
            "id": "<todo-id>",
            "name": "My first GraphQL task",
            "where": "Day 1",
            "when": "Friday Night",
            "description": "Learn more about GraphQL"
        }
    }
}
```

Operación de consulta de prueba: solicitud **`listTodos`**

```
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"}}}'
```

Operación de consulta de prueba: solicitud **`listTodos`**

```
{
  "data": {
    "listTodos": {
      "items": [
        {
          "description": "Learn more about GraphQL",
          "id": "<todo-id>",
          "name": "My first GraphQL task",
          "when": "Friday night",
          "where": "Day 1"
        }
      ]
    }
  }
}
```

Operación de prueba de suscripción: suscripción a una mutación **`createTodo`**

Para configurar las suscripciones de GraphQL en AWS AppSync, consulte Creación de un cliente de WebSocket en tiempo real. Desde una instancia de Amazon EC2 en una VPC, puede probar su punto de conexión de suscripción a la API privada de AWS AppSync mediante wscat. En el siguiente ejemplo, se utiliza una `API KEY` para autorización.

```
$ 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"}
```

Si lo prefiere, puede usar el nombre de dominio del punto de conexión de VPC y asegurarse de especificar el encabezado Host en el comando `wscat` para establecer el 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"}
```

Ejecute el siguiente código de mutación:

```
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"}}}'
```

Después, se activa una suscripción y el mensaje de notificación aparece como se muestra a continuación:

```
< {"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"}}}}
```

Uso de políticas de IAM para limitar la creación de API públicas

AWS AppSync es compatible con las declaraciones de `Condition` de IAM para su uso con API privadas. El campo `visibility` se puede incluir en las declaraciones de política de IAM para la operación `appsync:CreateGraphqlApi`, a fin de controlar qué usuarios y roles de IAM pueden crear API públicas y privadas. Esto permite al administrador de IAM definir una política de IAM que solo permitirá al usuario crear una API de GraphQL privada. Un usuario que intente crear una API pública recibirá un mensaje de no autorización.

Por ejemplo, un administrador de IAM podría crear la siguiente declaración de política de IAM para permitir la creación de API privadas:

```
{
    "Sid": "AllowPrivateAppSyncApis",
    "Effect": "Allow",
    "Action": "appsync:CreateGraphqlApi",
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "appsync:Visibility": "PRIVATE"
        }
    }
}
```

Un administrador de IAM también podría agregar la siguiente política de control de servicios para impedir que todos los usuarios de una organización de AWS creen API de AWS AppSync que no sean API privadas:

```
{
    "Sid": "BlockNonPrivateAppSyncApis",
    "Effect": "Deny",
    "Action": "appsync:CreateGraphqlApi",
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringNotEquals": {
            "appsync:Visibility": "PRIVATE"
        }
    }
}
```

Configurar la complejidad de ejecución, la profundidad de las consultas y la introspección de GraphQL con AWS AppSync

AWS AppSync permite activar o desactivar las características de introspección y establecer límites a la cantidad de niveles anidados y solucionadores en una sola consulta.

Uso de la característica de introspección

> **Tip**
>
> Para obtener más información sobre la introspección en GraphQL, consulte este artículo en el sitio web de la base de GraphQL.

De forma predeterminada, GraphQL le permite usar la introspección para consultar el propio esquema y descubrir sus tipos, campos, consultas, mutaciones, suscripciones, etc. Se trata de una característica importante para aprender cómo el servicio GraphQL moldea y procesa los datos. No obstante, hay algunos aspectos que se deben considerar en lo que respecta a la introspección. Es posible que tenga un caso de uso que se beneficie de la introspección desactivada, como un caso en el que los nombres de los campos sean confidenciales o estén ocultos, o que el esquema completo de la API se pretenda dejar sin documentar para los consumidores. En estos casos, la publicación

de los datos de esquema mediante introspección podría provocar la filtración intencionada de datos privados.

Para evitar que esto suceda, puede desactivar la introspección. Esto evitará que partes no autorizadas utilicen campos de introspección en su esquema. Sin embargo, es importante tener en cuenta que la introspección resulta útil para que los equipos de desarrollo aprendan cómo se procesan los datos de su servicio. Internamente, podría resultar útil mantener la introspección activada y, al mismo tiempo, desactivarla en el código de producción como una capa adicional de seguridad. Otra forma de controlar esto es añadir un método de autorización, que también proporciona AWS AppSync. Para obtener más información, consulte Autorización.

AWS AppSync permite activar o desactivar la introspección en el nivel de la API. Para activa o desactivar la introspección, haga lo siguiente:

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
2. En la página API, elija el nombre de una API de GraphQL.
3. En el panel de navegación de la página de inicio de la API de, elija Ajustes.
4. En Configuraciones de la API, elija Editar.
5. En Configuración general, haga lo siguiente:
    - Active o desactive Habilitar consultas de introspección.
6. Elija Guardar.

Cuando la introspección está activada (el comportamiento predeterminado), el uso del sistema de introspección funcionará como de costumbre. Por ejemplo, en la imagen siguiente se muestra un campo `__schema` que procesa todos los tipos disponibles en el esquema:

Al desactivar esta característica, en su lugar aparecerá un error de validación en la respuesta:

Configuración de los límites de profundidad de las consultas

Hay ocasiones en las que es posible que desee tener un control más detallado sobre el funcionamiento de la API durante una operación. Uno de estos controles consiste en añadir un límite a la cantidad de niveles anidados que puede procesar una consulta. De forma predeterminada, las consultas pueden procesar una cantidad ilimitada de niveles anidados. Limitar las consultas a una cantidad específica de niveles anidados tiene posibles implicaciones para el rendimiento y la flexibilidad del proyecto. Tomemos como ejemplo la siguiente consulta:

```
query MyQuery {
```

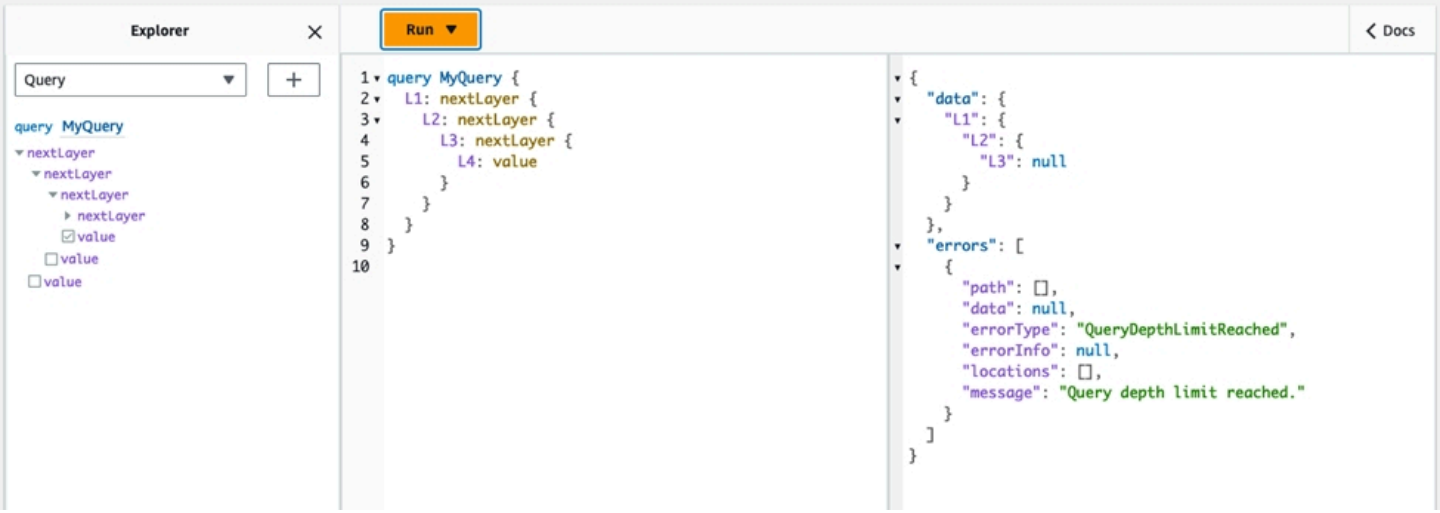
```
  L1: nextLayer {
    L2: nextLayer {
      L3: nextLayer {
        L4: value
      }
    }
  }
}
```

Es posible que su proyecto requiera limitar las consultas a L1 o L2 con algún propósito. De forma predeterminada, toda la consulta desde L1 hasta L4 se procesará sin forma de controlarla. Si establece un límite, puede impedir que las consultas accedan a datos que superen el nivel especificado.

Para añadir un límite de profundidad de las consultas, haga lo siguiente:

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
2. En la página API, elija el nombre de una API de GraphQL.
3. En el panel de navegación de la página de inicio de la API de, elija Ajustes.
4. En Configuraciones de la API, elija Editar.
5. En Profundidad de las consultas, haga lo siguiente:
   a. Active o desactive Habilitar profundidad de las consultas.
   b. En Profundidad máxima, establezca el límite de profundidad. Puede estar entre 1 y 75.
6. Elija Guardar.

Cuando se establece un límite, si se supera su límite superior se producirá un error `QueryDepthLimitReached`. Por ejemplo, en la imagen siguiente se muestra una consulta con un límite de profundidad de 2 que sobrepasa el límite de los niveles tercero (L3) y cuarto (L4):



Tenga en cuenta que los campos se pueden seguir marcando como anulables o no anulables en el esquema. Si un campo no anulable recibe un error `QueryDepthLimitReached`, ese error se transferirá al primer campo principal anulable.

Configuración de los límites de recuento de solucionadores

También puede controlar cuántos solucionadores puede procesar cada consulta. Al igual que la profundidad de la consulta, puede establecer un límite para esta cantidad. Tomemos como ejemplo la siguiente consulta que contiene tres solucionadores:

```
query MyQuery {
  resolver1: resolver
  resolver2: resolver
  resolver3: resolver
}
```

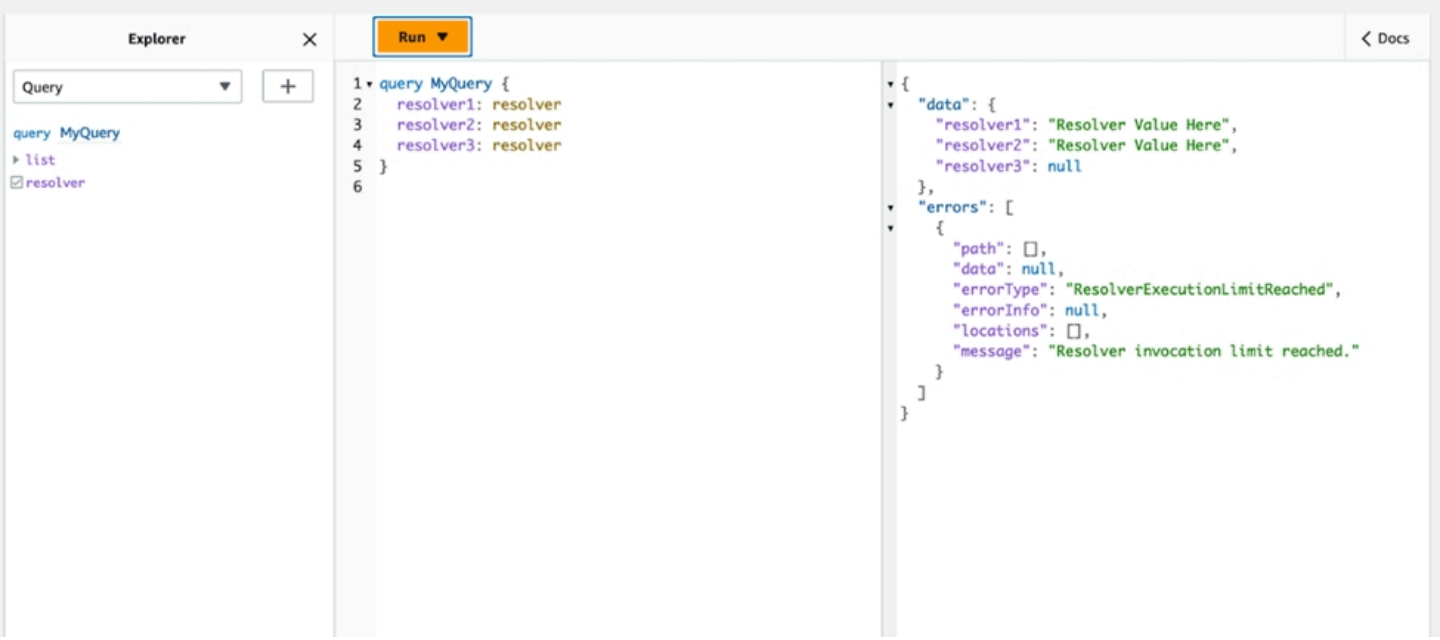
De forma predeterminada, cada consulta puede procesar hasta 10 000 solucionadores. En el ejemplo anterior, se procesarán `resolver1`, `resolver2` y `resolver3`. Sin embargo, es posible que su proyecto requiera limitar cada consulta a la gestión de uno o dos solucionadores en total. Al establecer un límite, puede indicarle a la consulta que no gestione ningún solucionador que supere un número determinado, como los primeros (`resolver1`) o los segundos (`resolver2`) solucionadores.

Para añadir un límite de recuento de solucionadores, haga lo siguiente:

1. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
2. En la página API, elija el nombre de una API de GraphQL.

3. En el panel de navegación de la página de inicio de la API de, elija Ajustes.
4. En Configuraciones de la API, elija Editar.
5. En Límite de recuento de solucionadores, haga lo siguiente:
   a. Active Habilitar recuento de solucionadores.
   b. Establezca el límite de recuento en Recuento máximo de resoluciones. Puede estar entre 1 y `10000`.
6. Elija Guardar.

Al igual que el límite de profundidad de las consultas, si se supera el límite de solucionadores configurado, la consulta finaliza con un error `ResolverExecutionLimitReached` en los solucionadores adicionales. En la imagen siguiente, una consulta con un límite de recuento de solucionadores de 2 intenta procesar tres solucionadores. Debido al límite, el tercer solucionador genera un error y no se ejecuta.



Uso de variables de entorno en AWS AppSync

Puede usar variables de entorno para ajustar el comportamiento de sus AWS AppSync resolutores y funciones sin necesidad de actualizar el código. Las variables de entorno son pares de cadenas almacenadas con la configuración de la API y que se ponen a disposición de los resolutores y las funciones para que las aprovechen durante el tiempo de ejecución. Son especialmente útiles en situaciones en las que debes hacer referencia a datos de configuración que solo están disponibles

durante la configuración inicial, pero que tus resolutores y funciones deben utilizar durante la ejecución. Las variables de entorno exponen los datos de configuración del código, lo que reduce la necesidad de codificar esos valores de forma rígida.

> **Note**
>
> Para aumentar la seguridad de la base de datos, le recomendamos que utilice Secrets Manager o AWS Systems Manager Parameter Store en lugar de variables de entorno para almacenar credenciales o información confidencial. Para aprovechar esta función, consulte Invocar AWS servicios con fuentes de datos AWS AppSync HTTP.

Las variables de entorno deben seguir varios comportamientos y reglas para funcionar correctamente:

- Tanto los JavaScript resolutores/funciones como las plantillas de VTL admiten variables de entorno.
- Las variables de entorno no se evalúan antes de la invocación de la función.
- Las variables de entorno solo admiten valores de cadena.
- Cualquier valor definido en una variable de entorno se considera una cadena literal y no se expande.
- Lo ideal es que las evaluaciones de variables se realicen en el código de la función.

Configuración de variables de entorno (consola)

Puedes configurar las variables de entorno para tu API de AWS AppSync GraphQL creando la variable y definiendo su par clave-valor. Tus resolutores y funciones utilizarán el nombre clave de la variable de entorno para recuperar el valor en tiempo de ejecución. Para configurar las variables de entorno en la AWS AppSync consola:

1. Inicie sesión en la AppSyncconsola AWS Management Console y ábrala.
2. En la página API, elija el nombre de una API de GraphQL.
3. En el panel de navegación de la página de inicio de la API de, elija Ajustes.
4. En Variables de entorno, selecciona Añadir variable de entorno.
5. Elija Add environment variable (Añadir variable de entorno).
6. Introduzca una clave y un valor.

7. Si es necesario, repita los pasos 5 y 6 para añadir más valores clave. Si necesita eliminar un valor clave, elija la opción Eliminar y las claves que desee eliminar.
8. Seleccione Submit (Enviar).

> **Tip**
>
> Hay algunas reglas que debes seguir al crear claves y valores:
>
> - Las claves deben empezar por una letra.
> - Las claves deben tener al menos dos caracteres.
> - Las claves solo pueden contener letras, números y el carácter de subrayado (_).
> - Los valores pueden tener una longitud máxima de 512 caracteres.
> - Puedes configurar hasta 50 pares clave-valor en una API de GraphQL.

Configuración de variables de entorno (API)

Para configurar una variable de entorno mediante API, puedes usar. `PutGraphqlApiEnvironmentVariables` El comando CLI correspondiente es`put-graphql-api-environment-variables`.

Para recuperar una variable de entorno mediante las API, puede utilizar`GetGraphqlApiEnvironmentVariables`. El comando CLI correspondiente es`get-graphql-api-environment-variables`.

El comando debe contener el ID de la API y la lista de variables de entorno:

```
aws appsync put-graphql-api-environment-variables \
  --api-id "<api-id>" \
  --environment-variables '{"key1":"value1","key2":"value2", …}'
```

El siguiente ejemplo establece dos variables de entorno en una API con el ID de `abcdefghijklmnopqrstuvwxyz` uso del `put-graphql-api-environment-variables` comando:

```
aws appsync put-graphql-api-environment-variables \
  --api-id "abcdefghijklmnopqrstuvwxyz" \
```

```
  --environment-variables '{"USER_TABLE":"users_prod","DEBUG":"true"}'
```

Tenga en cuenta que al aplicar variables de entorno con el `put-graphql-api-environment-variables` comando, el contenido de la estructura de las variables de entorno se sobrescribe; esto significa que se perderán las variables de entorno existentes. Para conservar las variables de entorno existentes al agregar otras nuevas, incluya todos los pares clave-valor existentes junto con los nuevos en su solicitud. Usando el ejemplo anterior, si quisieras agregar`"EMPTY":""`, podrías hacer lo siguiente:

```
aws appsync put-graphql-api-environment-variables \
  --api-id "abcdefghijklmnopqrstuvwxyz" \
  --environment-variables '{"USER_TABLE":"users_prod","DEBUG":"true", "EMPTY":""}'
```

Para recuperar la configuración actual, utilice el `get-graphql-api-environment-variables` comando:

```
aws appsync get-graphql-api-environment-variables --api-id "<api-id>"
```

Con el ejemplo anterior, puede utilizar el siguiente comando:

```
aws appsync get-graphql-api-environment-variables --api-id "abcdefghijklmnopqrstuvwxyz"
```

El resultado mostrará la lista de variables de entorno junto con sus valores clave:

```
{
    "environmentVariables": {
        "USER_TABLE": "users_prod",
        "DEBUG": "true",
        "EMPTY": ""
    }
}
```

Configuración de variables de entorno (CFN)

Puede usar la siguiente plantilla para crear variables de entorno:

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  GraphQLApiWithEnvVariables:
    Type: "AWS::AppSync::GraphQLApi"
```

```
    Properties:
      Name: "MyApiWithEnvVars"
      AuthenticationType: "AWS_IAM"
      EnvironmentVariables:
        EnvKey1: "non-empty"
        EnvKey2: ""
```

variables de entorno y API combinadas

Las variables de entorno definidas en las API de origen también están disponibles en las API combinadas. Las variables de entorno de las API combinadas son de solo lectura y no se pueden actualizar. Tenga en cuenta que las claves de las variables de entorno deben ser únicas en todas las API de origen para que las fusiones se realicen correctamente; las claves duplicadas siempre provocarán un error en la fusión.

Recuperación de variables de entorno

Para recuperar las variables de entorno del código de la función, recupere el valor del `ctx.env` objeto en los resolutores y las funciones. A continuación se muestran algunos ejemplos de esto en acción.

Publishing to Amazon SNS

En este ejemplo, nuestro solucionador HTTP envía un mensaje a un tema de Amazon SNS. El ARN del tema solo se conoce después de implementar la pila que define la API de GraphQL y el tema.

```
/**
 * 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

En este ejemplo, los nombres de la tabla de DynamoDB son diferentes si la API se implementa para la puesta en escena o si ya está en producción. No es necesario cambiar el código de

resolución. Los valores de las variables de entorno se actualizan en función del lugar en el que se despliegue la API.

```
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
      },
    ],
  };
}
```

Autorización y autenticación

En esta sección se describen las opciones para configurar la seguridad y la protección de datos de las aplicaciones.

Tipos de autorización

Existen cinco formas de autorizar que las aplicaciones interactúen con la API de AWS AppSync GraphQL. Para especificar el tipo de autorización que va a utilizar, especifique uno de los siguientes valores de tipo de autorización en su llamada a la AWS AppSync API o CLI:

- **`API_KEY`**

  Para utilizar claves de API.
- **`AWS_LAMBDA`**

  Para usar una AWS Lambda función.
- **`AWS_IAM`**

  Para usar permisos AWS Identity and Access Management (IAM).
- **`OPENID_CONNECT`**

  Para utilizar su proveedor de OpenID Connect.
- **`AMAZON_COGNITO_USER_POOLS`**

  Para utilizar con un grupo de usuarios de Amazon Cognito.

Estos tipos de autorización básicos funcionan para la mayoría de desarrolladores. Para casos de uso más avanzados, puede añadir modos de autorización adicionales mediante la consola, la CLI y AWS CloudFormation. Para modos de autorización adicionales, AWS AppSync proporciona un tipo de autorización que toma los valores enumerados anteriormente (es decir`API_KEY`,`AWS_LAMBDA`, `AWS_IAMOPENID_CONNECT`, y`AMAZON_COGNITO_USER_POOLS`).

Al especificar `API_KEY`, `AWS_LAMBDA` o `AWS_IAM` como tipo de autorización principal o predeterminado, no puede especificarlos de nuevo como uno de los modos de autorización adicionales. Del mismo modo, no puede duplicar `API_KEY`, `AWS_LAMBDA` ni `AWS_IAM` dentro de modos de autorización adicionales. Puede utilizar diferentes proveedores de OpenID Connect y

grupos de usuarios de Amazon Cognito. Sin embargo, no puede utilizar proveedores de OpenID Connect o grupos de usuarios de Amazon Cognito duplicados entre el modo de autorización predeterminado y cualquiera de los modos de autorización adicionales. Puede especificar diferentes clientes para el proveedor de OpenID Connect o el grupo de usuarios de Amazon Cognito mediante la expresión regular de configuración correspondiente.

Autorización API_KEY

Las API sin autenticar requieren un control de los límites más estricto que las API autenticadas. Una forma de supervisar la limitación controlada de puntos de enlace de GraphQL sin autenticar es mediante el uso de claves de API. Una clave de API es un valor codificado en tu aplicación que genera el AWS AppSync servicio al crear un punto final de GraphQL no autenticado. Puede rotar las claves de API desde la consola, desde la CLI o con la referencia de la API de AWS AppSync.

Console

1. Inicia sesión en la consola AWS Management Console y ábrela. AppSync

   a. En el panel de API, seleccione su API de GraphQL.

   b. En la barra lateral, seleccione Configuración.

2. En Modo de autorización predeterminado, seleccione Clave de API.
3. En la pestaña Claves de API, elija Agregar clave de API.

   Se generará una nueva clave de API en la tabla.

   - Para eliminar una clave de API antigua, selecciónela en la tabla y, a continuación, elija Eliminar.

4. Elija Guardar en la parte inferior de la página.

CLI

1. Si aún no lo ha hecho, configure el acceso a la AWS CLI. Para obtener más información, consulte Fundamentos de configuración.
2. Cree un objeto de API de GraphQL ejecutando el comando `update-graphql-api`.

   Deberá escribir dos parámetros para este comando concreto:

   1. El `api-id` de su API de GraphQL.

2. El nuevo `name` de su API. Puede utilizar el mismo `name`.
3. El `authentication-type`, que será `API_KEY`.

> **Note**
>
> Hay otros parámetros, como `Region`, que deben configurarse pero que normalmente se utilizarán de forma predeterminada en los valores de configuración de la CLI.

Un comando de ejemplo puede tener este aspecto:

```
aws appsync update-graphql-api --api-id abcdefghijklmnopqrstuvwxyz --name
 TestAPI --authentication-type API_KEY
```

Aparecerá un resultado en la CLI. A continuación se muestra un ejemplo en 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"
    }
}
```

Las claves de API se pueden configurar con una duración de hasta 365 días que pueden ampliarse hasta otros 365 días a partir de la fecha de vencimiento. Las claves de API se recomiendan con fines de desarrollo o para casos de uso en los que es seguro exponer una API pública.

En el cliente, la clave de API se especifica mediante el encabezado `x-api-key`.

Por ejemplo, si `API_KEY` es `'ABC123'`, puede enviar una consulta de GraphQL mediante `curl`, como se indica a continuación:

```
$ curl -XPOST -H "Content-Type:application/graphql" -H "x-api-key:ABC123" -d
 '{ "query": "query { movies { id } }" }' https://YOURAPPSYNCENDPOINT/graphql
```

Autorización de AWS_LAMBDA

Puede implementar su propia lógica de autorización de la API mediante una AWS Lambda función. Puede usar una función de Lambda para su autorizador principal o secundario, pero solo puede haber una función de autorización de Lambda por cada API. Cuando se utilizan funciones de Lambda para la autorización, es aplicable lo siguiente:

- Si la API tiene habilitados los modos de autorización `AWS_LAMBDA` y `AWS_IAM` , la firma SigV4 no se puede utilizar como token de autorización de `AWS_LAMBDA`.
- Si la API tiene habilitados los modos de autorización `AWS_LAMBDA` y `OPENID_CONNECT`, o el modo de autorización `AMAZON_COGNITO_USER_POOLS` activado, el token OIDC no se puede utilizar como token de autorización de `AWS_LAMBDA`. Tenga en cuenta que el token OIDC puede ser un esquema Bearer.
- Una función de Lambda no debe devolver más de 5 MB de datos contextuales para los solucionadores.

Por ejemplo, si su token de autorización es `'ABC123'`, puede enviar una consulta de GraphQL mediante curl, como se indica a continuación:

```
$ curl -XPOST -H "Content-Type:application/graphql" -H "Authorization:ABC123" -d
 '{ "query":
        "query { movies { id } }" }' https://YOURAPPSYNCENDPOINT/graphql
```

Las funciones de Lambda se invocan antes de cada consulta o mutación. El valor devuelto se puede almacenar en caché en función del ID de la API y el token de autenticación. De forma predeterminada, el almacenamiento en caché no está activado, pero se puede habilitar en la API o configurando el valor `ttlOverride` en un valor de retorno de una función.

Si lo desea, puede especificar una expresión regular que valide los tokens de autorización antes de que se llame a la función. Estas expresiones regulares se utilizan para validar que un token de autorización tiene el formato correcto antes de llamar a la función. Cualquier solicitud que utilice un token que no coincida con esta expresión regular se rechazará automáticamente.

Las funciones Lambda utilizadas para la autorización requieren que se `appsync.amazonaws.com` les aplique una política principal que permita AWS AppSync llamarlas. Esta acción se realiza automáticamente en la AWS AppSync consola; la AWS AppSync consola no elimina la política. Para obtener más información sobre cómo adjuntar políticas a las funciones de Lambda, consulte Políticas basadas en recursos en la Guía para desarrolladores. AWS Lambda

La función de Lambda que especifique recibirá un evento con la siguiente forma:

```
{
    "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
    }
}
```

El `event` objeto contiene los encabezados que se enviaron en la solicitud desde el cliente de la aplicación a. AWS AppSync

La función de autorización debe devolver al menos `isAuthorized` un booleano que indique si la solicitud está autorizada. AWS AppSync reconoce las siguientes claves devueltas por las funciones de autorización de Lambda:

Lista de funciones

`isAuthorized` (valor booleano, obligatorio)

> Un valor booleano que indica si el valor de `authorizationToken` está autorizado a realizar llamadas a la API de GraphQL.

Si este valor es true, la ejecución de la API de GraphQL continúa. Si este valor es false, se genera `UnauthorizedException`

`deniedFields` (lista de cadenas, opcional)

Una lista que se cambia forzosamente a `null`, incluso si un solucionador ha devuelto un valor.

Cada elemento es un ARN de campo totalmente cualificado en forma de `arn:aws:appsync:`*`us-east-1`*`:`*`111122223333`*`:apis/`*`GraphQLApiId`*`/types/`*`TypeName`*`/fields/`*`FieldName`* o una forma abreviada de *`TypeName`*`.`*`FieldName`*. El formulario ARN completo debe usarse cuando dos API comparten un autorizador de funciones Lambda y puede haber ambigüedad entre los tipos y campos comunes entre las dos API.

`resolverContext` (objeto JSON, opcional)

Un objeto JSON visible como `$ctx.identity.resolverContext` en las plantillas de solucionadores. Por ejemplo, si un solucionador devuelve la estructura siguiente:

```
{
  "isAuthorized":true
  "resolverContext": {
    "banana":"very yellow",
    "apple":"very green"
  }
}
```

El valor de `ctx.identity.resolverContext.apple` en las plantillas de solucionador será "`very green`". El objeto `resolverContext` solo admite pares clave-valor. No se admiten las claves anidadas.

> **Warning**
>
> El tamaño total de este objeto JSON no debe superar los 5 MB.

`ttlOverride` (entero, opcional)

El número de segundos durante los que se debe almacenar una respuesta en caché. Si no se devuelve ningún valor, se utiliza el valor de la API. Si es 0, la respuesta no se almacena en caché.

Los autorizadores de Lambda tienen un tiempo de espera de 10 segundos. Recomendamos diseñar funciones para que se ejecuten en el menor tiempo posible a fin de escalar el rendimiento de su API.

Varias AWS AppSync API pueden compartir una sola función Lambda de autenticación. No se permite el uso de autorizadores entre cuentas.

Al compartir una función de autorización entre varias API, tenga en cuenta que los nombres de campo abreviados (*typename*.*fieldname*) pueden ocultar campos de forma inadvertida. Para eliminar la ambigüedad de un campo en `deniedFields`, puede especificar un ARN de campo inequívoco en forma de `arn:aws:appsync:`*region*`:`*accountId*`:apis/`*GraphQLApiId*`/types/`*typeName*`/fields/`*fieldName*.

Para añadir una función de Lambda como modo de autorización predeterminado en AWS AppSync:

Console

1. Inicie sesión en la AWS AppSync consola y navegue hasta la API que desee actualizar.
2. Vaya a la página de Configuración de su API.

   Cambie la autorización a nivel de API a AWS Lambda.
3. Elija el ARN Región de AWS y el ARN de Lambda para autorizar las llamadas a la API.

   > **Note**
   >
   > Se añadirá automáticamente la política principal correspondiente, lo que permitirá a AWS AppSync llamar a la función de Lambda.
4. Si lo desea, configure el TTL de respuesta y la expresión regular de validación del token.

AWS CLI

1. Adjunte la política siguiente a la función de Lambda que se esté utilizando:

```
aws lambda add-permission --function-name "my-function" --statement-id "appsync"
 --principal appsync.amazonaws.com --action lambda:InvokeFunction --output text
```

> **Important**
>
> Si quiere que la política de la función se bloquee en una única API de GraphQL, puede ejecutar este comando:
>
> ```
> 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. Actualice su AWS AppSync API para usar la función de Lambda ARN dada como autorizador:

```
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**
>
> También puede incluir otras opciones de configuración, como la expresión regular del token.

En el siguiente ejemplo se describe una función de Lambda que demuestra los distintos estados de autenticación y error que puede tener una función de Lambda cuando se utiliza como mecanismo de autorización de 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 {}
```

Elusión de las limitaciones de autorización de los tokens SiGv4 y OIDC

Pueden usarse los métodos siguientes para eludir el problema de no poder usar su firma SigV4 o su token OIDC como token de autorización de Lambda cuando están habilitados ciertos modos de autorización.

Si desea utilizar la firma SigV4 como token de autorización de Lambda cuando estén habilitados los modos de autorización `AWS_IAM` y `AWS_LAMBDA` para la API de AWS AppSync, haga lo siguiente:

- Para crear un nuevo token de autorización de Lambda, añada sufijos o prefijos aleatorios a la firma SigV4.
- Para recuperar la firma SiGv4 original, actualice la función de Lambda eliminando los prefijos o sufijos aleatorios del token de autorización de Lambda. A continuación, utilice la firma SigV4 original para la autenticación.

Si desea utilizar el token OIDC como token de autorización de Lambda cuando el modo de autorización o los modos de `OPENID_CONNECT` `AWS_LAMBDA` autorización `AMAZON_COGNITO_USER_POOLS` y estén habilitados para la API, haga AWS AppSync lo siguiente:

- Para crear un nuevo token de autorización de Lambda, añada sufijos o prefijos aleatorios al token OIDC. El token de autorización de Lambda no debe contener un prefijo de esquema Bearer.
- Para recuperar el token OIDC original, actualice la función de Lambda eliminando los prefijos o sufijos aleatorios del token de autorización de Lambda. A continuación, utilice el token OIDC original para la autenticación.

Autorización AWS_IAM

Este tipo de autorización aplica el proceso de firma Signature Version 4 de AWS a la API de GraphQL. Con este tipo de autorización puede asociar políticas de acceso de Identity and Access Management (IAM). Su aplicación puede beneficiarse de esta asociación mediante el uso de una clave de acceso (que se compone de un ID de clave de acceso y una clave de acceso secreta) o mediante el uso de credenciales temporales con una vida útil corta proporcionadas por las identidades federadas de Amazon Cognito.

Si desea un rol que pueda realizar todas las operaciones de datos:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "appsync:GraphQL"
         ],
         "Resource": [
            "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/*"
         ]
      }
   ]
}
```

Puedes encontrarlo en la página principal `YourGraphQLApiId` de listados de API de la AppSync consola, justo debajo del nombre de tu API. También puede obtenerla desde la CLI: `aws appsync list-graphql-apis`

Si desea restringir el acceso a determinadas operaciones de GraphQL, puede hacerlo para los campos raíz `Query`, `Mutation` y `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>"
         ]
     }
   ]
}
```

Por ejemplo, supongamos que tiene el siguiente esquema y desea restringir el acceso a todas las publicaciones:

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

type Query {
   posts:[Post!]!
}

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

La política de IAM correspondiente a un rol (que podría asociar, por ejemplo, a un grupo de identidades de Amazon Cognito) tendría un aspecto similar al siguiente:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
            "appsync:GraphQL"
            ],
            "Resource": [
                "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/types/
Query/fields/posts"
            ]
        }
    ]
}
```

Autorización OPENID_CONNECT

Este tipo de autorización aplica tokens de OpenID Connect (OIDC) proporcionados por un servicio compatible con OIDC. Su aplicación puede aprovechar los usuarios y los privilegios definidos por su proveedor OIDC para controlar el acceso.

La URL del emisor es el único valor de configuración obligatorio que le proporcionas AWS AppSync (por ejemplo,`https://auth.example.com`). Esta URL debe poder direccionarse a través de HTTPS. AWS AppSync se añade `/.well-known/openid-configuration` a la URL del emisor y localiza la configuración de OpenID según la especificación OpenID Connect `https://auth.example.com/.well-known/openid-configuration` Discovery. Al hacerlo, espera obtener un documento JSON conforme a RFC5785 en la URL. Este documento JSON debe contener una `jwks_uri` clave que apunte al documento del conjunto de claves web JSON (JWKS) con las claves de firma. AWS AppSync requiere que los JWKS contengan los campos JSON de y. `kty` `kid`

AWS AppSync admite una amplia gama de algoritmos de firma.

Algoritmos de firma
RS256

Algoritmos de firma
RS384
RS512
PS256
PS384
PS512
HS256
HS384
HS512
ES256
ES384
ES512

Le recomendamos utilizar los algoritmos de RSA. Los tokens emitidos por el proveedor deben incluir la hora a la que se emitió el token (`iat`) y también pueden incluir la hora en que se autenticó (`auth_time`). Puede proporcionar valores de TTL para el tiempo de emisión (`iatTTL`) y para el tiempo de autenticación (`authTTL`) en la configuración de OpenID Connect para una validación adicional. Si su proveedor autoriza varias aplicaciones, también puede proporcionar una expresión regular (`clientId`) que se utiliza para autorizar por ID de cliente. Cuando `clientId` está presente en la configuración de OpenID Connect, AWS AppSync valida la afirmación exigiendo que coincida con la `clientId` afirmación `aud` o con la `azp` afirmación del token.

Para validar varios ID de cliente, utilice el operador de canalización (“|”) que equivale a "o" en una expresión regular. Por ejemplo, si su aplicación OIDC tiene cuatro clientes con ID de cliente como 0A1S2D, 1F4G9H, 1J6L4B, 6GS5MG, para validar solo los tres primeros ID de cliente, debe colocar 1F4G9H|1J6L4B|6GS5MG en el campo ID de cliente.

Autorización AMAZON_COGNITO_USER_POOLS

Este tipo de autorización aplica tokens de OIDC proporcionados por grupos de usuarios de Amazon Cognito. Su aplicación puede aprovechar los usuarios y grupos de sus grupos de usuarios y de otros grupos de usuarios de otra AWS cuenta y asociarlos a campos de GraphQL para controlar el acceso.

Si utiliza agrupaciones de usuarios de Amazon Cognito, puede crear grupos a los que pertenecen los usuarios. Esta información está codificada en un token JWT al que la aplicación envía AWS AppSync en un encabezado de autorización al enviar operaciones de GraphQL. Puede utilizar directivas de GraphQL en el esquema para controlar qué grupos pueden invocar cuáles solucionadores para un campo. De este modo otorgará un acceso más controlado a los usuarios.

Por ejemplo, suponga que tiene el siguiente esquema de GraphQL:

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

type Query {
   posts:[Post!]!
}

type Mutation {
   addPost(id:ID!, title:String!):Post!
}
...
```

Si tiene dos grupos en las agrupaciones de usuarios de Amazon Cognito (blogueros y lectores) y desea restringir el acceso de los lectores para que no puedan añadir nuevas entradas, entonces su esquema debería tener el siguiente aspecto:

```
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"])
}
...
```

Ten en cuenta que puedes omitir la `@aws_auth` directiva si quieres seguir una grant-or-deny estrategia de acceso específica de forma predeterminada. Puedes especificar la grant-or-deny estrategia en la configuración del grupo de usuarios al crear tu API de GraphQL mediante la consola o mediante el siguiente comando CLI:

```
$ 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"}'
```

Uso de modos de autorización adicionales

Al añadir modos de autorización adicionales, puede configurar directamente la configuración de autorización en el nivel de la API de AWS AppSync GraphQL (es decir, el `authenticationType` campo que puede configurar directamente en el `GraphqlApi` objeto) y actúa como predeterminado en el esquema. Esto supone que cualquier tipo que no tenga una directiva específica tiene que transmitir la configuración de autorización de nivel de API.

En el nivel de esquema, puede especificar modos de autorización adicionales mediante las directivas del esquema. Puede especificar modos de autorización en los campos individuales del esquema. Por ejemplo, para la autorización `API_KEY`, debería utilizar `@aws_api_key` en los campos o las definiciones de tipo de objeto del esquema. Las siguientes directivas se admiten en los campos y las definiciones de tipo de objeto del esquema:

- `@aws_api_key`: para especificar que el campo está autorizado por `API_KEY`.
- `@aws_iam`: para especificar que el campo está autorizado por `AWS_IAM`.
- `@aws_oidc`: para especificar que el campo está autorizado por `OPENID_CONNECT`.
- `@aws_cognito_user_pools`: para especificar que el campo está autorizado por `AMAZON_COGNITO_USER_POOLS`.
- `@aws_lambda`: para especificar que el campo está autorizado por `AWS_LAMBDA`.

No puede utilizar la directiva `@aws_auth` junto con modos de autorización adicionales. `@aws_auth` funciona únicamente en el contexto de la autorización `AMAZON_COGNITO_USER_POOLS` sin modos de autorización adicionales. No obstante, puede utilizar la directiva `@aws_cognito_user_pools` en lugar de la directiva `@aws_auth` con los mismos argumentos. La principal diferencia entre las dos es que puede especificar `@aws_cognito_user_pools` en cualquier campo y definición de tipo de objeto.

Para comprender cómo funcionan los modos de autorización adicionales y cómo se pueden especificar en un esquema, observemos el siguiente esquema:

```
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!
}
...
```

Para este esquema, supongamos que AWS_IAM es el tipo de autorización predeterminado en la API AWS AppSync GraphQL. Esto significa que los campos que no tienen una directiva se protegen mediante AWS_IAM. Por ejemplo, es el caso del campo getPost del tipo Query. Las directivas del esquema le permiten utilizar más de un modo de autorización. Por ejemplo, puedes API_KEY configurarlo como un modo de autorización adicional en la API de AWS AppSync GraphQL y puedes marcar un campo mediante la @aws_api_key directiva (por ejemplo, getAllPosts en este ejemplo). Las directivas funcionan a nivel de campo, por lo que tiene que permitir que API_KEY también acceda al tipo Post. Puede hacerlo marcando cada campo del tipo Post con una directiva o marcando el tipo Post con la directiva @aws_api_key.

Para restringir más el acceso a los campos del tipo Post, puede utilizar directivas contra los campos individuales del tipo Post como se muestra a continuación.

Por ejemplo, puede añadir un campo restrictedContent al tipo Post y restringir el acceso a él mediante la directiva @aws_iam. Las solicitudes autenticadas por AWS_IAM podrían acceder a restrictedContent, pero las solicitudes de API_KEY no.

```
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
}
...
```

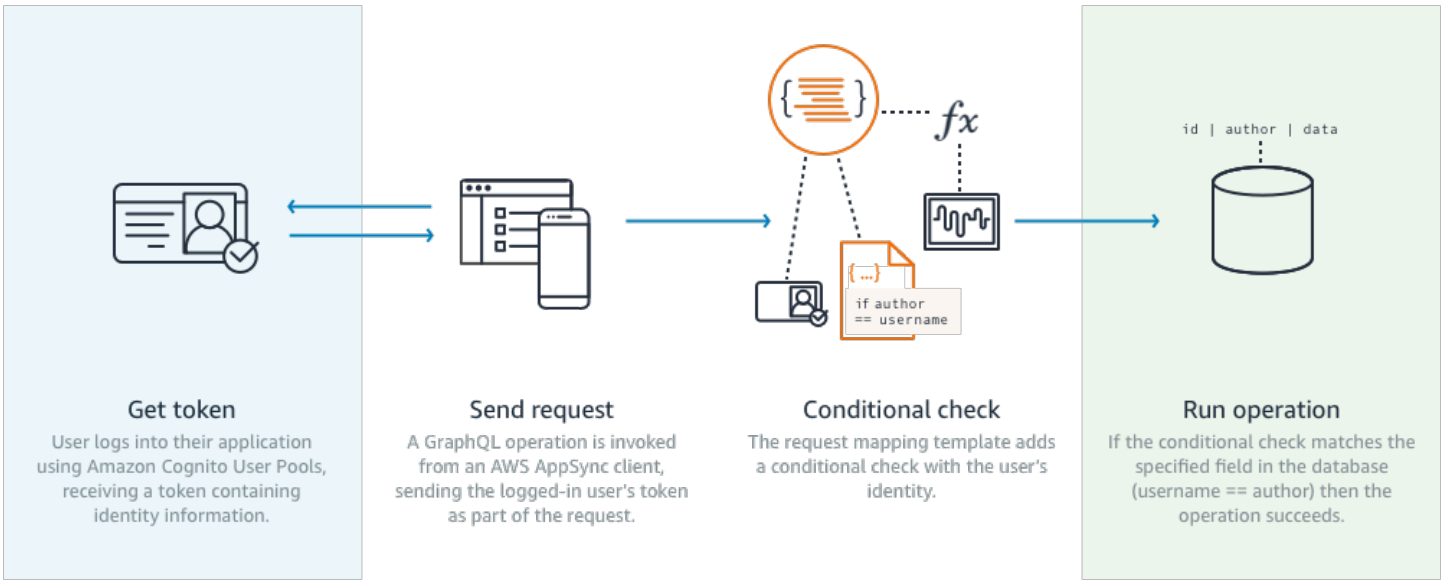
Control de acceso detallado

La información anterior muestra cómo restringir o conceder acceso a determinados campos de GraphQL. Si desea establecer controles de acceso a los datos en función de determinadas condiciones (por ejemplo, en función del usuario que realiza una llamada y de si es propietario de los datos), puede utilizar plantillas de mapeo en los solucionadores. También puede aplicar una lógica de negocio más compleja, como describimos más adelante en Filtrado de información.

En esta sección se muestra cómo establecer controles de acceso a los datos mediante una plantilla de asignación de solucionadores de DynamoDB.

Antes de continuar, si no está familiarizado con las plantillas de mapeo de Resolver AWS AppSync, puede revisar la referencia de plantillas de mapeo de Resolver y la referencia de plantillas de mapeo de Resolver para DynamoDB.

En el siguiente ejemplo con DynamoDB, imagine que utiliza el esquema de publicación del blog anterior y que solo los usuarios que han creado publicaciones tienen permiso para editarlas. El proceso de evaluación del usuario consistiría en obtener las credenciales de su aplicación, por ejemplo mediante las agrupaciones de usuarios de Amazon Cognito, y transferir entonces dichas credenciales como parte de una operación de GraphQL. A continuación, la plantilla de mapeo sustituirá un valor de las credenciales (como el nombre de usuario) en una instrucción condicional que entonces se comparará con un valor de la base de datos.



Para agregar esta funcionalidad, añada el campo de GraphQL `editPost` como se indica a continuación:

```
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!
}
...
```

La plantilla de mapeo del solucionador para `editPost` (que se muestra en un ejemplo al final de esta sección) debe realizar una comprobación lógica con el almacén de datos para permitir editar una publicación únicamente al usuario que la creó. Dado que se trata de una operación de edición, se corresponde con un `UpdateItem` de DynamoDB. Puede realizar una comprobación condicional antes de realizar esta acción utilizando el contexto transferido para validar la identidad del usuario. Esto se almacena en un objeto `Identity` que tiene los siguientes valores:

```
{
   "accountId" : "12321434323",
   "cognitoIdentityPoolId" : "",
   "cognitoIdentityId" : "",
   "sourceIP" : "",
   "caller" : "ThisistheprincipalARN",
   "username" : "username",
   "userArn" : "Sameasabove"
}
```

Para utilizar este objeto en una llamada de `UpdateItem` de DynamoDB, debe almacenar la información de identidad del usuario en la tabla para poder compararla. En primer lugar, la mutación `addPost` debe almacenar el creador. En segundo lugar, la mutación `editPost` debe realizar la comprobación condicional antes de actualizar.

El siguiente es un ejemplo del código de solucionador para `addPost` que almacena la identidad del usuario como una columna `Author`:

```
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;
```

Observe que el atributo `Author` se rellena a partir del objeto `Identity`, que procede de la aplicación.

Por último, el siguiente es un ejemplo del código de solucionador para `editPost`, que solo actualiza el contenido de la publicación del blog si la solicitud proviene del usuario que la creó:

```
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;
```

En este ejemplo se utiliza `PutItem`, lo que sobrescribe todos los valores, en lugar de `UpdateItem`, pero en ambos casos se aplica el mismo principio en el bloque de instrucciones `condition`.

Filtrado de información

Puede haber casos en los que, aunque no se pueda controlar la respuesta del origen de datos, no se desee enviar información innecesaria a los clientes tras una solicitud de escritura o lectura correctas al origen de datos. En estos casos puede filtrar la información utilizando una plantilla de mapeo de respuesta.

Por ejemplo, imagine que no dispone de un índice adecuado en una tabla DynamoDB de publicaciones de blog (por ejemplo, un índice por `Author`). Puede utilizar el siguiente solucionador:

```
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;
}
```

El controlador de solicitudes obtiene el elemento incluso si la intermediario no es el autor que creó la publicación. Para evitar que se devuelvan todos los datos, el controlador de respuestas comprueba que el intermediario coincide con el autor del elemento. Si el intermediario no coincide con esta comprobación, solo se devuelve una respuesta nula.

Acceso al origen de datos

AWS AppSync se comunica con las fuentes de datos mediante las funciones y políticas de acceso de Identity and Access Management (IAM). Si utiliza un rol existente, es necesario añadir una política de confianza AWS AppSync para poder asumir el rol. La relación de confianza tendrá este aspecto:

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

Es importante reducir el ámbito de la política de acceso para que el rol solo tenga permisos para actuar sobre el conjunto mínimo de recursos necesario. Al utilizar la AppSync consola para crear una fuente de datos y crear un rol, esto se hace automáticamente. Sin embargo, si utilizas una plantilla de ejemplo integrada en la consola de IAM para crear un rol fuera de la AWS AppSync consola, los permisos no se limitarán automáticamente a un recurso, por lo que debes realizar esta acción antes de pasar la aplicación a producción.

Casos de uso de autorizaciones

En la sección Seguridad ha aprendido cuáles son los diferentes modos de autorización para proteger la API. Además dicha sección contiene una introducción a los mecanismos de autorización precisos para comprender los conceptos y el flujo. Dado que AWS AppSync le permite llevar a cabo operaciones de lógica completas en los datos mediante el uso de plantillas de mapeo de solucionador de GraphQL, puede proteger los datos de lectura o escritura de una forma muy flexible mediante una combinación de identidad del usuario, condiciones e inyección de datos.

Si no está familiarizado con la edición de solucionadores de AWS AppSync, consulte la guía de programación.

Información general

La concesión de acceso a los datos de un sistema se realiza normalmente a través de una matriz de control de acceso, donde la intersección de una fila (recurso) y una columna (usuario/rol) es el permiso concedido.

AWS AppSync utiliza los recursos de su cuenta e integra la información de identidad (usuario/rol) en la solicitud y respuesta de GraphQL en forma de un objeto de contexto que se puede utilizar en el solucionador. Esto significa que los permisos se pueden conceder de forma adecuada en operaciones de lectura o escritura en función de la lógica del solucionador. Si esta lógica se aplica en el nivel de recursos, por ejemplo si únicamente determinados usuarios o grupos designados pueden leer/escribir en una fila de base de datos, entonces es necesario almacenar los “metadatos de autorización”. AWS AppSync no almacena datos, por lo que deberá almacenar estos metadatos de autorización con los recursos para que puedan determinarse los permisos. Normalmente los metadatos de la autorización son un atributo (columna) de una tabla de DynamoDB, por ejemplo un propietario o una lista de usuarios o grupos. Por ejemplo, podrían existir los atributos lectores y escritores.

En general esto significa que al leer un elemento individual de un origen de datos, ejecutará una instrucción condicional `#if () ... #end` para la plantilla de respuesta una vez que el solucionador haya leído el origen de datos. Normalmente la comprobación usará valores de usuario o grupo de `$context.identity` para realizar comprobaciones de pertenencia con los metadatos de autorización que devuelve una operación de lectura. Cuando haya varios registros, como en el caso de listas obtenidas con operaciones `Scan` o `Query` aplicadas a una tabla, la comprobación de condición se envía como parte de la operación al origen de datos usando valores de usuario o grupo similares.

Del mismo modo, al escribir datos se aplica una instrucción condicional a la acción (como `PutItem` o `UpdateItem`) a fin de comprobar si el usuario o el grupo que realiza la mutación tiene permiso para ello. A menudo, la instrucción condicional usará un valor de `$context.identity` para compararlo con los metadatos de autorización del recurso. Tanto para las plantillas de solicitud como para las de respuesta, también se pueden usar encabezados personalizados de clientes para realizar comprobaciones de validación.

Lectura de datos

Como se ha indicado anteriormente, los metadatos de autorización para realizar una comprobación se deben almacenar con el recurso o se deben transferir a la solicitud de GraphQL (identidad, encabezado, etc.). Como demostración, supongamos que tiene la tabla de DynamoDB siguiente:

ID	Data	PeopleCanAccess	GroupsCanAccess	Owner
123	{my: data,...}	[Mary, Joe]	[Admins, Editors]	Nadia

La clave principal es `id` y los datos a los que se debe obtener acceso son `Data`. El resto de columnas son ejemplos de comprobaciones que puede realizar para la autorización. `Owner` es de tipo `String`, mientras que `PeopleCanAccess` y `GroupsCanAccess` son `String Sets`, como se describe en Resolver mapping template reference for DynamoD.

El diagrama de Información general sobre plantillas de mapeo de solucionador muestra cómo la plantilla de respuesta no solo contiene el objeto de contexto, sino también los resultados del origen de datos. En las consultas de GraphQL referidas a elementos individuales puede utilizar la plantilla de respuesta para comprobar si el usuario tiene permiso para ver estos resultados o devolver un mensaje de error de autorización. Esto es lo que a veces se denomina “filtro de autorización”. En el caso de las consultas de GraphQL que devuelven listas como resultado de Scan o Query, es más eficaz hacer la comprobación en la plantilla de solicitud y devolver los datos solo si se cumple una condición de autorización. La implementación es entonces la siguiente:

1. GetItem: comprobación de autorización de registros individuales. Se realiza con instrucciones `#if() ... #end`.
2. Operaciones Scan o Query: la comprobación de autorización es una instrucción `"filter": {"expression":...}`. Algunas comprobaciones habituales son de igualdad (`attribute = :input`) o si un valor se encuentra en una lista (`contains(attribute, :input)`).

En el n.º 2, `attribute` representa en ambas instrucciones el nombre de columna del registro en una tabla, por ejemplo, `Owner` en el ejemplo anterior. Puede aplicar un alias con un signo # y utilizar, `"expressionNames":{...}`, pero no es obligatorio. `:input` es una referencia al valor que va a comparar con el atributo de base de datos, que definirá en `"expressionValues":{...}`. Verá estos ejemplos a continuación.

Caso de uso: el propietario puede leer

Conforme a la tabla anterior, si quiere que solo se devuelvan datos cuando `Owner == Nadia` en una operación de lectura individual (`GetItem`), la plantilla será similar a la siguiente:

```
#if($context.result["Owner"] == $context.identity.username)
    $utils.toJson($context.result)
#else
    $utils.unauthorized()
#end
```

Aquí deben tenerse en cuenta algunos puntos, ya que se volverán a usar en las secciones restantes. En primer lugar, la comprobación utiliza `$context.identity.username`, que será el nombre de inicio de sesión fácil de recordar si se usan grupos de usuario de Amazon Cognito y será la identidad de usuario si se utiliza IAM (incluidas las identidades federadas de Amazon Cognito). También deben almacenarse otros valores para el propietario, como el valor único “identidad de Amazon Cognito”, que es útil cuando se federan inicios de sesión desde varias ubicaciones, y no se olvide revisar las opciones disponibles en Resolver Mapping Template Context Reference.

En segundo lugar, la comprobación condicional "else" que responde con `$util.unauthorized()` es totalmente opcional, pero se aconseja como práctica recomendada al diseñar una API de GraphQL.

Caso de uso: acceso específico codificado

```
// 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
```

Caso de uso: filtro de una lista de resultados

En el ejemplo anterior pudimos realizar directamente una comprobación con `$context.result`, ya que se devolvía un único elemento. Sin embargo, algunas operaciones, como Scan, devuelven varios elementos en `$context.result.items`, por lo que hay que filtrar la autorización y devolver únicamente los resultados que el usuario tiene permiso para ver. Supongamos que esta vez el campo `0wner` tiene el IdentityID de Amazon Cognito establecido en el registro. En este caso, puede utilizar la siguiente plantilla de mapeo de respuestas para filtrar los registros y mostrar solo los que son propiedad del usuario:

```
#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)
```

Caso de uso: varias personas pueden leer

Otra opción de autorización frecuente consiste en permitir a un grupo de personas leer datos. En el ejemplo siguiente, `"filter":{"expression":...}` solo devuelve valores obtenidos de una tabla si el usuario que ejecuta la consulta GraphQL aparece en el conjunto `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)
        }
    }
}
```

Caso de uso: un grupo puede leer

Al igual que en el caso de uso anterior, puede ocurrir que solo las personas de uno o varios grupos tengan derecho a leer determinados elementos de una base de datos. El uso de la operación `"expression": "contains()"` es similar. Sin embargo se trata de una disyuntiva OR lógica de todos los grupos a los que puede pertenecer el usuario, lo que debe tenerse en cuenta en la pertenencia a conjuntos. En este caso hemos creado una instrucción `$expression` para cada grupo al que pertenece el usuario y la pasamos al filtro:

```
#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)
    }
}
```

Escritura de datos

La escritura de datos en las mutaciones siempre se controla en la plantilla de mapeo de solicitud. En el caso de los orígenes de datos de DynamoDB, la clave radica en utilizar una expresión de condición `"condition":{"expression"...}"` adecuada que efectúe validaciones aplicando los metadatos de autorización de la tabla. En Seguridad proporcionamos un ejemplo que puede utilizar para comprobar el campo `Author` de una tabla. Los casos de uso de esta sección exploran otras posibilidades.

Caso de uso: varios propietarios

Partiendo del diagrama de tabla del ejemplo anterior, observemos la lista `PeopleCanAccess`.

```
{
    "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)
        }
    }
}
```

Caso de uso: el grupo puede crear un registro nuevo

```
#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)
    }
}
```

Caso de uso: el grupo puede actualizar un registro existente

```
#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)
    }
}
```

Registros públicos y privados

Los filtros condicionales también le permiten marcar datos como privados, públicos o efectuar alguna comprobación booleana. Esto puede combinarse como parte de un filtro de autorización dentro de la plantilla de respuesta. El uso de esta comprobación es una buena manera de ocultar temporalmente datos o eliminarlos de la vista sin tener que controlar la pertenencia al grupo.

Por ejemplo, supongamos que añade un atributo a cada elemento de una tabla de DynamoDB denominada `public` con el valor `yes` o `no`. La plantilla de respuesta siguiente se podría utilizar en una llamada `GetItem` para mostrar datos solo si el usuario está en un grupo que tenga acceso Y si los datos están marcados como públicos:

```
#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
```

El código anterior también puede utilizar un operador lógico O (`||`) para permitir a los usuarios leer si tienen permiso sobre un registro o si este es público:

```
#if($hasPermission || $context.result.public == 'yes')
    $utils.toJson($context.result)
#else
    $utils.unauthorized()
#end
```

En general los operadores estándar `==`, `!=`, `&&` y `||` le resultarán útiles cuando realice comprobaciones de autorización.

Datos en tiempo real

Puede aplicar controles de acceso precisos en las suscripciones a GraphQL en el momento en que un cliente realice una suscripción utilizando las técnicas descritas anteriormente en esta documentación. Asocie un solucionador al campo de suscripción y ya podrá consultar un origen de datos y aplicar lógica condicional en la plantilla de mapeo de solicitud o en la de respuesta. También puede devolver datos adicionales al cliente, como los resultados iniciales de una suscripción, siempre que la estructura de datos coincida con la del tipo devuelto en la suscripción de GraphQL.

Caso de uso: el usuario solo puede suscribirse a conversaciones específicas

Un caso de uso común de datos en tiempo real con suscripciones de GraphQL consiste en crear una aplicación de mensajería o de chat privado. Al crear una aplicación de chat para varios usuarios, pueden producirse conversaciones entre dos personas o entre varias personas. Los usuarios pueden agruparse en “salas” que pueden ser privadas o públicas. En consecuencia, es necesario autorizar a cada usuario a suscribirse únicamente a las conversaciones (con otro usuario o dentro de un grupo) para las que se le haya concedido permiso. Con fines de demostración, el ejemplo siguiente muestra

un caso de uso sencillo de un usuario que envía un mensaje privado a otro. La configuración tiene dos tablas de Amazon DynamoDB:

- Tabla Messages (Mensajes): (clave principal) `toUser`, (clave de ordenación) `id`
- Tabla Permissions (Permisos): (clave principal) `username`

La tabla Messages almacena los mensajes que se envían a través de una mutación de GraphQL. La suscripción de GraphQL comprueba la tabla Permissions para consultar si existe una autorización en el tiempo de conexión del cliente. En el siguiente ejemplo se presupone que se usa el siguiente esquema de GraphQL:

```
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
}
```

Algunas de las operaciones estándar, como `createUserPermissions()`, no se tratan a continuación para ilustrar los solucionadores de suscripción, pero son implementaciones estándar de solucionadores de DynamoDB. En su lugar, vamos a centrarnos en los flujos de autorización de suscripción con solucionadores. Para enviar un mensaje de un usuario a otro, asocie un solucionador al campo `sendMessage()` y seleccione como origen de datos la tabla Messages con la plantilla de solicitud siguiente:

```
{
    "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),
    }
}
```

En este ejemplo, usaremos `$context.identity.username`. Esto devuelve la información de los usuarios de AWS Identity and Access Management o Amazon Cognito. La plantilla de respuesta simplemente traslada `$util.toJson($ctx.result)`. Guarde el resultado y vuelva a la página de

esquema. Asocie entonces un solucionador para la suscripción `newMessage()` utilizando la tabla Permissions como origen de datos y la siguiente plantilla de mapeo de solicitud:

```
{
    "version": "2018-05-29",
    "operation": "GetItem",
    "key": {
        "username": $util.dynamodb.toDynamoDBJson($ctx.identity.username),
    },
}
```

Use entonces la siguiente plantilla de mapeo de respuesta para realizar las comprobaciones de autorización con los datos de la tabla Permissions:

```
#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
```

En este caso se hacen tres comprobaciones de autorización. La primera asegura que se devuelva un resultado. La segunda garantiza que el usuario no se suscriba a mensajes destinados a otra persona. La tercera garantiza que el usuario tiene permiso para suscribirse a cualquier campo comprobando el atributo de `isAuthorizedForSubscriptions` de DynamoDB almacenado como BOOL.

Para probar que todo funciona bien, inicie sesión en la consola de AWS AppSync usando grupos de usuarios de Amazon Cognito y un usuario llamado “Nadia” y luego ejecute la siguiente suscripción de GraphQL:

```
subscription AuthorizedSubscription {
    newMessage(toUser: "Nadia") {
        id
        toUser
        fromUser
```

```
        content
    }
}
```

Si en la tabla Permissions hay un registro para el atributo de clave `username` `Nadia` con `isAuthorizedForSubscriptions` establecido en `true`, verá una respuesta correcta. Si prueba otro `username` en la consulta de `newMessage()` anterior, se devolverá un error.

Uso de AWS WAF para proteger sus API

AWS WAF es un firewall de aplicaciones web que ayuda a proteger las aplicaciones web y las API de ataques. Le permite configurar un conjunto de reglas denominadas lista de control de acceso web (ACL web) que permiten, bloquean o monitorizan (cuentan) solicitudes web en función de las reglas y condiciones de seguridad web personalizables que defina. Al integrar la API de AWS AppSync con AWS WAF, se obtiene más control y visibilidad del tráfico HTTP aceptado por la API. Para obtener más información acerca de AWS WAF, consulte How AWS WAF Works en la Guía para desarrolladores de AWS WAF.

Puede utilizar AWS WAF para proteger su API de AppSync de vulnerabilidades web comunes, como ataques de inyección de código SQL y scripting entre sitios (XSS). Esto podría afectar a la disponibilidad y el rendimiento de la API, comprometer la seguridad o consumir recursos excesivos. Por ejemplo, puede crear reglas para permitir o bloquear solicitudes de rangos de direcciones IP especificados, solicitudes de bloques CIDR, solicitudes que se originan en un país o región específico, solicitudes que contengan código SQL malintencionado o solicitudes que contengan secuencias de comandos malintencionadas.

También puede crear reglas que busquen una cadena o un patrón de expresión regular en encabezados HTTP, métodos, cadenas de consulta, URI y el cuerpo de la solicitud (limitado a los primeros 8 KB). Además, puede crear reglas para bloquear ataques de agentes de usuario específicos, bots malintencionados y scrapers de contenido. Por ejemplo, puede utilizar reglas basadas en la frecuencia para especificar el número de solicitudes web permitidas por IP de cliente en un periodo de 5 minutos actualizado constantemente.

Para obtener más información sobre los tipos de reglas compatibles y las características de AWS WAFadicionales, consulte la Guía para desarrolladores de AWS WAF y la Referencia de la API de AWS WAF.

> **Important**
>
> AWS WAF es su primera línea de defensa contra vulnerabilidades de la web. Cuando AWS WAF se habilita en una API, las reglas de AWS WAF se evalúan antes que otras características de control de acceso, como la autorización de claves API, las políticas de IAM, los tokens de OIDC y los grupos de usuarios de Amazon Cognito.

Integre una API de AppSync con AWS WAF

Puede integrar una API de Appsync con AWS WAF utilizando la AWS Management Console, la AWS CLI, AWS CloudFormation o cualquier otro cliente compatible.

Para integrar una API de AWS AppSync con AWS WAF

1. Cree una ACL web de AWS WAF. Para ver los pasos detallados para utilizar la consola de AWS WAF, consulte Crear una ACL web.
2. Defina las reglas de la ACL web. Se definen una o varias reglas en el proceso de creación de la ACL web. Para obtener información sobre cómo estructurar las reglas, consulte Reglas de AWS WAF. Para ver ejemplos de reglas útiles que puede definir para su API de AWS AppSync, consulte Creación de reglas para una ACL web.
3. Asocie la ACL web con una API de AWS AppSync. Puede realizar este paso en la consola de AWS WAF o en la consola de AppSync.
   - Para asociar la ACL web a una API de AWS AppSync en la consola de AWS WAF, siga las instrucciones para asociar o desasociar una ACL web con un recurso de AWS en la Guía para desarrolladores de AWS WAF.
   - Para asociar la ACL web con una API de AWS AppSync en la consola de AWS AppSync
     a. Inicie sesión en la AWS Management Console y abra la consola de AppSync.
     b. Elija la API que desee asociar a una ACL web.
     c. En el panel de navegación, seleccione Settings (Configuración).
     d. En la sección Firewall de aplicaciones web, active Habilitar AWS WAF.
     e. En la lista desplegable ACL web, elija el nombre de la ACL web que desee asociar a su API.
     f. Seleccione Guardar para asociar la ACL web con la API.

> **Note**
>
> Después de la creación de una ACL web en la consola de AWS WAF, la nueva ACL web puede tardar unos minutos en estar disponible. Si no ve una ACL web recién creada en el menú Firewall de aplicaciones web, espere unos minutos y vuelva a intentar los pasos para asociar la ACL web a su API.

> **Note**
>
> La integración de AWS WAF solo admite el evento `Subscription registration message` para puntos de conexión en tiempo real. AWS AppSyncresponderá con un mensaje de error en lugar de un mensaje `start_ack` para los `Subscription registration message` que bloquee AWS WAF.

Después de asociar una ACL web a una API de AWS AppSync, administrará la ACL web mediante las API de AWS WAF. No necesita volver a asociar la ACL web a su API de AWS AppSync a menos que desee asociar la API de AWS AppSync a una ACL web diferente.

Creación de reglas para una ACL web

Las reglas definen cómo inspeccionar las solicitudes web y qué hacer cuando una solicitud web coincida con los criterios de inspección. Las reglas no existen en AWS WAF de forma independiente. Puede acceder a una regla por su nombre en el grupo de reglas o en la ACL web donde está definida. Para obtener más información, consulte Reglas de AWS WAF. Los siguientes ejemplos muestran cómo definir y asociar reglas que son útiles para proteger una API de AppSync.

Example Regla de ACL web para limitar el tamaño del cuerpo de la solicitud

El siguiente es un ejemplo de una regla que limita el tamaño del cuerpo de las solicitudes. Esto se introduciría en el Editor JSON de reglas al crear una ACL web en la consola de AWS WAF.

```
{
    "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
        }
}
```

Después de crear la ACL web con la regla del ejemplo anterior, debe asociarla a la API de AppSync. Como alternativa al uso de AWS Management Console, puede realizar este paso en la AWS CLI ejecutando el siguiente comando.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Los cambios pueden tardar unos minutos en propagarse, pero después de ejecutar este comando, las solicitudes que contengan un cuerpo superior a 1024 bytes serán rechazadas por AWS AppSync.

> **Note**
>
> Tras crear una nueva ACL web en la consola de AWS WAF, la ACL web puede tardar unos minutos en estar disponible para asociarse a una API. Si ejecuta el comando de la CLI y aparece un error `WAFUnavailableEntityException`, espere unos minutos y vuelva a intentar ejecutar el comando.

Example Regla de ACL web para limitar las solicitudes desde una única dirección IP

El siguiente es un ejemplo de una regla que limita una API de AppSync a 100 solicitudes desde una sola dirección IP. Esto se introduciría en el Editor JSON de reglas al crear una ACL web con una regla basada en tasas en la consola de AWS WAF.

```
{
  "Name": "Throttle",
  "Priority": 0,
  "Action": {
    "Block": {}
  },
  "VisibilityConfig": {
    "SampledRequestsEnabled": true,
    "CloudWatchMetricsEnabled": true,
    "MetricName": "Throttle"
  },
  "Statement": {
    "RateBasedStatement": {
      "Limit": 100,
      "AggregateKeyType": "IP"
    }
  }
}
```

Después de crear la ACL web con la regla del ejemplo anterior, debe asociarla a la API de AppSync. Puede realizar este paso en la AWS CLI ejecutando el siguiente comando.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Example Regla de ACL web para evitar consultas de introspección __schema de GraphQL a una API

El siguiente es un ejemplo de una regla que impide las consultas de introspección __schema de GraphQL a una API. Se bloqueará cualquier cuerpo HTTP que incluya la cadena “__schema”. Esto se introduciría en el Editor JSON de reglas al crear una ACL web en la consola de AWS WAF.

```
{
  "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
        }
      ]
    }
  }
}
```

Después de crear la ACL web con la regla del ejemplo anterior, debe asociarla a la API de AppSync. Puede realizar este paso en la AWS CLI ejecutando el siguiente comando.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Seguridad en AWS AppSync

La seguridad en la nube AWS es la máxima prioridad. Como AWS cliente, usted se beneficia de los centros de datos y las arquitecturas de red diseñados para cumplir con los requisitos de las organizaciones más sensibles a la seguridad.

La seguridad es una responsabilidad compartida entre AWS usted y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta AWS los servicios en la AWS nube. AWS también le proporciona servicios que puede utilizar de forma segura. Los auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los AWS programas de de . Para obtener más información sobre los programas de cumplimiento aplicables AWS AppSync, consulte AWS Servicios incluidos en el ámbito de aplicación por programa de conformidad y AWS servicios incluidos .
- Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice. Usted también es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos aplicables.

Esta documentación le ayuda a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza AWS AppSync. Los siguientes temas muestran cómo configurarlo AWS AppSync para cumplir sus objetivos de seguridad y conformidad. También aprenderá a utilizar otros AWS servicios que le ayudan a supervisar y proteger sus AWS AppSync recursos.

Temas

- Protección de datos en AWS AppSync
- Validación de conformidad para AWS AppSync
- Seguridad de la infraestructura en AWS AppSync
- Resiliencia en AWS AppSync
- Gestión de identidad y acceso para AWS AppSync
- Registrar las llamadas a AWS AppSync la API con AWS CloudTrail
- Prácticas recomendadas de seguridad para AWS AppSync

Protección de datos en AWS AppSync

El modelo de responsabilidad AWS compartida modelo se aplica a la protección de datos en AWS AppSync. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global que ejecuta todos los Nube de AWS. Usted es responsable de mantener el control sobre el contenido alojado en esta infraestructura. Usted también es responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte las Preguntas frecuentes sobre la privacidad de datos. Para obtener información sobre la protección de datos en Europa, consulte la publicación de blog sobre el Modelo de responsabilidad compartida de AWS y GDPR en el Blog de seguridad de AWS .

Con fines de protección de datos, le recomendamos que proteja Cuenta de AWS las credenciales y configure los usuarios individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utilice la autenticación multifactor (MFA) en cada cuenta.
- Utilice SSL/TLS para comunicarse con los recursos. AWS Se recomienda el uso de TLS 1.2 y recomendamos TLS 1.3.
- Configure la API y el registro de actividad de los usuarios con. AWS CloudTrail
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados Servicios de AWS.
- Utilice servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.
- Si necesita módulos criptográficos validados por FIPS 140-2 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto final FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte Estándar de procesamiento de la información federal (FIPS) 140-2.

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como, por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Esto incluye cuando trabaja AWS AppSync o Servicios de AWS utiliza la consola, la API o los SDK. AWS CLI AWS Cualquier dato que ingrese en etiquetas o campos de formato libre utilizados para nombres se puede emplear para los registros de facturación o

diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya la información de las credenciales en la URL para validar la solicitud para ese servidor.

Cifrado en movimiento

AWS AppSync, como todos los AWS servicios, utiliza el TLS1.2 y versiones posteriores para la comunicación cuando utiliza las API y los AWS SDK publicados.

El uso AWS AppSync con otros AWS servicios, como Amazon DynamoDB, garantiza el cifrado en tránsito: AWS todos los servicios utilizan TLS 1.2 y versiones posteriores para comunicarse entre sí, a menos que se especifique lo contrario. En el caso de los solucionadores que utilizan Amazon EC2 CloudFront o, es su responsabilidad comprobar que TLS (HTTPS) esté configurado y sea seguro. Para obtener información sobre la configuración de HTTPS en Amazon EC2, consulte Configuración de SSL/TLS en Amazon Linux 2 en la Guía del usuario de Amazon EC2. Para obtener información sobre cómo configurar HTTPS en Amazon CloudFront, consulta HTTPS en Amazon CloudFront en la guía del CloudFront usuario.

Validación de conformidad para AWS AppSync

Los auditores externos evalúan la seguridad y el cumplimiento AWS AppSync como parte de varios programas de AWS cumplimiento. AWS AppSync cumple con los programas SOC, PCI, HIPAA/HIPAA BAA, IRAP, C5, ENS High, OSPAR e HITRUST CSF.

Para saber si uno Servicio de AWS está dentro del ámbito de aplicación de programas de cumplimiento específicos, consulte Alcance por programa de cumplimiento Servicios de AWS en Alcance por programa de cumplimiento y elija el programa de cumplimiento que le . Para obtener información general, consulte Programas de AWS cumplimiento > Programas AWS .

Puede descargar informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact .

Su responsabilidad de cumplimiento al Servicios de AWS utilizarlos viene determinada por la confidencialidad de sus datos, los objetivos de cumplimiento de su empresa y las leyes y reglamentos aplicables. AWS proporciona los siguientes recursos para ayudar con el cumplimiento:

- Guías de inicio rápido sobre seguridad y cumplimiento: estas guías de implementación analizan las consideraciones arquitectónicas y proporcionan los pasos para implementar entornos básicos centrados en AWS la seguridad y el cumplimiento.

- Diseño de arquitectura para garantizar la seguridad y el cumplimiento de la HIPAA en Amazon Web Services: en este documento técnico se describe cómo pueden utilizar AWS las empresas para crear aplicaciones aptas para la HIPAA.

  > **Note**
  >
  > No Servicios de AWS todas cumplen con los requisitos de la HIPAA. Para más información, consulte la Referencia de servicios compatibles con HIPAA.

- AWS Recursos de de cumplimiento: esta colección de libros de trabajo y guías puede aplicarse a su industria y ubicación.
- AWS Guías de cumplimiento para clientes: comprenda el modelo de responsabilidad compartida desde el punto de vista del cumplimiento. Las guías resumen las mejores prácticas para garantizar la seguridad Servicios de AWS y orientan los controles de seguridad en varios marcos (incluidos el Instituto Nacional de Estándares y Tecnología (NIST), el Consejo de Normas de Seguridad del Sector de Tarjetas de Pago (PCI) y la Organización Internacional de Normalización (ISO)).
- Evaluación de los recursos con reglas en la guía para AWS Config desarrolladores: el AWS Config servicio evalúa en qué medida las configuraciones de los recursos cumplen con las prácticas internas, las directrices del sector y las normas.
- AWS Security Hub— Esto Servicio de AWS proporciona una visión completa del estado de su seguridad interior AWS. Security Hub utiliza controles de seguridad para evaluar sus recursos de AWS y comprobar su cumplimiento con los estándares y las prácticas recomendadas del sector de la seguridad. Para obtener una lista de los servicios y controles compatibles, consulte la Referencia de controles de Security Hub.
- Amazon GuardDuty: Servicio de AWS detecta posibles amenazas para sus cargas de trabajo Cuentas de AWS, contenedores y datos mediante la supervisión de su entorno para detectar actividades sospechosas y maliciosas. GuardDuty puede ayudarlo a cumplir con varios requisitos de conformidad, como el PCI DSS, al cumplir con los requisitos de detección de intrusiones exigidos por ciertos marcos de cumplimiento.
- AWS Audit Manager— Esto le Servicio de AWS ayuda a auditar continuamente su AWS uso para simplificar la gestión del riesgo y el cumplimiento de las normativas y los estándares del sector.

Seguridad de la infraestructura en AWS AppSync

Como servicio gestionado, AWS AppSync está protegido por la seguridad de la red AWS global. Para obtener información sobre los servicios AWS de seguridad y cómo se AWS protege la infraestructura,

consulte Seguridad AWS en la nube. Para diseñar su AWS entorno utilizando las mejores prácticas de seguridad de la infraestructura, consulte Protección de infraestructuras en un marco de buena AWS arquitectura basado en el pilar de la seguridad.

Utiliza las llamadas a la API AWS publicadas para acceder a AWS AppSync través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad de seguridad de IAM principal. También puede utilizar AWS Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Resiliencia en AWS AppSync

La infraestructura AWS global se basa en AWS regiones y zonas de disponibilidad. AWS Las regiones proporcionan varias zonas de disponibilidad aisladas y separadas físicamente, que están conectadas mediante redes de baja latencia, alto rendimiento y alta redundancia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de uno o varios centros de datos.

Para obtener más información sobre AWS las regiones y las zonas de disponibilidad, consulte Infraestructura global.AWS

Además de la infraestructura AWS global, AWS AppSync permite definir la mayoría de los recursos mediante AWS CloudFormation plantillas; para ver un ejemplo del uso de AWS CloudFormation plantillas para declarar AWS AppSync recursos, consulte Casos prácticos de AWS AppSync Pipeline Resolvers en el AWS blog y en la Guía del AWS CloudFormation usuario.

Gestión de identidad y acceso para AWS AppSync

AWS Identity and Access Management (IAM) es una herramienta Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a los AWS recursos. Los administradores de IAM controlan quién puede autenticarse (iniciar sesión) y quién puede autorizarse (tener permisos) para usar los recursos. AWS AppSync La IAM es una Servicio de AWS opción que puede utilizar sin coste adicional.

Temas

- Público
- Autenticación con identidades
- Administración de acceso mediante políticas
- ¿Cómo AWS AppSync funciona con IAM
- Políticas de AWS AppSync basadas en identidades
- Solución de problemas AWS AppSync de identidad y acceso

Público

La forma de usar AWS Identity and Access Management (IAM) varía según el trabajo en el que se realice. AWS AppSync

Usuario del servicio: si utiliza el AWS AppSync servicio para realizar su trabajo, el administrador le proporcionará las credenciales y los permisos que necesita. A medida que vaya utilizando más AWS AppSync funciones para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarlo a solicitar los permisos correctos al administrador. Si no puede acceder a una función en AWS AppSync, consulteSolución de problemas AWS AppSync de identidad y acceso.

Administrador de servicios: si está a cargo de AWS AppSync los recursos de su empresa, probablemente tenga acceso total a ellos AWS AppSync. Su trabajo consiste en determinar a qué AWS AppSync funciones y recursos deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su administrador de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página para conocer los conceptos básicos de IAM. Para obtener más información sobre cómo su empresa puede utilizar la IAM AWS AppSync, consulte¿Cómo AWS AppSync funciona con IAM.

Administrador de IAM: si es administrador de IAM, puede que le interese obtener más información sobre cómo redactar políticas para administrar el acceso. AWS AppSync Para ver ejemplos de

políticas AWS AppSync basadas en la identidad que puede usar en IAM, consulte. Políticas de AWS AppSync basadas en identidades

Autenticación con identidades

La autenticación es la forma de iniciar sesión AWS con sus credenciales de identidad. Debe estar autenticado (con quien haya iniciado sesión AWS) como usuario de IAM o asumiendo una función de IAM. Usuario raíz de la cuenta de AWS

Puede iniciar sesión AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (Centro de identidades de IAM), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su administrador habrá configurado previamente la federación de identidades mediante roles de IAM. Cuando accedes AWS mediante la federación, estás asumiendo un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en el portal AWS Management Console o en el de AWS acceso. Para obtener más información sobre cómo iniciar sesión AWS, consulte Cómo iniciar sesión Cuenta de AWS en su Guía del AWS Sign-In usuario.

Si accede AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de línea de comandos (CLI) para firmar criptográficamente sus solicitudes con sus credenciales. Si no utilizas AWS herramientas, debes firmar las solicitudes tú mismo. Para obtener más información sobre cómo usar el método recomendado para firmar las solicitudes usted mismo, consulte Firmar las solicitudes de la AWS API en la Guía del usuario de IAM.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, le AWS recomienda que utilice la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte Autenticación multifactor en la Guía del usuario de AWS IAM Identity Center y Uso de la autenticación multifactor (MFA) en AWSen la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear una Cuenta de AWS, comienza con una identidad de inicio de sesión que tiene acceso completo a todos Servicios de AWS los recursos de la cuenta. Esta identidad se denomina usuario Cuenta de AWS raíz y se accede a ella iniciando sesión con la dirección de correo electrónico y la contraseña que utilizaste para crear la cuenta. Recomendamos encarecidamente que no utilice el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren

que inicie sesión como usuario raíz, consulte Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos, incluidos los que requieren acceso de administrador, que utilicen la federación con un proveedor de identidades para acceder Servicios de AWS mediante credenciales temporales.

Una identidad federada es un usuario del directorio de usuarios de su empresa, un proveedor de identidades web AWS Directory Service, el directorio del Centro de Identidad o cualquier usuario al que acceda Servicios de AWS mediante las credenciales proporcionadas a través de una fuente de identidad. Cuando las identidades federadas acceden Cuentas de AWS, asumen funciones y las funciones proporcionan credenciales temporales.

Para una administración de acceso centralizada, le recomendamos que utilice AWS IAM Identity Center. Puede crear usuarios y grupos en el Centro de identidades de IAM, o puede conectarse y sincronizarse con un conjunto de usuarios y grupos de su propia fuente de identidad para usarlos en todas sus Cuentas de AWS aplicaciones. Para obtener más información, consulte ¿Qué es el Centro de identidades de IAM? en la Guía del usuario de AWS IAM Identity Center .

Usuarios y grupos de IAM

Un usuario de IAM es una identidad propia Cuenta de AWS que tiene permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulte Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración en la Guía del usuario de IAM.

Un grupo de IAM es una identidad que especifica un conjunto de usuarios de IAM. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos de grandes conjuntos de usuarios. Por ejemplo, podría tener un grupo cuyo nombre fuese IAMAdmins y conceder permisos a dicho grupo para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales permanentes a largo plazo y los roles proporcionan credenciales temporales.

Para más información, consulte Cuándo crear un usuario de IAM (en lugar de un rol) en la Guía del usuario de IAM.

Roles de IAM

Un rol de IAM es una identidad dentro de usted Cuenta de AWS que tiene permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una determinada persona. Puede asumir temporalmente una función de IAM en el AWS Management Console cambiando de función. Puede asumir un rol llamando a una operación de AWS API AWS CLI o utilizando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulte Uso de roles de IAM en la Guía del usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles para federación, consulte Creación de un rol para un proveedor de identidades de terceros en la Guía del usuario de IAM. Si utiliza IAM Identity Center, debe configurar un conjunto de permisos. IAM Identity Center correlaciona el conjunto de permisos con un rol en IAM para controlar a qué pueden acceder las identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulte Conjuntos de permisos en la Guía del usuario de AWS IAM Identity Center .
- Permisos de usuario de IAM temporales: un usuario de IAM puede asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- Acceso entre cuentas: puede utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. Sin embargo, en algunos casos Servicios de AWS, puedes adjuntar una política directamente a un recurso (en lugar de usar un rol como proxy). Para obtener información acerca de la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del usuario de IAM.
- Acceso entre servicios: algunos Servicios de AWS utilizan funciones en otros Servicios de AWS. Por ejemplo, cuando realiza una llamada en un servicio, es común que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado al servicio.

- Sesiones de acceso directo (FAS): cuando utilizas un usuario o un rol de IAM para realizar acciones en ellas AWS, se te considera director. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. El FAS utiliza los permisos del principal que llama Servicio de AWS y los solicita Servicio de AWS para realizar solicitudes a los servicios descendentes. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.
- Rol de servicio: un rol de servicio es un rol de IAM que adopta un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.
- Función vinculada al servicio: una función vinculada a un servicio es un tipo de función de servicio que está vinculada a un. Servicio de AWS El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

- Aplicaciones que se ejecutan en Amazon EC2: puede usar un rol de IAM para administrar las credenciales temporales de las aplicaciones que se ejecutan en una instancia EC2 y realizan AWS CLI solicitudes a la API. AWS Es preferible hacerlo de este modo a almacenar claves de acceso en la instancia de EC2. Para asignar una AWS función a una instancia EC2 y ponerla a disposición de todas sus aplicaciones, debe crear un perfil de instancia adjunto a la instancia. Un perfil de instancia contiene el rol y permite a los programas que se ejecutan en la instancia de EC2 obtener credenciales temporales. Para más información, consulte Uso de un rol de IAM para conceder permisos a aplicaciones que se ejecutan en instancias Amazon EC2 en la Guía del usuario de IAM.

Para obtener información sobre el uso de los roles de IAM, consulte Cuándo crear un rol de IAM (en lugar de un usuario) en la Guía del usuario de IAM.

Administración de acceso mediante políticas

El acceso se controla AWS creando políticas y adjuntándolas a AWS identidades o recursos. Una política es un objeto AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando un director (usuario, usuario raíz o sesión de rol) realiza una

solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan AWS como documentos JSON. Para obtener más información sobre la estructura y el contenido de los documentos de política JSON, consulte Información general de políticas JSON en la Guía del usuario de IAM.

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan. A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Las políticas de IAM definen permisos para una acción independientemente del método que se utilice para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con esa política puede obtener información sobre el rol de la API AWS Management Console AWS CLI, la o la AWS API.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede adjuntar a varios usuarios, grupos y roles de su Cuenta de AWS empresa. Las políticas administradas incluyen políticas AWS administradas y políticas administradas por el cliente. Para más información sobre cómo elegir una política administrada o una política insertada, consulte Elegir entre políticas administradas y políticas insertadas en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico.

Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o. Servicios de AWS

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puedes usar políticas AWS gestionadas de IAM en una política basada en recursos.

Listas de control de acceso (ACL)

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3 y Amazon VPC son ejemplos de servicios que admiten las ACL. AWS WAF Para obtener más información sobre las ACL, consulte Información general de Lista de control de acceso (ACL) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite tipos de políticas adicionales y menos comunes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- Límites de permisos: un límite de permisos es una característica avanzada que le permite establecer los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM (usuario o rol de IAM). Puede establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifiquen el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulte Límites de permisos para las entidades de IAM en la Guía del usuario de IAM.
- Políticas de control de servicios (SCP): las SCP son políticas de JSON que especifican los permisos máximos para una organización o unidad organizativa (OU). AWS Organizations AWS Organizations es un servicio para agrupar y gestionar de forma centralizada varios de los Cuentas de AWS que son propiedad de su empresa. Si habilita todas las características en una organización, entonces podrá aplicar políticas de control de servicio (SCP) a una o a todas sus cuentas. El SCP limita los permisos de las entidades en las cuentas de los miembros, incluidas las de cada una. Usuario raíz de la cuenta de AWS Para obtener más información acerca de

Organizations y las SCP, consulte Funcionamiento de las SCP en la Guía del usuario de AWS Organizations .

- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para más información, consulte Políticas de sesión en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para saber cómo AWS determinar si se debe permitir una solicitud cuando se trata de varios tipos de políticas, consulte la lógica de evaluación de políticas en la Guía del usuario de IAM.

¿Cómo AWS AppSync funciona con IAM

Antes de utilizar IAM para gestionar el acceso AWS AppSync, infórmese sobre las funciones de IAM disponibles para su uso. AWS AppSync

Funciones de IAM que puede utilizar con AWS AppSync

Característica de IAM	AWS AppSync soporte
Políticas basadas en identidades	Sí
Políticas basadas en recursos	No
Acciones de políticas	Sí
Recursos de políticas	Sí
Claves de condición de política	No
ACL	No
ABAC (etiquetas en políticas)	Parcial

Característica de IAM	AWS AppSync soporte
Credenciales temporales	Sí
Sesiones de acceso directo (FAS)	Parcial
Roles de servicio	No
Roles vinculados al servicio	Parcial

Para obtener una visión general de cómo AWS AppSync funcionan otros AWS servicios con la mayoría de las funciones de IAM, consulte AWS los servicios que funcionan con IAM en la Guía del usuario de IAM.

Políticas basadas en la identidad para AWS AppSync

Compatibilidad con las políticas basadas en identidad	Sí

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Con las políticas basadas en identidades de IAM, puede especificar las acciones y los recursos permitidos o denegados, así como las condiciones en las que se permiten o deniegan las acciones. No es posible especificar la entidad principal en una política basada en identidad porque se aplica al usuario o rol al que está adjunto. Para más información sobre los elementos que puede utilizar en una política de JSON, consulte Referencia de los elementos de las políticas de JSON de IAM en la Guía del usuario de IAM.

Ejemplos de políticas basadas en la identidad para AWS AppSync

Para ver ejemplos de políticas AWS AppSync basadas en la identidad, consulte. Políticas de AWS AppSync basadas en identidades

Políticas basadas en recursos dentro de AWS AppSync

Compatibilidad con las políticas basadas en recursos	No

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o. Servicios de AWS

Para habilitar el acceso entre cuentas, puede especificar toda una cuenta o entidades de IAM de otra cuenta como la entidad principal de una política en función de recursos. Añadir a una política en función de recursos una entidad principal entre cuentas es solo una parte del establecimiento de una relación de confianza. Cuando el principal y el recurso son diferentes Cuentas de AWS, el administrador de IAM de la cuenta de confianza también debe conceder a la entidad principal (usuario o rol) permiso para acceder al recurso. Para conceder el permiso, adjunte la entidad a una política basada en identidad. Sin embargo, si la política en función de recursos concede el acceso a una entidad principal de la misma cuenta, no es necesaria una política basada en identidad adicional. Para más información, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del usuario de IAM.

Acciones políticas para AWS AppSync

Admite acciones de política	Sí

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Action` de una política JSON describe las acciones que puede utilizar para conceder o denegar el acceso en una política. Las acciones políticas suelen tener el mismo nombre que la operación de AWS API asociada. Hay algunas excepciones, como acciones de solo permiso que no

tienen una operación de API coincidente. También hay algunas operaciones que requieren varias acciones en una política. Estas acciones adicionales se denominan acciones dependientes.

Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Para ver una lista de AWS AppSync acciones, consulta las acciones definidas AWS AppSync en la Referencia de autorización del servicio.

Las acciones políticas AWS AppSync utilizan el siguiente prefijo antes de la acción:

```
appsync
```

Para especificar varias acciones en una única instrucción, sepárelas con comas.

```
"Action": [
      "appsync:action1",
      "appsync:action2"
         ]
```

Para ver ejemplos de políticas AWS AppSync basadas en la identidad, consulte. Políticas de AWS AppSync basadas en identidades

Recursos de políticas para AWS AppSync

Admite recursos de políticas	Sí

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Resource` de la política JSON especifica el objeto u objetos a los que se aplica la acción. Las instrucciones deben contener un elemento `Resource` o `NotResource`. Como práctica recomendada, especifique un recurso utilizando el Nombre de recurso de Amazon (ARN). Puede hacerlo para acciones que admitan un tipo de recurso específico, conocido como permisos de nivel de recurso.

Para las acciones que no admiten permisos de nivel de recurso, como las operaciones de descripción, utilice un carácter comodín (*) para indicar que la instrucción se aplica a todos los recursos.

```
"Resource": "*"
```

Para ver una lista de los tipos de AWS AppSync recursos y sus ARN, consulte los recursos definidos AWS AppSync en la Referencia de autorización de servicios. Para saber con qué acciones puede especificar el ARN de cada recurso, consulte Acciones definidas por. AWS AppSync

Para ver ejemplos de políticas AWS AppSync basadas en la identidad, consulte. Políticas de AWS AppSync basadas en identidades

Claves de condición de la política para AWS AppSync

Admite claves de condición de políticas específicas del servicio	No

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Condition` (o bloque de `Condition`) permite especificar condiciones en las que entra en vigor una instrucción. El elemento `Condition` es opcional. Puede crear expresiones condicionales que utilicen operadores de condición, tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud.

Si especifica varios elementos de `Condition` en una instrucción o varias claves en un único elemento de `Condition`, AWS las evalúa mediante una operación `AND` lógica. Si especifica varios valores para una única clave de condición, AWS evalúa la condición mediante una `OR` operación lógica. Se deben cumplir todas las condiciones antes de que se concedan los permisos de la instrucción.

También puede utilizar variables de marcador de posición al especificar condiciones. Por ejemplo, puede conceder un permiso de usuario de IAM para acceder a un recurso solo si está etiquetado con su nombre de usuario de IAM. Para más información, consulte Elementos de la política de IAM: variables y etiquetas en la Guía del usuario de IAM.

AWS admite claves de condición globales y claves de condición específicas del servicio. Para ver todas las claves de condición AWS globales, consulte las claves de contexto de condición AWS globales en la Guía del usuario de IAM.

Para ver una lista de claves de AWS AppSync condición, consulte las claves de condición AWS AppSync en la Referencia de autorización de servicio. Para saber con qué acciones y recursos puede utilizar una clave de condición, consulte Acciones definidas por AWS AppSync.

Para ver ejemplos de políticas AWS AppSync basadas en la identidad, consulte. Políticas de AWS AppSync basadas en identidades

Listas de control de acceso (ACL) en AWS AppSync

Admite las ACL	No

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Control de acceso basado en atributos (ABAC) con AWS AppSync

Admite ABAC (etiquetas en las políticas)	Parcial

El control de acceso basado en atributos (ABAC) es una estrategia de autorización que define permisos en función de atributos. En AWS, estos atributos se denominan etiquetas. Puede adjuntar etiquetas a las entidades de IAM (usuarios o roles) y a muchos AWS recursos. El etiquetado de entidades y recursos es el primer paso de ABAC. A continuación, designa las políticas de ABAC para permitir operaciones cuando la etiqueta de la entidad principal coincida con la etiqueta del recurso al que se intenta acceder.

ABAC es útil en entornos que crecen con rapidez y ayuda en situaciones en las que la administración de las políticas resulta engorrosa.

Para controlar el acceso en función de etiquetas, debe proporcionar información de las etiquetas en el elemento de condición de una política utilizando las claves de condición `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`.

Si un servicio admite las tres claves de condición para cada tipo de recurso, el valor es Sí para el servicio. Si un servicio admite las tres claves de condición solo para algunos tipos de recursos, el valor es Parcial.

Para obtener más información sobre ABAC, consulte ¿Qué es ABAC? en la Guía del usuario de IAM. Para ver un tutorial con los pasos para configurar ABAC, consulte Uso del control de acceso basado en atributos (ABAC) en la Guía del usuario de IAM.

Utilizar credenciales temporales con AWS AppSync

Compatible con el uso de credenciales temporales	Sí

Algunas Servicios de AWS no funcionan cuando inicias sesión con credenciales temporales. Para obtener información adicional, incluidas las que Servicios de AWS funcionan con credenciales temporales, consulta Cómo Servicios de AWS funcionan con IAM en la Guía del usuario de IAM.

Utiliza credenciales temporales si inicia sesión en ellas AWS Management Console mediante cualquier método excepto un nombre de usuario y una contraseña. Por ejemplo, cuando accedes AWS mediante el enlace de inicio de sesión único (SSO) de tu empresa, ese proceso crea automáticamente credenciales temporales. También crea credenciales temporales de forma automática cuando inicia sesión en la consola como usuario y luego cambia de rol. Para más información sobre el cambio de roles, consulte Cambio a un rol (consola) en la Guía del usuario de IAM.

Puedes crear credenciales temporales manualmente mediante la AWS CLI API o. AWS A continuación, puede utilizar esas credenciales temporales para acceder AWS. AWS recomienda generar credenciales temporales de forma dinámica en lugar de utilizar claves de acceso a largo plazo. Para más información, consulte Credenciales de seguridad temporales en IAM.

Sesiones de acceso directo para AWS AppSync

Admite sesiones de acceso directo (FAS)	Parcial

Cuando utiliza un usuario o un rol de IAM para realizar acciones en AWSél, se le considera director. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FAS utiliza los permisos del principal que llama y los que solicita Servicio de AWS para realizar solicitudes a los servicios descendentes. Servicio de AWS Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS recursos para completarse. En este caso, debe tener permisos para realizar

ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.

Roles de servicio para AWS AppSync

Compatible con roles de servicio	No

Un rol de servicio es un rol de IAM que asume un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.

> **Warning**
>
> Si se cambian los permisos de un rol de servicio, es posible que se interrumpa AWS AppSync la funcionalidad. Edite las funciones de servicio solo cuando se AWS AppSync proporcionen instrucciones para hacerlo.

Funciones vinculadas al servicio para AWS AppSync

Compatible con roles vinculados al servicio	Parcial

Un rol vinculado a un servicio es un tipo de rol de servicio que está vinculado a un. Servicio de AWS El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

Para obtener más información acerca de cómo crear o administrar roles vinculados a servicios, consulte Servicios de AWS que funcionan con IAM en la Guía de usuario de IAM. Busque un servicio en la tabla que incluya `Yes` en la columna Rol vinculado a un servicio. Seleccione el vínculo Sí para ver la documentación acerca del rol vinculado a servicios para ese servicio.

Políticas de AWS AppSync basadas en identidades

De forma predeterminada, los usuarios y los roles no tienen permiso para crear o modificar AWS AppSync recursos. Tampoco pueden realizar tareas mediante la AWS Management Console, AWS

Command Line Interface (AWS CLI) o la AWS API. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan. A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Para obtener información acerca de cómo crear una política basada en identidades de IAM mediante el uso de estos documentos de políticas JSON de ejemplo, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Para obtener más información sobre las acciones y los tipos de recursos definidos por cada uno de los tipos de recursos AWS AppSync, incluido el formato de los ARN para cada uno de los tipos de recursos, consulte las claves de condición, recursos y acciones de la Referencia de autorización de servicios. AWS AppSync

Para conocer las prácticas recomendadas para crear y configurar políticas de IAM basadas en la identidad, consulte the section called “Prácticas recomendadas sobre políticas de IAM”.

Para obtener una lista de las políticas de IAM basadas en la identidad, consulte. AWS AppSync AWS políticas gestionadas para AWS AppSync

Temas

- Mediante la consola de AWS AppSync
- Cómo permitir a los usuarios consultar sus propios permisos
- Acceso a un bucket de Amazon S3
- Visualización de AWS AppSync widgets basados en etiquetas
- AWS políticas gestionadas para AWS AppSync

Mediante la consola de AWS AppSync

Para acceder a la AWS AppSync consola, debe tener un conjunto mínimo de permisos. Estos permisos deben permitirle enumerar y ver detalles sobre los AWS AppSync recursos de su cuenta Cuenta de AWS. Si crea una política basada en identidades que sea más restrictiva que el mínimo de permisos necesarios, la consola no funcionará del modo esperado para las entidades (usuarios o roles) que tengan esa política.

No es necesario que concedas permisos mínimos de consola a los usuarios que solo realicen llamadas a la API AWS CLI o a la AWS API. En su lugar, permite acceso únicamente a las acciones que coincidan con la operación de API que intentan realizar.

Para garantizar que los usuarios y los roles de IAM puedan seguir utilizando la AWS AppSync consola, vincule también la política `ReadOnly` AWS gestionada AWS AppSync `ConsoleAccess` o la política gestionada a las entidades. Para más información, consulte Adición de permisos a un usuario en la Guía del usuario de IAM:

Cómo permitir a los usuarios consultar sus propios permisos

En este ejemplo, se muestra cómo podría crear una política que permita a los usuarios de IAM ver las políticas administradas e insertadas que se asocian a la identidad de sus usuarios. Esta política incluye permisos para completar esta acción en la consola o mediante programación mediante la AWS CLI API o. 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": "*"
        }
```

```
    ]
}
```

Acceso a un bucket de Amazon S3

En este ejemplo, quiere conceder a un usuario de IAM de su AWS cuenta acceso a uno de sus buckets de Amazon S3,. `examplebucket` También desea permitir al usuario añadir, actualizar o eliminar objetos.

Además de conceder los permisos `s3:PutObject`, `s3:GetObject` y `s3:DeleteObject` al usuario, la política también concede los permisos `s3:ListAllMyBuckets`, `s3:GetBucketLocation` y `s3:ListBucket`. Estos son los permisos adicionales que requiere la consola. Las acciones `s3:PutObjectAcl` y `s3:GetObjectAcl` también son necesarias para poder copiar, cortar y pegar objetos en la consola. Para ver un tutorial de ejemplo en el que se conceden permisos a los usuarios y se prueban con la consola, consulte Tutorial de ejemplo: uso de las políticas del usuario para controlar el acceso al bucket.

```
{
   "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/*"
      }
   ]
}
```

Visualización de AWS AppSync *widgets* basados en etiquetas

Puede utilizar las condiciones de su política basada en la identidad para controlar el acceso a AWS AppSync los recursos en función de las etiquetas. En este ejemplo, se muestra cómo crear una política que permita visualizar un *widget*. Sin embargo, los permisos solo se conceden si el *widget* de la etiqueta `Owner` tiene el valor del nombre de usuario de dicho usuario. Esta política también proporciona los permisos necesarios para llevar a cabo esta acción en la consola.

```
{
    "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}"}
            }
        }
    ]
}
```

También puede asociar esta política al usuario de IAM en su cuenta. Si un usuario llamado `richard-roe` intenta ver un AWS AppSync *widget, el widget* debe estar etiquetado `Owner=richard-roe` o. `owner=richard-roe` De lo contrario, se le deniega el acceso. La clave

de la etiqueta de condición `0wner` coincide con los nombres de las claves de condición `0wner` y `owner` porque no distinguen entre mayúsculas y minúsculas. Para obtener más información, consulte Elementos de la política de JSON de IAM: Condición en la Guía del usuario de IAM.

AWS políticas gestionadas para AWS AppSync

Para añadir permisos a usuarios, grupos y roles, es más fácil usar políticas AWS administradas que escribirlas usted mismo. Se necesita tiempo y experiencia para crear políticas de IAM administradas por el cliente que proporcionen a su equipo solo los permisos necesarios. Para empezar rápidamente, puedes usar nuestras políticas AWS gestionadas. Estas políticas cubren casos de uso comunes y están disponibles en su Cuenta de AWS. Para obtener más información sobre las políticas AWS administradas, consulte las políticas AWS administradas en la Guía del usuario de IAM.

AWS los servicios mantienen y AWS actualizan las políticas gestionadas. No puede cambiar los permisos en las políticas AWS gestionadas. En ocasiones, los servicios añaden permisos adicionales a una política AWS gestionada para admitir nuevas funciones. Este tipo de actualización afecta a todas las identidades (usuarios, grupos y roles) donde se asocia la política. Lo más probable es que los servicios actualicen una política AWS administrada cuando se lanza una nueva función o cuando hay nuevas operaciones disponibles. Los servicios no eliminan los permisos de una política AWS administrada, por lo que las actualizaciones de la política no afectarán a los permisos existentes.

Además, AWS admite políticas administradas para funciones laborales que abarcan varios servicios. Por ejemplo, la política ReadOnlyAccess AWS gestionada proporciona acceso de solo lectura a todos los AWS servicios y recursos. Cuando un servicio lanza una nueva función, AWS agrega permisos de solo lectura para nuevas operaciones y recursos. Para obtener una lista y descripciones de las políticas de funciones de trabajo, consulte Políticas administradas de AWS para funciones de trabajo en la Guía del usuario de IAM.

AWS política gestionada: AWSAppSyncInvokeFullAccess

Utilice la política `AWSAppSyncInvokeFullAccess` AWS gestionada para permitir a los administradores acceder al AWS AppSync servicio a través de la consola o de forma independiente.

Puede adjuntar la política de `AWSAppSyncInvokeFullAccess` a las identidades de IAM.

Detalles de los permisos

Esta política incluye los siguientes permisos.

- `AWS AppSync`— Permite el acceso administrativo total a todos los recursos de AWS AppSync

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "appsync:GraphQL",
                "appsync:GetGraphqlApi",
                "appsync:ListGraphqlApis",
                "appsync:ListApiKeys"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS política gestionada: AWSAppSyncSchemaAuthor

Utilice la política `AWSAppSyncSchemaAuthor` AWS gestionada para permitir a los usuarios de IAM acceder para crear, actualizar y consultar sus esquemas de GraphQL. Para obtener información sobre lo que los usuarios pueden hacer con estos permisos, consulte Diseño de API de GraphQL.

Puede adjuntar la política de `AWSAppSyncSchemaAuthor` a las identidades de IAM.

Detalles de los permisos

Esta política incluye los siguientes permisos.

- `AWS AppSync`: permite las siguientes acciones.

- Crear esquemas de GraphQL
- Permitir crear, modificar y eliminar tipos, solucionadores y funciones de GraphQL
- Evaluar la lógica de las plantillas de solicitud y respuesta
- Evaluar código con un tiempo de ejecución y contexto
- Enviar consultas de GraphQL a las API de GraphQL
- Recuperar datos de 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 política gestionada: AWSAppSyncPushToCloudWatchLogs

AWS AppSync utiliza Amazon CloudWatch para supervisar el rendimiento de su aplicación mediante la generación de registros que puede utilizar para solucionar problemas y optimizar sus solicitudes de GraphQL. Para obtener más información, consulte Supervisión y registro.

Utilice la política `AWSAppSyncPushToCloudWatchLogs` AWS gestionada para poder enviar registros AWS AppSync a la cuenta de un usuario de IAM. CloudWatch

Puede adjuntar la política de `AWSAppSyncPushToCloudWatchLogs` a las identidades de IAM.

Detalles de los permisos

Esta política incluye los siguientes permisos.

- `CloudWatch Logs`— Permite AWS AppSync crear grupos de registros y flujos con nombres específicos. AWS AppSyncenvía los eventos de registro al flujo de registro especificado.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

AWS política gestionada: AWSAppSyncAdministrator

Utilice la política `AWSAppSyncAdministrator` AWS gestionada para permitir a los administradores acceder a todas las AWS AppSync aplicaciones excepto a la AWS consola.

Puede adjuntar `AWSAppSyncAdministrator` a sus entidades de IAM. AWS AppSync también asocia esta política a un rol de servicio que le permite realizar acciones en su nombre.

Detalles de los permisos

Esta política incluye los siguientes permisos.

- `AWS AppSync`— Permite el acceso administrativo total a todos los recursos de AWS AppSync
- `IAM`: permite las siguientes acciones.
  - Crear funciones vinculadas a los servicios AWS AppSync para poder analizar los recursos de otros servicios en su nombre
  - Eliminar roles vinculados a servicios
  - Transferir las funciones vinculadas al servicio a otros AWS servicios para que las asuman más adelante y realicen acciones en tu nombre

```
{
    "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 política gestionada: AWSAppSyncServiceRolePolicy

Use la política `AWSAppSyncServiceRolePolicy` AWS administrada para permitir el acceso a AWS los servicios y recursos que AWS AppSync usa o administra.

No puede asociar `AWSAppSyncServiceRolePolicy` a sus entidades IAM. Esta política está asociada a un rol vinculado al servicio que permite AWS AppSync realizar acciones en su nombre. Para obtener más información, consulte Funciones vinculadas al servicio para AWS AppSync.

Detalles de los permisos

Esta política incluye los siguientes permisos.

- `X-Ray`— se AWS AppSync utiliza AWS X-Ray para recopilar datos sobre las solicitudes realizadas en su solicitud. Para obtener más información, consulte Rastreo con AWS X-Ray.

  Esta política permite las acciones siguientes:

  - Recuperar las reglas de muestreo y sus resultados
  - Enviar datos de seguimiento al daemon X-Ray

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingTargets",
                "xray:GetSamplingRules",
                "xray:GetSamplingStatisticSummaries"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Actualizaciones de AWS AppSync en las políticas administradas de AWS

Consulta los detalles sobre las actualizaciones de las políticas AWS gestionadas AWS AppSync desde que este servicio comenzó a rastrear estos cambios. Para recibir alertas automáticas sobre los cambios en esta página, suscríbase a la fuente RSS de la página del historial del AWS AppSync documento.

Cambio	Descripción	Fecha
AWSAppSyncSchemaAuthor: actualización de una política existente	Se añadió una acción de política de `EvaluateCode` para permitir a los usuarios evaluar código con un tiempo de ejecución y contexto.	7 de febrero de 2023
AWSAppSyncSchemaAuthor: actualización de una política existente	Se añadieron acciones de políticas para permitir las funciones de lista, obtención , creación, actualización y eliminación en una API. Se añadió una acción política `EvaluateMappingTemplate` para permitir a los usuarios evaluar la lógica de la plantilla de asignación de solucionadores de solicitudes y respuestas. Se añadieron acciones políticas para permitir el etiquetado de recursos.	25 de agosto de 2022
AWS AppSync comenzó a rastrear los cambios	AWS AppSync comenzó a realizar un seguimiento de los cambios de sus políticas AWS gestionadas.	25 de agosto de 2022

Solución de problemas AWS AppSync de identidad y acceso

Utilice la siguiente información como ayuda para diagnosticar y solucionar los problemas habituales que pueden surgir al trabajar con un AWS AppSync IAM.

No estoy autorizado a realizar ninguna acción en AWS AppSync

Si AWS Management Console le indica que no está autorizado a realizar una acción, debe ponerse en contacto con su administrador para obtener ayuda. Su administrador es la persona que le facilitó su nombre de usuario y contraseña.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM `mateojackson` intenta utilizar la consola para consultar los detalles acerca de un recurso ficticio *`my-example-widget`*, pero no tiene los permisos ficticios `appsync:`*`GetWidget`*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 appsync:GetWidget on resource: my-example-widget
```

En este caso, Mateo pide a su administrador que actualice sus políticas de forma que pueda obtener acceso al recurso *`my-example-widget`* mediante la acción `appsync:`*`GetWidget`*.

No estoy autorizado a realizar lo siguiente: PassRole

Si recibes un mensaje de error que indica que no estás autorizado a realizar la `iam:PassRole` acción, debes actualizar tus políticas para que puedas transferirle AWS AppSync una función.

Algunas Servicios de AWS permiten transferir una función existente a ese servicio en lugar de crear una nueva función de servicio o una función vinculada al servicio. Para ello, debe tener permisos para transferir el rol al servicio.

El siguiente ejemplo de error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en ella. AWS AppSync Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su AWS administrador. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero ver mis claves de acceso

Después de crear sus claves de acceso de usuario de IAM, puede ver su ID de clave de acceso en cualquier momento. Sin embargo, no puede volver a ver su clave de acceso secreta. Si pierde la clave de acceso secreta, debe crear un nuevo par de claves de acceso.

Las claves de acceso se componen de dos partes: un ID de clave de acceso (por ejemplo, `AKIAIOSFODNN7EXAMPLE`) y una clave de acceso secreta (por ejemplo, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). El ID de clave de acceso y la clave de acceso secreta se utilizan juntos, como un nombre de usuario y contraseña, para autenticar sus solicitudes. Administre sus claves de acceso con el mismo nivel de seguridad que para el nombre de usuario y la contraseña.

> **Important**
>
> No proporcione las claves de acceso a terceros, ni siquiera para que lo ayuden a buscar el ID de usuario canónico. De este modo, podrías dar a alguien acceso permanente a tu Cuenta de AWS.

Cuando crea un par de claves de acceso, se le pide que guarde el ID de clave de acceso y la clave de acceso secreta en un lugar seguro. La clave de acceso secreta solo está disponible en el momento de su creación. Si pierde la clave de acceso secreta, debe agregar nuevas claves de acceso a su usuario de IAM. Puede tener un máximo de dos claves de acceso. Si ya cuenta con dos, debe eliminar un par de claves antes de crear una nueva. Para consultar las instrucciones, consulte Administración de claves de acceso en la Guía del usuario de IAM.

Soy administrador y quiero permitir que otras personas accedan AWS AppSync

Para permitir el acceso de otras personas AWS AppSync, debe crear una entidad de IAM (usuario o rol) para la persona o aplicación a la que necesita acceso. Esta persona utilizará las credenciales de la entidad para acceder a AWS. A continuación, debe adjuntar una política a la entidad que les conceda los permisos correctos. AWS AppSync

Para comenzar de inmediato, consulte Creación del primer grupo y usuario delegado de IAM en la Guía del usuario de IAM.

Quiero permitir que personas ajenas a mi AWS cuenta accedan a mis AWS AppSync recursos

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que admitan las políticas basadas en recursos o las listas de control de acceso (ACL), puede utilizar dichas políticas para conceder a las personas acceso a sus recursos.

Para más información, consulte lo siguiente:

- Para saber si AWS AppSync es compatible con estas funciones, consulte¿Cómo AWS AppSync funciona con IAM.
- Para obtener información sobre cómo proporcionar acceso a los recursos de su Cuentas de AWS propiedad, consulte Proporcionar acceso a un usuario de IAM en otro usuario de su propiedad Cuenta de AWS en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso a tus recursos a terceros Cuentas de AWS, consulta Cómo proporcionar acceso a recursos que Cuentas de AWS son propiedad de terceros en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulte Proporcionar acceso a usuarios autenticados externamente (identidad federada) en la Guía del usuario de IAM.
- Para obtener información sobre la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del usuario de IAM.

Registrar las llamadas a AWS AppSync la API con AWS CloudTrail

AWS AppSync está integrado con AWS CloudTrail un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o un AWS servicio en AWS AppSync. CloudTrail captura las llamadas a la API AWS AppSync como eventos. Las llamadas capturadas incluyen llamadas desde la AWS AppSync consola y llamadas en código a las operaciones de la AWS AppSync API. Si crea una ruta, puede habilitar la entrega continua de CloudTrail eventos a un bucket de Amazon S3, incluidos los eventos para AWS AppSync. Si no configura una ruta, podrá ver los eventos más recientes en la CloudTrail consola, en el historial de eventos. Con la información recopilada por usted

CloudTrail, puede determinar el destinatario de la solicitud AWS AppSync, la dirección IP desde la que se realizó la solicitud, quién la realizó, cuándo se realizó y detalles adicionales.

Para obtener más información CloudTrail, consulte la Guía AWS CloudTrail del usuario.

AWS AppSync información en CloudTrail

CloudTrail está habilitada en su AWS cuenta al crear la cuenta. Cuando se produce una actividad en AWS AppSync, esa actividad se registra en un CloudTrail evento junto con otros eventos de AWS servicio en el historial de eventos. Puedes ver, buscar y descargar los eventos recientes en tu AWS cuenta. Para obtener más información, consulte Visualización de eventos con el historial de CloudTrail eventos.

Para tener un registro continuo de los eventos de tu AWS cuenta, incluidos los eventos de tu cuenta AWS AppSync, crea una ruta. Un rastro permite CloudTrail entregar archivos de registro a un bucket de Amazon S3. De forma predeterminada, cuando crea una ruta en la consola, la ruta se aplica a todas AWS las regiones. La ruta registra los eventos de todas las regiones de la AWS partición y envía los archivos de registro al bucket de Amazon S3 que especifique. Además, puede configurar otros AWS servicios para analizar más a fondo los datos de eventos recopilados en los CloudTrail registros y actuar en función de ellos. Para más información, consulte los siguientes temas:

- Introducción a la creación de registros de seguimiento
- CloudTrail servicios e integraciones compatibles
- Configuración de las notificaciones de Amazon SNS para CloudTrail
- Recibir archivos de CloudTrail registro de varias regiones y recibir archivos de CloudTrail registro de varias cuentas

AWS AppSync admite el registro de las llamadas realizadas a través de la AWS AppSync API. En este momento, las llamadas a tus API y las llamadas realizadas a los solucionadores no se registran.
AWS AppSync CloudTrail

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario lo ayuda a determinar lo siguiente:

- Si la solicitud se realizó con credenciales de usuario root o AWS Identity and Access Management (IAM).
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.

- Si la solicitud la realizó otro AWS servicio.

Para obtener más información, consulte el elemento CloudTrail UserIdentity.

Descripción AWS AppSync de las entradas de los archivos de registro

Un rastro es una configuración que permite la entrega de eventos como archivos de registro a un bucket de Amazon S3 que usted especifique. CloudTrail Los archivos de registro contienen una o más entradas de registro. Un evento representa una solicitud única de cualquier fuente e incluye información sobre la acción solicitada, la fecha y la hora de la acción, los parámetros de la solicitud, etc. CloudTrail Los archivos de registro no son un registro ordenado de las llamadas a la API pública, por lo que no aparecen en ningún orden específico.

En el siguiente ejemplo, se muestra una entrada de CloudTrail registro que muestra la `GetGraphqlApi` acción realizada a través de la AWS AppSync consola:


```
{
    "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"
}
```

El siguiente ejemplo muestra una entrada de CloudTrail registro que demuestra la `CreateApikey` acción realizada mediante 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"
}
```

Prácticas recomendadas de seguridad para AWS AppSync

Asegurar AWS AppSync es más que simplemente activar algunas palancas o configurar el registro. En las secciones siguientes se analizan las prácticas recomendadas de seguridad, que varían según el uso que se haga del servicio.

Métodos de autenticación

AWS AppSync proporciona varias formas de autenticar a los usuarios en las API de GraphQL. Cada método tiene ventajas y desventajas en cuanto a seguridad, auditabilidad y usabilidad.

Están disponibles los siguientes métodos de autenticación comunes.

- Los grupos de usuarios de Amazon Cognito permiten a la API de GraphQL utilizar los atributos de usuario para el filtrado y el control de acceso detallados.
- Los tokens de la API tienen una vida útil limitada y son adecuados para sistemas automatizados, como los sistemas de integración continua y la integración con API externas.
- AWS Identity and Access Management (IAM) es adecuado para las aplicaciones internas administradas en su. Cuenta de AWS
- OpenID Connect permite controlar y federar el acceso con el protocolo OpenID Connect.

Para obtener más información sobre la autenticación y la autorización en AWS AppSync, consulteAutorización y autenticación.

Uso de TLS para solucionadores de HTTP

Al utilizar solucionadores de HTTP, asegúrese de usar conexiones protegidas mediante TLS (HTTPS) siempre que sea posible. Para obtener una lista completa de los certificados TLS en los que se AWS AppSync confía, consulteEntidades de certificación (CA) reconocidas por AWS AppSync para los puntos de conexión HTTPS.

Utilice roles con el menor número de permisos posible

Al utilizar solucionadores como el solucionador de DynamoDB, use roles que proporcionen la vista más restrictiva de sus recursos, como las tablas de Amazon DynamoDB.

Prácticas recomendadas sobre políticas de IAM

Las políticas basadas en la identidad determinan si alguien puede crear AWS AppSync recursos de tu cuenta, acceder a ellos o eliminarlos. Estas acciones pueden generar costos adicionales para su Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comience con las políticas AWS administradas y avance hacia los permisos con privilegios mínimos: para empezar a conceder permisos a sus usuarios y cargas de trabajo, utilice las políticas AWS administradas que otorgan permisos para muchos casos de uso comunes. Están disponibles en su. Cuenta de AWS Le recomendamos que reduzca aún más los permisos definiendo políticas administradas por el AWS cliente que sean específicas para sus casos de uso. Con el fin de obtener más información, consulte las políticas administradas por AWS o las políticas administradas por AWS para funciones de trabajo en la Guía de usuario de IAM.
- Aplique permisos de privilegio mínimo: cuando establezca permisos con políticas de IAM, conceda solo los permisos necesarios para realizar una tarea. Para ello, debe definir las acciones que se pueden llevar a cabo en determinados recursos en condiciones específicas, también conocidos como permisos de privilegios mínimos. Con el fin de obtener más información sobre el uso de IAM para aplicar permisos, consulte Políticas y permisos en IAM en la Guía del usuario de IAM.
- Utilice condiciones en las políticas de IAM para restringir aún más el acceso: puede agregar una condición a sus políticas para limitar el acceso a las acciones y los recursos. Por ejemplo, puede escribir una condición de políticas para especificar que todas las solicitudes deben enviarse utilizando SSL. También puedes usar condiciones para conceder el acceso a las acciones del servicio si se utilizan a través de una acción específica Servicio de AWS, por ejemplo AWS CloudFormation. Para obtener más información, consulte Elementos de la política de JSON de IAM: Condición en la Guía del usuario de IAM.

- Utilice el analizador de acceso de IAM para validar las políticas de IAM con el fin de garantizar la seguridad y funcionalidad de los permisos: el analizador de acceso de IAM valida políticas nuevas y existentes para que respeten el lenguaje (JSON) de las políticas de IAM y las prácticas recomendadas de IAM. El analizador de acceso de IAM proporciona más de 100 verificaciones de políticas y recomendaciones procesables para ayudar a crear políticas seguras y funcionales. Para más información, consulte Política de validación de Analizador de acceso de IAM en la Guía de usuario de IAM.
- Requerir autenticación multifactor (MFA): si tiene un escenario que requiere usuarios de IAM o un usuario raíz en Cuenta de AWS su cuenta, active la MFA para mayor seguridad. Para solicitar la MFA cuando se invocan las operaciones de la API, agregue las condiciones de la MFA a sus políticas. Para más información, consulte Configuración del acceso a una API protegido por MFA en la Guía de usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte las Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.

Referencia de solucionadores (JavaScript)

En las secciones siguientes se describen los solucionadores de JavaScript y la versión ejecutable `APPSYNC_JS`.

Temas

- Descripción general de los solucionadores de JavaScript
- Referencia al objeto del contexto del solucionador
- Características de la versión ejecutable de JavaScript para solucionadores y funciones
- Referencia a la función de solucionador de JavaScript para DynamoDB
- Referencia a la función de solucionador de JavaScript para OpenSearch
- JavaScript referencia de la función de resolución para Lambda
- JavaScript referencia de la función de resolución para la fuente EventBridge de datos
- Referencia a la función de solucionador de JavaScript para el origen de datos None
- JavaScript referencia de función de resolución para HTTP
- JavaScript referencia de la función de resolución para Amazon RDS

Descripción general de los solucionadores de JavaScript

Con AWS AppSync puede responder a solicitudes de GraphQL efectuando operaciones en sus orígenes de datos. Para cada campo de GraphQL en el que desee ejecutar una consulta, mutación o suscripción, se debe asociar un solucionador.

Los solucionadores son los conectores entre GraphQL y un origen de datos. Indican a AWS AppSync cómo convertir una solicitud de GraphQL entrante en instrucciones para el origen de datos de backend y cómo convertir la respuesta de ese origen de datos en una respuesta de GraphQL. Con AWS AppSync, puede escribir sus solucionadores mediante JavaScript y ejecutarlos en el entorno de AWS AppSync (`APPSYNC_JS`).

AWS AppSync permite escribir solucionadores de unidad o solucionadores de canalización que constan de varias funciones de AWS AppSync en una canalización.

Características de la versión ejecutable compatibles

La versión ejecutable de JavaScript de AWS AppSync proporciona un subconjunto de bibliotecas, utilidades y características. Para obtener una lista completa de las características y funcionalidades compatibles con la versión ejecutable `APPSYNC_JS`, consulte el artículo sobre las características de la versión ejecutable de JavaScript para solucionadores y funciones.

Solucionadores de unidad

Un solucionador de unidad se compone de código que define un controlador de solicitudes y respuestas que se ejecutan con un origen de datos. El controlador de solicitudes toma un objeto de contexto como argumento y devuelve la carga de la solicitud utilizada para llamar al origen de datos. El controlador de respuestas recibe una carga útil del origen de datos con el resultado de la solicitud ejecutada. El controlador de respuestas transforma la carga útil en una respuesta de GraphQL para resolver el campo de GraphQL. En el ejemplo siguiente, un solucionador recupera un elemento de un origen de datos de 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;
```

Anatomía de un solucionador de canalización de JavaScript

Un solucionador de canalización se compone de un código que define un controlador de solicitudes y respuestas y una lista de funciones. Cada función tiene un controlador de solicitudes y respuestas que ejecuta con un origen de datos. Puesto que un solucionador de canalización delega las ejecuciones a una lista de funciones, no está vinculado a ningún origen de datos. Las funciones y los solucionadores de unidad son primitivos que ejecutan la operación frente a los orígenes de datos.

Controlador de solicitudes del solucionador de canalización

El controlador de solicitudes de un solucionador de canalización (el paso Before (Antes)) permite crear alguna lógica de preparación antes de ejecutar las funciones definidas.

Lista de funciones

La lista de funciones de un solucionador de canalización se ejecutará de forma secuencial. El resultado de la evaluación del controlador de solicitudes del solucionador de canalización estará disponible para la primera función como `ctx.prev.result`. El resultado de la evaluación de cada función está disponible para la siguiente función como `ctx.prev.result`.

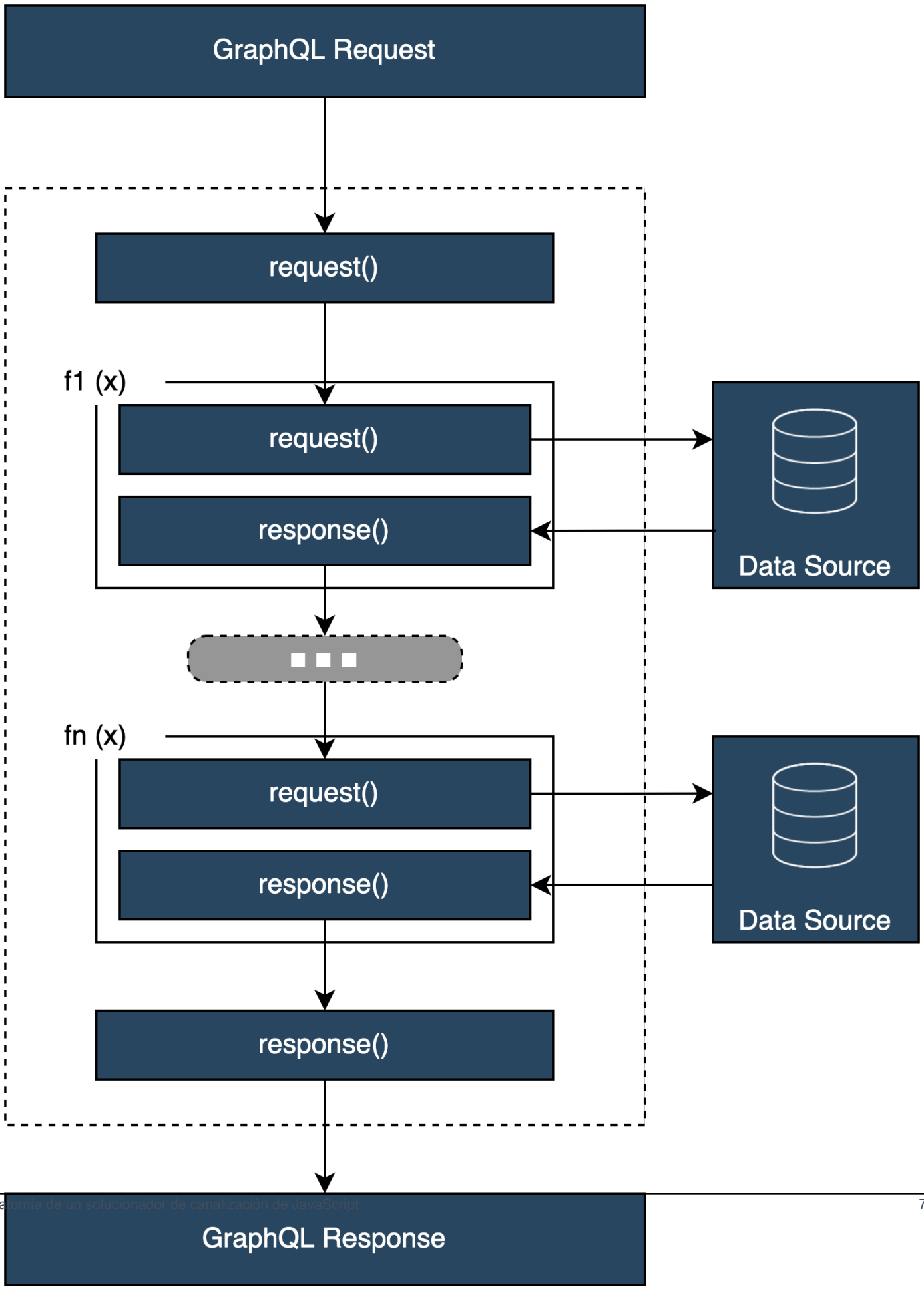
Controlador de respuestas del solucionador de canalización

El controlador de respuestas de un solucionador de canalización permite realizar alguna lógica final a partir del resultado de la última función al tipo de campo GraphQL esperado. El resultado de la última función en lista de funciones está disponible en el controlador de respuestas del solucionador de canalización como `ctx.prev.result` o `ctx.result`.

Flujo de la ejecución

Con un solucionador de canalización compuesto por dos funciones, la lista siguiente representa el flujo de ejecución cuando se invoca el solucionador:

1. Controlador de solicitudes del solucionador de canalización
2. Función 1: Controlador de solicitudes de función
3. Función 1: Invocación de origen de datos
4. Función 1: Controlador de respuestas de función
5. Función 2: Controlador de solicitudes de función
6. Función 2: Invocación de origen de datos
7. Función 2: Controlador de respuestas de función
8. Controlador de respuestas del solucionador de canalización


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

Utilidades integradas de tiempo de ejecución **APPSYNC_JS** prácticas

Las siguientes utilidades pueden ayudar cuando se trabaja con solucionadores de canalización.

ctx.stash

El stash es un objeto que está disponible dentro de cada solucionador y controlador de solicitudes y respuestas de función. La misma instancia stash vive en una única ejecución de solucionador. Esto significa que puede utilizar el stash para pasar datos arbitrarios a controladores de solicitudes y respuestas, así como a las funciones de un solucionador de canalización. Puede probar el stash como un objeto de JavaScript normal.

ctx.prev.result

El `ctx.prev.result` representa el resultado de la operación anterior que se ha ejecutado en la canalización. Si la operación anterior era el controlador de solicitudes del solucionador de canalización, entonces `ctx.prev.result` estará disponible para la primera función de la cadena. Si la operación anterior era la primera función, entonces `ctx.prev.result` representa el resultado de la primera función y estará disponible para la segunda función de la canalización. Si la operación anterior era la última función, entonces `ctx.prev.result` representa el resultado de la última función y estará disponible para el controlador de respuestas del solucionador de canalización.

util.error

La utilidad `util.error` es útil para lanzar un error de campo. El uso de `util.error` dentro de un controlador de solicitudes o respuestas de función genera un error de campo inmediatamente, lo que impide que se ejecuten funciones posteriores. Para obtener información más detallada y otras firmas de `util.error`, visite el artículo sobre las características de la versión ejecutable de JavaScript para solucionadores y funciones.

util.appendError

El `util.appendError` es similar a `util.error()`, siendo la principal diferencia que no interrumpe la evaluación del controlador. En su lugar, indica que hubo un error con el campo, pero permite evaluar el controlador y, por tanto, devolver los datos. El uso de `util.appendError` dentro de una función no interrumpe el flujo de ejecución de la canalización. Para obtener información más detallada y otras firmas de `util.error`, visite el artículo sobre las características de la versión ejecutable de JavaScript para solucionadores y funciones.

runtime.earlyReturn

La función `runtime.earlyReturn` permite volver de forma prematura de cualquier función de solicitud. Al utilizar `runtime.earlyReturn` dentro de un controlador de solicitudes del solucionador volverá desde el solucionador. Si se llama desde un controlador de solicitudes de función de AWS AppSync volverá desde la función y continuará la ejecución a la siguiente función del controlador de respuestas del solucionador o canalización.

Escritura de solucionadores de canalización

Un solucionador de canalización también tiene un controlador de solicitudes y otro de respuestas relacionados con la ejecución de las funciones en la canalización: su controlador de solicitudes se ejecuta antes de la solicitud de la primera función y su controlador de respuestas lo hace después de la respuesta de la última función. El controlador de solicitudes del solucionador puede configurar los datos para que las funciones de la canalización los utilicen. El controlador de respuestas del solucionador es responsable de devolver los datos que se mapean al tipo de salida del campo GraphQL. En el siguiente ejemplo, un controlador de solicitudes del solucionador define `allowedGroups`; los datos devueltos deben pertenecer a uno de estos grupos. Las funciones del solucionador pueden utilizar este valor para solicitar datos. El controlador de respuestas del solucionador realiza una comprobación final y filtra el resultado para asegurarse de que solo se devuelvan los elementos pertenecientes a los grupos permitidos.

```
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;
}
```

Escritura de las funciones de AWS AppSync

Las funciones de AWS AppSync permiten escribir lógica común que puede reutilizar en varios solucionadores de su esquema. Por ejemplo, puede tener una función de AWS AppSync llamada `QUERY_ITEMS` responsable de consultar los elementos de un origen de datos de Amazon DynamoDB. En el caso de los solucionadores con los que desea consultar elementos, solo tiene que añadir la función a la canalización del solucionador y proporcionar el índice de consulta que se va a usar. No es necesario volver a implementar la lógica.

Escritura de código

Supongamos que desea asociar un solucionador de canalización a un campo llamado `getPost(id:ID!)` que devuelve un tipo `Post` de un origen de datos de Amazon DynamoDB con la consulta de GraphQL siguiente:

```
getPost(id:1){
    id
    title
    content
}
```

En primer lugar, adjunte un solucionador sencillo a `Query.getPost` con el siguiente código. Este es un ejemplo de código de solucionador sencillo. No hay ninguna lógica definida en el controlador de solicitudes, y el controlador de respuestas simplemente devuelve el resultado de la última función.

```
/**
 * 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
}
```

A continuación, defina la función `GET_ITEM` que recupera un elemento postitem del origen de datos:

```
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
}
```

Si se produce un error durante la solicitud, el controlador de respuestas de la función agrega un error que se devolverá al cliente que realiza la llamada en la respuesta de GraphQL. Añada la

función `GET_ITEM` a su lista de funciones de solucionador. Al ejecutar la consulta, el controlador de solicitudes de la función `GET_ITEM` utiliza las utilidades proporcionadas en el módulo de DynamoDB de AWS AppSync para crear una solicitud `DynamoDBGetItem` con el `id` como clave. `ddb.get({ key: { id } })` genera la operación `GetItem` adecuada:

```
{
    "operation" : "GetItem",
    "key" : {
        "id" : { "S" : "1" }
    }
}
```

AWS AppSync utiliza la solicitud para recuperar datos de Amazon DynamoDB. Una vez devueltos los datos, los administra el controlador de respuestas de la función `GET_ITEM`, que comprueba si hay errores y, a continuación, devuelve el resultado.

```
{
  "result" : {
    "id": 1,
    "title": "hello world",
    "content": "<long story>"
  }
}
```

Por último, el controlador de respuestas del solucionador devuelve el resultado directamente.

Solución de errores

Si se produce un error en su función durante una solicitud, el error estará disponible en el controlador de respuestas de función en `ctx.error`. Puede agregar el error a su respuesta de GraphQL mediante la utilidad `util.appendError`. Puede hacer que el error esté disponible para otras funciones en la canalización con el stash. Vea el ejemplo siguiente:

```
/**
 * 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;
}
```

Utilidades

AWS AppSync proporciona dos bibliotecas que contribuyen al desarrollo de solucionadores con la versión ejecutable `APPSYNC_JS`:

- `@aws-appsync/eslint-plugin`: identifica y soluciona problemas rápidamente durante el desarrollo.
- `@aws-appsync/utils`: proporciona la validación de tipos y la función de autocompletar en editores de código.

Configuración del complemento de eslint

ESLint es una herramienta que analiza estáticamente su código para encontrar problemas de forma rápida. Puede ejecutar ESLint como parte de su canalización de integración continua. `@aws-appsync/eslint-plugin` es un complemento de ESLint que detecta la sintaxis no válida en su código al aprovechar la versión ejecutable `APPSYNC_JS`. El complemento permite recibir comentarios sobre el código de forma rápida durante el desarrollo sin tener que enviar los cambios a la nube.

`@aws-appsync/eslint-plugin` proporciona dos conjuntos de reglas que puede usar durante el desarrollo.

"plugin:@aws-appsync/base" configura un conjunto básico de reglas que puede aprovechar en su proyecto:

Regla	Descripción
no-async	No se admiten promesas ni procesos asíncronos.
no-await	No se admiten promesas ni procesos asíncronos.

Regla	Descripción
no-classes	No se admiten clases.
no-for	No se admite `for` (excepto para `for-in` y `for-of`, que sí se admiten)
no-continue	No se admite `continue`.
no-generators	No se admiten generadores.
no-yield	No se admite `yield`.
no-labels	No se admiten etiquetas.
no-this	No se admite la palabra clave `this`.
no-try	No se admite la estructura try/catch.
no-while	No se admiten los bucles WHILE.
no-disallowed-unary-operators	No se permiten los operadores unarios `++`, `--` y `~`.
no-disallowed-binary-operators	No se permite el operador `instanceof` .
no-promise	No se admiten promesas ni procesos asíncronos.

"plugin:@aws-appsync/recommended" proporciona algunas reglas adicionales, pero también requiere que añada configuraciones de TypeScript a su proyecto.

Regla	Descripción
no-recursion	No se permiten llamadas a funciones recursivas.

Regla	Descripción
no-disallowed-methods	No se permiten algunos métodos. Consulte la referencia para obtener un conjunto completo de funciones integradas compatibles.
no-function-passing	No se permite pasar funciones como argumentos de la función a funciones.
no-function-reassign	No se pueden reasignar funciones.
no-function-return	Las funciones no pueden ser el valor devuelto de las funciones.

Para añadir el complemento a su proyecto, siga los pasos de instalación y uso que se indican en Introducción a ESLint. A continuación, instale el complemento en su proyecto con el administrador de paquetes del proyecto (por ejemplo, npm, yarn o pnpm):

```
$ npm install @aws-appsync/eslint-plugin
```

En el archivo `.eslintrc.{js,yml,json}`, añada "plugin:@aws-appsync/base" o "plugin:@aws-appsync/recommended" a la propiedad `extends`. El siguiente fragmento de código es un ejemplo de configuración `.eslintrc` básica para JavaScript:

```
{
  "extends": ["plugin:@aws-appsync/base"]
}
```

Para usar el conjunto de reglas "plugin:@aws-appsync/recommended", instale la dependencia requerida:

```
$ npm install -D @typescript-eslint/parser
```

A continuación, cree un archivo `.eslintrc.js`:

```
{
  "parser": "@typescript-eslint/parser",
  "parserOptions": {
```

```
    "ecmaVersion": 2018,
    "project": "./tsconfig.json"
  },
  "extends": ["plugin:@aws-appsync/recommended"]
}
```

Agrupación, TypeScript y mapas de origen

Uso de bibliotecas y agrupación del código

En su código de solucionador y función, puede utilizar bibliotecas personalizadas y externas siempre que cumplan los requisitos de `APPSYNC_JS`. Esto permite reutilizar el código existente en su aplicación. Para utilizar bibliotecas definidas por varios archivos, debe usar una herramienta de agrupación, como esbuild, para combinar el código en un solo archivo que, a continuación, se puede guardar en su solucionador o función de AWS AppSync.

Al agrupar el código, tenga en cuenta lo siguiente:

- `APPSYNC_JS` solo admite módulos ECMAScript (ESM).
- Los módulos `@aws-appsync/*` están integrados en `APPSYNC_JS` y no deben agruparse con su código.
- El entorno de versión ejecutable `APPSYNC_JS` es similar a NodeJS en que el código no se ejecuta en un entorno del navegador.
- Puede incluir un mapa de origen opcional. Sin embargo, no incluya el contenido de origen.

  Para obtener más información sobre los mapas de origen, consulte Uso de mapas de origen.

Por ejemplo, para agrupar el código de solucionador ubicado en `src/appsync/getPost.resolver.js`, puede usar el siguiente comando de la CLI esbuild:

```
$ 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
```

Creación del código y trabajo con TypeScript

TypeScript es un lenguaje de programación desarrollado por Microsoft que ofrece todas las características de JavaScript junto con el sistema de escritura de TypeScript. Puede usar TypeScript para escribir código con seguridad de tipos y detectar errores en el momento de la compilación antes de guardar el código en AWS AppSync. El paquete `@aws-appsync/utils` está completamente escrito.

La versión ejecutable `APPSYNC_JS` no admite TypeScript directamente. Primero debe transpilar el código de TypeScript a código JavaScript que la versión ejecutable `APPSYNC_JS` admita antes de guardar el código en AWS AppSync. Puede usar TypeScript para escribir el código en su entorno de desarrollo integrado (IDE) local, pero tenga en cuenta que no puede crear código de TypeScript en la consola AWS AppSync.

Para empezar, asegúrese de tener TypeScript instalado en su proyecto. A continuación, establezca la configuración de transcompilación de TypeScript para trabajar con la versión ejecutable `APPSYNC_JS` mediante TSConfig. Este es un ejemplo de un archivo `tsconfig.json` básico que puede usar:

```
// tsconfig.json
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
   "noEmit": true,
   "moduleResolution": "node",
  }
}
```

A continuación, puede usar una herramienta de agrupación como esbuild para compilar y agrupar el código. Por ejemplo, en el caso de un proyecto con su código AWS AppSync ubicado en `src/appsync`, puede usar el siguiente comando para compilar y agrupar el código:

```
$ esbuild --bundle \
--sourcemap=inline \
--sources-content=false \
--target=esnext \
--platform=node \
--format=esm \
--external:@aws-appsync/utils \
--outdir=out/appsync \
```

```
  src/appsync/**/*.ts
```

Uso de CodeGen de Amplify

Puede usar la CLI de Amplify a fin de generar los tipos para su esquema. En el directorio donde se encuentra su archivo `schema.graphql`, ejecute el siguiente comando y revise las instrucciones para configurar el CodeGen:

```
$  npx @aws-amplify/cli codegen add
```

Para regenerar el CodeGen en determinadas circunstancias (por ejemplo, cuando se actualiza el esquema), ejecute el siguiente comando:

```
$ npx @aws-amplify/cli codegen
```

A continuación, puede usar los tipos generados en el código de solucionador. Por ejemplo, en el caso del esquema siguiente:

```
type Todo {
  id: ID!
  title: String!
  description: String
}

type Mutation {
  createTodo(title: String!, description: String): Todo
}

type Query {
  listTodos: Todo
}
```

Podría utilizar los tipos generados en la siguiente función de AWS AppSync de ejemplo:

```
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
}
```

Uso de genéricos en TypeScript

Puede usar genéricos con varios de los tipos proporcionados. Por ejemplo, el siguiente fragmento de código es un tipo `Todo`:

```
export type Todo = {
  __typename: "Todo",
  id: string,
  title: string,
  description?: string | null,
};
```

Puede escribir un solucionador para una suscripción que haga uso de `Todo`. En su IDE, las definiciones de tipo y las sugerencias de autocompletar servirán como guía para utilizar correctamente la utilidad de transformación `toSubscriptionFilter`:

```
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
}
```

Análisis de sus paquetes

Puede analizar sus paquetes automáticamente mediante la importación del complemento `esbuild-plugin-eslint`. Después, puede habilitarlo proporcionando un valor `plugins` que habilite

las capacidades de eslint. A continuación se muestra un fragmento de código que usa la API de JavaScript esbuild en un archivo llamado `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 })],
})
```

Uso de mapas de origen

Puede proporcionar un mapa de origen en línea (`sourcemap`) con su código JavaScript. Los mapas de origen son útiles al agrupar código JavaScript o de TypeScript y cuando desea ver referencias a los archivos de origen de entrada en los registros y mensajes de error de JavaScript de la versión ejecutable.

Su `sourcemap` debe aparecer al final del código. Se define mediante una única línea de comentario que sigue el siguiente formato:

```
//# sourceMappingURL=data:application/json;base64,<base64 encoded string>
```

A continuación se muestra un ejemplo:

```
//# sourceMappingURL=data:application/
json;base64,ewogICJ2ZXJzaW9uIjogMywKICAic291cmNlcyI6IFsibGliLmpzIiwgImNvZGUuanMiXSwKICAibWFwcGl
```

Los mapas de origen se pueden crear con esbuild. En el siguiente ejemplo se muestra cómo usar la API de JavaScript esbuild para incluir un mapa de origen en línea cuando al compilarse y agruparse el código:

```
/* 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 })],
})
```

En concreto, las opciones `sourcemap` y `sourcesContent` especifican que se debe añadir un mapa de origen en línea al final de cada compilación, pero no debe incluir el contenido de origen. Por convención, se recomienda no incluir el contenido de origen en el `sourcemap`. Establezca `sources-content` en `false` para deshabilitarlo en esbuild.

Para ilustrar el funcionamiento de los mapas de origen, revise el siguiente ejemplo, en el que un código de solucionador hace referencia a funciones auxiliares de una biblioteca auxiliar. El código contiene instrucciones de registro en el código de solucionador y en la biblioteca auxiliar:

./src/default.resolver.ts (su solucionador)

```
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 (un archivo auxiliar)

```
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..')
}
```

Al compilar y agrupar el archivo de resolución, el código de solucionador incluirá un mapa de origen en línea. Al ejecutarse el solucionador, aparecen las siguientes entradas en los registros de CloudWatch:

Si observa las entradas en el registro de CloudWatch, verá que se han agrupado las funcionalidades de los dos archivos y se ejecutan simultáneamente. El nombre original de cada archivo también aparece claramente reflejado en los registros.

Pruebas

Puede usar el comando de la API de `EvaluateCode` para probar de forma remota su solucionador y sus controladores de función con datos simulados antes de guardar el código en un solucionador o una función. Para comenzar a utilizar el comando, asegúrese de haber añadido el permiso `appsync:evaluatecode` a su política. Por ejemplo:

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "appsync:evaluateCode",
            "Resource": "arn:aws:appsync:<region>:<account>:*"
        }
    ]
}
```

Puede utilizar el comando mediante la AWSCLI o los AWSSDK. Por ejemplo, para probar el código con la CLI, basta con apuntar al archivo, proporcionar un contexto y especificar el controlador que desea evaluar:

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

La respuesta contiene un `evaluationResult` que incluye la carga útil devuelta por el controlador. También contiene un objeto `logs` que incluye la lista de registros generados por el controlador durante la evaluación. Esto facilita la depuración de la ejecución de código y la consulta de información sobre la evaluación como ayuda para solucionar problemas. Por ejemplo:

```
{
    "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\""
    ]
}
```

El resultado de la evaluación se puede analizar como JSON, lo que da como resultado:

```
{
  "operation": "PutItem",
  "key": {
    "id": {
      "S": "record-id"
    }
```

```
  },
  "attributeValues": {
    "owner": {
      "S": "John doe"
    },
    "expectedVersion": {
      "N": 2
    },
    "authorId": {
      "S": "Sammy Davis"
    }
  }
}
```

Cuando se utiliza el SDK, puede incorporar fácilmente pruebas de su conjunto de pruebas para validar el comportamiento del código. En nuestro ejemplo aquí mostrado, se utiliza el marco de pruebas de Jest, pero cualquier conjunto de pruebas funciona. En el siguiente fragmento de código se muestra una ejecución de validación hipotética. Tenga en cuenta que esperamos que la respuesta de la evaluación sea un JSON válido, por lo que utilizamos `JSON.parse` para recuperar el JSON de la respuesta de cadena:

```
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)
})
```

Esto produce el siguiente resultado:

```
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
```

Migración de VTL a JavaScript

AWS AppSync permite escribir la lógica empresarial para los solucionadores y las funciones mediante VTL o JavaScript. Con ambos lenguajes, escribe una lógica que indica al servicio AWS AppSync cómo interactuar con los orígenes de datos. Con VTL, escribe plantillas de mapeo que deben evaluarse como una cadena codificada en JSON válida. Con JavaScript, escribe controladores de solicitudes y respuestas que devuelven objetos. No devuelve una cadena codificada en JSON.

Por ejemplo, tome la siguiente plantilla de mapeo VTL para obtener un elemento de Amazon DynamoDB:

```
{
    "operation": "GetItem",
    "key": {
        "id": $util.dynamodb.toDynamoDBJson($ctx.args.id),
    }
}
```

La utilidad `$util.dynamodb.toDynamoDBJson` devuelve una cadena codificada en JSON. Si `$ctx.args.id` se establece en `<id>`, la plantilla se evalúa como una cadena codificada en JSON válida:

```
{
    "operation": "GetItem",
    "key": {
        "id": {"S": "<id>"},
    }
}
```

Al trabajar con JavaScript, no es necesario que imprima cadenas codificadas en JSON sin procesar dentro del código, ni tampoco que utilice una utilidad como `toDynamoDBJson`. Un ejemplo equivalente de la plantilla de mapeo anterior es:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'GetItem',
    key: {id: util.dynamodb.toDynamoDB(ctx.args.id)}
  };
}
```

Una alternativa es utilizar `util.dynamodb.toMapValues`, que es el enfoque recomendado para gestionar un objeto de valores:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'GetItem',
    key: util.dynamodb.toMapValues({ id: ctx.args.id }),
  };
}
```

Esto se evalúa como:

```
{
  "operation": "GetItem",
  "key": {
    "id": {
      "S": "<id>"
    }
  }
}
```

Note

Recomendamos utilizar el módulo de DynamoDB con orígenes de datos de DynamoDB:

```
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
```

```
 ddb.get({ key: { id: ctx.args.id } })
}
```

Como otro ejemplo, tome la siguiente plantilla de mapeo para colocar un elemento en un origen de datos de Amazon DynamoDB:

```
{
    "operation" : "PutItem",
    "key" : {
        "id": $util.dynamodb.toDynamoDBJson($util.autoId()),
    },
    "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
}
```

Durante su evaluación, esta cadena de plantilla de mapeo debe producir una cadena codificada en JSON válida. Al usar JavaScript, su código devuelve el objeto de solicitud directamente:

```
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),
  };
}
```

que se evalúa como:

```
{
  "operation": "PutItem",
  "key": {
    "id": { "S": "2bff3f05-ff8c-4ed8-92b4-767e29fc4e63" }
  },
  "attributeValues": {
    "firstname": { "S": "Shaggy" },
    "age": { "N": 4 }
  }
}
```

> **Note**
>
> Recomendamos utilizar el módulo de DynamoDB con orígenes de datos de 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 })
}
```

Elección entre acceso directo a los orígenes de datos y proxies a través de un origen de datos de Lambda

Con AWS AppSync y la versión ejecutable `APPSYNC_JS`, puede escribir su propio código que implemente su lógica empresarial personalizada mediante funciones de AWS AppSync para acceder a sus orígenes de datos. Esto facilita su interacción directa con orígenes de datos como Amazon DynamoDB, Aurora sin servidor, OpenSearch Service, API HTTP y otros servicios de AWS sin tener que implementar infraestructura ni servicios computacionales adicionales. AWS AppSync también facilita la interacción con una función AWS Lambda mediante la configuración de un origen de datos de Lambda. Los orígenes de datos de Lambda permiten ejecutar una lógica empresarial compleja mediante el conjunto completo de capacidades de AWS Lambda para resolver una solicitud de GraphQL. En la mayoría de los casos, una función de AWS AppSync conectada directamente a su origen de datos de destino proporcionará toda la funcionalidad que necesita. En situaciones en las que necesita implementar una lógica empresarial compleja incompatible con la versión ejecutable `APPSYNC_JS`, puede utilizar un origen de datos de Lambda como proxy para interactuar con el origen de datos de destino.

	Integración directa de origen de datos	Origen de datos de Lambda como proxy
Caso de uso	AWS AppSync functions interact directly with API data sources.	AWS AppSync functions call Lambdas that interact with API data sources.

Runtime	*APPSYNC_JS* (JavaScript)	Cualquier versión ejecutable de Lambda compatible
Maximum size of code	32.000 caracteres por función de AWS AppSync	50 MB (comprimido, para carga directa) por Lambda
External modules	Limitada (solo características compatibles con APPSYNC_JS)	Sí
Call any AWS service	Sí (uso de origen de datos HTTP de AWS AppSync)	Sí (uso de SDK de AWS)
Access to the request header	Sí	Sí
Network access	No	Sí
File system access	No	Sí
Logging and metrics	Sí	Sí
Build and test entirely within AppSync	Sí	No
Cold start	No	No (con simultaneidad aprovisionada)
Auto-scaling	Sí (AWS AppSync de forma transparente)	Sí (según se ha configurado en Lambda)
Pricing	Sin cargo adicional.	Cobro por el uso de Lambda

Las funciones de AWS AppSync que se integran directamente con su origen de datos de destino son ideales para casos de uso como los siguientes:

- Interacción con Amazon DynamoDB, Aurora sin servidor y OpenSearch Service
- Interacción con API HTTP y transferencia de encabezados entrantes
- Interacción con servicios de AWS mediante orígenes de datos HTTP (con firma de solicitudes por parte de AWS AppSync con el rol del origen de datos proporcionado)

- Implementación del control de acceso antes de acceder a orígenes de datos
- Implementación del filtrado de datos recuperados antes de cumplir una solicitud
- Implementación de orquestación sencilla con ejecución secuencial de funciones de AWS AppSync en una canalización del solucionador
- Control de conexiones de suscripción y almacenamiento en caché en consultas y mutaciones.

Las funciones de AWS AppSync que utilizan un origen de datos de Lambda como proxy son ideales para casos de uso como los siguientes:

- Uso de un lenguaje distinto de JavaScript o Velocity Template Language (VTL)
- Ajuste y control de la CPU o memoria para optimizar el rendimiento
- Importación de bibliotecas de terceros o solicitud de características no compatibles en `APPSYNC_JS`
- Realización de varias solicitudes de red u obtención de acceso al sistema de archivos para cumplir una consulta
- Procesamiento por lotes de solicitudes mediante la configuración de procesamiento por lotes.

Referencia al objeto del contexto del solucionador

AWS AppSync define un conjunto de variables y funciones para trabajar con controladores de solicitudes y respuestas. Esto simplifica las operaciones lógicas realizadas en los datos en GraphQL. En este documento se describen estas funciones y se proporcionan ejemplos.

Acceso a **`context`**

El argumento `context` de un controlador de solicitudes y respuestas es un objeto que contiene toda la información contextual para la invocación del solucionador. Tiene la estructura siguiente:

```
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**
>
> A menudo verá que se hace referencia al objeto `context` como `ctx`.

Cada campo del mapa `context` se define de la siguiente manera:

Campos **`context`**

`arguments`

Un mapa que contiene todos los argumentos de GraphQL de este campo.

`identity`

Un objeto que contiene información sobre el intermediario. Consulte Identidad para obtener más información acerca de la estructura de este campo.

`source`

Un mapa que contiene la resolución del campo principal.

`stash`

El stash es un objeto que está disponible dentro de cada solucionador y controlador de funciones. Una misma instancia stash perdura durante una única ejecución de solucionador. Esto significa que puede utilizar el stash para pasar datos arbitrarios a controladores de solicitudes y respuestas, así como a las funciones de un solucionador de canalización.

> **Note**
>
> No puede eliminar ni reemplazar todo el stash, pero sí puede agregar, actualizar, eliminar y leer sus propiedades.

Puede agregar elementos al stash modificando uno de los siguientes ejemplos de código:

```
//Example 1
ctx.stash.newItem = { key: "something" }

//Example 2
Object.assign(ctx.stash, {key1: value1, key2: value})
```

Puedes eliminar elementos del stash modificando el siguiente código:

```
delete ctx.stash.key
```

`result`

Un contenedor para los resultados de este solucionador. Este campo solo está disponible para los controladores de respuestas.

Por ejemplo, al solucionar el campo `author` de la siguiente consulta:

```
query {
    getPost(id: 1234) {
        postId
        title
        content
        author {
            id
            name
        }
    }
}
```

A continuación, la variable completa `context` está disponible cuando se evalúa un controlador de respuestas:

```
{
  "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`

Es el resultado de cualquier operación previa que se haya ejecutado en un solucionador de canalización.

Si la operación anterior era el controlador de solicitudes del solucionador de canalización, `ctx.prev.result` representa el resultado de la evaluación y estará disponible para la primera función de la canalización.

Si la operación anterior era la primera función, `ctx.prev.result` representa el resultado de la evaluación del primer controlador de respuesta de la función, y estará disponible para la segunda función de la canalización.

Si la operación anterior era la última función, `ctx.prev.result` representa el resultado de la evaluación de la última función, y estará disponible para el controlador de respuestas del solucionador de canalización.

`info`

Un objeto que contiene información sobre la solicitud de GraphQL. Para obtener información sobre la estructura de este campo, consulte Información.

Identidad

La sección `identity` contiene información sobre el intermediario. La forma de esta sección depende del tipo de autorización de su API de AWS AppSync.

Para obtener más información sobre las opciones de seguridad de AWS AppSync, consulte Autorización y autenticación.

Autorización de **API_KEY**

El campo `identity` no se rellena.

Autorización de **AWS_LAMBDA**

La `identity` tiene el siguiente formato:

```
type AppSyncIdentityLambda = {
  resolverContext: any;
};
```

La `identity` contiene la clave `resolverContext`, que contiene el mismo contenido `resolverContext` devuelto por la función de Lambda que autoriza la solicitud.

Autorización de **AWS_IAM**

La `identity` tiene el siguiente formato:

```
type AppSyncIdentityIAM = {
  accountId: string;
  cognitoIdentityPoolId: string;
  cognitoIdentityId: string;
  sourceIp: string[];
  username: string;
  userArn: string;
  cognitoIdentityAuthType: string;
  cognitoIdentityAuthProvider: string;
};
```

Autorización de **AMAZON_COGNITO_USER_POOLS**

La `identity` tiene el siguiente formato:

```
type AppSyncIdentityCognito = {
  sourceIp: string[];
  username: string;
  groups: string[] | null;
  sub: string;
  issuer: string;
  claims: any;
  defaultAuthStrategy: string;
};
```

Cada campo se define de la siguiente manera:

`accountId`

El ID de la cuenta de AWS del intermediario.

`claims`

Las notificaciones que tiene el usuario.

`cognitoIdentityAuthType`

Autenticado o no según el tipo de identidad.

`cognitoIdentityAuthProvider`

Una lista separada por comas de la información del proveedor de identidad externo utilizada para obtener las credenciales que se usan para firmar la solicitud.

`cognitoIdentityId`

El ID de identidad de Amazon Cognito del intermediario.

`cognitoIdentityPoolId`

El ID de grupo de identidades de Amazon Cognito asociado al intermediario.

`defaultAuthStrategy`

La estrategia de autorización predeterminada para este intermediario (`ALLOW` o `DENY`).

`issuer`

El emisor del token.

`sourceIp`

La dirección IP de origen de la persona que realiza la llamada que AWS AppSync recibe. Si la solicitud no incluye el encabezado `x-forwarded-for`, el valor de IP de origen solo contiene una dirección IP de la conexión TCP. Si la solicitud incluye un encabezado `x-forwarded-for`, la IP de origen será una lista de las direcciones IP del encabezado `x-forwarded-for`, además de la dirección IP de la conexión TCP.

`sub`

El UUID del usuario autenticado.

`user`

El usuario de IAM.

`userArn`

Es el nombre de recurso de Amazon (ARN) del usuario de IAM.

`username`

El nombre de usuario del usuario autenticado. En el caso de una autorización `AMAZON_COGNITO_USER_POOLS`, el valor de nombre de usuario es el valor del atributo cognito:username. En el caso de una autorización `AWS_IAM`, el valor de nombre de usuario es el valor de la entidad principal del usuario de AWS. Si utiliza una autorización de IAM con credenciales proporcionadas desde grupos de identidad de Amazon Cognito, le recomendamos que use `cognitoIdentityId`.

Acceso a los encabezados de solicitud

AWS AppSync permite enviar encabezados personalizados de los clientes y obtener acceso a ellos desde los solucionadores de GraphQL utilizando `ctx.request.headers`. Puede utilizar los valores de encabezado para acciones como insertar datos en un origen de datos o incluso efectuar comprobaciones de autorización. Puede usar uno o varios encabezados de solicitud usando `$curl` con una clave de API desde la línea de comandos, como se muestra en los siguientes ejemplos:

Ejemplo de encabezado único

Supongamos que establece un encabezado `custom` con un valor `nadia` como el siguiente:

```
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
```

Se puede obtener acceso a este encabezado con `ctx.request.headers.custom`. Por ejemplo, podría encontrarse en el siguiente código para DynamoDB:

```
"custom": util.dynamodb.toDynamoDB(ctx.request.headers.custom)
```

Ejemplo de varios encabezados

También puede enviar varios encabezados en una única solicitud y obtener acceso a ellos en el controlador del solucionador. Por ejemplo, si el encabezado `custom` se define con dos valores:

```
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
```

Puede obtener acceso a ellos como una matriz, por ejemplo `ctx.request.headers.custom[1]`.

> **Note**
>
> AWS AppSync no expone el encabezado de la cookie en `ctx.request.headers`.

Acceso al nombre de dominio personalizado de la solicitud

AWS AppSync admite la configuración de un dominio personalizado que puede usar para acceder a sus puntos de conexión de GraphQL y en tiempo real para sus API. Al hacer una solicitud con un nombre de dominio personalizado, puede obtener el nombre de dominio utilizando. `ctx.request.domainName`

Cuando se utiliza el nombre de dominio de punto de conexión predeterminado de GraphQL, el valor es. `null`

Información

La sección `info` contiene información sobre la solicitud de GraphQL. Esta sección incluye la siguiente forma:

```
type Info = {
  fieldName: string;
  parentTypeName: string;
  variables: any;
  selectionSetList: string[];
  selectionSetGraphQL: string;
};
```

Cada campo se define de la siguiente manera:

`fieldName`

El nombre del campo que se está resolviendo actualmente.

`parentTypeName`

El nombre del tipo principal del campo que se está resolviendo actualmente.

`variables`

Un mapa que contiene todas las variables que se pasan a la solicitud de GraphQL.

`selectionSetList`

Una representación de lista de los campos del conjunto de selección de GraphQL. Solo se hace referencia a los campos con alias por el nombre de alias, no por el nombre de campo. El ejemplo siguiente muestra detalladamente esta estructura.

`selectionSetGraphQL`

Una representación en forma de cadena del conjunto de selección, formateado como lenguaje de definición de esquema (SDL) de GraphQL. Aunque los fragmentos no se combinan en el conjunto de selección, los fragmentos insertados se conservan, como se muestra en el ejemplo siguiente.

> **Note**
>
> `JSON.stringify` no incluye ni `selectionSetGraphQL` ni `selectionSetList` en la serialización de cadenas. Debe hacer referencia a estas propiedades directamente.

Por ejemplo, al resolver el campo `getPost` de la siguiente consulta:

```
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
  }
}
```

La variable `ctx.info` completa que está disponible al procesar un controlador podría ser:

```
{
  "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` expone solo los campos que pertenecen al tipo actual. Si el tipo actual es una interfaz o una unión, solo se exponen los campos seleccionados que pertenecen a la interfaz. Por ejemplo, en el caso del esquema siguiente:

```
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
}
```

Y en la siguiente consulta:

```
query {
    node(id: "post1") {
        id
        ... on Post {
            title
        }

        ... on Blog {
            title
        }
    }
}
```

Cuando se llama a `ctx.info.selectionSetList` con la resolución del campo `Query.node`, solo se expone `id`:

```
"selectionSetList": [
    "id"
]
```

Características de la versión ejecutable de JavaScript para solucionadores y funciones

El entorno de versión ejecutable APPSYNC_JS proporciona una funcionalidad similar a la versión 6.0 de ECMAScript (ES). Es compatible con un subconjunto de sus características y proporciona algunos métodos adicionales (utilidades) que no forman parte de las especificaciones de ES. En los siguientes temas se mencionan todas las características de lenguaje admitidas.

> **Note**
>
> Actualmente, esta referencia solo se aplica a la versión ejecutable 1.0.0.

Temas

- Características de la versión ejecutable compatibles
- Utilidades integradas
- Módulos integrados
- Utilidades de tiempo de ejecución
- Aplicaciones auxiliares de tiempo en util.time
- Aplicaciones auxiliares de DynamoDB en util.dynamodb
- Aplicaciones auxiliares para HTTP en util.http
- Aplicaciones auxiliares de transformación en util.transform
- Aplicaciones auxiliares de cadena en util.str
- Extensiones
- Aplicaciones auxiliares para XML en util.xml

Características de la versión ejecutable compatibles

En las siguientes secciones se describe el conjunto de características compatibles de la versión ejecutable APPSYNC_JS.

Características principales

Se admiten las siguientes características principales.

Tipos

Se admiten los siguientes tipos:

- números
- cadenas
- booleanos
- objects
- matrices
- funciones

Operadores

Se admiten operadores, entre los que se incluyen:

- operadores matemáticos estándar (`+`, `-`, `/`, `%`, `*`, etc.)
- operador de fusión de NULL (`??`)
- Encadenamiento opcional (`?.`)
- operadores Bitwise
- operadores `void` y `typeof`

No se admiten los siguientes operadores:

- operadores unarios (`++`, `--`, y `~`)
- operador `in`

> **Note**
>
> Utilice el operador `Object.hasOwn` para comprobar si la propiedad especificada está en el objeto especificado.

Instrucciones

Se admiten las siguientes instrucciones:

- `const`

- `let`
- `var`
- `break`
- `else`
- `for-in`
- `for-of`
- `if`
- `return`
- `switch`
- sintaxis de distribución

No se admiten las siguientes:

- `catch`
- `continue`
- `do-while`
- `finally`
- `for(initialization; condition; afterthought)`

> **Note**
>
> Las excepciones son las expresiones `for-in` y `for-of`, que son compatibles.

- `throw`
- `try`
- `while`
- instrucciones etiquetadas

Literales

Se admiten los siguientes literales de plantilla de ES 6:

- Cadenas de varias líneas
- Interpolación de expresiones

- Plantillas de anidamiento

Funciones

Se admite la sintaxis de la función siguiente:

- Se admiten las declaraciones de funciones.
- Se admiten las funciones de flecha de ES 6.
- Se admite la sintaxis del parámetro rest de ES 6.

Modo estricto

Las funciones operan en modo estricto de forma predeterminada, por lo que no necesita agregar una instrucción `use_strict` en su código de función. Esto no se puede cambiar.

Objetos primitivos

Se admiten los siguientes objetos primitivos de ES y sus funciones.

Objeto

Los siguientes objetos son compatibles:

- `Object.assign()`
- `Object.entries()`
- `Object.hasOwn()`
- `Object.keys()`
- `Object.values()`
- `delete`

Cadena

Se admiten las siguientes cadenas:

- `String.prototype.length()`
- `String.prototype.charAt()`
- `String.prototype.concat()`

- `String.prototype.endsWith()`
- `String.prototype.indexOf()`
- `String.prototype.lastIndexOf()`
- `String.raw()`
- `String.prototype.replace()`

> **Note**
>
> No se admiten expresiones regulares.

- `String.prototype.replaceAll()`

> **Note**
>
> No se admiten expresiones regulares.

- `String.prototype.slice()`
- `String.prototype.split()`
- `String.prototype.startsWith()`
- `String.prototype.toLowerCase()`
- `String.prototype.toUpperCase()`
- `String.prototype.trim()`
- `String.prototype.trimEnd()`
- `String.prototype.trimStart()`

Número

Se admiten los siguientes números:

- `Number.isFinite`
- `Number.isNaN`

Objetos y funciones integrados

Se admiten las siguientes funciones y objetos.

Math (Matemática)

Las siguientes funciones matemáticas son compatibles:

- `Math.random()`
- `Math.min()`
- `Math.max()`
- `Math.round()`
- `Math.floor()`
- `Math.ceil()`

Array (Matriz)

Se admiten los siguientes métodos de matriz:

- `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()` no admite argumentos.

- `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`

Consola

El objeto de la consola está disponible para su depuración. Durante la ejecución de la consulta en tiempo real, las instrucciones de registro o error de la consola se envían a Registros de Amazon CloudWatch (si el registro está habilitado). Durante la evaluación de código con `evaluateCode`, las instrucciones de registro se devuelven en la respuesta del comando.

- `console.error()`
- `console.log()`

JSON

Se admiten los siguientes métodos JSON:

- `JSON.parse()`

> **Note**
>
> Devuelve una cadena en blanco si la cadena analizada no es un JSON válido.

- `JSON.stringify()`

Función

- No se admiten los métodos `apply`, `bind` y `call`.
- Los constructores de funciones no son compatibles.
- No se admite la transferencia de una función como argumento.
- No se admiten llamadas a funciones recursivas.

Promesas

No se admiten promesas ni procesos asíncronos.

> **Note**
>
> No se admite el acceso a la red y al sistema de archivos en la versión ejecutable `APPSYNC_JS` en AWS AppSync. AWS AppSync gestiona todas las operaciones de E/S en función de las solicitudes realizadas por el solucionador de AWS AppSync o la función de AWS AppSync.

Globals

Se admiten las siguientes constantes globales:

- `NaN`
- `Infinity`
- `undefined`
- `util`
- `extensions`
- `runtime`

Tipos de error

No se admite la generación de errores con `throw`. Puede devolver un error mediante la función `util.error()`. Puede incluir un error en su respuesta de GraphQL mediante la función `util.appendError`.

Para obtener más información, consulte la sección sobre utilidades de error.

Utilidades integradas

La variable `util` contiene métodos de utilidad generales que ayudan a trabajar con los datos. A menos que se especifique lo contrario, todas las utilidades usan el juego de caracteres UTF-8.

Utilidades de codificación

Lista de utilidades de codificación

`util.urlEncode(String)`

Devuelve la cadena de entrada como una cadena `application/x-www-form-urlencoded` codificada.

`util.urlDecode(String)`

Descodifica una cadena `application/x-www-form-urlencoded` codificada y la devuelve a su forma no codificada.

`util.base64Encode(string) : string`

Codifica la entrada en una cadena codificada en base64.

`util.base64Decode(string) : string`

Descodifica los datos de una cadena codificada en base64.

Utilidades de generación de ID

Lista de utilidades de generación de ID

`util.autoId()`

Devuelve un UUID de 128 bits generado de forma aleatoria.

`util.autoUlid()`

Devuelve un ULID (identificador ordenable lexicográficamente único y universal) de 128 bits generado de forma aleatoria.

`util.autoKsuid()`

Devuelve un KSUID (identificador único ordenable por K) de 128 bits generado de forma aleatoria codificado en base62 como una cadena con una longitud de 27.

Utilidades de error

Lista de utilidades de error

`util.error(String, String?, Object?, Object?)`

Genera un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. También se pueden especificar los campos `errorType`, `data` y `errorInfo`. El valor de `data` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL.

> **Note**
>
> `data` se filtrará en función de la selección de consulta establecida. El valor de `errorInfo` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL.
> `errorInfo` no se filtrará en función de la selección de consulta establecida.

`util.appendError(String, String?, Object?, Object?)`

Adjunta un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. También se pueden especificar los campos `errorType`, `data` y `errorInfo`. A diferencia de `util.error(String, String?, Object?, Object?)`, la evaluación de la plantilla no se interrumpirá, de modo que podrán devolverse datos al intermediario. El valor de `data` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL.

> **Note**
>
> `data` se filtrará en función de la selección de consulta establecida. El valor de `errorInfo` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL.
> `errorInfo` no se filtrará en función de la selección de consulta establecida.

Utilidades de coincidencia de tipos y patrones

Lista de utilidades de coincidencia de tipos y patrones

`util.matches(String, String) : Boolean`

Devuelve un valor true si el patrón especificado en el primer argumento coincide con los datos proporcionados en el segundo argumento. El patrón tiene que ser una expresión regular, por ejemplo `util.matches("a*b", "aaaaab")`. La funcionalidad se basa en Pattern, que puede consultar para obtener más información.

`util.authType()`

Devuelve una cadena que describe el tipo de autenticación múltiple que utiliza una solicitud y devuelve "Autorización de IAM", "Autorización del grupo de usuarios", "Autorización de Open ID Connect" o "Autorización de la clave de API".

Utilidades de comportamiento del valor devuelto

Lista de utilidades de comportamiento del valor devuelto

`util.escapeJavaScript(String)`

Devuelve la cadena de entrada como una cadena de escape de JavaScript.

Utilidades de autorización del solucionador

Lista de utilidades de autorización del solucionador

`util.unauthorized()`

Genera el código `Unauthorized` para el campo que se está resolviendo. Utilícela en las plantillas de mapeo de solicitudes o de respuestas para determinar si se debe permitir al intermediario que resuelva el campo.

Módulos integrados

Los módulos forman parte de la versión ejecutable `APPSYNC_JS` y proporcionan utilidades para ayudar a escribir solucionadores y funciones de JavaScript.

Funciones del módulo de DynamoDB

Las funciones del módulo de DynamoDB proporcionan una experiencia mejorada al interactuar con los orígenes de datos de DynamoDB. Puede realizar solicitudes a sus orígenes de datos de DynamoDB mediante las funciones y sin añadir asignación de tipos.

Los módulos se importan mediante `@aws-appsync/utils/dynamodb`:

```
// Modules are imported using @aws-appsync/utils/dynamodb
import * as ddb from '@aws-appsync/utils/dynamodb';
```

Funciones

Lista de funciones

`get<T>(payload: GetInput): DynamoDBGetItemRequest`

> **Tip**
>
> Consulte the section called “Entradas” para obtener información sobre GetInput.

Genera un objeto `DynamoDBGetItemRequest` para realizar una solicitud GetItem a DynamoDB.

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

```
export function request(ctx) {
 return get({ key: { id: ctx.args.id } });
}
```

`put<T>(payload): DynamoDBPutItemRequest`

Genera un objeto `DynamoDBPutItemRequest` para realizar una solicitud PutItem a 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`

Genera un objeto `DynamoDBDeleteItemRequest` para realizar una solicitud DeleteItem a 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`

Genera un objeto `DynamoDBScanRequest` para realizar una solicitud Scan a 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`

Genera un objeto `DynamoDBSyncRequest` para realizar una solicitud Sync. La solicitud solo recibe los datos modificados desde la última consulta (actualizaciones delta). Las solicitudes solo se pueden realizar a orígenes de datos de DynamoDB con control de versiones.

```
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`

Genera un objeto `DynamoDBUpdateItemRequest` para realizar una solicitud UpdateItem a DynamoDB.

Operaciones

Las aplicaciones auxiliares de operación permiten realizar acciones específicas en partes de sus datos durante las actualizaciones. Para empezar, importe `operations` desde `@aws-appsync/utils/dynamodb`:

```
// Modules are imported using operations
import {operations} from '@aws-appsync/utils/dynamodb';
```

Lista de operaciones

`add<T>(payload)`

Función auxiliar que añade un nuevo elemento de atributo al actualizar DynamoDB.

Ejemplo

Para añadir una dirección (calle, ciudad y código postal) a un elemento de DynamoDB existente mediante el valor de 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)`

Función auxiliar que añade una carga a la lista existente en DynamoDB.

Ejemplo

Para añadir los ID de amigo recién agregados (`newFriendIds`) a una lista de amigos existente (`friendsIds`) durante una actualización:

```
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?)`

Función auxiliar que reduce el valor del atributo existente en el elemento al actualizar DynamoDB.

Ejemplo

Para reducir un contador de amigos (`friendsCount`) en 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?)`

Función auxiliar que incrementa el valor del atributo existente en el elemento al actualizar DynamoDB.

Ejemplo

Para incrementar un contador de amigos (`friendsCount`) en 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)`

Función auxiliar que se antepone a la lista existente en DynamoDB.

Ejemplo

Para anteponer los ID de amigo recién agregados (`newFriendIds`) a una lista de amigos existente (`friendsIds`) durante una actualización:

```
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)`

Función auxiliar que sustituye un atributo existente al actualizar un elemento en DynamoDB. Esto resulta útil cuando se desea actualizar todo el objeto o subobjeto en el atributo y no solo las claves en la carga.

Ejemplo

Para sustituir una dirección (calle, ciudad y código postal) en un objeto `info`:

```
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)`

Función auxiliar que sustituye un elemento en una lista.

Ejemplo

En el ámbito de la actualización (`newFriendIds`), en este ejemplo se utiliza `updateListItem` para actualizar los valores de ID del segundo elemento (índice: `1`, nuevo ID: `102`) y del tercer elemento (índice: 2, nuevo ID: `112`) en una lista (`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 });
}
```

Entradas

Lista de entradas

`Type GetInput<T>`

```
GetInput<T>: {
    consistentRead?: boolean;
```

```
    key: DynamoDBKey<T>;
}
```

Declaración de tipo

- `consistentRead?: boolean` (opcional)

  Valor booleano opcional para especificar si desea realizar una lectura altamente coherente con DynamoDB.

- `key: DynamoDBKey<T>` (obligatorio)

  Parámetro obligatorio que especifica la clave del elemento en DynamoDB. Los elementos de DynamoDB pueden tener una sola clave hash o claves hash y de ordenación.

`Type PutInput<T>`

```
PutInput<T>: {
    _version?: number;
    condition?: DynamoDBFilterObject<T> | null;
    customPartitionKey?: string;
    item: Partial<T>;
    key: DynamoDBKey<T>;
    populateIndexFields?: boolean;
}
```

Declaración de tipo

- `_version?: number` (opcional)
- `condition?: DynamoDBFilterObject<T> | null` (opcional)

  Al colocar un objeto en una tabla de DynamoDB, puede especificar opcionalmente una expresión condicional que determine si la solicitud se debe atender o no en función del estado del objeto que ya está en DynamoDB antes de ejecutar la operación.

- `customPartitionKey?: string` (opcional)

  Cuando se habilita, este valor de cadena modifica el formato de los registros `ds_sk` y `ds_pk` que utiliza la tabla Delta Sync una vez habilitado el control de versiones. Cuando se habilita, también lo hace el procesamiento de la entrada `populateIndexFields`.

- `item: Partial<T>` (obligatorio)

  El resto de los atributos del elemento que deben incluirse en DynamoDB.

- `key: DynamoDBKey<T>` (obligatorio)

  Parámetro obligatorio que especifica la clave del elemento en DynamoDB donde se realizará la operación put. Los elementos de DynamoDB pueden tener una sola clave hash o claves hash y de ordenación.

- `populateIndexFields?: boolean` (opcional)

  Valor booleano que, cuando se habilita junto con la `customPartitionKey`, crea nuevas entradas para cada registro de la tabla Delta Sync, específicamente en las columnas `gsi_ds_pk` y `gsi_ds_sk`. Para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync.

`Type QueryInput<T>`

```
QueryInput<T>: ScanInput<T> & {
    query: DynamoDBKeyCondition<Required<T>>;
}
```

Declaración de tipo

- `query: DynamoDBKeyCondition<Required<T>>` (obligatorio)

  Especifica una condición clave que describe los elementos que se van a consultar. Para un índice determinado, la condición de una clave de partición debe ser una igualdad y la clave de ordenación una comparación o un elemento `beginsWith` (cuando es una cadena). Solo se admiten los tipos Number y String para las claves de partición y ordenación.

  Ejemplo

  Tome el tipo `User` a continuación:

  ```
  type User = {
    id: string;
    name: string;
    age: number;
    isVerified: boolean;
    friendsIds: string[]
  }
  ```

  La consulta solo puede incluir los campos siguientes: `id`, `name` y `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;
}
```

Declaración de tipo

- `_version?: number` (opcional)
- `condition?: DynamoDBFilterObject<T>` (opcional)

  Al quitar un objeto en DynamoDB, puede especificar opcionalmente una expresión condicional que determine si la solicitud se debe atender o no en función del estado del objeto que ya está en DynamoDB antes de ejecutar la operación.

  Ejemplo

  El siguiente ejemplo es una expresión `DeleteItem` que contiene una condición que permite que la operación se realice correctamente solo si el propietario del documento coincide con el usuario que realiza la solicitud.

```
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` (opcional)

  Cuando se habilita, el valor `customPartitionKey` modifica el formato de los registros `ds_sk` y `ds_pk` que utiliza la tabla Delta Sync una vez habilitado el control de versiones. Cuando se habilita, también lo hace el procesamiento de la entrada `populateIndexFields`.

- `key: DynamoDBKey<T>` (obligatorio)

  Parámetro obligatorio que especifica la clave del elemento de DynamoDB que se va a quitar. Los elementos de DynamoDB pueden tener una sola clave hash o claves hash y de ordenación.

  Ejemplo

  Si un `User` solo tiene la clave hash con un `id` de usuario, la clave tendría el siguiente aspecto:

```
type User = {
 id: number
 name: string
 age: number
 isVerified: boolean
}
const key: DynamoDBKey<User> = {
 id: 1,
}
```

  Si el usuario de la tabla tiene una clave hash (`id`) y una clave de ordenación (`name`), la clave tendría el siguiente aspecto:

```
type User = {
 id: number
 name: string
 age: number
 isVerified: boolean
 friendsIds: string[]
```

```
}

const key: DynamoDBKey<User> = {
 id: 1,
 name: 'XXXXXXXXXX',
}
```

- `populateIndexFields?: boolean` (opcional)

  Valor booleano que, cuando se habilita junto con la `customPartitionKey`, crea nuevas entradas para cada registro de la tabla Delta Sync, específicamente en las columnas `gsi_ds_pk` y `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;
}
```

Declaración de tipo

- `consistentRead?: boolean | null` (opcional)

  Valor booleano opcional para indicar lecturas coherentes al consultar DynamoDB. El valor predeterminado es `false`.

- `filter?: DynamoDBFilterObject<T> | null` (opcional)

  Filtro opcional para aplicar a los resultados después de recuperarlos de la tabla.

- `index?: string | null` (opcional)

  Nombre opcional del índice a examinar.

- `limit?: number | null` (opcional)

  Número máximo opcional de resultados a devolver.

- `nextToken?: string | null` (opcional)

  Token de paginación opcional para continuar una consulta anterior. Se debe obtener de una consulta anterior.

- `scanIndexForward?: boolean | null` (opcional)

  Valor booleano opcional para indicar si la consulta se realiza en orden ascendente o descendente. De forma predeterminada, este valor se establece en `true`.

- `segment?: number` (opcional)
- `select?: DynamoDBSelectAttributes` (opcional)

  Atributos que se devolverán de DynamoDB. De forma predeterminada, el solucionador de DynamoDB de AWS AppSync solo devuelve los atributos que se proyectan en el índice. Los valores admitidos son:

  - `ALL_ATTRIBUTES`

    Devuelve todos los atributos de elementos de la tabla o el índice especificados. Si consulta un índice secundario local, DynamoDB recupera todo el elemento de la tabla principal para cada elemento coincidente en el índice. Si el índice está configurado para proyectar todos los atributos de los elementos, todos los datos se pueden obtener del índice secundario local y no es necesario efectuar una recuperación.

  - `ALL_PROJECTED_ATTRIBUTES`

    Devuelve todos los atributos que se han proyectado en el índice. Si el índice está configurado para proyectar todos los atributos, este valor de retorno equivale a especificar `ALL_ATTRIBUTES`.

  - `SPECIFIC_ATTRIBUTES`

    Devuelve solo los atributos que aparecen en `ProjectionExpression`. Este valor devuelto equivale a especificar `ProjectionExpression` sin especificar ningún valor para `AttributesToGet`.

- `totalSegments?: number` (opcional)

`Type DynamoDBSyncInput<T>`

```
DynamoDBSyncInput<T>: {
    basePartitionKey?: string;
    deltaIndexName?: string;
    filter?: DynamoDBFilterObject<T> | null;
```

```
    lastSync?: number;
    limit?: number | null;
    nextToken?: string | null;
}
```

Declaración de tipo

- `basePartitionKey?: string` (opcional)

  Clave de partición de la tabla base que se utilizará al realizar una operación Sync. Este campo permite realizar una operación Sync cuando la tabla utiliza una clave de partición personalizada.

- `deltaIndexName?: string` (opcional)

  Índice utilizado para la operación Sync. Este índice es necesario para habilitar una operación Sync en toda la tabla de almacenamiento Delta cuando la tabla utiliza una clave de partición personalizada. La operación Sync se realizará en el GSI (creado en `gsi_ds_pk` y `gsi_ds_sk`).

- `filter?: DynamoDBFilterObject<T> | null` (opcional)

  Filtro opcional para aplicar a los resultados después de recuperarlos de la tabla.

- `lastSync?: number` (opcional)

  Momento, en milisegundos transcurridos desde la fecha de inicio, en el que comenzó la última operación Sync que se ha realizado correctamente. Si se especifica, solo se devuelven los elementos que han cambiado después de `lastSync`. Este campo solo debe rellenarse después de haber recuperado todas las páginas de una operación Sync inicial. Si se omite, se devolverán los resultados de la tabla base. De lo contrario, se devolverán los resultados de la tabla Delta.

- `limit?: number | null` (opcional)

  Número máximo opcional de elementos que se evalúan en una sola vez. Si se omite, el límite predeterminado se establecerá en `100` elementos. El valor máximo de este campo son `1000` elementos.

- `nextToken?: string | null` (opcional)

`Type DynamoDBUpdateInput<T>`

```
DynamoDBUpdateInput<T>: {
    _version?: number;
```

```
    condition?: DynamoDBFilterObject<T>;
    customPartitionKey?: string;
    key: DynamoDBKey<T>;
    populateIndexFields?: boolean;
    update: DynamoDBUpdateObject<T>;
}
```

Declaración de tipo

- `_version?: number` (opcional)
- `condition?: DynamoDBFilterObject<T>` (opcional)

  Al actualizar un objeto en DynamoDB, puede especificar opcionalmente una expresión condicional que determine si la solicitud se debe atender o no en función del estado del objeto que ya está en DynamoDB antes de ejecutar la operación.

- `customPartitionKey?: string` (opcional)

  Cuando se habilita, el valor `customPartitionKey` modifica el formato de los registros `ds_sk` y `ds_pk` que utiliza la tabla Delta Sync una vez habilitado el control de versiones. Cuando se habilita, también lo hace el procesamiento de la entrada `populateIndexFields`.

- `key: DynamoDBKey<T>` (obligatorio)

  Parámetro obligatorio que especifica la clave del elemento de DynamoDB que se actualiza. Los elementos de DynamoDB pueden tener una sola clave hash o claves hash y de ordenación.

- `populateIndexFields?: boolean` (opcional)

  Valor booleano que, cuando se habilita junto con la `customPartitionKey`, crea nuevas entradas para cada registro de la tabla Delta Sync, específicamente en las columnas `gsi_ds_pk` y `gsi_ds_sk`.

- `update: DynamoDBUpdateObject<T>`

  Objeto que especifica los atributos que se van a actualizar junto con sus nuevos valores. El objeto de actualización se puede utilizar con `add`, `remove`, `replace`, `increment`, `decrement`, `append`, `prepend` y `updateListItem`.

Funciones del módulo Amazon RDS

Las funciones del módulo Amazon RDS ofrecen una experiencia mejorada al interactuar con bases de datos configuradas con la API de datos de Amazon RDS. El módulo se importa mediante `@aws-appsync/utils/rds`:

```
import * as rds from '@aws-appsync/utils/rds';
```

Las funciones también se pueden importar de forma individual. Por ejemplo, la importación siguiente utiliza `sql`:

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

Funciones

Puede utilizar las aplicaciones auxiliares de utilidades del módulo AWS AppSync RDS para interactuar con su base de datos.

Seleccionar

La utilidad `select` crea una instrucción `SELECT` para consultar la base de datos relacional.

Uso básico

En su forma básica, puede especificar la tabla que desea consultar:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

    // Generates statement:
    // "SELECT * FROM "persons"
    return createPgStatement(select({table: 'persons'}));
}
```

Tenga en cuenta que también puede especificar el esquema en el identificador de la tabla:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
```

```
    // Generates statement:
    // SELECT * FROM "private"."persons"
    return createPgStatement(select({table: 'private.persons'}));
}
```

Especificación de columnas

Puede especificar columnas con la propiedad `columns`. Si no se establece en un valor, el valor predeterminado es *:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name']
    }));
}
```

También puede especificar la tabla de una columna:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "persons"."name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'persons.name']
    }));
}
```

Límites y desplazamientos

Puede aplicar `limit` y `offset` a la consulta:

```
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
    }));
}
```

Ordenar por

Puede ordenar los resultados con la propiedad `orderBy`. Proporcione una matriz de objetos especificando la columna y una propiedad `dir` opcional:

```
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'}]
    }));
}
```

Filtros

Puede crear filtros con el objeto de condición especial:

```
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'}}
    }));
}
```

También puede combinar filtros:

```
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}}
    }));
}
```

También puede crear instrucciones 0R:

```
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 } }
        ]}
    }));
}
```

También puede negar una condición con `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 } }
        ]}
    }));
}
```

También puede utilizar los siguientes operadores para comparar valores:

Operador	Descripción	Tipos de valor posibles
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

Inserte

La utilidad `insert` ofrece una forma sencilla de insertar elementos de una sola fila en la base de datos con la operación `INSERT`.

Inserciones de un solo elemento

Para insertar un elemento, especifique la tabla y, a continuación, transfiera su objeto de valores. Las claves de objetos se asignan a las columnas de la tabla. Los nombres de las columnas se escapan automáticamente y los valores se envían a la base de datos mediante el mapa de variables:

```
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)
}
```

Caso de uso de MySQL

Puede combinar un `insert` seguido de un `select` para recuperar la fila insertada:

```
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)
}
```

Caso de uso de Postgres

Con Postgres, puede usar `returning` para obtener datos de la fila que insertó. Acepta `*` o una matriz de nombres de columna:

```
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)
}
```

Actualización

La utilidad `update` le permite actualizar las filas existentes. Puede utilizar el objeto de condición para aplicar cambios a las columnas especificadas en todas las filas que cumplan la condición. Por ejemplo, supongamos que tenemos un esquema que nos permite realizar esta mutación. Queremos actualizar el `name` de `Person` con el valor `id` de `3`, pero solo si los conocemos (`known_since`) desde el año `2000`:

```
mutation Update {
    updatePerson(
        input: {id: 3, name: "Jon"},
        condition: {known_since: {ge: "2000"}}
    ) {
    id
```

```
    name
  }
}
```

Nuestro solucionador de actualización tendrá este aspecto:

```
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)
}
```

Podemos añadir una comprobación a nuestra condición para asegurarnos de que solo se actualice la fila en la que la clave principal `3` sea igual a `id`. Del mismo modo, en el caso de `inserts` de Postgres, se puede utilizar `returning` para devolver los datos modificados.

Remove

La utilidad `remove` le permite eliminar las filas existentes. Puede utilizar el objeto de condición en todas las filas que cumplan la condición. Tenga en cuenta que `delete` es una palabra clave reservada en JavaScript. En su lugar, debería usarse `remove`:

```
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)
}
```

Conversión

En algunos casos, es posible que desee más especificidad sobre el tipo de objeto correcto para usar en su instrucción. Puede utilizar las sugerencias de tipo proporcionadas para especificar el tipo de parámetros. AWS AppSync admite las mismas sugerencias de tipo que la API de datos. Puede convertir sus parámetros mediante las funciones `typeHint` del módulo `rds` de AWS AppSync.

En el siguiente ejemplo puede enviar una matriz como un valor que se convierte en un objeto JSON. Usamos el operador `->` para recuperar el elemento en el `index` 2 en la matriz 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
}
```

La conversión también resulta útil al manipular y comparar `DATE`, `TIME` y `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)
}
```

Aquí se muestra otro ejemplo de cómo puede enviar la fecha y hora actuales:

```
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)}`)
}
```

Sugerencias de tipos disponibles

- `typeHint.DATE`: el parámetro correspondiente se envía como un objeto de tipo `DATE` a la base de datos. El formato aceptado es `YYYY-MM-DD`.
- `typeHint.DECIMAL`: el parámetro correspondiente se envía como un objeto de tipo `DECIMAL` a la base de datos.
- `typeHint.JSON`: el parámetro correspondiente se envía como un objeto de tipo `JSON` a la base de datos.
- `typeHint.TIME`: el valor del parámetro de cadena correspondiente se envía como un objeto de tipo `TIME` a la base de datos. El formato aceptado es `HH:MM:SS[.FFF]`.
- `typeHint.TIMESTAMP`: el valor del parámetro de cadena correspondiente se envía como un objeto de tipo `TIMESTAMP` a la base de datos. El formato aceptado es `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `typeHint.UUID`: el valor del parámetro de cadena correspondiente se envía como un objeto de tipo `UUID` a la base de datos.

Utilidades de tiempo de ejecución

La biblioteca de `runtime` proporciona utilidades para controlar o modificar las propiedades de tiempo de ejecución de sus solucionadores y funciones.

Lista de utilidades de tiempo de ejecución

`runtime.earlyReturn(obj?: unknown): never`

La invocación de esta función detendrá la ejecución de la función o el solucionador de AWS AppSync actual (solucionador de unidad o canalización) según el contexto actual. El objeto especificado se devuelve como resultado.

- Al llamarse en un controlador de solicitudes de función de AWS AppSync, se omiten el origen de datos y el controlador de respuestas y se llama al siguiente controlador de solicitudes de función (o al controlador de respuestas del solucionador de canalización en caso de tratarse de la última función de AWS AppSync).
- Al llamarse en un controlador de solicitudes del solucionador de canalización de AWS AppSync, se omite la ejecución de la canalización y se llama inmediatamente al controlador de respuestas del solucionador de canalización.

Ejemplo

```
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
}
```

Aplicaciones auxiliares de tiempo en util.time

La variable `util.time` contiene métodos de fecha y hora útiles para ayudar a generar marcas de tiempo, convertir entre formatos de fecha y hora y analizar cadenas de fecha y hora. La sintaxis de los formatos de fecha y hora se basa en DateTimeFormatter, que puede consultar para obtener más información. A continuación se ofrecen algunos ejemplos, así como una lista de métodos disponibles y sus descripciones.

Utilidades de tiempo

Lista de utilidades de tiempo

`util.time.nowISO8601()`

Devuelve una representación de cadena de la hora UTC en formato ISO8601.

`util.time.nowEpochSeconds()`

Devuelve el número de segundos desde la fecha de inicio de 1970-01-01T00:00:00Z hasta ahora.

`util.time.nowEpochMilliSeconds()`

Devuelve el número de milisegundos desde la fecha de inicio de 1970-01-01T00:00:00Z hasta ahora.

`util.time.nowFormatted(String)`

Devuelve una cadena con la marca de tiempo actual en UTC utilizando el formato especificado en un tipo de entrada String.

`util.time.nowFormatted(String, String)`

Devuelve una cadena con la marca de tiempo actual de una zona horaria utilizando el formato y la zona horaria especificados en tipos de entrada String.

`util.time.parseFormattedToEpochMilliSeconds(String, String)`

Analiza una marca de tiempo pasada como String junto con un formato y, a continuación, devuelve la marca de tiempo como milisegundos transcurridos desde la fecha de inicio.

`util.time.parseFormattedToEpochMilliSeconds(String, String, String)`

Analiza una marca de tiempo pasada como String junto con un formato y una zona horaria y, a continuación, la devuelve como milisegundos transcurridos desde la fecha de inicio.

`util.time.parseISO8601ToEpochMilliSeconds(String)`

Analiza una marca de tiempo ISO8601 pasada como String y, a continuación, la devuelve como milisegundos transcurridos desde la fecha de inicio.

`util.time.epochMilliSecondsToSeconds(long)`

Convierte una marca de tiempo en milisegundos desde la fecha de inicio en una marca de tiempo en segundos de la fecha de inicio.

`util.time.epochMilliSecondsToISO8601(long)`

Convierte una marca de tiempo en milisegundos desde la fecha de inicio en una marca de tiempo ISO8601.

`util.time.epochMilliSecondsToFormatted(long, String)`

Convierte una marca de tiempo en milisegundos desde la fecha de inicio, pasada como tipo long, en una marca de tiempo UTC con el formato suministrado.

`util.time.epochMilliSecondsToFormatted(long, String, String)`

Convierte una marca de tiempo en milisegundos desde la fecha de inicio, pasada como tipo long, en una marca de tiempo para la zona horaria y con el formato suministrados.

Aplicaciones auxiliares de DynamoDB en util.dynamodb

`util.dynamodb` contiene métodos auxiliares que facilitan la escritura y la lectura de datos en Amazon DynamoDB, como el mapeo y el formato automáticos de los tipos de datos.

toDynamoDB

Lista de utilidades toDynamoDB

`util.dynamodb.toDynamoDB(Object)`

Herramienta de conversión general de objetos para DynamoDB que convierte objetos de entrada en la representación de DynamoDB correspondiente. Es algo inflexible en cuanto al modo en que representa algunos tipos: por ejemplo, utiliza listas ("L") en lugar de conjuntos ("SS", "NS" "BS"). Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

Ejemplo de cadena

```
Input:      util.dynamodb.toDynamoDB("foo")
Output:     { "S" : "foo" }
```

Ejemplo de número

```
Input:      util.dynamodb.toDynamoDB(12345)
Output:     { "N" : 12345 }
```

Ejemplo de booleano

```
Input:      util.dynamodb.toDynamoDB(true)
Output:     { "BOOL" : true }
```

Ejemplo de lista

```
Input:      util.dynamodb.toDynamoDB([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

Ejemplo de mapa

```
Input:      util.dynamodb.toDynamoDB({ "foo": "bar", "baz" : 1234, "beep":
 [ "boop"] })
Output:     {
               "M" : {
                   "foo"  : { "S" : "bar" },
                   "baz"  : { "N" : 1234 },
                   "beep" : {
                       "L" : [
                           { "S" : "boop" }
                       ]
                   }
               }
           }
```

Utilidades toString

Lista de utilidades toString

`util.dynamodb.toString(String)`

Convierte una cadena de entrada al formato de cadena de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      util.dynamodb.toString("foo")
Output:     { "S" : "foo" }
```

`util.dynamodb.toStringSet(List<String>)`

Convierte una lista con cadenas al formato de conjunto de cadenas de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      util.dynamodb.toStringSet([ "foo", "bar", "baz" ])
Output:     { "SS" : [ "foo", "bar", "baz" ] }
```

Utilidades toNumber

Lista de utilidades toNumber

`util.dynamodb.toNumber(Number)`

Convierte un número al formato de número de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      util.dynamodb.toNumber(12345)
Output:     { "N" : 12345 }
```

`util.dynamodb.toNumberSet(List<Number>)`

Convierte una lista de números al formato de conjunto de números de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      util.dynamodb.toNumberSet([ 1, 23, 4.56 ])
Output:     { "NS" : [ 1, 23, 4.56 ] }
```

Utilidades toBinary

Lista de utilidades toBinary

`util.dynamodb.toBinary(String)`

Convierte datos binarios codificados como una cadena en base64 al formato binario de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      util.dynamodb.toBinary("foo")
Output:     { "B" : "foo" }
```

`util.dynamodb.toBinarySet(List<String>)`

Convierte una lista de datos binarios codificados como cadenas en base64 al formato de conjunto binario de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      util.dynamodb.toBinarySet([ "foo", "bar", "baz" ])
Output:     { "BS" : [ "foo", "bar", "baz" ] }
```

Utilidades toBoolean

Lista de utilidades toBoolean

`util.dynamodb.toBoolean(Boolean)`

Convierte un valor booleano al formato booleano correspondiente de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      util.dynamodb.toBoolean(true)
Output:     { "BOOL" : true }
```

Utilidades toNull

Lista de utilidades toNull

`util.dynamodb.toNull()`

Devuelve un valor nulo con el formato nulo de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      util.dynamodb.toNull()
Output:     { "NULL" : null }
```

Utilidades toList

Lista de utilidades toList

`util.dynamodb.toList(List)`

Convierte una lista de objetos al formato de lista de DynamoDB. Cada elemento de la lista se convierte también al formato correspondiente de DynamoDB. Es algo inflexible en cuanto al modo en que representa algunos objetos anidados: por ejemplo, utiliza listas ("L") en lugar de conjuntos ("SS", "NS" "BS"). Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      util.dynamodb.toList([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

Utilidades toMap

Lista de utilidades toMap

`util.dynamodb.toMap(Map)`

Convierte un mapa al formato de mapa de DynamoDB. Cada valor del mapa se convierte también al formato de DynamoDB correspondiente. Es algo inflexible en cuanto al modo en que representa algunos objetos anidados: por ejemplo, utiliza listas ("L") en lugar de conjuntos ("SS", "NS" "BS"). Esto devuelve un objeto que describe el valor del atributo de 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)`

Crea una copia del mapa en la que cada valor se convierte al formato correspondiente de DynamoDB. Es algo inflexible en cuanto al modo en que representa algunos objetos anidados: por ejemplo, utiliza listas ("L") en lugar de conjuntos ("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**
>
> Esto es ligeramente diferente de `util.dynamodb.toMap(Map)`, ya que solo devuelve el contenido del valor de atributo de DynamoDB y no todo el valor de atributo en sí. Por ejemplo, las siguientes instrucciones son exactamente lo mismo:
>
> ```
> util.dynamodb.toMapValues(<map>)
> util.dynamodb.toMap(<map>)("M")
> ```

Utilidades S3Object

Lista de utilidades S3Object

`util.dynamodb.toS3Object(String key, String bucket, String region)`

Convierte la clave, el bucket y la región a la representación de objeto de S3 de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de 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)`

Convierte la clave, el bucket, la región y la versión opcional a la representación de objeto de S3 de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      util.dynamodb.toS3Object("foo", "bar", "baz", "beep")
Output:     { "S" : "{ \"s3\" : { \"key\" : \"foo\", \"bucket\" : \"bar\", \"region
\" : \"baz\", \"version\" = \"beep\" } }" }
```

`util.dynamodb.fromS3ObjectJson(String)`

Acepta el valor de cadena de un objeto de S3 de DynamoDB y devuelve un mapa que contiene la clave, el bucket, la región y la versión opcional.

```
Input:      util.dynamodb.fromS3ObjectJson({ "S" : "{ \"s3\" : { \"key\" : \"foo\",
 \"bucket\" : \"bar\", \"region\" : \"baz\", \"version\" = \"beep\" } }" })
Output:     { "key" : "foo", "bucket" : "bar", "region" : "baz", "version" :
 "beep" }
```

Aplicaciones auxiliares para HTTP en util.http

La utilidad `util.http` proporciona métodos auxiliares que puede utilizar para gestionar parámetros de solicitud HTTP y añadir encabezados de respuesta.

Lista de utilidades util.http

`util.http.copyHeaders(headers)`

Copia el encabezado del mapa sin el conjunto restringido de los encabezados HTTP. Se puede usar para reenviar encabezados de solicitud al siguiente punto de conexión HTTP.

`util.http.addResponseHeader(String, Object)`

Añade un único encabezado personalizado con el nombre (`String`) y el valor (`Object`) de la respuesta. Se aplican las siguientes restricciones:

- Los nombres del encabezado no pueden coincidir con ninguno de los encabezados de AWS o AWS AppSync existentes o restringidos.
- Los nombres del encabezado no pueden comenzar por prefijos restringidos, como `x-amzn-` o `x-amz-`.
- El tamaño de los encabezados de respuesta personalizada no puede superar los 4 KB. Esto incluye los nombres y valores del encabezado.
- Debe definir cada encabezado de respuesta una vez por operación de GraphQL. Sin embargo, si define un encabezado personalizado con el mismo nombre varias veces, la definición más reciente aparecerá en la respuesta. Todos los encabezados se contabilizan para el límite de tamaño del encabezado independientemente de los nombres.

`util.http.addResponseHeaders(Map)`

Añade varios encabezados de respuesta a la respuesta desde el mapa de nombres (`String`) y valores (`Object`) especificado. Las mismas limitaciones enumeradas para el método `addResponseHeader(String, Object)` también se aplican a este método.

Aplicaciones auxiliares de transformación en util.transform

`util.transform` contiene métodos auxiliares que facilitan las operaciones complejas sobre orígenes de datos.

Lista de utilidades de aplicaciones auxiliares de transformación

`util.transform.toDynamoDBFilterExpression(filterObject: DynamoDBFilterObject) : string`

Convierte una cadena de entrada en una expresión de filtro que puede usarse en DynamoDB. Recomendamos usar `toDynamoDBFilterExpression` con las funciones del módulo integradas.

`util.transform.toElasticsearchQueryDSL(object: OpenSearchQueryObject) : string`

Convierte la entrada dada en su expresión DSL de consulta OpenSearch equivalente y la devuelve como cadena JSON.

Ejemplo de entrada:

```
util.transform.toElasticsearchQueryDSL({
    "upvotes":{
        "ne":15,
        "range":[
            10,
            20
        ]
    },
    "title":{
        "eq":"hihihi",
        "wildcard":"h*i"
    }
  })
```

Ejemplo de salida:

```
{
    "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**
>
> Se entiende que el operador predeterminado es AND.

`util.transform.toSubscriptionFilter(objFilter, ignoredFields?, rules?): SubscriptionFilter`

Convierte un objeto de entrada `Map` en un objeto de expresión `SubscriptionFilter`. El método `util.transform.toSubscriptionFilter` se utiliza como entrada a la extensión `extensions.setSubscriptionFilter()`. Para obtener más información, consulte el artículo sobre extensiones.

> **Note**
>
> Los parámetros y la instrucción return se indican a continuación:
> Parámetros
>
> - `objFilter`: `SubscriptionFilterObject`
>
> Objeto de entrada `Map` que se convierte en el objeto de expresión `SubscriptionFilter`.
> - `ignoredFields`: `SubscriptionFilterExcludeKeysType` (opcional)
>
> `List` de nombres de campo en el primer objeto que se omitirán.
> - `rules`: `SubscriptionFilterRuleObject` (opcional)
>
> Objeto de entrada `Map` con reglas estrictas que se incluye al construir el objeto de expresión `SubscriptionFilter`. Estas reglas estrictas se incluirán en el objeto de expresión `SubscriptionFilter` de tal forma que se cumpla al menos una de las reglas para pasar el filtro de suscripción.
>
> Respuesta
> Devuelve `SubscriptionFilter`.

`util.transform.toSubscriptionFilter(Map, List)`

Convierte un objeto de entrada `Map` en un objeto de expresión `SubscriptionFilter`. El método `util.transform.toSubscriptionFilter` se utiliza como entrada a la extensión `extensions.setSubscriptionFilter()`. Para obtener más información, consulte el artículo sobre extensiones.

El primer argumento es el objeto de entrada `Map` que se convierte en el objeto de expresión `SubscriptionFilter`. El segundo argumento es una `List` de nombres de campo

que se omiten en el primer objeto de entrada `Map` al construir el objeto de expresión `SubscriptionFilter`.

`util.transform.toSubscriptionFilter(Map, List, Map)`

Convierte un objeto de entrada `Map` en un objeto de expresión `SubscriptionFilter`. El método `util.transform.toSubscriptionFilter` se utiliza como entrada a la extensión `extensions.setSubscriptionFilter()`. Para obtener más información, consulte el artículo sobre extensiones.

`util.transform.toDynamoDBConditionExpression(conditionObject)`

Crea una expresión de condición de DynamoDB.

Argumentos de filtro de suscripción

En la siguiente tabla se explica cómo se definen los argumentos de las siguientes utilidades:

- `Util.transform.toSubscriptionFilter(objFilter, ignoredFields?, rules?): SubscriptionFilter`

Argument 1: Map

El argumento 1 es un objeto `Map` con los siguientes valores clave:

- nombres de los campos
- "and"
- "or"

En el caso de los nombres de campo como claves, las condiciones de las entradas de estos campos adoptan la forma de `"operator" : "value"`.

En el siguiente ejemplo se muestra cómo se pueden añadir entradas a `Map`:

```
"field_name" : {
                    "operator1" : value
                }

## We can have multiple conditions for the same field_name:

"field_name" : {
```

```
                    "operator1" : value
                    "operator2" : value
                    .
                    .
                    .
                }
```

Cuando un campo tiene dos o más condiciones, se considera que todas estas condiciones utilizan la operación OR.

La entrada `Map` también puede tener "and" y "or" como claves, lo que implica que todas las entradas que incluyen deben unirse con la lógica AND u OR, según la clave. Los valores clave "and" y "or" esperan una matriz de condiciones.

```
"and" : [

            {
                "field_name1" : {
                    "operator1" : value
                }
             },

             {
                "field_name2" : {
                    "operator1" : value
                }
             },
             .
             .
        ].
```

Tenga en cuenta que puede anidar "and" y "or". Es decir, puede tener "and" y "or" anidados dentro de otro bloque "and" y "or". Sin embargo, no funciona para campos simples.

```
"and" : [

            {
                "field_name1" : {
                    "operator" : value
                }
             },
```

```
            {
               "or" : [
                           {
                               "field_name2" : {
                                   "operator" : value
                               }
                           },

                           {
                               "field_name3" : {
                                   "operator" : value
                               }
                           }

                       ].
```

En el siguiente ejemplo se muestra una entrada de argumento 1 con `util.transform.toSubscriptionFilter(Map) : Map`.

Entradas

Argumento 1: mapa:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 2000
      }
    }
  ],
  "or": [
    {
      "author": {
```

```
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Salida

El resultado es un objeto `Map`:

```
{
  "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

El argumento 2 contiene una `List` de nombres de campo que no deberían tenerse en cuenta en la entrada `Map` (argumento 1) al construir el objeto de expresión `SubscriptionFilter`. `List` también puede estar vacía.

En el siguiente ejemplo se muestran las entradas de argumento 1 y argumento 2 con `util.transform.toSubscriptionFilter(Map, List) : Map`.

Entradas

Argumento 1: mapa:

```
{
  "percentageUp": {
```

```
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Argumento 2: lista:

```
["percentageUp", "author"]
```

Salida

El resultado es un objeto `Map`:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "title",
```

```
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    }
  ]
}
```

Argument 3: Map

El argumento 3 es un objeto `Map` que tiene nombres de campo como valores clave (no puede tener "and" u "or"). En el caso de los nombres de campo como claves, las condiciones de estos campos son entradas en forma de `"operator" : "value"`. A diferencia del argumento 1, el argumento 3 no puede tener varias condiciones en la misma clave. Además, el argumento 3 no tiene una cláusula "and" u "or", por lo que tampoco hay anidamiento involucrado.

El argumento 3 representa una lista de reglas estrictas, que se añaden al objeto de expresión `SubscriptionFilter` para que se cumpla al menos una de estas condiciones para pasar el filtro.

```
{
  "fieldname1": {
    "operator": value
  },
  "fieldname2": {
    "operator": value
  }
}
.
.
.
```

En el siguiente ejemplo se muestran las entradas de argumento 1, argumento 2 y argumento 3 con `util.transform.toSubscriptionFilter(Map, List, Map) : Map`.

Entradas

Argumento 1: mapa:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "lt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Argumento 2: lista:

```
["percentageUp", "author"]
```

Argumento 3: mapa:

```
{
  "upvotes": {
    "gte": 250
  },
  "author": {
    "eq": "Person1"
  }
}
```

Salida

El resultado es un objeto `Map`:

```
{
  "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"
        }
      ]
    }
  ]
}
```

Aplicaciones auxiliares de cadena en util.str

`util.str` contiene métodos para ayudar con operaciones de cadena comunes.

Lista de utilidades util.str

`util.str.normalize(String, String)`

Normaliza una cadena con uno de los cuatro formatos de normalización de Unicode: NFC, NFD, NFKC o NFKD. El primer argumento es la cadena que se va a normalizar. El segundo argumento es "nfc", "nfd", "nfkc" o "nfkd" y especifica el tipo de normalización que se utilizará en el proceso de normalización.

Extensiones

`extensions` contiene un conjunto de métodos para realizar acciones adicionales en sus solucionadores.

Almacenamiento en caché de extensiones

`extensions.evictFromApiCache(typeName: string, fieldName: string, keyValuePair: Record<string, string>) : Object`

Expulsa un elemento de la memoria caché del lado del servidor de AWS AppSync. El primer argumento es el nombre de tipo. El segundo argumento es el nombre de campo. El tercer argumento es un objeto que contiene elementos de pares clave-valor que especifican el valor clave de almacenamiento en caché. Debe colocar los elementos del objeto en el mismo orden que las claves de almacenamiento en caché del elemento `cachingKey` del solucionador almacenado en caché. Para obtener más información acerca del almacenamiento en caché, consulte la sección sobre el comportamiento de almacenamiento en caché.

Ejemplo 1:

En este ejemplo, se expulsan los elementos que se almacenaron en caché para un solucionador llamado `Query.allClasses` en el que se usó una clave de almacenamiento en caché denominada `context.arguments.semester`. Cuando se llama a la mutación y se ejecuta el solucionador, si una entrada se borra correctamente, la respuesta contiene un valor `apiCacheEntriesDeleted` en el objeto de extensiones que muestra cuántas entradas se eliminaron.

```
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**
>
> Esta función solo funciona para mutaciones, no para consultas.

Extensiones de suscripción

`extensions.setSubscriptionFilter(filterJsonObject)`

Define filtros de suscripción mejorados. Cada evento de notificación de suscripción se evalúa con respecto a los filtros de suscripción proporcionados y envía notificaciones a los clientes si todos los filtros se evalúan como `true`. El argumento es `filterJsonObject` (a continuación se puede encontrar más información sobre este argumento, en la sección sobre el argumento filterJsonObject). Consulte el artículo sobre filtrado de suscripciones mejorado.

> **Note**
>
> Puede usar esta función de extensión solo en el controlador de respuestas de un solucionador de suscripción. Además, recomendamos usar `util.transform.toSubscriptionFilter` para crear su filtro.

`extensions.setSubscriptionInvalidationFilter(filterJsonObject)`

Define los filtros de invalidación de suscripciones. Los filtros de suscripción se evalúan con respecto a la carga de invalidación y, a continuación, invalidan una suscripción determinada si los filtros se evalúan como `true`. El argumento es `filterJsonObject` (a continuación se puede encontrar más información sobre este argumento, en la sección sobre el argumento filterJsonObject). Consulte el artículo sobre filtrado de suscripciones mejorado.

> **Note**
>
> Puede usar esta función de extensión solo en el controlador de respuestas de un solucionador de suscripción. Además, recomendamos usar `util.transform.toSubscriptionFilter` para crear su filtro.

`extensions.invalidateSubscriptions(invalidationJsonObject)`

Se utiliza para iniciar la invalidación de una suscripción a partir de una mutación. El argumento es `invalidationJsonObject` (a continuación se puede encontrar más información sobre este argumento, en la sección sobre el argumento invalidationJsonObject).

> **Note**
>
> Esta extensión solo se puede usar en las plantillas de mapeo de respuestas de los solucionadores de mutaciones.
> Solo puede usar como máximo cinco llamadas al método `extensions.invalidateSubscriptions()` únicas en una sola solicitud. Si supera este límite, recibirá un error de GraphQL.

Argumento: filterJsonObject

El objeto JSON define los filtros de suscripción o invalidación. Es una matriz de filtros en un `filterGroup`. Cada filtro es una colección de filtros individuales.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                }
           ]

        },
        {
           "filters" : [
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
           ]

        }
    ]
}
```

Cada filtro tiene tres atributos:

- `fieldName`: campo de esquema de GraphQL.
- `operator`: tipo de operador.
- `value`: valores que se van a comparar con el valor `fieldName` de notificación de suscripción.

A continuación se muestra un ejemplo de asignación de estos atributos:

```
{
 "fieldName" : "severity",
 "operator" : "le",
 "value" : context.result.severity
}
```

Argumento: invalidationJsonObject

El `invalidationJsonObject` define lo siguiente:

- `subscriptionField`: suscripción del esquema de GraphQL a invalidar. Se baraja la posibilidad de invalidar una suscripción única, definida como cadena en el `subscriptionField`.
- `payload`: lista de pares clave-valor que se utiliza como entrada para la invalidación de suscripciones si el filtro de invalidación se evalúa como `true` en comparación con sus valores.

  En el siguiente ejemplo se invalida a los clientes suscritos y conectados que utilizan la suscripción `onUserDelete` cuando el filtro de invalidación definido en el solucionador de suscripción se evalúa como `true` en comparación con el valor `payload`.

  ```
  export const request = (ctx) => ({ payload: null });

  export function response(ctx) {
   extensions.invalidateSubscriptions({
    subscriptionField: 'onUserDelete',
    payload: { group: 'Developer', type: 'Full-Time' },
   });
   return ctx.result;
  }
  ```

Aplicaciones auxiliares para XML en util.xml

`util.xml` contiene métodos para ayudar con la conversión de cadena XML.

Lista de utilidades util.xml

`util.xml.toMap(String) : Object`

Convierte una cadena XML a un diccionario.

Ejemplo 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"
      }
    }
}
```

Ejemplo 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`

Convierte una cadena XML en una cadena JSON. Esto es similar a `toMap`, salvo que el resultado es una cadena. Esto resulta útil si se desea convertir directamente y devolver la respuesta XML de un objeto HTTP a JSON. Puede establecer un parámetro booleano opcional para determinar si desea codificar la cadena JSON.

Referencia a la función de solucionador de JavaScript para DynamoDB

La función de DynamoDB de AWS AppSync permite utilizar GraphQL para almacenar y recuperar datos de tablas de Amazon DynamoDB ya existentes en su cuenta. Para funcionar, este solucionador permite mapear una solicitud de GraphQL entrante a una llamada de DynamoDB y, a continuación, mapear la respuesta de DynamoDB a GraphQL. En esta sección se describen los controladores de solicitudes y respuestas para las operaciones de DynamoDB admitidas.

GetItem

La solicitud `GetItem` permite indicar a la función de DynamoDB de AWS AppSync que realice una solicitud `GetItem` a DynamoDB, así como especificar lo siguiente:

- La clave del elemento de DynamoDB
- Si se utiliza una lectura consistente o no.

La solicitud `GetItem` tiene la estructura siguiente:

```
type DynamoDBGetItem = {
  operation: 'GetItem';
  key: { [key: string]: any };
  consistentRead?: ConsistentRead;
  projection?: {
    expression: string;
    expressionNames?: { [key: string]: string };
  };
};
```

Los campos se definen de la siguiente manera:

Campos GetItem

Lista de campos GetItem

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `GetItem`, este valor se debe establecer en `GetItem`. Este valor es obligatorio.

`key`

La clave del elemento de DynamoDB. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`consistentRead`

Indica si se realizará o no una lectura lectura altamente coherente con DynamoDB. Este valor es opcional y de forma predeterminada es `false`.

`projection`

Proyección que se utiliza para especificar los atributos que se devolverán de la operación de DynamoDB. Para obtener más información acerca de las proyecciones, consulte la sección Proyecciones. Este campo es opcional.

El elemento que se devuelve desde DynamoDB se convierte automáticamente a tipos primitivos de GraphQL y JSON, y se encuentra disponible en el resultado del contexto (`context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información sobre los solucionadores de JavaScript, consulte el artículo sobre la descripción general de los solucionadores de JavaScript.

Ejemplo

El siguiente ejemplo es un controlador de solicitudes de función para una consulta de 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
  }
}
```

Para obtener más información sobre la API `GetItem` de DynamoDB, consulte la documentación de la API de DynamoDB.

PutItem

El documento de mapeo de solicitudes `PutItem` permite indicar a la función de DynamoDB de AWS AppSync que realice una solicitud `PutItem` a DynamoDB, así como especificar lo siguiente:

- La clave del elemento de DynamoDB
- El contenido completo del elemento (compuesto de `key` y `attributeValues`).
- Condiciones para que la operación se lleve a cabo correctamente.

La solicitud `PutItem` tiene la estructura siguiente:

```
type DynamoDBPutItemRequest = {
  operation: 'PutItem';
  key: { [key: string]: any };
```

```
  attributeValues: { [key: string]: any};
  condition?: ConditionCheckExpression;
  customPartitionKey?: string;
  populateIndexFields?: boolean;
  _version?: number;
};
```

Los campos se definen de la siguiente manera:

Campos PutItem

Lista de campos PutItem

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `PutItem`, este valor se debe establecer en `PutItem`. Este valor es obligatorio.

`key`

La clave del elemento de DynamoDB. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`attributeValues`

El resto de los atributos del elemento que debe colocarse en DynamoDB. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este campo es opcional.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `PutItem` sobrescribe todas las entradas existentes para dicho elemento. Para obtener más información sobre las condiciones, consulte Expresiones de condición. Este valor es opcional.

`_version`

Valor numérico que representa la última versión conocida de un elemento. Este valor es opcional. Este campo se utiliza para detectar conflictos y solo se admite en orígenes de datos con control de versiones.

`customPartitionKey`

Cuando se habilita, este valor de cadena modifica el formato de los registros `ds_sk` y `ds_pk` que utiliza la tabla Delta Sync una vez habilitado el control de versiones (para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync). Cuando se habilita, también lo hace el procesamiento de la entrada `populateIndexFields`. Este campo es opcional.

`populateIndexFields`

Valor booleano que, cuando se habilita junto con la **`customPartitionKey`**, crea nuevas entradas para cada registro de la tabla Delta Sync, específicamente en las columnas `gsi_ds_pk` y `gsi_ds_sk`. Para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync. Este campo es opcional.

El elemento que se escribe en DynamoDB se convierte automáticamente a los tipos primitivos de GraphQL y JSON, y está disponible en el resultado del contexto (`context.result`).

El elemento que se escribe en DynamoDB se convierte automáticamente a los tipos primitivos de GraphQL y JSON, y está disponible en el resultado del contexto (`context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información sobre los solucionadores de JavaScript, consulte el artículo sobre la descripción general de los solucionadores de JavaScript.

Ejemplo 1

El siguiente ejemplo es un controlador de solicitudes de función para una mutación de GraphQL `updateThing(foo: String!, bar: String!, name: String!, version: Int!)`.

Si no existe ningún elemento con la clave especificada, dicho elemento se crea. Si ya existe un elemento con la clave especificada, dicho elemento se sobrescribe.

```
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),
  };
}
```

Ejemplo 2

El siguiente ejemplo es un controlador de solicitudes de función para una mutación de GraphQL `updateThing(foo: String!, bar: String!, name: String!, expectedVersion: Int!)`.

En este ejemplo se comprueba que el elemento que se encuentra actualmente en DynamoDB tiene el campo `version` establecido en `expectedVersion`.

```
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,
  };
}
```

Para obtener más información sobre la API `PutItem` de DynamoDB, consulte la documentación de la API de DynamoDB.

UpdateItem

La solicitud `UpdateItem` permite indicar a la función de DynamoDB de AWS AppSync que realice una solicitud `UpdateItem` a DynamoDB, así como especificar lo siguiente:

- La clave del elemento de DynamoDB
- Una expresión de actualización que describe cómo se actualiza el elemento de DynamoDB

- Condiciones para que la operación se lleve a cabo correctamente.

La solicitud `UpdateItem` tiene la estructura siguiente:

```
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;
};
```

Los campos se definen de la siguiente manera:

Campos UpdateItem

Lista de campos UpdateItem

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `UpdateItem`, este valor se debe establecer en `UpdateItem`. Este valor es obligatorio.

`key`

La clave del elemento de DynamoDB. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`update`

La sección `update` permite especificar una expresión de actualización que describe cómo se actualiza el elemento en DynamoDB. Para obtener más información sobre el modo de escribir expresiones de actualización, consulte la documentación de DynamoDB UpdateExpressions. Esta sección es obligatoria.

La sección `update` tiene tres componentes:

`expression`

La expresión de actualización. Este valor es obligatorio.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre usado en la `expression` y el valor tiene que ser una cadena que corresponda al nombre de atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con las sustituciones de marcadores de posición de nombre de atributo de expresión que se usen en la `expression`.

`expressionValues`

Las sustituciones de los marcadores de posición de valor de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de valor usado en la `expression` y el valor tiene que ser un valor con tipo. Para obtener más información sobre cómo especificar un "valor con tipo", consulte Sistema de tipos (mapeo de solicitud). Este valor debe especificarse. Este campo es opcional y solo debe rellenarse con sustituciones de los marcadores de posición de valor de atributo de expresión que se usan en la `expression`.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `UpdateItem` actualiza todas las entradas existentes independientemente de su estado actual. Para obtener más información sobre las condiciones, consulte Expresiones de condición. Este valor es opcional.

`_version`

Valor numérico que representa la última versión conocida de un elemento. Este valor es opcional. Este campo se utiliza para detectar conflictos y solo se admite en orígenes de datos con control de versiones.

`customPartitionKey`

Cuando se habilita, este valor de cadena modifica el formato de los registros `ds_sk` y `ds_pk` que utiliza la tabla Delta Sync una vez habilitado el control de versiones (para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para

desarrolladores de AWS AppSync). Cuando se habilita, también lo hace el procesamiento de la entrada `populateIndexFields`. Este campo es opcional.

`populateIndexFields`

Valor booleano que, cuando se habilita junto con la **`customPartitionKey`**, crea nuevas entradas para cada registro de la tabla Delta Sync, específicamente en las columnas `gsi_ds_pk` y `gsi_ds_sk`. Para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync. Este campo es opcional.

El elemento que se actualiza en DynamoDB se convierte automáticamente a los tipos primitivos de GraphQL y JSON, y está disponible en el resultado del contexto (`context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información sobre los solucionadores de JavaScript, consulte el artículo sobre la descripción general de los solucionadores de JavaScript.

Ejemplo 1

El siguiente ejemplo es un controlador de solicitudes de función para la mutación de GraphQL `upvote(id: ID!)`.

En este ejemplo, se han incrementado en 1 los campos `upvotes` y `version` de un elemento de DynamoDB.

```
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 } },
    },
  };
}
```

Ejemplo 2

El siguiente ejemplo es un controlador de solicitudes de función para una mutación de GraphQL `updateItem(id: ID!, title: String, author: String, expectedVersion: Int!)`.

Se trata de un ejemplo complejo que inspecciona los argumentos y genera de manera dinámica la expresión de actualización que solo incluye los argumentos que ha proporcionado el cliente. Por ejemplo, si se omiten `title` y `author`, no se actualizan. Si se especifica un argumento pero su valor es `null`, ese campo se elimina del objeto en DynamoDB. Por último, la operación tiene una condición que comprueba si el elemento que está actualmente en DynamoDB tiene el campo `version` establecido en `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),
    },
  };
}
```

Para obtener más información sobre la API `UpdateItem` de DynamoDB, consulte la documentación de la API de DynamoDB.

DeleteItem

La solicitud `DeleteItem` permite indicar a la función de DynamoDB de AWS AppSync que realice una solicitud `DeleteItem` a DynamoDB, así como especificar lo siguiente:

- La clave del elemento de DynamoDB
- Condiciones para que la operación se lleve a cabo correctamente.

La solicitud `DeleteItem` tiene la estructura siguiente:

```
type DynamoDBDeleteItemRequest = {
  operation: 'DeleteItem';
  key: { [key: string]: any };
```

```
  condition?: ConditionCheckExpression;
  customPartitionKey?: string;
  populateIndexFields?: boolean;
  _version?: number;
};
```

Los campos se definen de la siguiente manera:

Campos DeleteItem

Lista de campos DeleteItem

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `DeleteItem`, este valor se debe establecer en `DeleteItem`. Este valor es obligatorio.

`key`

La clave del elemento de DynamoDB. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `DeleteItem` elimina un elemento independientemente de su estado actual. Para obtener más información sobre las condiciones, consulte Expresiones de condición. Este valor es opcional.

`_version`

Valor numérico que representa la última versión conocida de un elemento. Este valor es opcional. Este campo se utiliza para detectar conflictos y solo se admite en orígenes de datos con control de versiones.

`customPartitionKey`

Cuando se habilita, este valor de cadena modifica el formato de los registros `ds_sk` y `ds_pk` que utiliza la tabla Delta Sync una vez habilitado el control de versiones (para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para

desarrolladores de AWS AppSync). Cuando se habilita, también lo hace el procesamiento de la entrada `populateIndexFields`. Este campo es opcional.

`populateIndexFields`

Valor booleano que, cuando se habilita junto con la **`customPartitionKey`**, crea nuevas entradas para cada registro de la tabla Delta Sync, específicamente en las columnas `gsi_ds_pk` y `gsi_ds_sk`. Para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync. Este campo es opcional.

El elemento que se elimina de DynamoDB se convierte automáticamente a los tipos primitivos de GraphQL y JSON, y está disponible en el resultado del contexto (`context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información sobre los solucionadores de JavaScript, consulte el artículo sobre la descripción general de los solucionadores de JavaScript.

Ejemplo 1

El siguiente ejemplo es un controlador de solicitudes de función para una mutación de GraphQL `deleteItem(id: ID!)`. Si ya existe un elemento con este ID, se eliminará.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'DeleteItem',
    key: util.dynamodb.toMapValues({ id: ctx.args.id }),
  };
}
```

Ejemplo 2

El siguiente ejemplo es un controlador de solicitudes de función para una mutación de GraphQL `deleteItem(id: ID!, expectedVersion: Int!)`. Si ya existe un elemento con este ID, se eliminará, pero solo si su campo `version` está establecido en `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),
  };
}
```

Para obtener más información sobre la API `DeleteItem` de DynamoDB, consulte la documentación de la API de DynamoDB.

Consulta

El objeto de solicitud `Query` permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `Query` a DynamoDB, así como especificar lo siguiente:

- Expresión de clave.
- Qué índice utilizar.
- Todos los filtros adicionales.
- Cuántos elementos deben devolverse.
- Si se utilizarán lecturas consistentes.
- Dirección de consulta (hacia delante o hacia atrás).
- Token de paginación

El objeto de solicitud `Query` tiene la siguiente estructura:

```
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 };
  };
};
```

Los campos se definen de la siguiente manera:

Campos de consulta

Lista de campos de consulta

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `Query`, este valor se debe establecer en `Query`. Este valor es obligatorio.

`query`

La sección `query` permite especificar una expresión de condición de clave que describe qué elementos deben recuperarse de DynamoDB. Para obtener más información sobre cómo escribir expresiones de condición de clave, consulte DynamoDB KeyConditions documentation. Esta sección debe especificarse.

`expression`

La expresión de la consulta. Este campo debe especificarse.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre usado en la `expression` y el valor tiene que ser una cadena que corresponda al nombre de atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con las

sustituciones de marcadores de posición de nombre de atributo de expresión que se usen en la `expression`.

`expressionValues`

Las sustituciones de los marcadores de posición de valor de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de valor usado en la `expression` y el valor tiene que ser un valor con tipo. Para obtener más información sobre cómo especificar un "valor con tipo", consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio. Este campo es opcional y solo debe rellenarse con sustituciones de los marcadores de posición de valor de atributo de expresión que se usan en la `expression`.

`filter`

Un filtro adicional que se puede utilizar para filtrar los resultados de DynamoDB antes de que se devuelvan. Para obtener más información acerca de los filtros, consulte Filtros. Este campo es opcional.

`index`

El nombre del índice que se consultará. La operación de consulta de DynamoDB permite escanear en índices secundarios locales y globales además de hacerlo en el índice de clave principal de una clave hash. Si se especifica, indica a DynamoDB que debe consultar el índice especificado. Si se omite, se consultará el índice de clave principal.

`nextToken`

El token de paginación para continuar una consulta anterior. Se debe obtener de una consulta anterior. Este campo es opcional.

`limit`

Número máximo de elementos que se van a evaluar, que no es necesariamente el número de elementos coincidentes. Este campo es opcional.

`scanIndexForward`

Valor booleano que indica si se debe consultar hacia delante o hacia atrás. Este campo es opcional y de forma predeterminada es `true`.

`consistentRead`

Valor booleano que indica si se utilizarán lecturas coherentes al consultar DynamoDB. Este campo es opcional y de forma predeterminada es `false`.

`select`

De forma predeterminada, el solucionador de DynamoDB de AWS AppSync solo devuelve los atributos que se proyectan en el índice. Si se necesitan más atributos, puede configurar este campo. Este campo es opcional. Los valores admitidos son:

`ALL_ATTRIBUTES`

Devuelve todos los atributos de elementos de la tabla o el índice especificados. Si consulta un índice secundario local, DynamoDB recupera todo el elemento de la tabla principal para cada elemento coincidente en el índice. Si el índice está configurado para proyectar todos los atributos de los elementos, todos los datos se pueden obtener del índice secundario local y no es necesario efectuar una recuperación.

`ALL_PROJECTED_ATTRIBUTES`

Permitido solo al consultar un índice. Recupera todos los atributos que se han proyectado en el índice. Si el índice está configurado para proyectar todos los atributos, este valor de retorno equivale a especificar `ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Devuelve solo los atributos que aparecen en la `expression` de la `projection`. Este valor devuelto equivale a especificar la `expression` de la `projection` sin especificar ningún valor para `Select`.

`projection`

Proyección que se utiliza para especificar los atributos que se devolverán de la operación de DynamoDB. Para obtener más información acerca de las proyecciones, consulte la sección Proyecciones. Este campo es opcional.

Los resultados de DynamoDB se convierten automáticamente a los tipos primitivos de GraphQL y JSON, y están disponibles en el resultado del contexto (`context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información sobre los solucionadores de JavaScript, consulte el artículo sobre la descripción general de los solucionadores de JavaScript.

Los resultados tienen la estructura siguiente:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Los campos se definen de la siguiente manera:

`items`

Una lista que contiene los elementos que devuelve la consulta de DynamoDB.

`nextToken`

Si hubiera más resultados, `nextToken` contiene un token de paginación que puede usar en otra solicitud. Observe que AWS AppSync cifra y oculta el token de paginación devuelto de DynamoDB. Esto evita que los datos de las tablas se filtren accidentalmente al intermediario. Además, tenga en cuenta que estos tokens de paginación no se pueden utilizar en diferentes funciones o solucionadores.

`scannedCount`

El número de elementos que coincidían con la expresión de condición de consulta antes de aplicar una expresión de filtro (si la hay).

Ejemplo

El siguiente ejemplo es un controlador de solicitudes de función para una consulta de GraphQL `getPosts(owner: ID!)`.

En este ejemplo, se consulta un índice secundario global en una tabla para que devuelva todas las publicaciones que pertenecen al ID especificado.

```
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',
  };
}
```

Para obtener más información sobre la API `Query` de DynamoDB, consulte la documentación de la API de DynamoDB.

Examen

La solicitud `Scan` permite indicar a la función de DynamoDB de AWS AppSync que realice una solicitud `Scan` a DynamoDB, así como especificar lo siguiente:

- Un filtro para excluir resultados
- Qué índice utilizar.
- Cuántos elementos deben devolverse.
- Si se utilizarán lecturas consistentes.
- Token de paginación
- Exámenes en paralelo

El objeto de solicitud `Scan` tiene la siguiente estructura:

```
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 };
  };
```

```
};
```

Los campos se definen de la siguiente manera:

Campos de Scan

Lista de campos de Scan

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `Scan`, este valor se debe establecer en `Scan`. Este valor es obligatorio.

`filter`

Un filtro que se puede utilizar para filtrar los resultados de DynamoDB antes de que se devuelvan. Para obtener más información acerca de los filtros, consulte Filtros. Este campo es opcional.

`index`

El nombre del índice que se consultará. La operación de consulta de DynamoDB permite escanear en índices secundarios locales y globales además de hacerlo en el índice de clave principal de una clave hash. Si se especifica, indica a DynamoDB que debe consultar el índice especificado. Si se omite, se consultará el índice de clave principal.

`limit`

El número máximo de elementos que se evalúan en una sola vez. Este campo es opcional.

`consistentRead`

Valor booleano que indica si se utilizarán lecturas coherentes al consultar DynamoDB. Este campo es opcional y de forma predeterminada es `false`.

`nextToken`

El token de paginación para continuar una consulta anterior. Se debe obtener de una consulta anterior. Este campo es opcional.

`select`

De forma predeterminada, la función de DynamoDB de AWS AppSync solo devuelve los atributos que se proyecten en el índice. Si se necesitan más atributos, este campo se puede configurar. Este campo es opcional. Los valores admitidos son:

`ALL_ATTRIBUTES`

Devuelve todos los atributos de elementos de la tabla o el índice especificados. Si consulta un índice secundario local, DynamoDB recupera todo el elemento de la tabla principal para cada elemento coincidente en el índice. Si el índice está configurado para proyectar todos los atributos de los elementos, todos los datos se pueden obtener del índice secundario local y no es necesario efectuar una recuperación.

`ALL_PROJECTED_ATTRIBUTES`

Permitido solo al consultar un índice. Recupera todos los atributos que se han proyectado en el índice. Si el índice está configurado para proyectar todos los atributos, este valor de retorno equivale a especificar `ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Devuelve solo los atributos que aparecen en la `expression` de la `projection`. Este valor devuelto equivale a especificar la `expression` de la `projection` sin especificar ningún valor para `Select`.

`totalSegments`

El número de segmentos para dividir en particiones la tabla al realizar un examen paralelo. Este campo es opcional, pero debe especificarse si se indica `segment`.

`segment`

El segmento de tabla de esta operación al realizar un examen en paralelo. Este campo es opcional, pero debe especificarse si se indica `totalSegments`.

`projection`

Proyección que se utiliza para especificar los atributos que se devolverán de la operación de DynamoDB. Para obtener más información acerca de las proyecciones, consulte la sección Proyecciones. Este campo es opcional.

Los resultados que el examen de DynamoDB devuelve se convierten automáticamente a los tipos primitivos de GraphQL y JSON, y están disponibles en el resultado del contexto (`context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información sobre los solucionadores de JavaScript, consulte el artículo sobre la descripción general de los solucionadores de JavaScript.

Los resultados tienen la estructura siguiente:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Los campos se definen de la siguiente manera:

`items`

Una lista que contiene los elementos que devuelve el análisis de DynamoDB.

`nextToken`

Si hubiera más resultados, `nextToken` contiene un token de paginación que puede usar en otra solicitud. AWS AppSync cifra y oculta el token de paginación devuelto de DynamoDB. Esto evita que los datos de las tablas se filtren accidentalmente al intermediario. Además, estos tokens de paginación no se pueden utilizar en diferentes funciones o solucionadores.

`scannedCount`

El número de elementos que DynamoDB ha recuperado antes de aplicar una expresión de filtro (en caso de incluirse).

Ejemplo 1

El siguiente ejemplo es un controlador de solicitudes de función para la consulta de GraphQL: `allPosts`.

En este ejemplo, se devuelven todas las entradas de la tabla.

```
export function request(ctx) {
  return { operation: 'Scan' };
}
```

Ejemplo 2

El siguiente ejemplo es un controlador de solicitudes de función para la consulta de GraphQL: `postsMatching(title: String!)`.

En este ejemplo, todas las entradas de la tabla se devuelven donde el título comienza con el argumento `title`.

```
export function request(ctx) {
  const { title } = ctx.args;
  const filter = { filter: { beginsWith: title } };
  return {
    operation: 'Scan',
    filter: JSON.parse(util.transform.toDynamoDBFilterExpression(filter)),
  };
}
```

Para obtener más información sobre la API `Scan` de DynamoDB, consulte la documentación de la API de DynamoDB.

Sync (Sincronizar)

El objeto de solicitud `Sync` permite recuperar todos los resultados de una tabla de DynamoDB y, a continuación, recibir tan solo los datos alterados desde la última consulta (las actualizaciones delta). Únicamente se pueden realizar solicitudes `Sync` a orígenes de datos con control de versiones de DynamoDB. Puede especificar lo siguiente:

- Un filtro para excluir resultados
- Cuántos elementos deben devolverse.
- Token de paginación
- Cuando se inició la última operación `Sync`

El objeto de solicitud `Sync` tiene la siguiente estructura:

```
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 };
```

```
  };
};
```

Los campos se definen de la siguiente manera:

Campos de Sync

Lista de campos de Sync

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación `Sync` de , en este valor debe establecerse `Sync`. Este valor es obligatorio.

`filter`

Un filtro que se puede utilizar para filtrar los resultados de DynamoDB antes de que se devuelvan. Para obtener más información acerca de los filtros, consulte Filtros. Este campo es opcional.

`limit`

El número máximo de elementos que se evalúan en una sola vez. Este campo es opcional. Si se omite, el límite predeterminado se establecerá en `100` elementos. El valor máximo de este campo son `1000` elementos.

`nextToken`

El token de paginación para continuar una consulta anterior. Se debe obtener de una consulta anterior. Este campo es opcional.

`lastSync`

El momento, en milisegundos transcurridos desde la fecha de inicio, en el que comenzó la última operación `Sync` que se ha realizado correctamente. Si se especifica, solo se devuelven los elementos que han cambiado después de `lastSync`. Este campo es opcional y solo debe rellenarse después de haber recuperado todas las páginas de una operación `Sync` inicial. Si se omite, se devolverán los resultados de la tabla Base; de lo contrario, se devolverán los resultados de la tabla Delta.

`basePartitionKey`

Clave de partición de la tabla Base utilizada al realizar una operación `Sync`. Este campo permite realizar una operación `Sync` cuando la tabla utiliza una clave de partición personalizada. Se trata de un campo opcional.

`deltaIndexName`

Índice utilizado para la operación `Sync`. Este índice es necesario para habilitar una operación `Sync` en toda la tabla de almacenamiento Delta cuando la tabla utiliza una clave de partición personalizada. La operación `Sync` se realizará en el GSI (creado en `gsi_ds_pk` y `gsi_ds_sk`). Este campo es opcional.

Los resultados que la sincronización de DynamoDB devuelve se convierten automáticamente a los tipos primitivos de GraphQL y JSON, y están disponibles en el resultado del contexto (`context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información sobre los solucionadores de JavaScript, consulte el artículo sobre la descripción general de los solucionadores de JavaScript.

Los resultados tienen la estructura siguiente:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10,
    startedAt = 1550000000000
}
```

Los campos se definen de la siguiente manera:

`items`

Una lista que contiene los elementos que devuelve la sincronización.

`nextToken`

Si hubiera más resultados, `nextToken` contiene un token de paginación que puede usar en otra solicitud. AWS AppSync cifra y oculta el token de paginación devuelto de DynamoDB. Esto evita que los datos de las tablas se filtren accidentalmente al intermediario. Además, estos tokens de paginación no se pueden utilizar en diferentes funciones o solucionadores.

`scannedCount`

El número de elementos que DynamoDB ha recuperado antes de aplicar una expresión de filtro (en caso de incluirse).

`startedAt`

El momento, en milisegundos transcurridos desde la fecha de inicio, en el que comenzó la operación de sincronización que puede almacenar localmente y usar en otra solicitud como argumento de `lastSync`. Si se incluyó un token de paginación en la solicitud, este valor será el mismo que el devuelto por la solicitud para la primera página de resultados.

Ejemplo 1

El siguiente ejemplo es un controlador de solicitudes de función para la consulta de GraphQL: `syncPosts(nextToken: String, lastSync: AWSTimestamp)`.

En este ejemplo, si se omite `lastSync`, se devuelven todas las entradas de la tabla base. Si se proporciona `lastSync`, solo se devuelven las entradas de la tabla Delta Sync que han cambiado desde `lastSync`.

```
export function request(ctx) {
  const { nextToken, lastSync } = ctx.args;
  return { operation: 'Sync', limit: 100, nextToken, lastSync };
}
```

BatchGetItem

El objeto de solicitud `BatchGetItem` permite indicar a la función de DynamoDB de AWS AppSync que realice una solicitud `BatchGetItem` a DynamoDB para recuperar varios elementos, potencialmente en varias tablas. Para este objeto de solicitud, debe especificar lo siguiente:

- Los nombres de tabla de los que recuperar los elementos
- Las claves de los elementos que recuperar de cada tabla

Se aplican los límites de `BatchGetItem` de DynamoDB y no puede proporcionar ninguna expresión de condición.

El objeto de solicitud `BatchGetItem` tiene la siguiente estructura:

```
type DynamoDBBatchGetItemRequest = {
  operation: 'BatchGetItem';
  tables: {
    [tableName: string]: {
      keys: { [key: string]: any }[];
      consistentRead?: boolean;
      projection?: {
        expression: string;
        expressionNames?: { [key: string]: string };
      };
    };
  };
};
```

Los campos se definen de la siguiente manera:

Campos BatchGetItem

Lista de campos BatchGetItem

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `BatchGetItem`, este valor se debe establecer en `BatchGetItem`. Este valor es obligatorio.

`tables`

Las tablas de DynamoDB de las que recuperar los elementos. El valor es un mapa en el que se especifican los nombres de las tablas como claves del mapa. Al menos debe proporcionarse una tabla. Este valor `tables` es obligatorio.

`keys`

Lista de claves de DynamoDB que representan la clave principal de los elementos que se van a recuperar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud).

`consistentRead`

Si se utiliza una lectura consistente a la hora de ejecutar una operación GetItem. Este valor es opcional y de forma predeterminada es falso.

`projection`

Proyección que se utiliza para especificar los atributos que se devolverán de la operación de DynamoDB. Para obtener más información acerca de las proyecciones, consulte la sección Proyecciones. Este campo es opcional.

Cosas que tener en cuenta:

- Si un elemento no se ha recuperado de la tabla, aparece un elemento nulo en el bloque de datos para esa tabla.
- Los resultados de invocación se ordenan por tabla, según el orden en el que se hayan proporcionado dentro del objeto de solicitud.
- Cada comando `Get` dentro de un `BatchGetItem` es atómico, sin embargo, un lote se puede procesar parcialmente. Si un lote se procesa parcialmente debido a un error, la claves sin procesar se devuelven como parte del resultado de invocación dentro del bloque unprocessedKeys.
- `BatchGetItem` está limitado a 100 claves.

Para el siguiente controlador de solicitudes de función de ejemplo:

```
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 })],
    },
  };
}
```

El resultado de invocación disponible en `ctx.result` es el siguiente:

```
{
   "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": []
    }
}
```

El `ctx.error` contiene detalles acerca del error. Está garantizado que los datos de claves, unprocessedKeys y cada clave de tabla que se proporcionó en el resultado del objeto de solicitud de función estarán presentes en el resultado de invocación. Los elementos que se han eliminado aparecen en el bloque de datos. Los elementos que no se hayan procesado se marcan como null dentro del bloque de datos y se colocan dentro del bloque unprocessedKeys.

BatchDeleteItem

El objeto de solicitud `BatchDeleteItem` permite indicar a la función de DynamoDB de AWS AppSync que realice una solicitud `BatchWriteItem` a DynamoDB para eliminar varios elementos, potencialmente en varias tablas. Para este objeto de solicitud, debe especificar lo siguiente:

- Los nombres de tabla de los que eliminar los elementos
- Las claves de los elementos que eliminar de cada tabla

Se aplican los límites de `BatchWriteItem` de DynamoDB y no puede proporcionar ninguna expresión de condición.

El objeto de solicitud `BatchDeleteItem` tiene la siguiente estructura:

```
type DynamoDBBatchDeleteItemRequest = {
  operation: 'BatchDeleteItem';
```

```
  tables: {
    [tableName: string]: { [key: string]: any }[];
  };
};
```

Los campos se definen de la siguiente manera:

Campos BatchDeleteItem

Lista de campos BatchDeleteItem

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `BatchDeleteItem`, este valor se debe establecer en `BatchDeleteItem`. Este valor es obligatorio.

`tables`

Las tablas de DynamoDB de las que eliminar los elementos. Cada tabla es una lista de claves de DynamoDB que representan la clave principal de los elementos que se van a eliminar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Al menos debe proporcionarse una tabla. El valor `tables` es obligatorio.

Cosas que tener en cuenta:

- Al contrario que en la operación `DeleteItem`, el elemento completamente eliminado no se devuelve en la respuesta. Solo se devuelve la clave pasada.
- Si un elemento no se ha eliminado de la tabla, aparece un elemento nulo en el bloque de datos para esa tabla.
- Los resultados de invocación se ordenan por tabla, según el orden en el que se hayan proporcionado dentro del objeto de solicitud.
- Cada comando `Delete` dentro de un `BatchDeleteItem` es atómico. Sin embargo, un lote puede procesarse parcialmente. Si un lote se procesa parcialmente debido a un error, la claves sin procesar se devuelven como parte del resultado de invocación dentro del bloque unprocessedKeys.
- `BatchDeleteItem` está limitado a 25 claves.

Para el siguiente controlador de solicitudes de función de ejemplo:

```
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 })],
    },
  };
}
```

El resultado de invocación disponible en `ctx.result` es el siguiente:

```
{
   "data": {
     "authors": [null],
     "posts": [
        // Was deleted
        {
          "authorId": "a1",
          "postId": "p2"
        }
     ]
   },
   "unprocessedKeys": {
     "authors": [
        // This key was not processed due to an error
        {
          "authorId": "a1"
        }
      ],
     "posts": []
   }
}
```

El `ctx.error` contiene detalles acerca del error. Está garantizado que los datos de claves, unprocessedKeys y cada clave de tabla que se proporcionó en el objeto de solicitud de función estarán presentes en el resultado de invocación. Los elementos que se han eliminado están

presentes en el bloque de datos. Los elementos que no se hayan procesado se marcan como null dentro del bloque de datos y se colocan dentro del bloque unprocessedKeys.

BatchPutItem

El objeto de solicitud `BatchPutItem` permite indicar a la función de DynamoDB de AWS AppSync que realice una solicitud `BatchWriteItem` a DynamoDB para colocar varios elementos, potencialmente en varias tablas. Para este objeto de solicitud, debe especificar lo siguiente:

- Los nombres de tabla en los que poner los elementos
- Los elementos completos que poner en cada tabla

Se aplican los límites de `BatchWriteItem` de DynamoDB y no puede proporcionar ninguna expresión de condición.

El objeto de solicitud `BatchPutItem` tiene la siguiente estructura:

```
type DynamoDBBatchPutItemRequest = {
  operation: 'BatchPutItem';
  tables: {
    [tableName: string]: { [key: string]: any}[];
  };
};
```

Los campos se definen de la siguiente manera:

Campos BatchPutItem

Lista de campos BatchPutItem

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `BatchPutItem`, este valor se debe establecer en `BatchPutItem`. Este valor es obligatorio.

`tables`

Las tablas de DynamoDB en las que poner los elementos. Cada entrada de la tabla representa una lista de elementos de DynamoDB que insertar para esta tabla específica. Al menos debe proporcionarse una tabla. Este valor es obligatorio.

Cosas que tener en cuenta:

- Los elementos totalmente insertados se devuelven en la respuesta, si la operación se realiza correctamente.
- Si un elemento no se ha insertado en la tabla, aparece un elemento nulo en el bloque de datos para esa tabla.
- Los elementos insertados se ordenan por tabla, según el orden en el que se hayan proporcionado dentro del objeto de solicitud.
- Cada comando `Put` dentro de un `BatchPutItem` es atómico, sin embargo, un lote se puede procesar parcialmente. Si un lote se procesa parcialmente debido a un error, la claves sin procesar se devuelven como parte del resultado de invocación dentro del bloque unprocessedKeys.
- `BatchPutItem` está limitado a 25 elementos.

Para el siguiente controlador de solicitudes de función de ejemplo:

```
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 })],
    },
  };
}
```

El resultado de invocación disponible en `ctx.result` es el siguiente:

```
{
   "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": []
   }
}
```

El `ctx.error` contiene detalles acerca del error. Está garantizado que los datos de claves, unprocessedItems y cada clave de tabla que se proporcionó en el objeto de solicitud estarán presentes en el resultado de invocación. Los elementos que se han insertado están en el bloque de datos. Los elementos que no se hayan procesado se marcan como null dentro del bloque de datos y se colocan dentro del bloque unprocessedItems.

TransactGetItems

El objeto de solicitud `TransactGetItems` permite indicar a la función de DynamoDB de AWS AppSync que realice una solicitud `TransactGetItems` a DynamoDB para recuperar varios elementos, potencialmente en varias tablas. Para este objeto de solicitud, debe especificar lo siguiente:

- El nombre de la tabla de cada elemento de solicitud de la que se va a recuperar el elemento
- La clave de cada elemento de solicitud que se va a recuperar de cada tabla

Se aplican los límites de `TransactGetItems` de DynamoDB y no puede proporcionar ninguna expresión de condición.

El objeto de solicitud `TransactGetItems` tiene la siguiente estructura:

```
type DynamoDBTransactGetItemsRequest = {
  operation: 'TransactGetItems';
```

```
  transactItems: { table: string; key: { [key: string]: any }; projection?:
 { expression: string; expressionNames?: { [key: string]: string }; }[];
  };
};
```

Los campos se definen de la siguiente manera:

Campos TransactGetItems

Lista de campos TransactGetItems

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `TransactGetItems`, este valor se debe establecer en `TransactGetItems`. Este valor es obligatorio.

`transactItems`

Los elementos de solicitud que se van a incluir. El valor es una matriz de elementos de solicitud. Se debe proporcionar al menos un elemento de solicitud. Este valor `transactItems` es obligatorio.

`table`

La tabla de DynamoDB de la que se va a recuperar el elemento. El valor es una cadena con el nombre de la tabla. Este valor `table` es obligatorio.

`key`

La clave de DynamoDB que representa la clave principal del elemento que se desea recuperar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud).

`projection`

Proyección que se utiliza para especificar los atributos que se devolverán de la operación de DynamoDB. Para obtener más información acerca de las proyecciones, consulte la sección Proyecciones. Este campo es opcional.

Cosas que tener en cuenta:

- Si una transacción se realiza correctamente, el orden de los elementos recuperados en el bloque `items` será el mismo que el orden de los elementos de solicitud.
- Las transacciones se realizan en régimen de todo o nada. Si algún elemento de solicitud causa un error, no se realizará la transacción completa y se devolverán los detalles del error.
- El hecho de no poder recuperar un elemento de solicitud no es un error. En su lugar, aparece un elemento null en el bloque items (elementos) en la posición correspondiente.
- Si el error de una transacción es TransactionCanceledException, se rellenará el bloque `cancellationReasons`. El orden de los motivos de cancelación en el bloque `cancellationReasons` será el mismo que el orden de los elementos de solicitud.
- `TransactGetItems` está limitado a 25 elementos de solicitud.

Para el siguiente controlador de solicitudes de función de ejemplo:

```
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 }),
      },
    ],
  };
}
```

Si la transacción se realiza correctamente y solo se recupera el primer elemento solicitado, el resultado de la invocación disponible en `ctx.result` es el siguiente:

```
{
    "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
}
```

Si la transacción no se realiza correctamente debido a la excepción TransactionCanceledException causada por el primer elemento de solicitud, el resultado de la invocación disponible en `ctx.result` es el siguiente:

```
{
    "items": null,
    "cancellationReasons": [
        {
            "type":"Sample error type",
            "message":"Sample error message"
        },
        {
            "type":"None",
            "message":"None"
        }
    ]
}
```

El `ctx.error` contiene detalles acerca del error. Los valores de items de claves y cancellationReasons estarán presentes sin duda en `ctx.result`.

TransactWriteItems

El objeto de solicitud `TransactWriteItems` permite indicar a la función de DynamoDB de AWS AppSync que realice una solicitud `TransactWriteItems` a DynamoDB para escribir varios elementos, potencialmente en varias tablas. Para este objeto de solicitud, debe especificar lo siguiente:

- El nombre de la tabla de destino de cada elemento de solicitud
- La operación de cada elemento de solicitud que se va a realizar. Hay cuatro tipos de operaciones compatibles: PutItem, UpdateItem, DeleteItem y ConditionCheck.

- La clave de cada elemento de solicitud que se va a escribir

Se aplican los límites de `TransactWriteItems` de DynamoDB.

El objeto de solicitud `TransactWriteItems` tiene la siguiente estructura:

```
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;
};
```

Campos TransactWriteItems

Lista de campos TransactWriteItems

Los campos se definen de la siguiente manera:

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `TransactWriteItems`, este valor se debe establecer en `TransactWriteItems`. Este valor es obligatorio.

`transactItems`

Los elementos de solicitud que se van a incluir. El valor es una matriz de elementos de solicitud. Se debe proporcionar al menos un elemento de solicitud. Este valor `transactItems` es obligatorio.

Para `PutItem`, los campos se definen de la siguiente manera:

`table`

La tabla de DynamoDB de destino. El valor es una cadena con el nombre de la tabla. Este valor `table` es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `PutItem`, este valor se debe establecer en `PutItem`. Este valor es obligatorio.

`key`

La clave de DynamoDB que representa la clave principal del elemento que se desea colocar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`attributeValues`

El resto de los atributos del elemento que debe colocarse en DynamoDB. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este campo es opcional.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `PutItem` sobrescribe todas las entradas existentes para dicho elemento. Puede especificar si desea recuperar el elemento existente cuando se produzca un error en la comprobación de condiciones. Para obtener más información acerca de las condiciones transaccionales, consulte Expresiones de condición de transacción. Este valor es opcional.

Para `UpdateItem`, los campos se definen de la siguiente manera:

`table`

La tabla de DynamoDB que se va a actualizar. El valor es una cadena con el nombre de la tabla. Este valor `table` es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `UpdateItem`, este valor se debe establecer en `UpdateItem`. Este valor es obligatorio.

`key`

La clave de DynamoDB que representa la clave principal del elemento que se va a actualizar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`update`

La sección `update` permite especificar una expresión de actualización que describe cómo se actualiza el elemento en DynamoDB. Para obtener más información sobre el modo de escribir expresiones de actualización, consulte la documentación de DynamoDB UpdateExpressions. Esta sección es obligatoria.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `UpdateItem` actualiza todas las entradas existentes independientemente de su estado actual. Puede especificar si desea recuperar el elemento existente cuando se produzca un error en la comprobación de condiciones. Para obtener más información acerca de las condiciones transaccionales, consulte Expresiones de condición de transacción. Este valor es opcional.

Para `DeleteItem`, los campos se definen de la siguiente manera:

`table`

La tabla de DynamoDB en la que se elimina el elemento. El valor es una cadena con el nombre de la tabla. Este valor `table` es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `DeleteItem`, este valor se debe establecer en `DeleteItem`. Este valor es obligatorio.

`key`

La clave de DynamoDB que representa la clave principal del elemento que se desea eliminar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `DeleteItem` elimina un elemento independientemente de su estado actual. Puede especificar si desea recuperar el elemento existente cuando se produzca un error en la comprobación de condiciones. Para obtener más información acerca de las condiciones transaccionales, consulte Expresiones de condición de transacción. Este valor es opcional.

Para `ConditionCheck`, los campos se definen de la siguiente manera:

`table`

La tabla de DynamoDB en la que se comprueba la condición. El valor es una cadena con el nombre de la tabla. Este valor `table` es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `ConditionCheck`, este valor se debe establecer en `ConditionCheck`. Este valor es obligatorio.

`key`

La clave de DynamoDB que representa la clave principal del elemento que hay que someter a una comprobación de condición. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Puede especificar si desea recuperar el elemento existente cuando se produzca un error en la comprobación de condiciones. Para obtener más información acerca de las condiciones transaccionales, consulte Expresiones de condición de transacción. Este valor es obligatorio.

Cosas que tener en cuenta:

- Solo las claves de los elementos de solicitud se devuelven en la respuesta, si se realiza correctamente. El orden de las claves será el mismo que el orden de los elementos de solicitud.
- Las transacciones se realizan en régimen de todo o nada. Si algún elemento de solicitud causa un error, no se realizará la transacción completa y se devolverán los detalles del error.
- No se pueden dirigir dos elementos de solicitud al mismo elemento. De lo contrario, causarán un error de TransactionCanceledException.
- Si el error de una transacción es TransactionCanceledException, se rellenará el bloque `cancellationReasons`. Si se produce un error en la comprobación de condición de un elemento de solicitud y no se ha especificado que `returnValuesOnConditionCheckFailure` sea `false`, el elemento existente en la tabla se recuperará y almacenará en `item` en la posición correspondiente del bloque `cancellationReasons`.

- `TransactWriteItems` está limitado a 25 elementos de solicitud.

Para el siguiente controlador de solicitudes de función de ejemplo:

```
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 }),
        },
      },
    ],
  };
}
```

Si la transacción se realiza correctamente, el resultado de la invocación disponible en `ctx.result` es el siguiente:

```
{
    "keys": [
       // Key of the PutItem request
       {
           "post_id": "p1",
       },
```

```
        // Key of the UpdateItem request
        {
            "author_id": "a1"
        }
    ],
    "cancellationReasons": null
}
```

Si la transacción no se realiza correctamente debido a un error de comprobación de condición de la solicitud `PutItem`, el resultado de la invocación disponible en `ctx.result` es el siguiente:

```
{
    "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"
        }
    ]
}
```

El `ctx.error` contiene detalles acerca del error. Los valores de keys y cancellationReasons estarán presentes sin duda en `ctx.result`.

Sistema de tipos (mapeo de solicitud)

Cuando se utiliza la función de DynamoDB de AWS AppSync para llamar a sus tablas de DynamoDB, AWS AppSync debe conocer el tipo de cada valor que se va a utilizar en dicha llamada. Esto se debe a que DynamoDB admite más tipos primitivos que los disponibles en GraphQL o JSON (por ejemplo, conjuntos y datos binarios). AWS AppSync necesita indicaciones para hacer conversiones entre GraphQL y DynamoDB; de lo contrario, tendría que suponer cómo se estructuran los datos de la tabla.

Para obtener más información sobre los tipos de datos de DynamoDB, consulte la documentación relativa a los descriptores de tipos de datos y los tipos de datos de DynamoDB.

Un valor de DynamoDB se representa mediante un objeto JSON que contiene un único par de clave-valor. La clave especifica el tipo de DynamoDB y el valor especifica el valor en sí. En el siguiente ejemplo, la clave S indica que el valor es una cadena y el valor `identifier` es el valor de la cadena en sí.

```
{ "S" : "identifier" }
```

Tenga en cuenta que el objeto JSON no puede tener más de un par de clave-valor. Si se especifica más de un par de clave-valor, el objeto de solicitud no se analizará.

Siempre que necesite especificar un valor en un objeto de solicitud, deberá usar un valor de DynamoDB. Entre los lugares donde necesitará realizar esta operación figuran: las secciones `key` y `attributeValue`, y la sección `expressionValues` de secciones de expresión. En el siguiente ejemplo, el valor de cadena de DynamoDB `identifier` se asigna al campo `id` de una sección `key` (puede que en un objeto de solicitud `GetItem`).

```
"key" : {
   "id" : { "S" : "identifier" }
}
```

Tipos admitidos

AWS AppSync admite los siguientes tipos escalares, de documento y de conjunto de DynamoDB:

Tipo cadena **S**

Valor de cadena único. Un valor de cadena de DynamoDB se indica de la siguiente manera:

```
{ "S" : "some string" }
```

Ejemplo de uso:

```
"key" : {
   "id" : { "S" : "some string" }
}
```

Tipo conjunto de cadenas **SS**

Conjunto de valores de cadena. Un valor de conjunto de cadenas de DynamoDB se indica de la siguiente manera:

```
{ "SS" : [ "first value", "second value", ... ] }
```

Ejemplo de uso:

```
"attributeValues" : {
   "phoneNumbers" : { "SS" : [ "+1 555 123 4567", "+1 555 234 5678" ] }
}
```

Tipo número **N**

Valor numérico único. Un valor de número de DynamoDB se indica de la siguiente manera:

```
{ "N" : 1234 }
```

Ejemplo de uso:

```
"expressionValues" : {
   ":expectedVersion" : { "N" : 1 }
}
```

Tipo conjunto de números **NS**

Conjunto de valores de número. Un valor de conjunto de números de DynamoDB se indica de la siguiente manera:

```
{ "NS" : [ 1, 2.3, 4 ... ] }
```

Ejemplo de uso:

```
"attributeValues" : {
   "sensorReadings" : { "NS" : [ 67.8, 12.2, 70 ] }
}
```

Tipo binario **B**

Valor binario. Un valor binario de DynamoDB se indica de la siguiente manera:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

Observe que el valor es en realidad una cadena que contiene la representación codificada en base64 de los datos binarios. AWS AppSync descodifica esta cadena de nuevo a su valor binario antes de enviarlo a DynamoDB. AWS AppSync utiliza el esquema de descodificación base64 como se define en RFC 2045. Cualquier carácter que no esté en el alfabeto base64 no se tiene en cuenta.

Ejemplo de uso:

```
"attributeValues" : {
   "binaryMessage" : { "B" : "SGVsbG8sIFdvcmxkIQo=" }
}
```

Tipo conjunto binario **BS**

Conjunto de valores binarios. Un valor de conjunto binario de DynamoDB se indica de la siguiente manera:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

Observe que el valor es en realidad una cadena que contiene la representación codificada en base64 de los datos binarios. AWS AppSync descodifica esta cadena de nuevo a su valor binario antes de enviarlo a DynamoDB. AWS AppSync utiliza el esquema de descodificación base64 como se define en RFC 2045: cualquier carácter que no esté en el alfabeto base64 no se tiene en cuenta.

Ejemplo de uso:

```
"attributeValues" : {
   "binaryMessages" : { "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ] }
}
```

Tipo booleano **BOOL**

Un valor booleano. Un valor booleano de DynamoDB se indica de la siguiente manera:

```
{ "BOOL" : true }
```

Observe que solo los valores `true` y `false` son válidos.

Ejemplo de uso:

```
"attributeValues" : {
   "orderComplete" : { "BOOL" : false }
}
```

Tipo lista **L**

Lista del resto de valores de DynamoDB admitidos. Un valor de lista de DynamoDB se indica de la siguiente manera:

```
{ "L" : [ ... ] }
```

Observe que se trata de un valor compuesto, y que la lista puede contener cero o más de cualquiera de los valores de DynamoDB admitidos (incluidas otras listas). La lista también puede contener una combinación de diferentes tipos.

Ejemplo de uso:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

Tipo de mapa **M**

Representa una colección sin ordenar de pares de clave-valor de otros valores de DynamoDB admitidos. Un valor de mapa de DynamoDB se indica de la siguiente manera:

```
{ "M" : { ... } }
```

Observe que un mapa puede contener cero o más pares clave-valor. La clave tiene que ser una cadena y el valor puede ser cualquier valor de DynamoDB admitido (incluidos otros mapas). El mapa también puede contener una combinación de diferentes tipos.

Ejemplo de uso:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
```

```
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

Tipo nulo **`NULL`**

Valor nulo. Un valor nulo de DynamoDB se indica de la siguiente manera:

```
{ "NULL" : null }
```

Ejemplo de uso:

```
"attributeValues" : {
   "phoneNumbers" : { "NULL" : null }
}
```

Para obtener más información sobre cada tipo, consulte la documentación de DynamoDB.

Sistema de tipos (mapeo de respuestas)

Cuando recibe una respuesta de DynamoDB, AWS AppSync la convierte automáticamente a los tipos primitivos de GraphQL y JSON. Cada atributo de DynamoDB se descodifica y se devuelve en el contexto del controlador de respuestas.

Por ejemplo si DynamoDB devuelve lo siguiente:

```
{
    "id" : { "S" : "1234" },
    "name" : { "S" : "Nadia" },
    "age" : { "N" : 25 }
}
```

Cuando el solucionador de canalización devuelve el resultado, AWS AppSync lo convierte a los tipos de GraphQL y JSON como:

```
{
    "id" : "1234",
    "name" : "Nadia",
    "age" : 25
```

```
}
```

En esta sección se explica cómo AWS AppSync convierte los tipos escalares, de documento y de conjunto de DynamoDB indicados:

Tipo cadena **S**

Valor de cadena único. Se devuelve un valor de cadena de DynamoDB en forma de cadena.

Por ejemplo, si DynamoDB devuelve el siguiente valor de cadena de DynamoDB:

```
{ "S" : "some string" }
```

AWS AppSync lo convierte en una cadena:

```
"some string"
```

Tipo conjunto de cadenas **SS**

Conjunto de valores de cadena. Un valor de conjunto de cadenas de DynamoDB se devuelve como una lista de cadenas.

Por ejemplo, si DynamoDB devuelve el siguiente valor de conjunto de cadenas de DynamoDB:

```
{ "SS" : [ "first value", "second value", ... ] }
```

AWS AppSync lo convierte en una lista de cadenas:

```
[ "+1 555 123 4567", "+1 555 234 5678" ]
```

Tipo número **N**

Valor numérico único. Un valor de número de DynamoDB se devuelve como número.

Por ejemplo, si DynamoDB devuelve el siguiente valor de número de DynamoDB:

```
{ "N" : 1234 }
```

AWS AppSync lo convierte en un número:

```
1234
```

Tipo conjunto de números **NS**

Conjunto de valores de número. Un valor de conjunto de números de DynamoDB se devuelve como una lista de números.

Por ejemplo, si DynamoDB devuelve el siguiente valor de conjunto de números de DynamoDB:

```
{ "NS" : [ 67.8, 12.2, 70 ] }
```

AWS AppSync lo convierte en una lista de números:

```
[ 67.8, 12.2, 70 ]
```

Tipo binario **B**

Valor binario. Un valor binario de DynamoDB se devuelve en forma de cadena que contiene la representación base64 de dicho valor.

Por ejemplo, si DynamoDB devuelve el siguiente valor binario de DynamoDB:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

AWS AppSync lo convierte en una cadena con la representación base64 del valor:

```
"SGVsbG8sIFdvcmxkIQo="
```

Observe que los datos binarios se codifican con el esquema base64 como se especifica en RFC 4648 y en RFC 2045.

Tipo conjunto binario **BS**

Conjunto de valores binarios. Un valor de conjunto binario de DynamoDB se devuelve en forma de una lista de cadenas con la representación base64 de los valores.

Por ejemplo, si DynamoDB devuelve el siguiente valor de conjunto binario de DynamoDB:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

AWS AppSync lo convierte en una lista de cadenas que contienen la representación base64 de los valores:

```
[ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ]
```

Observe que los datos binarios se codifican con el esquema base64 como se especifica en RFC 4648 y en RFC 2045.

Tipo booleano **BOOL**

Un valor booleano. Un valor booleano de DynamoDB se devuelve en forma de valor booleano.

Por ejemplo, si DynamoDB devuelve el siguiente valor booleano de DynamoDB:

```
{ "BOOL" : true }
```

AWS AppSync lo convierte en un valor booleano:

```
true
```

Tipo lista **L**

Lista del resto de valores de DynamoDB admitidos. Un valor de lista de DynamoDB se devuelve en forma de lista de valores, donde el valor de cada elemento también se convierte.

Por ejemplo, si DynamoDB devuelve el siguiente valor de lista de DynamoDB:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

AWS AppSync lo convierte en una lista de valores convertidos:

```
[ "A string value", 1, [ "Another string value", "Even more string values!" ] ]
```

Tipo de mapa **M**

Colección de claves y valores de cualquier otro valor de DynamoDB admitido. Un valor de mapa de DynamoDB se devuelve en forma de objeto JSON en el que también se convierte cada clave y valor.

Por ejemplo, si DynamoDB devuelve el siguiente valor de mapa de DynamoDB:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

AWS AppSync lo convierte en un objeto JSON:

```
{
   "someString" : "A string value",
   "someNumber" : 1,
   "stringSet"  : [ "Another string value", "Even more string values!" ]
}
```

Tipo nulo **`NULL`**

Valor nulo.

Por ejemplo, si DynamoDB devuelve el siguiente valor nulo de DynamoDB:

```
{ "NULL" : null }
```

AWS AppSync lo convierte en un valor nulo:

```
null
```

Filtros

Al consultar objetos de DynamoDB mediante las operaciones `Query` y `Scan`, tiene la posibilidad de especificar un `filter` para evaluar los resultados y devolver solo los valores deseados.

La propiedad de filtro de una solicitud `Query` o `Scan` tiene la siguiente estructura:

```
type DynamoDBExpression = {
  expression: string;
```

```
  expressionNames?: { [key: string]: string};
  expressionValues?: { [key: string]: any};
};
```

Los campos se definen de la siguiente manera:

`expression`

La expresión de la consulta. Para obtener más información sobre cómo escribir expresiones de filtro, consulte la documentación de DynamoDB QueryFilter y DynamoDB ScanFilter. Este campo debe especificarse.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre utilizado en la `expression`. El valor debe ser una cadena que corresponda al nombre del atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con las sustituciones de marcadores de posición de nombre de atributo de expresión que se usen en la `expression`.

`expressionValues`

Las sustituciones de los marcadores de posición de valor de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de valor usado en la `expression` y el valor tiene que ser un valor con tipo. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor debe especificarse. Este campo es opcional y solo debe rellenarse con sustituciones de los marcadores de posición de valor de atributo de expresión que se usan en la `expression`.

Ejemplo

El siguiente ejemplo es una sección de filtro para una solicitud en la que las entradas obtenidas de DynamoDB solo se devuelven si el título comienza con el argumento `title`.

Aquí utilizamos la `util.transform.toDynamoDBFilterExpression` para crear automáticamente un filtro a partir de un objeto:

```
const filter = util.transform.toDynamoDBFilterExpression({
  title: { beginsWith: 'far away' },
});
```

```
const request = {};
request.filter = JSON.parse(filter);
```

Esto genera el filtro siguiente:

```
{
  "filter": {
    "expression": "(begins_with(#title,:title_beginsWith))",
    "expressionNames": { "#title": "title" },
    "expressionValues": {
      ":title_beginsWith": { "S": "far away" }
    }
  }
}
```

Expresiones de condición

Al mutar objetos en DynamoDB utilizando las operaciones de DynamoDB `PutItem`, `UpdateItem` y `DeleteItem`, puede especificar opcionalmente una expresión de condición que determine si la solicitud se debe atender o no en función del estado del objeto que ya está en DynamoDB antes de ejecutar la operación.

La función de DynamoDB de AWS AppSync permite especificar una expresión de condición en los objetos de solicitud `PutItem`, `UpdateItem` y `DeleteItem`, así como la estrategia que debe seguirse en caso de que la condición no se cumpla y el objeto no se actualice.

Ejemplo 1

El siguiente objeto de solicitud `PutItem` no tiene una expresión de condición. Como resultado, pone un elemento en DynamoDB incluso si ya existe un elemento con la misma clave, lo que permite sobrescribir el elemento existente.

```
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),
```

```
  };
}
```

Ejemplo 2

El siguiente objeto `PutItem` tiene una expresión de condición que permitirá que la operación se complete solo si no existe un elemento con la misma clave en 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)" }
  };
}
```

De forma predeterminada, si se produce un error en la comprobación de condición, la función de DynamoDB de AWS AppSync proporciona un error en `ctx.error`. Puede devolver el error para la mutación y el valor actual del objeto en DynamoDB en un campo `data` de la sección `error` de la respuesta de GraphQL.

Sin embargo, la función de DynamoDB de AWS AppSync ofrece algunas características adicionales para ayudar a los desarrolladores a gestionar algunos casos de periferia habituales:

- Si las funciones de DynamoDB de AWS AppSync pueden determinar que el valor actual de DynamoDB coincide con el resultado deseado, trata la operación como si se hubiera realizado de todos modos.
- En lugar de devolver un error, puede configurar la función de modo que invoque una función de Lambda personalizada para decidir cómo debe gestionar el error la función de DynamoDB de AWS AppSync.

Estos casos se describen con más detalle en la sección de gestión de un error de comprobación de la condición.

Para obtener más información sobre las expresiones de condiciones de DynamoDB, consulte la documentación de expresiones de condición de DynamoDB.

Especificación de una condición

Todos los objetos de solicitud `PutItem`, `UpdateItem` y `DeleteItem` permiten especificar una sección `condition` opcional. Si se omite, no se comprobará ninguna condición. Si se especifica, la condición debe ser true para que la operación se lleve a cabo correctamente.

Las secciones `condition` tienen la siguiente estructura:

```
type ConditionCheckExpression = {
  expression: string;
  expressionNames?: { [key: string]: string};
  expressionValues?: { [key: string]: any};
  equalsIgnore?: string[];
  consistentRead?: boolean;
  conditionalCheckFailedHandler?: {
    strategy: 'Custom' | 'Reject';
    lambdaArn?: string;
  };
};
```

Los campos siguientes especifican la condición:

`expression`

La misma expresión de actualización. Para obtener más información sobre cómo escribir expresiones de condición, consulte la documentación de DynamoDB ContitionExpressions. Este campo debe especificarse.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre usado en la expresión, y el valor tiene que ser una cadena que corresponda al nombre de atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con sustituciones de los marcadores de posición de nombre de atributo de expresión que se usan en la expresión.

`expressionValues`

Las sustituciones de los marcadores de posición de valor de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de valor usado en la expresión y el valor tiene que ser un valor con tipo. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor debe

especificarse. Este campo es opcional y solo debe rellenarse con sustituciones de los marcadores de posición de valor de atributo de expresión que se usan en la expresión.

El resto de los campos indican a la función de DynamoDB de AWS AppSync cómo gestionar los casos en que no se cumpla la condición:

`equalsIgnore`

Cuando no se cumple la condición para la operación `PutItem`, la función de DynamoDB de AWS AppSync compara el elemento que hay actualmente en DynamoDB con el elemento que ha intentado escribir. Si son iguales, trata la operación como si se hubiera realizado correctamente. Puede utilizar el campo `equalsIgnore` para especificar una lista de atributos que AWS AppSync no debe tener en cuenta al realizar la comparación. Por ejemplo, si la única diferencia ha sido un atributo `version`, trata la operación como si se hubiera realizado satisfactoriamente. Este campo es opcional.

`consistentRead`

Cuando una condición no se cumple, AWS AppSync obtiene el valor actual del elemento de DynamoDB mediante una lectura altamente coherente. Puede utilizar este campo para indicar a la función de DynamoDB de AWS AppSync que use una lectura coherente posterior en su lugar. Este campo es opcional y de forma predeterminada es `true`.

`conditionalCheckFailedHandler`

Esta sección permite especificar la forma en que la función de DynamoDB de AWS AppSync trata los casos en que no se cumpla la condición después de haber comparado el valor actual en DynamoDB con el resultado esperado. Esta sección es opcional. Si se omite, el valor predeterminado es una estrategia `Reject`.

`strategy`

La estrategia que la función de DynamoDB de AWS AppSync sigue después de comparar el valor actual en DynamoDB con el resultado esperado. Este campo es obligatorio y tiene los siguientes valores posibles:

`Reject`

La mutación fracasa y se obtiene un error y un error para la mutación y el valor actual del objeto en DynamoDB en un campo `data` de la sección `error` de la respuesta de GraphQL.

`Custom`

La función de DynamoDB de AWS AppSync invoca una función de Lambda personalizada para decidir cómo gestionar el incumplimiento de la condición. Cuando `strategy` tiene el valor `Custom`, el campo `lambdaArn` debe contener el ARN de la función Lambda que se va a invocar.

`lambdaArn`

El ARN de la función de Lambda que se invoca que determina cómo debe gestionar la función de DynamoDB de AWS AppSync el incumplimiento de la condición. Este campo solo tiene que especificarse cuando `strategy` tiene el valor `Custom`. Para obtener más información acerca de cómo utilizar esta característica, consulte Gestión de un error de comprobación de la condición.

Gestión de un error de comprobación de la condición

Cuando la condición no se cumple, la función de DynamoDB de AWS AppSync puede transferir el error para la mutación y el valor actual del objeto mediante la utilidad `util.appendError`. Esto agregará el campo `data` en la sección `error` de la respuesta de GraphQL. Sin embargo, la función de DynamoDB de AWS AppSync ofrece algunas características adicionales para ayudar a los desarrolladores a gestionar algunos casos de periferia habituales:

- Si las funciones de DynamoDB de AWS AppSync pueden determinar que el valor actual de DynamoDB coincide con el resultado deseado, trata la operación como si se hubiera realizado de todos modos.
- En lugar de devolver un error, puede configurar la función de modo que invoque una función de Lambda personalizada para decidir cómo debe gestionar el error la función de DynamoDB de AWS AppSync.

El diagrama de flujo de este proceso es:

Comprobación del resultado deseado

Cuando la condición no se cumple, la función de DynamoDB de AWS AppSync realiza una solicitud de DynamoDB `GetItem` para obtener el valor actual del elemento de DynamoDB. De forma predeterminada, utiliza una lectura altamente coherente; sin embargo, esto puede configurarse

mediante el campo `consistentRead` en el bloque `condition` y compararlo con el resultado esperado:

- En la operación `PutItem`, la función de DynamoDB de AWS AppSync compara el valor actual con el que intentó escribir, excluyendo de la comparación los atributos especificados en `equalsIgnore`. Si los elementos son los mismos, trata la operación como si se hubiera realizado y devuelve el elemento obtenido de DynamoDB. De lo contrario, sigue la estrategia configurada.

  Por ejemplo, si el objeto de solicitud `PutItem` tenía el siguiente aspecto:

```
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']
    }
  };
}
```

  Y el elemento que está actualmente en DynamoDB fuese de la siguiente manera:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

  La función de DynamoDB de AWS AppSync compararía el elemento que intentó escribir con el valor actual y vería que la única diferencia es el campo `version`, pero como la configuración pasa el campo `version` por alto, trataría la operación como correcta y devolvería el elemento obtenido de DynamoDB.

- En la operación `DeleteItem`, la función de DynamoDB de AWS AppSync realiza una comprobación para verificar que se ha devuelto un elemento de DynamoDB. Si no se devuelve

ningún elemento, trata la operación como si se hubiera realizado correctamente. De lo contrario, sigue la estrategia configurada.

- En la operación `UpdateItem`, la función de DynamoDB de AWS AppSync no tiene suficiente información para determinar si el elemento que está actualmente en DynamoDB coincide con el resultado esperado y, por lo tanto, sigue la estrategia configurada.

Si el estado actual del objeto en DynamoDB es diferente del resultado esperado, la función de DynamoDB de AWS AppSync sigue la estrategia configurada para rechazar la mutación o invocar una función de Lambda para determinar qué hacer a continuación.

Aplicación de la estrategia de rechazo

Si se sigue la estrategia `Reject`, la función de DynamoDB de AWS AppSync devuelve un error para la mutación y el valor actual del objeto en DynamoDB también se devuelve en un campo `data` de la sección `error` de la respuesta de GraphQL. El elemento que se devuelve desde DynamoDB pasa por el controlador de respuestas de función para convertirlo a un formato que el cliente espera y también se filtra según el conjunto de selección.

Por ejemplo, si se recibe la solicitud de mutación siguiente:

```
mutation {
    updatePerson(id: 1, name: "Steve", expectedVersion: 1) {
        Name
        theVersion
    }
}
```

Si el elemento devuelto de DynamoDB tiene un aspecto similar al siguiente:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

Y el controlador de respuestas de función tiene el siguiente aspecto:

```
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
}
```

La respuesta GraphQL tiene este aspecto:

```
{
  "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
      },
      ...
    }
  ]
}
```

Además, si hay algún campo en el objeto devuelto que hayan cumplimentado otros solucionadores y si la mutación se ha efectuado satisfactoriamente, el objeto no se resuelve cuando se devuelve en la sección `error`.

Aplicación de la estrategia personalizada

Si se sigue la estrategia `Custom`, la función de DynamoDB de AWS AppSync invoca una función de Lambda para decidir qué hacer a continuación. La función Lambda selecciona una de las siguientes opciones:

- `reject` (rechazar) la mutación. Esto le indica a la función de DynamoDB de AWS AppSync que se comporte como si la estrategia configurada fuera `Reject` y que devuelva un error para la mutación y el valor actual del objeto de DynamoDB, tal como se describe en la sección anterior.

- `discard` (rechazar) la mutación. Esto indica a la función de DynamoDB de AWS AppSync que no notifique el incumplimiento de la condición y que devuelva el valor en DynamoDB.
- `retry` (rechazar) la mutación. Esto indica a la función de DynamoDB de AWS AppSync que vuelva a intentar la mutación con un nuevo objeto de solicitud.

La solicitud de invocación Lambda

La función de DynamoDB de AWS AppSync invoca la función de Lambda especificada en el `lambdaArn`. Se utiliza el mismo `service-role-arn` configurado en el origen de datos. La carga de la invocación tiene la siguiente estructura:

```
{
    "arguments": { ... },
    "requestMapping": {... },
    "currentValue": { ... },
    "resolver": { ... },
    "identity": { ... }
}
```

Los campos se definen de la siguiente manera:

`arguments`

Los argumentos de la mutación de GraphQL. Esto es lo mismo que los argumentos disponibles en el objeto de solicitud en `context.arguments`.

`requestMapping`

El objeto de solicitud de esta operación.

`currentValue`

El valor actual del objeto en DynamoDB.

`resolver`

Información sobre el solucionador o la función de AWS AppSync.

`identity`

Información sobre el intermediario. Equivale a la información de identidad disponible en el objeto de solicitud en `context.identity`.

Un ejemplo completo de la carga:

```
{
    "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"
    }
```

```
}
```

La respuesta de invocación Lambda

La función de Lambda puede inspeccionar la carga de invocación y aplicar cualquier lógica de negocio para decidir la forma en que la función de DynamoDB de AWS AppSync debe gestionar el error. Existen tres opciones para gestionar el error de comprobación de condición:

- `reject` (rechazar) la mutación. La carga de respuesta de esta opción debe tener esta estructura:

  ```
  {
      "action": "reject"
  }
  ```

  Esto indica a la función de DynamoDB de AWS AppSync que se comporte como si la estrategia configurada fuera `Reject` y que devuelva un error para la mutación y el valor actual del objeto de DynamoDB, tal como se describe en la sección anterior.
- `discard` (rechazar) la mutación. La carga de respuesta de esta opción debe tener esta estructura:

  ```
  {
      "action": "discard"
  }
  ```

  Esto indica a la función de DynamoDB de AWS AppSync que no notifique el incumplimiento de la condición y que devuelva el valor en DynamoDB.
- `retry` (rechazar) la mutación. La carga de respuesta de esta opción debe tener esta estructura:

  ```
  {
      "action": "retry",
      "retryMapping": { ... }
  }
  ```

  Esto indica a la función de DynamoDB de AWS AppSync que vuelva a intentar la mutación con un nuevo objeto de solicitud. La estructura de la sección `retryMapping` depende de la operación de DynamoDB y es un subconjunto del objeto de solicitud completo de esa operación.

  Para `PutItem`, la sección `retryMapping` tiene la siguiente estructura. Para ver una descripción del campo `attributeValues`, consulte PutItem.

```
{
    "attributeValues": { ... },
    "condition": {
        "equalsIgnore" = [ ... ],
        "consistentRead" = true
    }
}
```

Para `UpdateItem`, la sección `retryMapping` tiene la siguiente estructura. Para ver una descripción de la sección `update`, consulte UpdateItem.

```
{
    "update" : {
        "expression" : "someExpression"
        "expressionNames" : {
            "#foo" : "foo"
        },
        "expressionValues" : {
            ":bar" : ... typed value
        }
    },
    "condition": {
        "consistentRead" = true
    }
}
```

Para `DeleteItem`, la sección `retryMapping` tiene la siguiente estructura.

```
{
    "condition": {
        "consistentRead" = true
    }
}
```

No hay forma de especificar otra operación o clave en la que trabajar. La función de DynamoDB de AWS AppSync solo permite reintentos de la misma operación en el mismo objeto. Asimismo, la sección `condition` no permite especificar un `conditionalCheckFailedHandler`. Si el reintento fracasa, la función de DynamoDB de AWS AppSync sigue la estrategia `Reject`.

A continuación se muestra un ejemplo de función Lambda para tratar una solicitud `PutItem` sin éxito. La lógica de negocio examina quién realizó la llamada. Si la realizó `jeffTheAdmin`, vuelve a intentar realizar la solicitud, actualizando `version` y `expectedVersion` desde el elemento actualmente en DynamoDB. De lo contrario, rechaza la mutación.

```
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)
};
```

Expresiones de condición de transacción

Las expresiones de condición de transacción están disponibles en las solicitudes de los cuatro tipos de operaciones en `TransactWriteItems`, a saber, `PutItem`, `DeleteItem`, `UpdateItem` y `ConditionCheck`.

Para `PutItem`, `DeleteItem` y `UpdateItem`, la expresión de condición de transacción es opcional. Para `ConditionCheck`, se requiere la expresión de condición de transacción.

Ejemplo 1

El siguiente controlador de solicitudes de función `DeleteItem` de carácter transaccional no tiene una expresión de condición. Como resultado, elimina el elemento en 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 }),
      }
    ],
  };
}
```

Ejemplo 2

El siguiente controlador de solicitudes de función `DeleteItem` de carácter transaccional tiene una expresión de condición de transacción que permite que la operación tenga éxito únicamente si el autor de esa publicación es igual a un nombre determinado.

```
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 },
        }),
      }
    ],
  };
}
```

Si la comprobación de condición no se realiza correctamente, provocará la excepción `TransactionCanceledException` y se devolverá el detalle del error en `ctx.result.cancellationReasons`. Tenga en cuenta que, de forma predeterminada, el elemento anterior de DynamoDB que provocó el error en esa comprobación de condición se devolverá en `ctx.result.cancellationReasons`.

Especificación de una condición

Todos los objetos de solicitud `PutItem`, `UpdateItem` y `DeleteItem` permiten especificar una sección `condition` opcional. Si se omite, no se comprobará ninguna condición. Si se especifica, la condición debe ser true para que la operación se lleve a cabo correctamente. `ConditionCheck` debe tener una sección `condition` sección para especificarla. La condición debe ser verdadera para que toda la transacción se realice correctamente.

Las secciones `condition` tienen la siguiente estructura:

```
type TransactConditionCheckExpression = {
  expression: string;
  expressionNames?: { [key: string]: string };
  expressionValues?: { [key: string]: string };
  returnValuesOnConditionCheckFailure: boolean;
};
```

Los campos siguientes especifican la condición:

`expression`

La misma expresión de actualización. Para obtener más información sobre cómo escribir expresiones de condición, consulte la documentación de DynamoDB ContitionExpressions. Este campo debe especificarse.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre usado

en la expresión, y el valor tiene que ser una cadena que corresponda al nombre de atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con sustituciones de los marcadores de posición de nombre de atributo de expresión que se usan en la expresión.

`expressionValues`

Las sustituciones de los marcadores de posición de valor de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de valor usado en la expresión y el valor tiene que ser un valor con tipo. Para obtener más información sobre cómo especificar un "valor con tipo", consulte Sistema de tipos (mapeo de solicitud). Este valor debe especificarse. Este campo es opcional y solo debe rellenarse con sustituciones de los marcadores de posición de valor de atributo de expresión que se usan en la expresión.

`returnValuesOnConditionCheckFailure`

Especifique si desea recuperar el elemento en DynamoDB cuando se produzca un error en la comprobación de condición. El elemento recuperado estará en `ctx.result.cancellationReasons[<index>].item`, donde `<index>` es el índice del elemento de solicitud que no ha superado la comprobación de condición. Este valor se establece de forma predeterminada en true.

Proyecciones

Al leer objetos de DynamoDB mediante las operaciones `GetItem`, `Scan`, `Query`, `BatchGetItem` y `TransactGetItems`, tiene la posibilidad de especificar una proyección para identificar los atributos deseados. La propiedad de proyección tiene la siguiente estructura, que es similar a los filtros:

```
type DynamoDBExpression = {
  expression: string;
  expressionNames?: { [key: string]: string}
};
```

Los campos se definen de la siguiente manera:

`expression`

La expresión de proyección, que es una cadena. Para recuperar un solo atributo, especifique su nombre. Si desea obtener varios atributos, separe sus nombres mediante comas. Para obtener más información sobre la redacción de expresiones de proyección, consulte la documentación relativa a las expresiones de proyección de DynamoDB. Este campo es obligatorio.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre utilizado en la `expression`. El valor debe ser una cadena que corresponda al nombre del atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con las sustituciones de marcadores de posición de nombre de atributo de expresión que se usen en la `expression`. Para obtener más información acerca de `expressionNames`, consulte la documentación de DynamoDB.

Ejemplo 1

El siguiente ejemplo es una sección de proyección para una función de JavaScript en la que solo los atributos `author` y `id` se devuelven de DynamoDB:

```
projection : {
    expression : "#author, id",
    expressionNames : {
        "#author" : "author"
    }
}
```

> **Tip**
>
> Puede acceder a su conjunto de selección de solicitudes de GraphQL mediante selectionSetList. Este campo permite enmarcar su expresión de proyección de forma dinámica según sus requisitos.

> **Note**
>
> Al utilizar expresiones de proyección con las operaciones `Query` y `Scan`, el valor de `select` debe ser `SPECIFIC_ATTRIBUTES`. Para obtener más información, consulte la documentación de DynamoDB.

Referencia a la función de solucionador de JavaScript para OpenSearch

El solucionador de AWS AppSync para Amazon OpenSearch Service permite utilizar GraphQL para almacenar y recuperar datos de dominios de OpenSearch Service ya existentes en su cuenta. Para funcionar, este solucionador permite mapear una solicitud de GraphQL entrante a una solicitud de OpenSearch Service y, a continuación, mapear la respuesta de OpenSearch Service a GraphQL. En esta sección se describen los controladores de solicitudes y respuestas de función para las operaciones de OpenSearch Service admitidas.

Solicitud

La mayoría de objetos de solicitud de OpenSearch Service presentan una estructura común en la que tan solo cambian unos pocos elementos. En el siguiente ejemplo se ejecuta una búsqueda en un dominio de OpenSearch Service donde los documentos son del tipo `post` y se indexan por `id`. Los parámetros de búsqueda se definen en la sección `body` y muchas de las cláusulas de consulta comunes se definen en el campo `query`. En este ejemplo se buscan documentos que contengan `"Nadia"`, `"Bailey"` o ambos en el campo `author` de un documento:

```
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' } },
            ],
          },
        },
      },
    },
  };
```

```
}
```

Respuesta

Al igual que ocurre con otros orígenes de datos, OpenSearch Service envía una respuesta a AWS AppSync que debe convertirse a GraphQL.

La mayoría de consultas de GraphQL buscan el campo `_source` de una respuesta de OpenSearch Service. Puesto que puede hacer búsquedas para devolver un documento individual o una lista de documentos, en OpenSearch Service se utilizan dos patrones de respuesta comunes:

Lista de resultados

```
export function response(ctx) {
  const entries = [];
  for (const entry of ctx.result.hits.hits) {
    entries.push(entry['_source']);
  }
  return entries;
}
```

Elemento individual

```
export function response(ctx) {
  return ctx.result['_source']
}
```

Campo **`operation`**

(Solo el controlador de SOLICITUDES)

Método o verbo HTTP (GET, POST, PUT, HEAD o DELETE) que envía AWS AppSync al dominio de OpenSearch Service. Tanto la clave como el valor deben ser cadenas.

```
"operation" : "PUT"
```

Campo **`path`**

(Solo el controlador de SOLICITUDES)

Ruta de búsqueda de una solicitud de OpenSearch Service desde AWS AppSync. Esto constituye una URL para el verbo HTTP de la operación. Tanto la clave como el valor deben ser cadenas.

```
"path" : "/indexname/type"

"path" : "/indexname/type/_search"
```

Cuando se evalúa el controlador de solicitudes, esta ruta se envía como parte de la solicitud HTTP, incluido el dominio de OpenSearch Service. Por ejemplo, el ejemplo anterior puede convertirse como:

```
GET https://opensearch-domain-name.REGION.es.amazonaws.com/indexname/type/_search
```

Campo **`params`**

(Solo el controlador de SOLICITUDES)

Se utiliza para especificar la acción que realiza la búsqueda, normalmente estableciendo el valor query dentro de body. Sin embargo, se pueden configurar otras funcionalidades, como, por ejemplo, el formato de las respuestas.

- headers

  Es la información del encabezado en forma de pares clave-valor. Tanto la clave como el valor deben ser cadenas. Por ejemplo:

  ```
  "headers" : {
      "Content-Type" : "application/json"
  }
  ```

  > **Note**
  >
  > Actualmente, AWS AppSync solo admite JSON como `Content-Type`.

- queryString

  Son los pares clave-valor que especifican opciones comunes, como el formato de código de las respuestas JSON. Tanto la clave como el valor deben ser cadenas. Por ejemplo, si desea JSON con formato pretty, puede especificar:

```
"queryString" : {
    "pretty" : "true"
}
```

- body

  Esta es la parte principal de la solicitud, la que permite a AWS AppSync crear una petición de búsqueda bien formada dirigida al dominio de OpenSearch Service. La clave debe ser una cadena compuesta por un objeto. A continuación se muestran algunos ejemplos.

Ejemplo 1

Devuelve todos los documentos que incluyan la ciudad “seattle”:

```
export function request(ctx) {
  return {
    operation: 'GET',
    path: '/id/post/_search',
    params: {
      headers: {},
      queryString: {},
      body: { from: 0, size: 50, query: { match: { city: 'seattle' } } },
    },
  };
}
```

Ejemplo 2

Devuelve todos los documentos que incluyan “washington” como ciudad o estado:

```
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'] },
        },
      },
    },
  };
}
```

Variables de transferencia

(Solo el controlador de SOLICITUDES)

También puede pasar variables como parte de la evaluación en su controlador de solicitudes. Por ejemplo, suponga que ha tenido la siguiente consulta de GraphQL:

```
query {
    searchForState(state: "washington"){
        ...
    }
}
```

El controlador de solicitudes de función podría ser el siguiente:

```
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 referencia de la función de resolución para Lambda

Puede usar AWS AppSync funciones y resolutores para invocar las funciones de Lambda ubicadas en su cuenta. Puede configurar las cargas útiles de sus solicitudes y la respuesta de sus funciones Lambda antes de devolverlas a sus clientes. También puede especificar el tipo de operación que se va a realizar en el objeto de solicitud. En esta sección se describen las solicitudes para las operaciones de Lambda admitidas.

Objeto Request (solicitud)

El objeto de solicitud Lambda gestiona los campos relacionados con la función Lambda:

```
export type LambdaRequest = {
  operation: 'Invoke' | 'BatchInvoke';
  invocationType?: 'RequestResponse' | 'Event';
  payload: unknown;
};
```

Este es un ejemplo en el que se usa una `invoke` operación cuyos datos de carga útil son el `getPost` campo de un esquema de GraphQL junto con sus argumentos del contexto:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    payload: { field: 'getPost', arguments: ctx.args },
  };
}
```

El documento de mapeo completo se pasa como entrada a la función Lambda, de modo que el ejemplo anterior ahora tiene este aspecto:

```
{
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": {
      "input": {
        "id": "postId1",
      }
    }
```

```
  }
}
```

Operación

La fuente de datos Lambda le permite definir dos operaciones en el `operation` campo: `Invoke` y. `BatchInvoke` La `Invoke` operación permite llamar a AWS AppSync la función Lambda para cada solucionador de campos de GraphQL. `BatchInvoke`indica AWS AppSync que se agrupen las solicitudes para el campo GraphQL actual. El campo `operation` es obligatorio.

Para`Invoke`, la solicitud resuelta coincide con la carga útil de entrada de la función Lambda. Modifiquemos el ejemplo anterior:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    payload: { field: 'getPost', arguments: ctx.args },
  };
}
```

Esto se resuelve y se pasa a la función Lambda, que podría tener un aspecto similar al siguiente:

```
{
  "operation": "Invoke",
  "payload": {
    "arguments": {
      "id": "postId1"
    }
  }
}
```

`BatchInvoke`En efecto, la solicitud se aplica a todos los solucionadores de campos del lote. Para mayor concisión, AWS AppSync fusiona todos los `payload` valores de la solicitud en una lista bajo un único objeto que coincida con el objeto de la solicitud. El siguiente controlador de solicitudes de ejemplo muestra esta combinación:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    payload: ctx,
  };
```

```
}
```

Esta solicitud se evalúa y resuelve para dar el siguiente documento de mapeo:

```
{
  "operation": "BatchInvoke",
  "payload": [
    {...}, // context for batch item 1
    {...}, // context for batch item 2
    {...}  // context for batch item 3
  ]
}
```

Cada elemento de la `payload` lista corresponde a un único elemento del lote. También se espera que la función Lambda devuelva una respuesta en forma de lista que coincida con el orden de los elementos enviados en la solicitud:

```
[
  { "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
]
```

Carga

El `payload` campo es un contenedor que se utiliza para pasar cualquier dato a la función Lambda. Si el `operation` campo está establecido en`BatchInvoke`, AWS AppSync agrupa los `payload` valores existentes en una lista. El campo `payload` es opcional.

Tipo de invocación

La fuente de datos Lambda le permite definir dos tipos de invocación: y. `RequestResponse` `Event` Los tipos de invocación son sinónimos de los tipos de invocación definidos en la API Lambda. El tipo `RequestResponse` de invocación permite AWS AppSync llamar a la función Lambda de forma sincrónica para esperar una respuesta. La `Event` invocación le permite invocar la función Lambda de forma asíncrona. Para obtener más información sobre cómo Lambda gestiona las solicitudes de tipo de `Event` invocación, consulte Invocación asíncrona. El campo `invocationType` es opcional.

Si este campo no está incluido en la solicitud, se AWS AppSync utilizará de forma predeterminada el tipo de invocación. `RequestResponse`

Para cualquier `invocationType` campo, la solicitud resuelta coincide con la carga útil de entrada de la función Lambda. Modifiquemos el ejemplo anterior:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    invocationType: 'Event',
    payload: { field: 'getPost', arguments: ctx.args },
  };
}
```

Esto se resuelve y se pasa a la función Lambda, que podría tener un aspecto similar al siguiente:

```
{
  "operation": "Invoke",
  "invocationType": "Event",
  "payload": {
    "arguments": {
      "id": "postId1"
    }
  }
}
```

Cuando la `BatchInvoke` operación se usa junto con el campo de tipo de `Event` invocación, AWS AppSync fusiona el solucionador de campos de la misma manera que se mencionó anteriormente y la solicitud se pasa a la función Lambda como un evento asíncrono, siendo una lista de valores. `payload` La respuesta de una solicitud de tipo de `Event` invocación da como resultado un valor sin un controlador de respuesta: `null`

```
{
  "data": {
    "field": null
  }
}
```

Le recomendamos que deshabilite el almacenamiento en caché de los resolutores de tipo `Event` invocación, ya que estos no se enviarían a Lambda si se produjera un error en la memoria caché.

Objeto de respuesta

Al igual que con otras fuentes de datos, la función Lambda envía una respuesta AWS AppSync que debe convertirse a un tipo GraphQL. El resultado de la función Lambda está contenido en la propiedad `context` result ()`context.result`.

Si la forma de la respuesta de la función Lambda coincide con la forma del tipo GraphQL, puede reenviar la respuesta mediante el siguiente controlador de respuesta de la función:

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

No hay campos obligatorios ni restricciones de forma aplicables al objeto de respuesta. Sin embargo, dado que los tipos de GraphQL son estrictos, la respuesta resuelta debe coincidir con el tipo de GraphQL previsto.

Repuesta de la función de Lambda en lotes

Si el `operation` campo está establecido en`BatchInvoke`, AWS AppSync espera una lista de elementos de la función Lambda. Para volver AWS AppSync a asignar cada resultado al elemento de la solicitud original, la lista de respuestas debe coincidir en tamaño y orden. Es válido tener `null` elementos en la lista de respuestas; `ctx.result` se establece en nulo en consecuencia.

JavaScript referencia de la función de resolución para la fuente EventBridge de datos

La solicitud y la respuesta de la función de AWS AppSync resolución que se utilizan con la fuente de EventBridge datos te permiten enviar eventos personalizados al EventBridge bus de Amazon.

Solicitud

El controlador de solicitudes le permite enviar varios eventos personalizados a un bus de EventBridge eventos:

```
export function request(ctx) {
  return {
    "operation" : "PutEvents",
    "events" : [{}]
```

```
  }
}
```

Una EventBridge `PutEvents` solicitud tiene la siguiente definición de tipo:

```
type PutEventsRequest = {
  operation: 'PutEvents'
  events: {
    source: string
    detail: { [key: string]: any }
    detailType: string
    resources?: string[]
    time?: string // RFC3339 Timestamp format
  }[]
}
```

Respuesta

Si la `PutEvents` operación se realiza correctamente, la respuesta de EventBridge se incluye en`ctx.result`:

```
export function response(ctx) {
  if(ctx.error)
    util.error(ctx.error.message, ctx.error.type, ctx.result)
  else
    return ctx.result
}
```

Los errores que se produzcan al realizar operaciones `PutEvents` como `InternalExceptions` o `Timeouts` aparecerán en `ctx.error`. Para obtener una lista EventBridge de los errores más comunes, consulta la referencia de errores EventBridge comunes.

El `result` tendrá la siguiente definición de tipos:

```
type PutEventsResult = {
  Entries: {
    ErrorCode: string
    ErrorMessage: string
    EventId: string
  }[]
  FailedEntryCount: number
```

```
}
```

- Entradas

  Resultados de los eventos ingeridos, tanto correctos como incorrectos. Si la ingesta se realizó correctamente, la entrada contiene el `EventID`. De lo contrario, puede usar `ErrorCode` y `ErrorMessage` para identificar el problema con la entrada.

  Para cada registro, el índice del elemento de respuesta es el mismo que el de la matriz de solicitudes.
- FailedEntryCount

  Número de entradas con error. Este valor se representa como un entero.

Para obtener más información sobre la respuesta de`PutEvents`, consulte PutEvents.

Ejemplo de respuesta de muestra 1

El siguiente ejemplo es una operación `PutEvents` con dos eventos correctos:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        }
    ],
    "FailedEntryCount" : 0
}
```

Ejemplo de respuesta de muestra 2

El siguiente ejemplo es una operación `PutEvents` con tres eventos, dos correctos y uno incorrecto:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
```

```
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        },
        {
            "ErrorCode" : "SampleErrorCode",
            "ErrorMessage" : "Sample Error Message"
        }
    ],
    "FailedEntryCount" : 1
}
```

Campo **PutEvents**

- Versión

  El campo `version` es común a todas las plantillas de mapeo de solicitudes y define la versión utilizada por la plantilla. Este campo es obligatorio. El valor `2018-05-29` es la única versión compatible con las plantillas de EventBridge mapeo.
- Operación

  La única operación admitida es `PutEvents`. Esta operación permite añadir eventos personalizados a su bus de eventos.
- Eventos

  Una matriz de eventos que se añadirán al bus de eventos. Esta matriz debe tener una asignación de entre 1 y 10 elementos.

  El objeto `Event` tiene los siguientes campos:
  - `"source"`: cadena que define el origen del evento.
  - `"detail"`: objeto JSON que puede usar para asociar información sobre el evento. Este campo puede ser un mapa vacío (`{ }`).
  - `"detailType`: cadena que identifica el tipo de evento
  - `"resources"`: matriz JSON de cadenas que identifica los recursos involucrados en el evento. Este campo puede ser una matriz vacía.
  - `"time"`: marca temporal del evento proporcionada como cadena. Debe seguir el formato de marca temporal RFC3339.

Los siguientes fragmentos de código son algunos ejemplos de objetos `Event` válidos:

Ejemplo 1

```
{
    "source" : "source1",
    "detail" : {
        "key1" : [1,2,3,4],
        "key2" : "strval"
    },
    "detailType" : "sampleDetailType",
    "resources" : ["Resouce1", "Resource2"],
    "time" : "2022-01-10T05:00:10Z"
}
```

Ejemplo 2

```
{
    "source" : "source1",
    "detail" : {},
    "detailType" : "sampleDetailType"
}
```

Ejemplo 3

```
{
    "source" : "source1",
    "detail" : {
        "key1" : 1200
    },
    "detailType" : "sampleDetailType",
    "resources" : []
}
```

Referencia a la función de solucionador de JavaScript para el origen de datos None

La solicitud y respuesta de función del solucionador de AWS AppSync utilizadas con el origen de datos de tipo None permiten dar forma a las solicitudes de operaciones locales de AWS AppSync.

Solicitud

El controlador de solicitudes puede ser sencillo y permite transferir toda la información contextual posible a través del campo `payload`.

```
type NONERequest = {
  payload: any;
};
```

En el siguiente ejemplo se transfieren los argumentos del campo a la carga:

```
export function request(ctx) {
  return {
    payload: context.args
  };
}
```

El valor del campo `payload` se reenviará al controlador de respuestas de función y está disponible en `context.result`.

Carga

El campo `payload` es un contenedor que se puede utilizar para transferir cualquier dato que luego se pone a disposición del controlador de respuestas de función.

El campo `payload` es opcional.

Respuesta

Dado que no hay ningún origen de datos, el valor del campo `payload` se reenviará al controlador de respuestas de función y se establecerá en la propiedad `context.result`.

Si la forma del valor del campo `payload` coincide exactamente con la forma del tipo de GraphQL, puede reenviar la respuesta mediante el siguiente controlador de respuestas:

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

No hay campos obligatorios ni restricciones de forma aplicables a la respuesta de devolución. Sin embargo, dado que los tipos de GraphQL son estrictos, la respuesta resuelta debe coincidir con el tipo de GraphQL previsto.

JavaScript referencia de función de resolución para HTTP

Las funciones de resolución de AWS AppSync HTTP le permiten enviar solicitudes desde AWS AppSync cualquier punto de enlace HTTP y respuestas desde su punto de enlace HTTP de vuelta a AWS AppSync. Con su controlador de solicitudes, puede proporcionar sugerencias AWS AppSync sobre la naturaleza de la operación que se va a invocar. En esta sección se describen las distintas configuraciones para el solucionador de HTTP admitido.

Solicitud

```
type HTTPRequest = {
  method: 'PUT' | 'POST' | 'GET' | 'DELETE' | 'PATCH';
  params?: {
    query?: { [key: string]: any };
    headers?: { [key: string]: string };
    body?: any;
  };
  resourcePath: string;
};
```

El siguiente fragmento de código es un ejemplo de una solicitud HTTP POST con cuerpo `text/plain`:

```
export function request(ctx) {
  return {
    method: 'POST',
    params: {
      headers: { 'Content-Type': 'text/plain' },
      body: 'this is an example of text body',
    },
    resourcePath: '/',
  };
}
```

Método

Solo el controlador de solicitudes

Método o verbo HTTP (GET, POST, PUT, PATCH o DELETE) que envía AWS AppSync al punto de enlace HTTP.

```
"method": "PUT"
```

ResourcePath

Solo el controlador de solicitudes

La ruta de recurso a la que desea acceso. Junto con el punto de enlace del origen de datos HTTP, la ruta del recurso forma la URL a la que el servicio AWS AppSync envía la solicitud.

```
"resourcePath": "/v1/users"
```

Cuando se evalúa la solicitud, esta ruta se envía como parte de la solicitud HTTP, incluido el punto de conexión HTTP. Por ejemplo, el ejemplo anterior puede convertirse como:

```
PUT <endpoint>/v1/users
```

Campo params

Solo el controlador de solicitudes

Se utiliza para especificar la acción que realiza la búsqueda, normalmente estableciendo el valor query dentro de body. Sin embargo, se pueden configurar otras funcionalidades, como, por ejemplo, el formato de las respuestas.

headers

Es la información del encabezado en forma de pares clave-valor. Tanto la clave como el valor deben ser cadenas.

Por ejemplo:

```
"headers" : {
```

```
    "Content-Type" : "application/json"
}
```

Los encabezados `Content-Type` admitidos actualmente son:

```
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
```

No puede definir los siguientes encabezados HTTP:

```
HOST
CONNECTION
USER-AGENT
EXPECTATION
TRANSFER_ENCODING
CONTENT_LENGTH
```

consulta

Son los pares clave-valor que especifican opciones comunes, como el formato de código de las respuestas JSON. Tanto la clave como el valor deben ser cadenas. En el siguiente ejemplo se muestra el modo de enviar una cadena de consulta como `?type=json`:

```
"query" : {
    "type" : "json"
}
```

body

La sección body contiene el cuerpo de la solicitud HTTP que defina. El cuerpo de la solicitud siempre es una cadena con codificación UTF-8, a menos que el tipo de contenido especifique un conjunto de caracteres.

```
"body":"body string"
```

Respuesta

Puede ver un ejemplo aquí.

JavaScript referencia de la función de resolución para Amazon RDS

La función y el solucionador de AWS AppSync RDS permiten a los desarrolladores enviar SQL consultas a una base de datos de clústeres de Amazon Aurora mediante la API de datos de RDS y obtener el resultado de estas consultas. Puede escribir SQL declaraciones que se envíen a la API de datos mediante la plantilla `sql` etiquetada como `rds` módulo AWS AppSync del módulo o mediante las funciones`select`, `insertupdate`, y `remove` auxiliar del `rds` módulo. AWS AppSync utiliza la `ExecuteStatement`acción del servicio de datos de RDS para ejecutar sentencias SQL en la base de datos.

Temas

- Plantilla con etiquetas SQL
- Creación de instrucciones
- Recuperación de datos
- Funciones de utilidad
- Selección de SQL
- Inserción de SQL
- Actualización de SQL
- Eliminación de SQL
- Conversión

Plantilla con etiquetas SQL

AWS AppSync La plantilla `sql` etiquetada permite crear una sentencia estática que puede recibir valores dinámicos en tiempo de ejecución mediante el uso de expresiones de plantilla. AWS AppSync crea un mapa de variables a partir de los valores de la expresión para crear una `SqlParameterized`consulta que se envía a la API de datos sin servidor de Amazon Aurora. Con este método, no es posible que los valores dinámicos pasados en tiempo de ejecución modifiquen

la instrucción original, lo que podría provocar una ejecución no intencionada. Todos los valores dinámicos se pasan como parámetros, no pueden modificar la instrucción original y la base de datos no los ejecuta. Esto hace que la consulta sea menos vulnerable a los ataques de inyección de SQL.

> **Note**
>
> En todos los casos, al escribir instrucciones SQL, debe seguir las pautas de seguridad para gestionar correctamente los datos que reciba como entrada.

> **Note**
>
> La plantilla con etiquetas `sql` solo admite la transferencia de valores variables. No puede usar una expresión para especificar dinámicamente los nombres de las columnas o tablas. No obstante, puede usar funciones de utilidad para crear instrucciones dinámicas.

En el siguiente ejemplo, creamos una consulta que filtra en función del valor del argumento `col` que se establece dinámicamente en la consulta GraphQL en tiempo de ejecución. El valor solo se puede añadir a la instrucción mediante la expresión de etiquetas:

```
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);
}
```

Al pasar todos los valores dinámicos por el mapa de variables, confiamos en el motor de la base de datos para gestionar y sanear los valores de forma segura.

Creación de instrucciones

Las funciones y los solucionadores pueden interactuar con las bases de datos MySQL y PostgreSQL. Utilice `createMySQLStatement` y `createPgStatement`, respectivamente, para crear instrucciones. Por ejemplo, `createMySQLStatement` puede crear una consulta MySQL. Estas

funciones aceptan hasta dos instrucciones, lo que resulta útil cuando una solicitud debe recuperar los resultados de forma inmediata. Con SQL, podría hacer:

```
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` y `createMySQLStatement` no escapa ni cita las instrucciones creadas con la plantilla etiquetada con `sql`.

Recuperación de datos

El resultado de la instrucción SQL ejecutada está disponible en el controlador de respuestas del objeto `context.result`. El resultado es una cadena JSON con los elementos de respuesta de la acción `ExecuteStatement`. Por ejemplo, puede utilizar la siguiente forma:

```
type SQLStatementResults = {
    sqlStatementResults: {
        records: any[];
        columnMetadata: any[];
        numberOfRecordsUpdated: number;
        generatedFields?: any[]
    }[]
}
```

Puede utilizar la utilidad `toJsonObject` para transformar el resultado en una lista de objetos JSON que representen las filas devueltas. Por ejemplo:

```
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]
}
```

Tenga en cuenta que `toJsonObject` devuelve una matriz de resultados de la instrucción. Si proporcionó una instrucción, la longitud de la matriz es `1`. Si proporcionó dos instrucciones, la longitud de la matriz es `2`. Cada resultado de la matriz contiene `0` o más filas. `toJsonObject` devuelve `null` si el valor del resultado no es válido o no es esperado.

Funciones de utilidad

Puede utilizar las utilidades auxiliares del módulo AWS AppSync RDS para interactuar con la base de datos.

Selección de SQL

La utilidad `select` crea una instrucción `SELECT` para consultar la base de datos relacional.

Uso básico

En su forma básica, puede especificar la tabla que desea consultar:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

    // Generates statement:
    // "SELECT * FROM "persons"
    return createPgStatement(select({table: 'persons'}));
}
```

Tenga en cuenta que también puede especificar el esquema en el identificador de la tabla:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';
```

```
export function request(ctx) {

    // Generates statement:
    // SELECT * FROM "private"."persons"
    return createPgStatement(select({table: 'private.persons'}));
}
```

Especificación de columnas

Puede especificar columnas con la propiedad `columns`. Si no se establece en un valor, el valor predeterminado es *:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name']
    }));
}
```

También puede especificar la tabla de una columna:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "persons"."name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'persons.name']
    }));
}
```

Límites y desplazamientos

Puede aplicar `limit` y `offset` a la consulta:

```
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
    }));
}
```

Ordenar por

Puede ordenar los resultados con la propiedad `orderBy`. Proporcione una matriz de objetos especificando la columna y una propiedad `dir` opcional:

```
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'}]
    }));
}
```

Filtros

Puede crear filtros con el objeto de condición especial:

```
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'}}
    }));
}
```

También puede combinar filtros:

```
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}}
    }));
}
```

También puede crear instrucciones 0R:

```
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 } }
        ]}
    }));
}
```

También puede negar una condición con `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 } }
        ]}
    }));
}
```

También puede utilizar los siguientes operadores para comparar valores:

Operador	Descripción	Tipos de valor posibles
eq	Igualdad	número, cadena, booleano
Uno	Desigualdad	número, cadena, booleano
una mentira	Menor que o igual a	número, cadena
lt	Menor que	número, cadena
edad	Mayor que o igual a	número, cadena
gt	Mayor que	número, cadena
contiene	¿Como	cadena
No contiene	No como	cadena
Empieza con	Empieza con el prefijo	cadena
entre	Entre dos valores	número, cadena
El atributo existe	El atributo no es nulo	número, cadena, booleano
size	comprueba la longitud del elemento	cadena

Inserción de SQL

La utilidad `insert` ofrece una forma sencilla de insertar elementos de una sola fila en la base de datos con la operación `INSERT`.

Inserciones de un solo elemento

Para insertar un elemento, especifique la tabla y, a continuación, transfiera su objeto de valores. Las claves de objetos se asignan a las columnas de la tabla. Los nombres de las columnas se escapan automáticamente y los valores se envían a la base de datos mediante el mapa de variables:

```
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)
}
```

Caso de uso de MySQL

Puede combinar un `insert` seguido de un `select` para recuperar la fila insertada:

```
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)
}
```

Caso de uso de Postgres

Con Postgres, puede usar `returning` para obtener datos de la fila que insertó. Acepta `*` o una matriz de nombres de columna:

```
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)
}
```

Actualización de SQL

La utilidad `update` le permite actualizar las filas existentes. Puede utilizar el objeto de condición para aplicar cambios a las columnas especificadas en todas las filas que cumplan la condición. Por ejemplo, supongamos que tenemos un esquema que nos permite realizar esta mutación. Queremos actualizar el `name` de `Person` con el valor `id` de `3`, pero solo si los conocemos (`known_since`) desde el año `2000`:

```
mutation Update {
    updatePerson(
        input: {id: 3, name: "Jon"},
        condition: {known_since: {ge: "2000"}}
    ) {
```

```
    id
    name
  }
}
```

Nuestro solucionador de actualización tendrá este aspecto:

```
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)
}
```

Podemos añadir una comprobación a nuestra condición para asegurarnos de que solo se actualice la fila en la que la clave principal `3` sea igual a `id`. Del mismo modo, en el caso de `inserts` de Postgres, se puede utilizar `returning` para devolver los datos modificados.

Eliminación de SQL

La utilidad `remove` le permite eliminar las filas existentes. Puede utilizar el objeto de condición en todas las filas que cumplan la condición. Tenga en cuenta que `delete` es una palabra clave reservada en JavaScript. `remove`debería usarse en su lugar:

```
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)
}
```

Conversión

En algunos casos, es posible que desee más especificidad sobre el tipo de objeto correcto para usar en su instrucción. Puede utilizar las sugerencias de tipo proporcionadas para especificar el tipo de parámetros. AWS AppSync admite las mismas sugerencias de tipo que la API de datos. Puede convertir sus parámetros mediante las `typeHint` funciones del AWS AppSync `rds` módulo.

En el siguiente ejemplo puede enviar una matriz como un valor que se convierte en un objeto JSON. Usamos el operador `->` para recuperar el elemento en el `index 2` en la matriz 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
}
```

La conversión también resulta útil al manipular y comparar `DATE`, `TIME` y `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)
}
```

Aquí se muestra otro ejemplo de cómo puede enviar la fecha y hora actuales:

```
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)}`)
}
```

Sugerencias de tipos disponibles

- `typeHint.DATE`: el parámetro correspondiente se envía como un objeto de tipo `DATE` a la base de datos. El formato aceptado es `YYYY-MM-DD`.
- `typeHint.DECIMAL`: el parámetro correspondiente se envía como un objeto de tipo `DECIMAL` a la base de datos.
- `typeHint.JSON`: el parámetro correspondiente se envía como un objeto de tipo `JSON` a la base de datos.
- `typeHint.TIME`: el valor del parámetro de cadena correspondiente se envía como un objeto de tipo `TIME` a la base de datos. El formato aceptado es `HH:MM:SS[.FFF]`.
- `typeHint.TIMESTAMP`: el valor del parámetro de cadena correspondiente se envía como un objeto de tipo `TIMESTAMP` a la base de datos. El formato aceptado es `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `typeHint.UUID`: el valor del parámetro de cadena correspondiente se envía como un objeto de tipo `UUID` a la base de datos.

Referencia de plantillas de mapeo de solucionador (VTL)

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

En las siguientes secciones se describe cómo se pueden utilizar las operaciones de utilidad en las plantillas de mapeo.

Temas

- Información general sobre las plantillas de mapeo de solucionador
- Guía de programación de plantillas de mapeo de solucionador
- Referencia de contexto de las plantillas de mapeo del solucionador
- Referencia de utilidad de la plantilla de mapeo de solucionador
- Referencia de las plantillas de mapeo de solucionador para DynamoDB
- Referencia de plantillas de mapeo de solucionador para RDS
- Referencia de plantillas de mapeo de solucionador para OpenSearch
- Referencia de plantillas de mapeo de solucionador para Lambda
- Referencia de plantilla de mapeo de Resolver para EventBridge
- Referencia de plantillas de mapeo de solucionador para el origen de datos None
- Referencia de plantillas de mapeo de solucionador para HTTP
- Registro de cambios de plantillas de mapeo de solucionador

Información general sobre las plantillas de mapeo de solucionador

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

Con AWS AppSync puede responder a solicitudes de GraphQL efectuando operaciones en sus recursos. Para cada campo de GraphQL en el que desee ejecutar una consulta o mutación, se debe asociar un solucionador a fin de comunicarse con un origen de datos. La comunicación se suele realizar a través de parámetros u operaciones que son exclusivos para cada origen de datos.

Los solucionadores son los conectores entre GraphQL y un origen de datos. Indican a AWS AppSync cómo convertir una solicitud de GraphQL entrante en instrucciones para el origen de datos de backend y cómo convertir la respuesta de ese origen de datos en una respuesta de GraphQL. Están escritas en Apache Velocity Template Language (VTL), que toma su solicitud como entrada y genera una salida en forma de documento JSON con las instrucciones para el solucionador. Puede utilizar plantillas de mapeo para instrucciones sencillas, como transferencias en argumentos de campos de GraphQL, o para instrucciones más complejas, como bucles que recorran argumentos para crear un elemento antes de insertarlo en DynamoDB.

Existen dos tipos de solucionadores en AWS AppSync que aprovechan las plantillas de mapeo de maneras ligeramente distintas:

- Solucionadores de unidad
- Solucionadores de canalización

Solucionadores de unidad

Los solucionadores de unidad son entidades autónomas que incluyen solo una plantilla de solicitud y respuesta. Utilícelos para operaciones sencillas y únicas, como enumerar elementos de un único origen de datos.

- Plantillas de solicitud: toman la solicitud entrante después de analizar una operación de GraphQL y la convierten en una configuración de solicitud para la operación del origen de datos seleccionado.
- Plantillas de respuesta: interpretan respuestas del origen de datos y las mapean a la forma del tipo de resultado del campo de GraphQL.

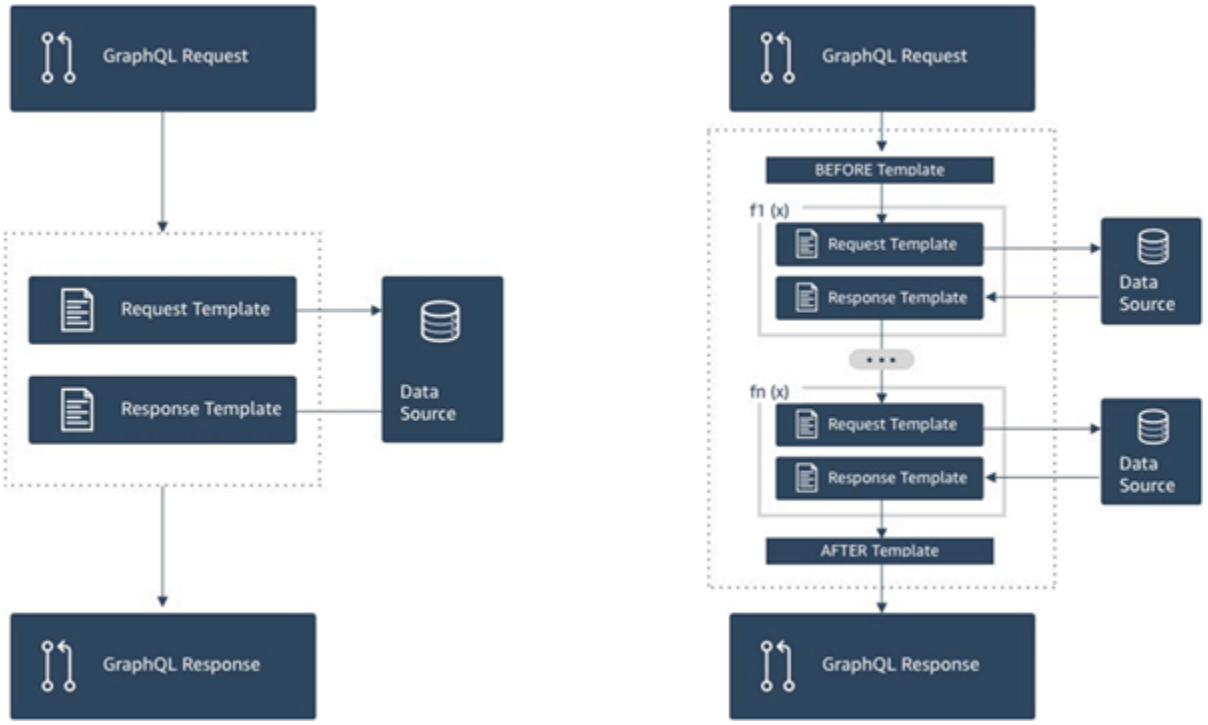
Solucionadores de canalización

Los solucionadores de canalización contienen una o varias funciones que se realizan en orden secuencial. Cada función incluye una plantilla de solicitud y otra de respuesta. Un solucionador de canalización también tiene una plantilla Antes y Después alrededor de la secuencia de funciones que contiene la plantilla. La plantilla Después se mapea al tipo de resultado del campo de GraphQL.

Los solucionadores de canalización se diferencian de los solucionadores de unidad en la forma en que la plantilla de respuesta mapea el resultado. Un solucionador de canalización puede mapearse a cualquier resultado que desee, incluida la entrada de otra función o la plantilla Después del solucionador de canalización.

Las funciones de solucionador de canalización permiten escribir lógica común que puede reutilizar en varios solucionadores de su esquema. Las funciones están asociadas directamente a un origen de datos y, como un solucionador de unidad, contienen el mismo formato de plantilla de mapeo de solicitudes y respuestas.

En el siguiente diagrama se muestra el flujo de procesos de un solucionador de unidad a la izquierda y un solucionador de canalización a la derecha.



Los solucionadores de canalización contienen un superconjunto de la funcionalidad que admiten los solucionadores de unidad, entre otros, por el coste de un poco más de complejidad.

Anatomía de un solucionador de canalización

Un solucionador de canalización se compone de una plantilla de mapeo Antes, una plantilla de mapeo Después y una lista de funciones. Cada función tiene una plantilla de mapeo de solicitudes y respuestas que ejecuta con un origen de datos. Puesto que un solucionador de canalización delega la ejecución a una lista de funciones, no está vinculado a ningún origen de datos. Las funciones y los solucionadores de unidad son primitivos que ejecutan la operación frente a los orígenes de

datos. Para obtener más información, consulte Información general sobre las plantillas de mapeo de solucionador.

Plantilla de mapeo Antes

La plantilla de mapeo de solicitudes de un solucionador de canalización, también denominado paso Antes, permite realizar alguna lógica de preparación antes de ejecutar las funciones definidas.

Lista de funciones

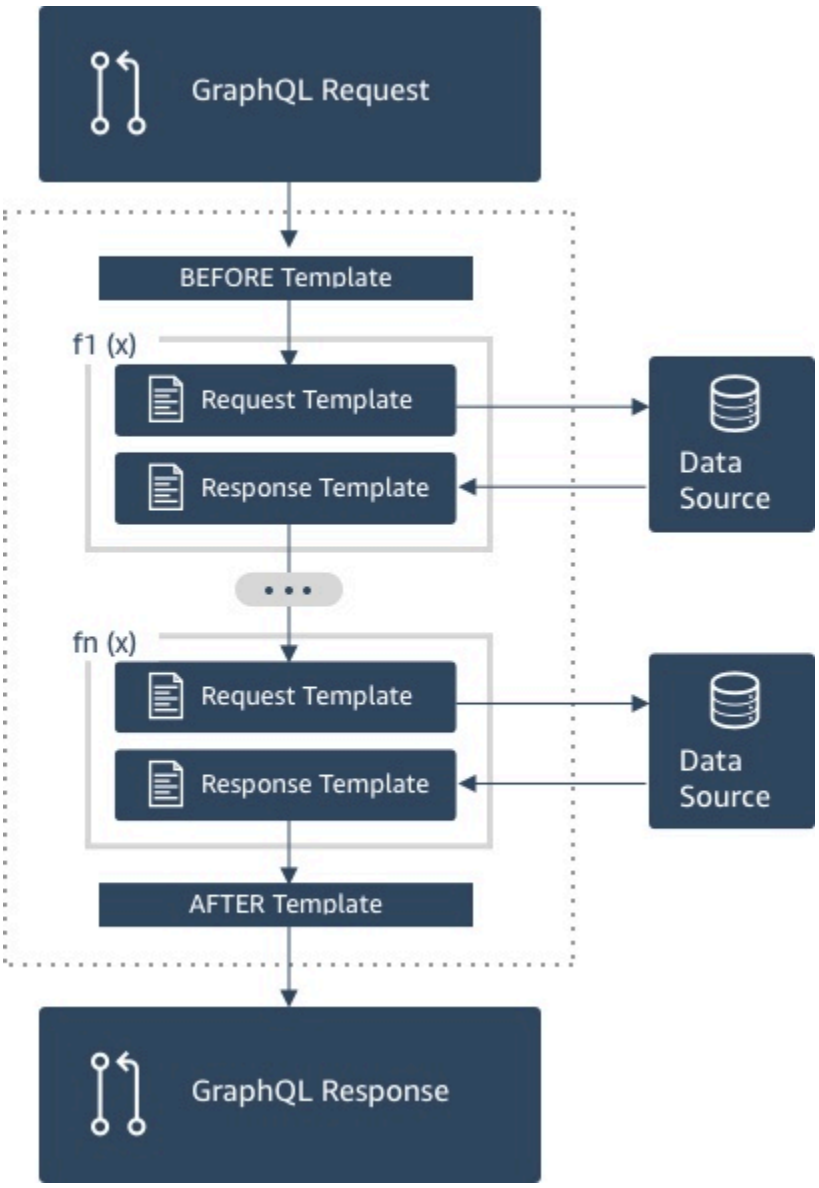
La lista de funciones de un solucionador de canalización se ejecutará de forma secuencial. El resultado evaluado de la plantilla de mapeo de solicitud del solucionador de canalización estará disponible para la primera función como `$ctx.prev.result`. Cada resultado de función está disponible para la siguiente función como `$ctx.prev.result`.

Plantilla de mapeo After (Después)

La plantilla de mapeo de respuestas de un solucionador de canalización, también denominado paso Después, permite realizar alguna lógica de mapeo final a partir del resultado de la última función al tipo de campo de GraphQL esperado. El resultado de la última función en lista de funciones está disponible en la plantilla de mapeo del solucionador de canalización como `$ctx.prev.result` o `$ctx.result`.

Flujo de la ejecución

Con un solucionador de canalización compuesto por dos funciones, la lista siguiente representa el flujo de ejecución cuando se invoca el solucionador:



1. Plantilla de mapeo Antes de solucionador de canalización
2. Función 1: Plantilla de mapeo de solicitud de función
3. Función 1: Invocación de origen de datos
4. Función 1: Plantilla de mapeo de respuesta de función
5. Función 2: Plantilla de mapeo de solicitud de función
6. Función 2: Invocación de origen de datos
7. Función 2: Plantilla de mapeo de respuesta de función
8. Plantilla de mapeo Después de solucionador de canalización

> **Note**
>
> El flujo de ejecución del solucionador de canalización es unidireccional y se define estáticamente en el solucionador.

Utilidades prácticas de Apache Velocity Template Language (VTL)

A medida que aumenta la complejidad de una aplicación, las utilidades y directivas de VTL están para facilitar la productividad de desarrollo. Las siguientes utilidades pueden ayudar cuando se trabaja con solucionadores de canalización.

$ctx.stash

El "stash" es un `Map` que está disponible dentro de cada solucionador y plantilla de mapeo de funciones. La misma instancia stash vive en una única ejecución de solucionador. Esto significa que puede utilizar el stash para transferir datos arbitrarios en plantillas de mapeo de solicitudes y respuestas, así como en funciones de un solucionador de canalización. El stash expone los mismos métodos que la estructura de datos del mapa de Java.

$ctx.prev.result

El `$ctx.prev.result` representa el resultado de la operación anterior que se ha ejecutado en el solucionador de canalización.

Si la operación anterior era la plantilla de mapeo Antes del solucionador de canalización, entonces `$ctx.prev.result` representa el resultado de la evaluación de la plantilla y estará disponible para la primera función de la canalización. Si la operación anterior era la primera función, entonces `$ctx.prev.result` representa el resultado de la primera función y estará disponible para la segunda función de la canalización. Si la operación anterior era la última función, entonces `$ctx.prev.result` representa el resultado de la última función y estará disponible para la plantilla de mapeo Después del solucionador de canalización.

#return(data: Object)

La directiva `#return(data: Object)` es útil si necesita para devolver de forma prematura de cualquier plantilla de mapeo. `#return(data: Object)` es análogo a la palabra clave de return (devolución) en los lenguajes de programación, ya que vuelve desde el bloque de lógica del ámbito más cercano. Esto significa que utilizar `#return` dentro de una plantilla de mapeo vuelve desde

el solucionador. El uso de `#return(data: Object)` en una plantilla de mapeo de solucionador establece `data` en el campo GraphQL. Además, el uso de `#return(data: Object)` partir de una plantilla de mapeo de función vuelve desde la función y continúa la ejecución a la siguiente función en la canalización o a la plantilla de mapeo de respuesta del solucionador.

#return

Equivale a `#return(data: Object)`, pero se devolverá `null` en su lugar.

$util.error

La utilidad `$util.error` es útil para lanzar un error de campo. El uso de `$util.error` dentro de una plantilla de mapeo de función genera un error de campo inmediatamente, lo que impide que se ejecuta funciones posteriores. Para obtener información más detallada y otras firmas de `$util.error`, visite el artículo sobre la referencia de utilidad de la plantilla de mapeo de solucionador.

$util.appendError

El `$util.appendError` es similar a `$util.error()`, siendo la principal diferencia que no interrumpe la evaluación de la plantilla de mapeo. En su lugar, indica que hubo un error con el campo, pero permite evaluar la plantilla y, por tanto, devolver los datos. El uso de `$util.appendError` dentro de una función no interrumpe el flujo de ejecución de la canalización. Para obtener información más detallada y otras firmas de `$util.error`, visite el artículo sobre la referencia de utilidad de la plantilla de mapeo de solucionador.

Ejemplo de plantilla de de

Supongamos que tiene un origen de datos de DynamoDB y un solucionador de unidad para un campo llamado `getPost(id:ID!)` que devuelve un tipo `Post` con la consulta de GraphQL siguiente:

```
getPost(id:1){
    id
    title
    content
}
```

La plantilla del solucionador puede ser parecida a la siguiente:

```
     {
    "version" : "2018-05-29",
    "operation" : "GetItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    }
}
```

Esto sustituiría el valor 1 del parámetro de entrada `id` para `${ctx.args.id}` y generaría el siguiente JSON:

```
     {
    "version" : "2018-05-29",
    "operation" : "GetItem",
    "key" : {
        "id" : { "S" : "1" }
    }
}
```

AWS AppSync utiliza esta plantilla para generar instrucciones para comunicarse con DynamoDB y obtener datos (o realizar otras operaciones, según corresponda). Una vez que se devuelven los datos, AWS AppSync los pasa por una plantilla de mapeo de respuestas opcional que puede utilizar para dar forma a los datos o efectuar operaciones lógicas. Por ejemplo, cuando se obtienen los resultados de DynamoDB podrían tener este aspecto:

```
{
        "id" : 1,
        "theTitle" : "AWS AppSync works offline!",
        "theContent-part1" : "It also has realtime functionality",
        "theContent-part2" : "using GraphQL"
}
```

Puede elegir unir dos de los campos en uno solo con la siguiente plantilla de mapeo de respuesta:

```
{
        "id" : $util.toJson($context.data.id),
        "title" : $util.toJson($context.data.theTitle),
        "content" : $util.toJson("${context.data.theContent-part1}
 ${context.data.theContent-part2}")
}
```

Esta es la forma de los datos después de aplicarles la plantilla:

```
{
        "id" : 1,
        "title" : "AWS AppSync works offline!",
        "content" : "It also has realtime functionality using GraphQL"
}
```

Estos datos se entregan como respuesta al cliente de este modo:

```
{
        "data": {
                "getPost":      {
                        "id" : 1,
                        "title" : "AWS AppSync works offline!",
                        "content" : "It also has realtime functionality using GraphQL"
                }
        }
}
```

Observe que, en la mayoría de los casos, las plantillas de mapeo de respuesta simplemente transfieren los datos, y en ellas la máxima diferencia consiste en devolver un elemento individual o una lista de elementos. En el caso de un elemento individual la transferencia es:

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

Para las listas, la transferencia suele ser:

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

Para ver más ejemplos de solucionadores de unidad y canalización, consulte los tutoriales de solucionadores.

Reglas de deserialización de plantillas de mapeo evaluadas

Las plantillas de asignación se evalúan en función de una cadena. En AWS AppSync, la cadena resultante debe seguir una estructura JSON para ser válida.

Además, se aplican las siguientes reglas de deserialización.

No se permiten claves duplicadas en objetos JSON

Si la cadena de la plantilla de asignación evaluada representa un objeto JSON o contiene un objeto con claves duplicadas, la plantilla de asignación devuelve el siguiente mensaje de error:

`Duplicate field 'aField' detected on Object. Duplicate JSON keys are not allowed.`

Ejemplo de una clave duplicada en una plantilla de asignación de solicitudes evaluada:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "postId": "1",
        "field": "getPost" ## key 'field' has been redefined
    }
}
```

Para corregir este error, no redefina las claves en objetos JSON.

No se permiten caracteres finales en objetos JSON

Si la cadena de la plantilla de asignación evaluada representa un objeto JSON y contiene caracteres extraños al final, la plantilla de asignación devuelve el siguiente mensaje de error:

`Trailing characters at the end of the JSON string are not allowed.`

Ejemplo de caracteres finales en una plantilla de asignación de solicitudes evaluada:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "postId": "1",
    }
}extraneouschars
```

Para corregir este error, asegúrese de que las plantillas evaluadas se evalúen estrictamente en función de JSON.

Guía de programación de plantillas de mapeo de solucionador

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

Este documento es un tutorial de programación similar a un libro de recetas para programar en AWS AppSync con el lenguaje Apache Velocity Template Language (VTL). Si está familiarizado con otros lenguajes de programación, como JavaScript, C o Java, debería resultarle bastante sencillo.

AWS AppSync utiliza VTL para convertir solicitudes de GraphQL de los clientes en una solicitud para el origen de datos. A continuación, invierte el proceso para traducir la respuesta del origen de datos a una respuesta de GraphQL. VTL es un lenguaje de plantilla lógico que permite manipular tanto la solicitud como la respuesta en el flujo de solicitud/respuesta estándar de una aplicación web, utilizando técnicas como:

- Valores predeterminados para nuevos elementos
- Validación y formato de entrada
- Transformación y moldeado de datos
- Iteración en listas, mapas y matrices para puntear o modificar valores
- Filtrado/cambio de las respuestas en función de la identidad del usuario
- Comprobaciones de autorización complejas

Por ejemplo, puede ser conveniente validar un número de teléfono del servicio en un argumento de GraphQL o convertir un parámetro de entrada a mayúsculas antes de almacenarlo en DynamoDB. O quizás quiera que los sistemas cliente proporcionen un código como parte de un argumento de GraphQL, de una reclamación de token JWT o de un encabezado HTTP, y responder únicamente con datos si el código coincide con una cadena concreta de una lista. Son todas comprobaciones lógicas que puede llevar a cabo con VTL en AWS AppSync.

VTL le permite aplicar lógica utilizando técnicas de programación que quizá ya le sean familiares. Sin embargo, solo se puede ejecutar dentro del flujo de solicitud/respuesta estándar, para asegurar que su API de GraphQL sea escalable a medida que crece su base de usuarios. Dado que AWS AppSync también admite AWS Lambda como solucionador, puede escribir funciones de Lambda

en el lenguaje de programación que prefiera (Node.js, Python, Go, Java, etc.) si requiere más flexibilidad.

Configuración

Una técnica habitual que se emplea mientras se aprende un lenguaje es imprimir los resultados (por ejemplo, `console.log(variable)` en JavaScript) para ver lo que ocurre. En este tutorial, lo demostramos mediante la creación de un esquema de GraphQL sencillo y la transferencia de un mapa de valores a una función Lambda. La función Lambda imprime los valores y, a continuación, los utiliza para responder. Esto le permitirá comprender el flujo de solicitud/respuesta y ver diferentes técnicas de programación.

Empiece con la creación del siguiente esquema de GraphQL:

```
type Query {
    get(id: ID, meta: String): Thing
}

type Thing {
    id: ID!
    title: String!
    meta: String
}

schema {
    query: Query
}
```

Ahora, cree la siguiente función AWS Lambda con el lenguaje Node.js:

```
exports.handler = (event, context, callback) => {
    console.log('VTL details: ', event);
    callback(null, event);
};
```

En el panel Orígenes de datos de la consola de AWS AppSync, añada esta función de Lambda como un nuevo origen de datos. Vuelva a la página Esquema de la consola y, a continuación, haga clic en el botón ASOCIAR de la derecha, junto a la consulta `get(...):Thing`. Para la plantilla de la solicitud, elija la plantilla existente en el menú Invoke and forward arguments (Invocar y reenviar argumentos). Para la plantilla de la respuesta, elija Return Lambda result (Devolver resultado Lambda).

Abra Registros de Amazon CloudWatch para la función de Lambda en una ubicación y ejecute la siguiente consulta de GraphQL en la pestaña Consultas de la consola de AWS AppSync:

```
query test {
  get(id:123 meta:"testing"){
    id
    meta
  }
}
```

La respuesta de GraphQL debería incluir `id:123` y `meta:testing`, ya que la función Lambda los devuelve. Al cabo de unos segundos, también debería ver un registro en los Registros de CloudWatch con esta información.

Variables

VTL emplea referencias que puede utilizar para almacenar o manipular los datos. Existen tres tipos de referencias en VTL: variables, propiedades y métodos. Las variables van precedidas de un signo $ y se crean con la directiva `#set`:

```
#set($var = "a string")
```

Las variables almacenan tipos similares a los de otros lenguajes que ya conoce, como números, cadenas, matrices, listas y mapas. Quizá haya observado que en la plantilla de solicitud predeterminada para los solucionadores Lambda se envía una carga JSON:

```
"payload": $util.toJson($context.arguments)
```

Un par de consideraciones importantes: en primer lugar, AWS AppSync proporciona varias funciones útiles para operaciones comunes. En este ejemplo, `$util.toJson` convierte una variable a JSON. En segundo lugar, la variable `$context.arguments` se rellena automáticamente desde una solicitud de GraphQL como objeto de mapa. Puede crear un nuevo mapa del modo siguiente:

```
#set( $myMap = {
  "id": $context.arguments.id,
  "meta": "stuff",
  "upperMeta" : $context.arguments.meta.toUpperCase()
} )
```

Ya ha creado una variable llamada `$myMap` que contiene las claves `id`, `meta` y `upperMeta`. Esto también ilustra algunos puntos:

- `id` se rellena con una clave tomada de los argumentos de GraphQL. Esto es habitual en VTL para obtener argumentos de los clientes.
- `meta` está codificada literalmente con un valor, lo que ilustra el uso de valores predeterminados.
- `upperMeta` transforma el argumento `meta` con el método `.toUpperCase()`.

Coloque el código anterior en la parte superior de la plantilla de solicitud y cambie `payload` para que utilice la nueva variable `$myMap`:

```
"payload": $util.toJson($myMap)
```

Ejecute la función Lambda y verá el cambio en la respuesta y los datos en los registros de CloudWatch. A medida que vaya avanzando por este tutorial, iremos rellenando `$myMap` para que pueda ejecutar pruebas similares.

También puede definir properties_ para las variables, que pueden ser cadenas simples, matrices o JSON:

```
#set($myMap.myProperty = "ABC")
#set($myMap.arrProperty = ["Write", "Some", "GraphQL"])
#set($myMap.jsonProperty = {
    "AppSync" : "Offline and Realtime",
    "Cognito" : "AuthN and AuthZ"
})
```

Referencias silenciosas

Dado que VTL es un lenguaje de plantillas, cada referencia que haga generará un `.toString()` de forma predeterminada. Si la referencia no está definida, imprime la representación de la referencia real en forma de cadena. Por ejemplo:

```
#set($myValue = 5)
##Prints '5'
$myValue

##Prints '$somethingelse'
```

```
$somethingelse
```

Para solucionarlo, VTL emplea una sintaxis de referencia implícita o referencia silenciosa que indica al motor de plantillas que suprima ese comportamiento. La sintaxis utilizada es `$!{}`. Por ejemplo, si cambiamos ligeramente el código anterior para utilizar `$!{somethingelse}`, se suprimirá la impresión:

```
#set($myValue = 5)
##Prints '5'
$myValue

##Nothing prints out
$!{somethingelse}
```

Llamada a métodos

En un ejemplo anterior vimos cómo crear una variable y definir valores de forma simultánea. Esto también puede hacerse en dos pasos, añadiendo datos a un mapa como se muestra a continuación:

```
#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")}
```

SIN EMBARGO, hay que saber algo acerca de este comportamiento. Aunque la notación de referencia silenciosa `$!{}` le permite llamar a métodos, como en el ejemplo anterior, no suprimirá el valor devuelto por el método ejecutado. Esta es la razón por la que observamos `##Prints "first value"` y `##Prints true` en el ejemplo anterior. Esto puede provocar errores al iterar en mapas o listas, por ejemplo al insertar un valor donde ya hay una clave, ya que la salida añadirá cadenas inesperadas a la plantilla durante la evaluación.

La forma de evitarlo consiste a veces en llamar a los métodos utilizando una directiva `#set` y pasar por alto la variable. Por ejemplo:

```
#set ($myMap = {})
```

```
#set($discard = $myMap.put("id", "first value"))
```

Aunque podría utilizar esta técnica en las plantillas, ya que impide que se impriman en ellas cadenas inesperadas, AWS AppSync ofrece una práctica función alternativa que consigue el mismo efecto con una notación más concisa. Esto le evitará preocuparse por estos detalles de implementación. El acceso a esta función puede obtenerse con `$util.quiet()` o su alias `$util.qr()`. Por ejemplo:

```
#set ($myMap = {})
#set ($myList = [])

##Nothing prints out
$util.quiet($myMap.put("id", "first value"))
##Nothing prints out
$util.qr($myList.add("something"))
```

Strings

Al igual que ocurre en muchos lenguajes de programación, puede ser difícil tratar con las cadenas, en especial si se quiere crearlas a partir de variables. Existen algunos aspectos habituales que también aparecen en VTL.

Supongamos que desea insertar datos en forma de cadena en un origen de datos como DynamoDB, pero que los rellena desde una variable, como un argumento de GraphQL. En este caso, la cadena tendría comillas dobles, pero para hacer referencia a la variable en una cadena solo se necesita `"${}"` (y no `!` como en la notación de referencia silenciosa). Esto es similar a una plantilla literal en JavaScript: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

```
#set($firstname = "Jeff")
$!{myMap.put("Firstname", "${firstname}")}
```

Esto puede observarse en las plantillas de solicitud de DynamoDB, como `"author": { "S" : "${context.arguments.author}"}`, cuando se usan argumentos de clientes GraphQL o para la generación automática de ID, como `"id" : { "S" : "$util.autoId()"}`. Esto significa que puede hacer referencia a una variable o al resultado de un método dentro de una cadena para rellenar los datos.

También puede utilizar métodos públicos de la clase String de Java, por ejemplo para extraer una subcadena:

```
#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}"))
```

La concatenación de cadenas es también una tarea muy frecuente. Puede efectuarla solo con referencias a variables o con valores estáticos:

```
#set($s1 = "Hello")
#set($s2 = " World")

$util.qr($myMap.put("concat","$s1$s2"))
$util.qr($myMap.put("concat2","Second $s1 World"))
```

Bucles

Ahora que ha creado variables y ha llamado a métodos, puede agregar algo de lógica a su código. A diferencia de otros lenguajes, VTL solo permite bucles en los que el número de iteraciones está predeterminado. En Velocity no existe `do..while`. Este diseño garantiza que el proceso de evaluación termine siempre y proporciona límites para la escalabilidad cuando se ejecutan las operaciones de GraphQL.

Los bucles se crean con `#foreach` y requieren que proporcione una variable de bucle y un objeto iterable, como una matriz, una lista, un mapa o una colección. Un ejemplo típico de programación con un bucle `#foreach` es iterar para todos los elementos de una colección e imprimirlos, por lo que en nuestro caso los punteamos y los añadimos al mapa:

```
#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
```

En este ejemplo se ilustran varios puntos. El primero es el uso de variables con el operador de rango `[..]` para crear un objeto iterable. A continuación, se hace referencia a cada elemento mediante una variable `$i` con la que se puede operar. En el ejemplo anterior, también puede ver comentarios, que van precedidos de dos almohadillas ##. Esto también ilustra el uso de la variable de bucle tanto en las claves como en los valores, así como de diferentes métodos de concatenación con cadenas.

Observe que `$i` es un número entero, por lo que puede llamar a un método `.toString()`. Esto puede ser útil para los tipos INT de GraphQL.

También puede utilizar un operador de rango directamente, por ejemplo:

```
#foreach($item in [1..5])
    ...
#end
```

Matrices

Hasta ahora ha manipulado un mapa, pero las matrices también son muy comunes en VTL. Las matrices también dan acceso a algunos métodos subyacentes, como `.isEmpty()`, `.size()`, `.set()`, `.get()` y `.add()`, como se muestra a continuación:

```
#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)))
```

El ejemplo anterior utiliza la notación de índice de matriz para recuperar un elemento con `arr2[$idx]`. Puede buscar por nombre en un mapa/diccionario de forma similar:

```
#set($result = {
    "Author" : "Nadia",
    "Topic" : "GraphQL"
})

$util.qr($myMap.put("Author", $result["Author"]))
```

Esto es muy habitual para filtrar los resultados devueltos desde los orígenes de datos en plantillas de respuesta aplicando condiciones.

Comprobaciones condicionales

La sección anterior con `#foreach` muestra algunos ejemplos de uso de lógica para transformar datos con VTL. También puede aplicar comprobaciones condicionales para evaluar los datos en tiempo de ejecución:

```
#if(!$array.isEmpty())
    $util.qr($myMap.put("ifCheck", "Array not empty"))
#else
    $util.qr($myMap.put("ifCheck", "Your array is empty"))
#end
```

La comprobación `#if()` anterior de una expresión booleana está bien, pero también puede utilizar operadores y `#elseif()` para seguir ramificaciones:

```
#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
```

Estos dos ejemplos emplean la negación (!) y la igualdad (==). También pueden usarse ||, &&, **>**, **<**, **>=**, **<=** y **!=**.

```
#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
```

Nota: En las condiciones solo se consideran falsos los valores `Boolean.FALSE` y `null`. El cero (0) y las cadenas vacías (“”) no equivalen a falso.

Operadores

Ningún lenguaje de programación estaría completo sin algunos operadores para realizar algunas acciones matemáticas. A continuación mostramos algunos ejemplos para comenzar:

```
#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))
```

Bucles y condicionales juntos

Es muy común que, al transformar los datos en VTL, por ejemplo antes de escribir o leer de un origen de datos, haya iteraciones sobre objetos y se hagan comprobaciones antes de ejecutar una acción. Combinando algunas de las herramientas de las secciones anteriores se consigue una gran variedad de funciones. Resulta muy útil saber que `#foreach` proporciona automáticamente `.count` para cada elemento:

```
#foreach ($item in $arr2)
  #set($idx = "item" + $foreach.count)
  $util.qr($myMap.put($idx, $item))
```

```
#end
```

Por ejemplo, puede que solo desee puntear los valores de un mapa si su tamaño es menor que un valor determinado. Con el recuento, condiciones y la instrucción `#break` puede hacer lo siguiente:

```
#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
```

El `#foreach` anterior se itera con `.keySet()`, que puede utilizar en mapas. De este modo tiene acceso para obtener `$key` y hacer referencia al valor con `.get($key)`. Los argumentos de GraphQL provenientes de los clientes de AWS AppSync se almacenan en forma de mapa. También es posible iterar por ellos con `.entrySet()`, lo que le da acceso tanto a las claves como a los valores como un conjunto, y le permite rellenar otras variables o efectuar comprobaciones condicionales complejas, como una validación o una transformación de la entrada:

```
#foreach( $entry in $context.arguments.entrySet() )
#if ($entry.key == "XYZ" && $entry.value == "BAD")
    #set($myvar = "...")
  #else
    #break
  #end
#end
```

Otros ejemplos comunes son el rellenado automático con información predeterminada, como las versiones iniciales de los objetos al sincronizar datos (muy importante en la resolución de conflictos) o el propietario predeterminado de un objeto para las comprobaciones de autorización. Mary creó esta publicación del blog, de modo que:

```
#set($myMap.owner ="Mary")
```

```
#set($myMap.defaultOwners = ["Admins", "Editors"])
```

Contexto

Ahora que conoce mejor las comprobaciones lógicas en los solucionadores de AWS AppSync con VTL, podemos fijarnos en el objeto de contexto:

```
$util.qr($myMap.put("context", $context))
```

este objeto contiene toda la información a la que tiene acceso desde una solicitud de GraphQL. Para obtener una explicación detallada, consulte la referencia del contexto.

Filtrado

Hasta ahora, en este tutorial toda la información de la función Lambda se ha devuelto a la consulta de GraphQL con una transformación JSON muy sencilla:

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

La lógica de VTL es igual de eficaz cuando se obtienen respuestas de un origen de datos, en especial para realizar las comprobaciones de autorización para los recursos. Veamos algunos ejemplos. En primer lugar, intente cambiar la plantilla de respuesta del siguiente modo:

```
#set($data = {
    "id" : "456",
    "meta" : "Valid Response"
})

$util.toJson($data)
```

Independientemente de lo que ocurra con la operación de GraphQL, se devuelven al cliente los valores codificados literalmente. Cambie esto ligeramente de forma que el campo `meta` se rellene con la respuesta Lambda definida previamente en el tutorial con el valor `elseIfCheck` cuando explicamos las instrucciones condicionales:

```
#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)
```

`$context.result` es un mapa, por lo que puede utilizar `entrySet()` para aplicar la lógica a las claves o a los valores devueltos. Debido a que `$context.identity` contiene información sobre el usuario que ha realizado la operación de GraphQL, si devuelve información de autorización del origen de datos, entonces podrá decidir si quiere devolver todos, parte o ningún dato a un usuario en función de la lógica aplicada. Cambie la plantilla de respuesta para que tenga el siguiente aspecto:

```
#if($context.result["id"] == 123)
    $util.toJson($context.result)
  #else
    $util.unauthorized()
#end
```

Si ejecuta la consulta de GraphQL, los datos se devolverán con normalidad. Sin embargo, si cambia el argumento id por algo que no sea 123 (`query test { get(id:456 meta:"badrequest") {} }`), recibirá un mensaje de error de autorización.

Encontrará más ejemplos de situaciones de autorización en la sección sobre casos de uso de autorización.

Apéndice: Ejemplo de plantilla

Si ha seguido todo el tutorial, probablemente ya habrá creado esta plantilla paso a paso. En caso de que no lo haya hecho, lo incluimos a continuación para copiarlo y realizar pruebas.

Plantilla de solicitud

```
#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)
}
```

Plantilla de respuesta

```
#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
```

Referencia de contexto de las plantillas de mapeo del solucionador

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSync define un conjunto de variables y funciones para trabajar con plantillas de mapeo de solucionador. Esto simplifica las operaciones lógicas realizadas en los datos en GraphQL. En este documento se describen estas funciones y se proporcionan ejemplos para trabajar con plantillas.

Acceso a `$context`

La variable `$context` es un mapa que contiene toda la información contextual de la invocación al solucionador. Tiene la estructura siguiente:

```
{
   "arguments" : { ... },
```

```
    "source" : { ... },
    "result" : { ... },
    "identity" : { ... },
    "request" : { ... },
    "info": { ... }
}
```

> **Note**
>
> Si intenta obtener acceso a una entrada de diccionario o de mapa (como una entrada de `context`) usando su clave para obtener el valor, Velocity Template Language (VTL) le permite usar directamente la notación `<dictionary-element>.<key-name>`. Sin embargo, esto podría no funcionar en todos los casos, por ejemplo cuando los nombres de clave tengan caracteres especiales (como el guion bajo "_"). Recomendamos usar siempre la notación `<dictionary-element>.get("<key-name>")`.

Cada campo del mapa `$context` se define de la siguiente manera:

Campos **`$context`**

`arguments`

Un mapa que contiene todos los argumentos de GraphQL de este campo.

`identity`

Un objeto que contiene información sobre el intermediario. Consulte Identidad para obtener más información acerca de la estructura de este campo.

`source`

Un mapa que contiene la resolución del campo principal.

`stash`

El "stash" es un mapa que está disponible dentro de cada solucionador y plantilla de mapeo de funciones. La misma instancia stash vive en una única ejecución de solucionador. Esto significa que puede utilizar el "stash" para pasar datos arbitrarios a plantillas de mapeo de solicitudes y respuestas, así como a funciones de un solucionador de canalización. El stash expone los mismos métodos que la estructura de datos del mapa de Java.

`result`

Un contenedor para los resultados de este solucionador. Este campo solo está disponible para las plantillas de mapeo de respuesta.

Por ejemplo, al solucionar el campo `author` de la siguiente consulta:

```
query {
    getPost(id: 1234) {
        postId
        title
        content
        author {
            id
            name
        }
    }
}
```

La variable `$context` completa que está disponible al procesar una plantilla de mapeo de respuesta podría ser:

```
{
  "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`

Es el resultado de cualquier operación previa que se haya ejecutado en un solucionador de canalización.

Si la operación anterior era la plantilla de mapeo Antes del solucionador de canalización, entonces `$ctx.prev.result` representa el resultado de la evaluación de la plantilla y estará disponible para la primera función de la canalización.

Si la operación anterior era la primera función, entonces `$ctx.prev.result` representa el resultado de la primera función y estará disponible para la segunda función de la canalización.

Si la operación anterior era la última función, entonces `$ctx.prev.result` representa el resultado de la última función y estará disponible para la plantilla de mapeo Después del solucionador de canalización.

`info`

Un objeto que contiene información sobre la solicitud de GraphQL. Para obtener información sobre la estructura de este campo, consulte Información.

Identidad

La sección `identity` contiene información sobre el intermediario. La forma de esta sección depende del tipo de autorización de su API de AWS AppSync.

Para obtener más información sobre las opciones de seguridad de AWS AppSync, consulte Autorización y autenticación.

Autorización de **`API_KEY`**

El campo `identity` no se rellena.

Autorización de **`AWS_LAMBDA`**

La `identity` contiene la clave `resolverContext`, que contiene el mismo contenido `resolverContext` devuelto por la función de Lambda que autoriza la solicitud.

Autorización de **`AWS_IAM`**

La `identity` tiene el siguiente formato:

```
{
    "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
}
```

Autorización de **AMAZON_COGNITO_USER_POOLS**

La `identity` tiene el siguiente formato:

```
{
    "sub" : "uuid",
    "issuer" : "string",
    "username" : "string"
    "claims" : { ... },
    "sourceIp" : ["x.x.x.x"],
    "defaultAuthStrategy" : "string"
}
```

Cada campo se define de la siguiente manera:

accountId

El ID de la cuenta de AWS del intermediario.

claims

Las notificaciones que tiene el usuario.

cognitoIdentityAuthType

Autenticado o no según el tipo de identidad.

cognitoIdentityAuthProvider

Una lista separada por comas de la información del proveedor de identidad externo utilizada para obtener las credenciales que se usan para firmar la solicitud.

`cognitoIdentityId`

El ID de identidad de Amazon Cognito del intermediario.

`cognitoIdentityPoolId`

El ID de grupo de identidades de Amazon Cognito asociado al intermediario.

`defaultAuthStrategy`

La estrategia de autorización predeterminada para este intermediario (`ALLOW` o `DENY`).

`issuer`

El emisor del token.

`sourceIp`

La dirección IP de origen de la persona que realiza la llamada que AWS AppSync recibe. Si la solicitud no incluye el encabezado `x-forwarded-for`, el valor de IP de origen solo contiene una dirección IP de la conexión TCP. Si la solicitud incluye un encabezado `x-forwarded-for`, la IP de origen será una lista de las direcciones IP del encabezado `x-forwarded-for`, además de la dirección IP de la conexión TCP.

`sub`

El UUID del usuario autenticado.

`user`

El usuario de IAM.

`userArn`

Es el nombre de recurso de Amazon (ARN) del usuario de IAM.

`username`

El nombre de usuario del usuario autenticado. En el caso de una autorización `AMAZON_COGNITO_USER_POOLS`, el valor de nombre de usuario es el valor del atributo cognito:username. En el caso de una autorización `AWS_IAM`, el valor de nombre de usuario es el valor de la entidad principal del usuario de AWS. Si utiliza una autorización de IAM con credenciales proporcionadas desde grupos de identidad de Amazon Cognito, le recomendamos que use `cognitoIdentityId`.

Acceso a los encabezados de solicitud

AWS AppSync permite enviar encabezados personalizados de los clientes y obtener acceso a ellos desde los solucionadores de GraphQL utilizando `$context.request.headers`. Puede utilizar los valores de encabezado para acciones como insertar datos en un origen de datos o incluso efectuar comprobaciones de autorización. Puede usar uno o varios encabezados de solicitud usando `$curl` con una clave de API desde la línea de comandos, como se muestra en los siguientes ejemplos:

Ejemplo de encabezado único

Supongamos que establece un encabezado `custom` con un valor `nadia` como el siguiente:

```
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
```

Se puede obtener acceso a este encabezado con `$context.request.headers.custom`. Por ejemplo, podría encontrarse en la siguiente VTL para DynamoDB:

```
"custom": $util.dynamodb.toDynamoDBJson($context.request.headers.custom)
```

Ejemplo de varios encabezados

También puede pasar varios encabezados en una única solicitud y obtener acceso a ellos en la plantilla de mapeo de solucionadores. Por ejemplo, si el encabezado `custom` se define con dos valores:

```
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
```

Puede obtener acceso a ellos como una matriz, por ejemplo `$context.request.headers.custom[1]`.

> **Note**
>
> AWS AppSync no expone el encabezado de la cookie en `$context.request.headers`.

Acceso al nombre de dominio personalizado de la solicitud

AWS AppSync admite la configuración de un dominio personalizado que puede usar para acceder a sus puntos de conexión de GraphQL y en tiempo real para sus API. Al hacer una solicitud con un nombre de dominio personalizado, puede obtener el nombre de dominio utilizando. `$context.request.domainName`

Cuando se utiliza el nombre de dominio de punto de conexión predeterminado de GraphQL, el valor es. `null`

Información

La sección `info` contiene información sobre la solicitud de GraphQL. Esta sección incluye la siguiente forma:

```
{
    "fieldName": "string",
    "parentTypeName": "string",
    "variables": { ... },
    "selectionSetList": ["string"],
    "selectionSetGraphQL": "string"
}
```

Cada campo se define de la siguiente manera:

`fieldName`

El nombre del campo que se está resolviendo actualmente.

`parentTypeName`

El nombre del tipo principal del campo que se está resolviendo actualmente.

`variables`

Un mapa que contiene todas las variables que se pasan a la solicitud de GraphQL.

`selectionSetList`

Una representación de lista de los campos del conjunto de selección de GraphQL. Solo se hace referencia a los campos con alias por el nombre de alias, no por el nombre de campo. El ejemplo siguiente muestra detalladamente esta estructura.

`selectionSetGraphQL`

Una representación en forma de cadena del conjunto de selección, formateado como lenguaje de definición de esquema (SDL) de GraphQL. Aunque los fragmentos no se combinan en el conjunto de selección, los fragmentos insertados se conservan, como se muestra en el ejemplo siguiente.

> **Note**
>
> Cuando se utiliza `$utils.toJson()` en `context.info`, los valores devueltos por `selectionSetGraphQL` y `selectionSetList` no se serializarán de forma predeterminada.

Por ejemplo, al resolver el campo `getPost` de la siguiente consulta:

```
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
  }
```

```
}
```

La variable `$context.info` completa que está disponible al procesar una plantilla de mapeo podría ser:

```
{
  "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` expone solo los campos que pertenecen al tipo actual. Si el tipo actual es una interfaz o una unión, solo se exponen los campos seleccionados que pertenecen a la interfaz. Por ejemplo, en el caso del esquema siguiente:

```
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
}
```

Y en la siguiente consulta:

```
query {
    node(id: "post1") {
        id
        ... on Post {
            title
        }

        ... on Blog {
            title
        }
    }
}
```

Cuando se llama a `$ctx.info.selectionSetList` con la resolución del campo `Query.node`, solo se expone `id`:

```
"selectionSetList": [
    "id"
]
```

Sanear datos entrantes

Las aplicaciones deben sanear datos entrantes que no sean de confianza para evitar que cualquier parte externa dé un uso fuera de lo previsto a una aplicación. Como el `$context` contiene datos

entrantes del usuario en las propiedades como `$context.arguments`, `$context.identity`, `$context.result`, `$context.info.variables` y `$context.request.headers`, hay que tener cuidado de sanear sus valores en las plantillas de mapeo.

Dado que las plantillas de asignación representan archivos JSON, el saneamiento de la entrada la toma forma de escapar de caracteres reservados JSON a partir de cadenas que representan las entradas del usuario. Se recomienda usar la utilidad `$util.toJson()` para escapar caracteres JSON reservados de valores de cadenas sensibles al colocarlos en una plantilla de asignación.

Por ejemplo, en la siguiente plantilla de mapeo de la solicitud de Lambda, como hemos accedido a una cadena de entrada de cliente no segura (`$context.arguments.id`), la envolvemos con `$util.toJson()` para evitar que los caracteres JSON no escapados rompan la plantilla JSON.

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "postId": $util.toJson($context.arguments.id)
    }
}
```

A diferencia de la siguiente plantilla de asignación, donde insertamos `$context.arguments.id` directamente sin sanear. Esto no funciona en cadenas que contienen comillas sin escapar u otros caracteres reservados en JSON, y podrían dejar la plantilla abierta a errores.

```
## 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.
    }
}
```

Referencia de utilidad de la plantilla de mapeo de solucionador

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSyncdefine un conjunto de utilidades que se pueden utilizar en un solucionador de GraphQL para simplificar las interacciones con las fuentes de datos. Algunas de estas utilidades son para uso general con cualquier origen de datos, como la generación de identificadores o marcas de tiempo. Otras son específicas de un tipo de origen de datos.

Temas

- Aplicaciones auxiliares de utilidades en $util
- AWS AppSync directivas
- Aplicaciones auxiliares de tiempo en $util.time
- Aplicaciones auxiliares de lista en $util.list
- Aplicaciones auxiliares de mapas en $util.map
- Aplicaciones auxiliares de DynamoDB en $util.dynamodb
- Auxiliares de Amazon RDS en $util.rds
- Aplicaciones auxiliares para HTTP en $util.http
- Aplicaciones auxiliares para XML en $util.xml
- Aplicaciones auxiliares de transformación en $util.transform
- Aplicaciones auxiliares de matemáticas en $util.math
- Aplicaciones auxiliares de cadena en $util.str
- Extensiones

Aplicaciones auxiliares de utilidades en $util

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

La variable `$util` contiene métodos de utilidad generales que ayudan a trabajar con los datos. A menos que se especifique lo contrario, todas las utilidades usan el juego de caracteres UTF-8.

Utilidades de análisis de JSON

Lista de utilidades de análisis de JSON

`$util.parseJson(String) : Object`

Toma un elemento JSON en forma de cadena y devuelve una representación del resultado en forma de objeto.

`$util.toJson(Object) : String`

Toma un objeto y devuelve una representación JSON en forma de cadena de dicho objeto.

Utilidades de codificación

Lista de utilidades de codificación

`$util.urlEncode(String) : String`

Devuelve la cadena de entrada como una cadena `application/x-www-form-urlencoded` codificada.

`$util.urlDecode(String) : String`

Descodifica una cadena `application/x-www-form-urlencoded` codificada y la devuelve a su forma no codificada.

`$util.base64Encode( byte[] ) : String`

Codifica la entrada en una cadena codificada en base64.

`$util.base64Decode(String) : byte[]`

Descodifica los datos de una cadena codificada en base64.

Utilidades de generación de ID

Lista de utilidades de generación de ID

`$util.autoId() : String`

Devuelve un UUID de 128 bits generado de forma aleatoria.

`$util.autoUlid() : String`

Devuelve un ULID (identificador ordenable lexicográficamente único y universal) de 128 bits generado de forma aleatoria.

`$util.autoKsuid() : String`

Devuelve un KSUID (identificador único ordenable por K) de 128 bits generado de forma aleatoria codificado en base62 como una cadena con una longitud de 27.

Utilidades de error

Lista de utilidades de error

`$util.error(String)`

Genera un error personalizado. Utilícela en las plantillas de mapeo de solicitudes o de respuestas para detectar un error en la solicitud o en el resultado de la invocación.

`$util.error(String, String)`

Genera un error personalizado. Utilícela en las plantillas de mapeo de solicitudes o de respuestas para detectar un error en la solicitud o en el resultado de la invocación. También puede especificar un `errorType`.

`$util.error(String, String, Object)`

Genera un error personalizado. Utilícela en las plantillas de mapeo de solicitudes o de respuestas para detectar un error en la solicitud o en el resultado de la invocación. También puede especificar un campo `errorType` y `data`. El valor de `data` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL.

> **Note**
>
> `data` se filtrará en función de la selección de consulta establecida.

`$util.error(String, String, Object, Object)`

Genera un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. También se pueden especificar los campos `errorType`, `data` y `errorInfo`. El valor de `data` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL.

> **Note**
>
> `data` se filtrará en función de la selección de consulta establecida. El valor de `errorInfo` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL.
> `errorInfo` NO se filtrará en función de la selección de consulta establecida.

`$util.appendError(String)`

Adjunta un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. A diferencia de `$util.error(String)`, la evaluación de la plantilla no se interrumpirá, de modo podrán devolverse datos al intermediario.

`$util.appendError(String, String)`

Adjunta un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. También se puede especificar un valor `errorType`. A diferencia de `$util.error(String, String)`, la evaluación de la plantilla no se interrumpirá, de modo podrán devolverse datos al intermediario.

`$util.appendError(String, String, Object)`

Adjunta un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. También se puede especificar un valor `errorType` y un campo `data`. A diferencia de `$util.error(String, String, Object)`, la evaluación de la plantilla no se interrumpirá,

de modo podrán devolverse datos al intermediario. El valor de `data` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL.

> **Note**
>
> `data` se filtrará en función de la selección de consulta establecida.

`$util.appendError(String, String, Object, Object)`

Adjunta un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. También se pueden especificar los campos `errorType`, `data` y `errorInfo`. A diferencia de `$util.error(String, String, Object, Object)`, la evaluación de la plantilla no se interrumpirá, de modo podrán devolverse datos al intermediario. El valor de `data` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL.

> **Note**
>
> `data` se filtrará en función de la selección de consulta establecida. El valor de `errorInfo` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL.
> `errorInfo` NO se filtrará en función de la selección de consulta establecida.

Utilidades de validación condicional

Lista de utilidades de validación condicional

`$util.validate(Boolean, String) : void`

Si la condición es falsa, lanza una CustomTemplateException con el mensaje especificado.

`$util.validate(Boolean, String, String) : void`

Si la condición es falsa, lanza un CustomTemplateException con el mensaje y el tipo de error especificados.

`$util.validate(Boolean, String, String, Object) : void`

Si la condición es falsa, arroja a CustomTemplateException con el mensaje y el tipo de error especificados, así como los datos que desee devolver en la respuesta.

Utilidades de comportamiento nulo

Lista de utilidades de comportamiento nulo

`$util.isNull(Object) : Boolean`

Devuelve el valor true si el objeto suministrado es nulo.

`$util.isNullOrEmpty(String) : Boolean`

Devuelve el valor true si los datos proporcionados son nulos o una cadena vacía. De lo contrario, devuelve el valor false.

`$util.isNullOrBlank(String) : Boolean`

Devuelve el valor true si los datos proporcionados son nulos o una cadena en blanco. De lo contrario, devuelve el valor false.

`$util.defaultIfNull(Object, Object) : Object`

Devuelve el primer objeto si no es nulo. De lo contrario devuelve el segundo objeto como "objeto predeterminado".

`$util.defaultIfNullOrEmpty(String, String) : String`

Devuelve la primera cadena si no es nula ni está vacía. De lo contrario devuelve la segunda cadena como "cadena predeterminada".

`$util.defaultIfNullOrBlank(String, String) : String`

Devuelve la primera cadena si no es nula ni está en blanco. De lo contrario devuelve la segunda cadena como "cadena predeterminada".

Utilidades de coincidencia de patrones

Lista de utilidades de coincidencia de tipos y patrones

`$util.typeOf(Object) : String`

Devuelve una cadena que describe el tipo de objeto. Las identificaciones de tipos admitidas son: "Null", "Number", "String", "Map", "List" y "Boolean". Si no puede identificarse un tipo, el tipo devuelto es "Object".

```
$util.matches(String, String) : Boolean
```

Devuelve un valor true si el patrón especificado en el primer argumento coincide con los datos proporcionados en el segundo argumento. El patrón tiene que ser una expresión regular, por ejemplo `$util.matches("a*b", "aaaaab")`. La funcionalidad se basa en Pattern, que puede consultar para obtener más información.

```
$util.authType() : String
```

Devuelve una cadena que describe el tipo de autenticación múltiple que utiliza una solicitud y devuelve "Autorización de IAM", "Autorización del grupo de usuarios", "Autorización de Open ID Connect" o "Autorización de la clave de API".

Utilidades de validación de objetos

Lista de utilidades de validación de objetos

```
$util.isString(Object) : Boolean
```

Devuelve el valor true si el objeto es una cadena.

```
$util.isNumber(Object) : Boolean
```

Devuelve el valor true si el objeto es un número.

```
$util.isBoolean(Object) : Boolean
```

Devuelve el valor true si el objeto es un valor booleano.

```
$util.isList(Object) : Boolean
```

Devuelve el valor true si el objeto es una lista.

```
$util.isMap(Object) : Boolean
```

Devuelve el valor true si el objeto es un mapa.

CloudWatch utilidades de registro

CloudWatch lista de utilidades de registro

`$util.log.info(Object) : Void`

Registra la representación en cadena del objeto proporcionado en el flujo de registro solicitado cuando el registro a nivel de solicitud y de campo está habilitado con el nivel de CloudWatch registro en una API. `ALL`

`$util.log.info(String, Object...) : Void`

Registra la representación en cadena de los objetos proporcionados en el flujo de registro solicitado cuando el registro a nivel de solicitud y de campo está habilitado con el nivel de CloudWatch registro en una API. `ALL` Esta utilidad reemplazará todas las variables indicadas con "{}" en la primera cadena de formato de entrada por la representación de cadena de los objetos proporcionados en orden.

`$util.log.error(Object) : Void`

Registra la representación en cadena del objeto proporcionado en el flujo de registro solicitado cuando el registro a nivel de campo está habilitado con el nivel de CloudWatch registro `ERROR` o el nivel de registro en una API. `ALL`

`$util.log.error(String, Object...) : Void`

Registra la representación en cadena de los objetos proporcionados en el flujo de registro solicitado cuando el registro a nivel de campo está habilitado con el nivel de CloudWatch registro `ERROR` o el nivel de registro en una API. `ALL` Esta utilidad reemplazará todas las variables indicadas con "{}" en la primera cadena de formato de entrada por la representación de cadena de los objetos proporcionados en orden.

Utilidades de comportamiento del valor devuelto

Lista de utilidades de comportamiento del valor devuelto

`$util.qr()` y `$util.quiet()`

Ejecuta una instrucción VTL y suprime el valor devuelto. Esto resulta útil para ejecutar métodos sin utilizar marcadores de posición temporales, por ejemplo para añadir elementos a un mapa. Por ejemplo:

```
#set ($myMap = {})
#set($discard = $myMap.put("id", "first value"))
```

se convierte en:

```
#set ($myMap = {})
$util.qr($myMap.put("id", "first value"))
```

`$util.escapeJavaScript(String) : String`

Devuelve la cadena de entrada como cadena de JavaScript escape.

`$util.urlEncode(String) : String`

Devuelve la cadena de entrada como una cadena `application/x-www-form-urlencoded` codificada.

`$util.urlDecode(String) : String`

Descodifica una cadena `application/x-www-form-urlencoded` codificada y la devuelve a su forma no codificada.

`$util.base64Encode( byte[] ) : String`

Codifica la entrada en una cadena codificada en base64.

`$util.base64Decode(String) : byte[]`

Descodifica los datos de una cadena codificada en base64.

`$util.parseJson(String) : Object`

Toma un elemento JSON en forma de cadena y devuelve una representación del resultado en forma de objeto.

`$util.toJson(Object) : String`

Toma un objeto y devuelve una representación JSON en forma de cadena de dicho objeto.

`$util.autoId() : String`

Devuelve un UUID de 128 bits generado de forma aleatoria.

`$util.autoUlid() : String`

Devuelve un ULID (identificador ordenable lexicográficamente único y universal) de 128 bits generado de forma aleatoria.

`$util.autoKsuid() : String`

Devuelve un KSUID (identificador único ordenable por K) de 128 bits generado de forma aleatoria codificado en base62 como una cadena con una longitud de 27.

`$util.unauthorized()`

Genera el código `Unauthorized` para el campo que se está resolviendo. Utilícela en las plantillas de mapeo de solicitudes o de respuestas para determinar si se debe permitir al intermediario que resuelva el campo.

`$util.error(String)`

Genera un error personalizado. Utilícela en las plantillas de mapeo de solicitudes o de respuestas para detectar un error en la solicitud o en el resultado de la invocación.

`$util.error(String, String)`

Genera un error personalizado. Utilícela en las plantillas de mapeo de solicitudes o de respuestas para detectar un error en la solicitud o en el resultado de la invocación. También puede especificar un `errorType`.

`$util.error(String, String, Object)`

Genera un error personalizado. Utilícela en las plantillas de mapeo de solicitudes o de respuestas para detectar un error en la solicitud o en el resultado de la invocación. También puede especificar un campo `errorType` y `data`. El valor de `data` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL. Nota: `data` se filtrará en función de la selección de consulta establecida.

`$util.error(String, String, Object, Object)`

Genera un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. También se puede especificar un campo `errorType`, un campo `data` y un campo `errorInfo`. El valor de `data` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL. Nota: `data` se filtrará en función de la selección de consulta establecida. El valor de `errorInfo` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL. Nota: `errorInfo` NO se filtrará en función de la selección de consulta establecida.

`$util.appendError(String)`

Adjunta un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. A diferencia de `$util.error(String)`, la evaluación de la plantilla no se interrumpirá, de modo podrán devolverse datos al intermediario.

`$util.appendError(String, String)`

Adjunta un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. También se puede especificar un valor `errorType`. A diferencia de `$util.error(String, String)`, la evaluación de la plantilla no se interrumpirá, de modo podrán devolverse datos al intermediario.

`$util.appendError(String, String, Object)`

Adjunta un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. También se puede especificar un valor `errorType` y un campo `data`. A diferencia de `$util.error(String, String, Object)`, la evaluación de la plantilla no se interrumpirá, de modo podrán devolverse datos al intermediario. El valor de `data` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL. Nota: `data` se filtrará en función de la selección de consulta establecida.

`$util.appendError(String, String, Object, Object)`

Adjunta un error personalizado. Se puede utilizar en las plantillas de mapeo de solicitud o de respuesta si la plantilla detecta un error en la solicitud o en el resultado de la invocación. También se puede especificar un campo `errorType`, un campo `data` y un campo `errorInfo`. A diferencia de `$util.error(String, String, Object, Object)`, la evaluación de la plantilla no se interrumpirá, de modo podrán devolverse datos al intermediario. El valor de `data` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL. Nota: `data` se filtrará en función de la selección de consulta establecida. El valor de `errorInfo` se añadirá al bloque `error` correspondiente dentro de `errors` en la respuesta de GraphQL. Nota: `errorInfo` NO se filtrará en función de la selección de consulta establecida.

`$util.validate(Boolean, String) : void`

Si la condición es falsa, lanza a CustomTemplateException con el mensaje especificado.

`$util.validate(Boolean, String, String) : void`

Si la condición es falsa, lanza un CustomTemplateException con el mensaje y el tipo de error especificados.

`$util.validate(Boolean, String, String, Object) : void`

Si la condición es falsa, arroja a CustomTemplateException con el mensaje y el tipo de error especificados, así como los datos que desee devolver en la respuesta.

`$util.isNull(Object) : Boolean`

Devuelve el valor true si el objeto suministrado es nulo.

`$util.isNullOrEmpty(String) : Boolean`

Devuelve el valor true si los datos proporcionados son nulos o una cadena vacía. De lo contrario, devuelve el valor false.

`$util.isNullOrBlank(String) : Boolean`

Devuelve el valor true si los datos proporcionados son nulos o una cadena en blanco. De lo contrario, devuelve el valor false.

`$util.defaultIfNull(Object, Object) : Object`

Devuelve el primer objeto si no es nulo. De lo contrario devuelve el segundo objeto como "objeto predeterminado".

`$util.defaultIfNullOrEmpty(String, String) : String`

Devuelve la primera cadena si no es nula ni está vacía. De lo contrario devuelve la segunda cadena como "cadena predeterminada".

`$util.defaultIfNullOrBlank(String, String) : String`

Devuelve la primera cadena si no es nula ni está en blanco. De lo contrario devuelve la segunda cadena como "cadena predeterminada".

`$util.isString(Object) : Boolean`

Devuelve el valor true si el objeto es una cadena.

`$util.isNumber(Object) : Boolean`

Devuelve el valor true si el objeto es un número.

`$util.isBoolean(Object) : Boolean`

Devuelve el valor true si el objeto es un valor booleano.

`$util.isList(Object) : Boolean`

Devuelve el valor true si el objeto es una lista.

`$util.isMap(Object) : Boolean`

Devuelve el valor true si el objeto es un mapa.

`$util.typeOf(Object) : String`

Devuelve una cadena que describe el tipo de objeto. Las identificaciones de tipos admitidas son: "Null", "Number", "String", "Map", "List" y "Boolean". Si no puede identificarse un tipo, el tipo devuelto es "Object".

`$util.matches(String, String) : Boolean`

Devuelve un valor true si el patrón especificado en el primer argumento coincide con los datos proporcionados en el segundo argumento. El patrón tiene que ser una expresión regular, por ejemplo `$util.matches("a*b", "aaaaab")`. La funcionalidad se basa en Pattern, que puede consultar para obtener más información.

`$util.authType() : String`

Devuelve una cadena que describe el tipo de autenticación múltiple que utiliza una solicitud y devuelve "Autorización de IAM", "Autorización del grupo de usuarios", "Autorización de Open ID Connect" o "Autorización de la clave de API".

`$util.log.info(Object) : Void`

Registra la representación en cadena del objeto proporcionado en el flujo de registro solicitado cuando el registro a nivel de solicitud y de campo está habilitado con el nivel de CloudWatch registro en una API. `ALL`

`$util.log.info(String, Object...) : Void`

Registra la representación en cadena de los objetos proporcionados en el flujo de registro solicitado cuando el registro a nivel de solicitud y de campo está habilitado con el nivel de CloudWatch registro en una API. `ALL` Esta utilidad reemplazará todas las variables indicadas con "{}" en la primera cadena de formato de entrada por la representación de cadena de los objetos proporcionados en orden.

`$util.log.error(Object) : Void`

Registra la representación en cadena del objeto proporcionado en el flujo de registro solicitado cuando el registro a nivel de campo está habilitado con el nivel de CloudWatch registro `ERROR` o el nivel de registro en una API. `ALL`

`$util.log.error(String, Object...) : Void`

Registra la representación en cadena de los objetos proporcionados en el flujo de registro solicitado cuando el registro a nivel de campo está habilitado con el nivel de CloudWatch registro `ERROR` o el nivel de registro en una API. `ALL` Esta utilidad reemplazará todas las variables indicadas con "{}" en la primera cadena de formato de entrada por la representación de cadena de los objetos proporcionados en orden.

`$util.escapeJavaScript(String) : String`

Devuelve la cadena de entrada como cadena de JavaScript escape.

Autorización del solucionador

Lista de autorizaciones del solucionador

`$util.unauthorized()`

Genera el código `Unauthorized` para el campo que se está resolviendo. Utilícela en las plantillas de mapeo de solicitudes o de respuestas para determinar si se debe permitir al intermediario que resuelva el campo.

AWS AppSync directivas

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

AWS AppSync expone directivas para facilitar la productividad de los desarrolladores al escribir en VTL.

Utilidades de directiva

`#return(Object)`

`#return(Object)` permite volver de forma prematura de cualquier plantilla de mapeo. `#return(Object)` es análoga a la palabra clave return (volver) en los lenguajes de

programación, ya que volverá del bloque de lógica del ámbito más cercano. Al utilizar `#return(Object)` dentro de una plantilla de mapeo de solucionador volverá desde el solucionador. Además, al usar `#return(Object)` desde una plantilla de mapeo de función volverá desde la función y continuará la ejecución a la siguiente función de la plantilla de mapeo de respuestas de solucionador o canalización.

`#return`

La directiva `#return` exhibe los mismos comportamientos que `#return(Object)`, pero se devolverá `null` en su lugar.

Aplicaciones auxiliares de tiempo en $util.time

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

La variable `$util.time` contiene métodos de fecha y hora útiles para ayudar a generar marcas de tiempo, convertir entre formatos de fecha y hora y analizar cadenas de fecha y hora. La sintaxis de los formatos de fecha y hora se basa en ella y puede DateTimeFormatterconsultarla para obtener más documentación. A continuación se ofrecen algunos ejemplos, así como una lista de métodos disponibles y sus descripciones.

Utilidades de tiempo

Lista de utilidades de tiempo

`$util.time.nowISO8601() : String`

Devuelve una representación de cadena de la hora UTC en formato ISO8601.

`$util.time.nowEpochSeconds() : long`

Devuelve el número de segundos desde la fecha de inicio de 1970-01-01T00:00:00Z hasta ahora.

`$util.time.nowEpochMilliSeconds() : long`

Devuelve el número de milisegundos desde la fecha de inicio de 1970-01-01T00:00:00Z hasta ahora.

`$util.time.nowFormatted(String) : String`

Devuelve una cadena con la marca de tiempo actual en UTC utilizando el formato especificado en un tipo de entrada String.

`$util.time.nowFormatted(String, String) : String`

Devuelve una cadena con la marca de tiempo actual de una zona horaria utilizando el formato y la zona horaria especificados en tipos de entrada String.

`$util.time.parseFormattedToEpochMilliSeconds(String, String) : Long`

Analiza una marca de tiempo pasada como String junto con un formato y, a continuación, devuelve la marca de tiempo como milisegundos transcurridos desde la fecha de inicio.

`$util.time.parseFormattedToEpochMilliSeconds(String, String, String) : Long`

Analiza una marca de tiempo pasada como String junto con un formato y una zona horaria y, a continuación, la devuelve como milisegundos transcurridos desde la fecha de inicio.

`$util.time.parseISO8601ToEpochMilliSeconds(String) : Long`

Analiza una marca de tiempo ISO8601 pasada como String y, a continuación, la devuelve como milisegundos transcurridos desde la fecha de inicio.

`$util.time.epochMilliSecondsToSeconds(long) : long`

Convierte una marca de tiempo en milisegundos desde la fecha de inicio en una marca de tiempo en segundos de la fecha de inicio.

`$util.time.epochMilliSecondsToISO8601(long) : String`

Convierte una marca de tiempo en milisegundos desde la fecha de inicio en una marca de tiempo ISO8601.

`$util.time.epochMilliSecondsToFormatted(long, String) : String`

Convierte una marca de tiempo en milisegundos desde la fecha de inicio, pasada como tipo long, en una marca de tiempo UTC con el formato suministrado.

`$util.time.epochMilliSecondsToFormatted(long, String, String) : String`

Convierte una marca de tiempo en milisegundos desde la fecha de inicio, pasada como tipo long, en una marca de tiempo para la zona horaria y con el formato suministrados.

Ejemplos de funciones independientes

```
$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
```

Ejemplos de conversión

```
#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
```

Ejemplos de análisis

```
$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
```

Uso con escalares definidos AWS AppSync

Los siguientes formatos son compatibles con `AWSDate`, `AWSDateTime` y `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
```

Aplicaciones auxiliares de lista en $util.list

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

`$util.list` contiene métodos útiles para ayudar con operaciones de lista habituales, como eliminar o retener elementos de una lista para casos de uso de filtro.

Utilidades de lista

`$util.list.copyAndRetainAll(List, List) : List`

Hace una copia superficial de la lista suministrada en el primer argumento y retiene solo los elementos especificados en el segundo argumento, si están presentes. Todos los demás elementos se eliminan de la copia.

`$util.list.copyAndRemoveAll(List, List) : List`

Hace una copia superficial de la lista suministrada en el primer argumento y elimina los elementos especificados en el segundo argumento, si están presentes. Todos los demás elementos se conservan en la copia.

`$util.list.sortList(List, Boolean, String) : List`

Ordena una lista de objetos, que se proporciona en el primer argumento. Si el segundo argumento tiene el valor true, la lista se ordena de forma descendente; si el segundo argumento tiene el valor false, la lista se ordena de forma ascendente. El tercer argumento es el nombre de la cadena de la propiedad utilizada para ordenar una lista de objetos personalizados. Si se trata de una lista compuesta únicamente por los tipos String, Integer, Float o Double, el tercer argumento puede ser cualquier cadena aleatoria. Si no todos los objetos son de la misma clase, se devolverá la lista original. Solo se admiten aquellas listas con un máximo de 1000 objetos. A continuación se muestra un ejemplo del uso de esta utilidad:

```
 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}]
```

Aplicaciones auxiliares de mapas en $util.map

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

`$util.map` contiene métodos útiles para ayudar con operaciones de mapa habituales, como eliminar o retener elementos de un mapa para casos de uso de filtro.

Utilidades de mapa

`$util.map.copyAndRetainAllKeys(Map, List) : Map`

Hace una copia superficial del primer mapa y retiene solo las claves especificadas en la lista, si están presentes. Todas las demás claves se eliminan de la copia.

`$util.map.copyAndRemoveAllKeys(Map, List) : Map`

Hace una copia superficial del primer mapa y elimina todas las entradas cuyas claves se especifican en la lista, si están presentes. Todas las demás claves se conservan en la copia.

Aplicaciones auxiliares de DynamoDB en $util.dynamodb

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

`$util.dynamodb` contiene métodos auxiliares que facilitan la escritura y la lectura de datos en Amazon DynamoDB, como el mapeo y el formato automáticos de los tipos de datos. Estos métodos

están diseñados para mapear automáticamente los tipos primitivos y las listas al formato de entrada de DynamoDB correspondiente, creando una estructura `Map` con el formato `{ "TYPE" : VALUE }`.

Por ejemplo, anteriormente, una plantilla de mapeo de solicitudes para crear un elemento nuevo en DynamoDB podía tener el siguiente aspecto:

```
{
    "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) }
    }
}
```

Si queríamos añadir campos al objeto teníamos que actualizar la consulta de GraphQL en el esquema, y también la plantilla de mapeo de solicitud. Sin embargo, ahora podemos reestructurar nuestra plantilla de mapeo de solicitudes para que recoja automáticamente los nuevos campos de nuestro esquema y los añada a DynamoDB con los tipos correctos:

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

En el ejemplo anterior utilizamos la aplicación auxiliar `$util.dynamodb.toDynamoDBJson(...)` para que tome automáticamente el ID generado y lo convierta en la representación en DynamoDB de un atributo de cadena. A continuación, tomamos todos los argumentos, los convertimos en sus representaciones en DynamoDB y los incluimos en el campo `attributeValues` de la plantilla.

Cada aplicación auxiliar tiene dos versiones: una que devuelve un objeto (por ejemplo, `$util.dynamodb.toString(...)`) y otra que devuelve el objeto como una cadena JSON (por ejemplo, `$util.dynamodb.toStringJson(...)`). En el ejemplo anterior, utilizamos la versión

que devuelve los datos como una cadena JSON. Si desea manipular el objeto antes de usarlo en la plantilla, puede elegir que se devuelva en un objeto en su lugar, como se muestra a continuación:

```
{
    "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)
}
```

En el ejemplo anterior devolvemos los argumentos convertidos como un mapa en lugar de una cadena JSON y después añadimos los campos `version` y `timestamp` antes de incluirlos finalmente en el campo `attributeValues` de la plantilla mediante `$util.toJson(...)`.

La versión JSON de cada una de las aplicaciones auxiliares equivale a encapsular la versión que no es de JSON en `$util.toJson(...)`. Por ejemplo, las siguientes instrucciones son exactamente lo mismo:

```
$util.toStringJson("Hello, World!")
$util.toJson($util.toString("Hello, World!"))
```

toDynamoDB

Lista de utilidades toDynamoDB

`$util.dynamodb.toDynamoDB(Object) : Map`

Herramienta de conversión general de objetos para DynamoDB que convierte objetos de entrada en la representación de DynamoDB correspondiente. Es algo inflexible en cuanto al modo en que representa algunos tipos: por ejemplo, utiliza listas ("L") en lugar de conjuntos ("SS", "NS" "BS"). Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

Ejemplo de cadena

```
Input:      $util.dynamodb.toDynamoDB("foo")
Output:     { "S" : "foo" }
```

Ejemplo de número

```
Input:      $util.dynamodb.toDynamoDB(12345)
Output:     { "N" : 12345 }
```

Ejemplo de booleano

```
Input:      $util.dynamodb.toDynamoDB(true)
Output:     { "BOOL" : true }
```

Ejemplo de lista

```
Input:      $util.dynamodb.toDynamoDB([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

Ejemplo de mapa

```
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`

Lo mismo que `$util.dynamodb.toDynamoDB(Object) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

Utilidades toString

Lista de utilidades toString

`$util.dynamodb.toString(String) : String`

Convierte una cadena de entrada al formato de cadena de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      $util.dynamodb.toString("foo")
Output:     { "S" : "foo" }
```

`$util.dynamodb.toStringJson(String) : Map`

Lo mismo que `$util.dynamodb.toString(String) : String`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

`$util.dynamodb.toStringSet(List<String>) : Map`

Convierte una lista con cadenas al formato de conjunto de cadenas de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      $util.dynamodb.toStringSet([ "foo", "bar", "baz" ])
Output:     { "SS" : [ "foo", "bar", "baz" ] }
```

`$util.dynamodb.toStringSetJson(List<String>) : String`

Lo mismo que `$util.dynamodb.toStringSet(List<String>) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

Utilidades toNumber

Lista de utilidades toNumber

`$util.dynamodb.toNumber(Number) : Map`

Convierte un número al formato de número de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      $util.dynamodb.toNumber(12345)
Output:     { "N" : 12345 }
```

`$util.dynamodb.toNumberJson(Number) : String`

Lo mismo que `$util.dynamodb.toNumber(Number) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

`$util.dynamodb.toNumberSet(List<Number>) : Map`

Convierte una lista de números al formato de conjunto de números de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      $util.dynamodb.toNumberSet([ 1, 23, 4.56 ])
Output:     { "NS" : [ 1, 23, 4.56 ] }
```

`$util.dynamodb.toNumberSetJson(List<Number>) : String`

Lo mismo que `$util.dynamodb.toNumberSet(List<Number>) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

Utilidades toBinary

Lista de utilidades toBinary

`$util.dynamodb.toBinary(String) : Map`

Convierte datos binarios codificados como una cadena en base64 al formato binario de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      $util.dynamodb.toBinary("foo")
```

```
Output:     { "B" : "foo" }
```

`$util.dynamodb.toBinaryJson(String) : String`

Lo mismo que `$util.dynamodb.toBinary(String) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

`$util.dynamodb.toBinarySet(List<String>) : Map`

Convierte una lista de datos binarios codificados como cadenas en base64 al formato de conjunto binario de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      $util.dynamodb.toBinarySet([ "foo", "bar", "baz" ])
Output:     { "BS" : [ "foo", "bar", "baz" ] }
```

`$util.dynamodb.toBinarySetJson(List<String>) : String`

Lo mismo que `$util.dynamodb.toBinarySet(List<String>) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

Utilidades toBoolean

Lista de utilidades toBoolean

`$util.dynamodb.toBoolean(Boolean) : Map`

Convierte un valor booleano al formato booleano correspondiente de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      $util.dynamodb.toBoolean(true)
Output:     { "BOOL" : true }
```

`$util.dynamodb.toBooleanJson(Boolean) : String`

Lo mismo que `$util.dynamodb.toBoolean(Boolean) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

Utilidades toNull

Lista de utilidades toNull

`$util.dynamodb.toNull() : Map`

Devuelve un valor nulo con el formato nulo de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      $util.dynamodb.toNull()
Output:     { "NULL" : null }
```

`$util.dynamodb.toNullJson() : String`

Lo mismo que `$util.dynamodb.toNull() : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

Utilidades toList

Lista de utilidades toList

`$util.dynamodb.toList(List) : Map`

Convierte una lista de objetos al formato de lista de DynamoDB. Cada elemento de la lista se convierte también al formato correspondiente de DynamoDB. Es algo inflexible en cuanto al modo en que representa algunos objetos anidados: por ejemplo, utiliza listas ("L") en lugar de conjuntos ("SS", "NS" "BS"). Esto devuelve un objeto que describe el valor del atributo de DynamoDB.

```
Input:      $util.dynamodb.toList([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

`$util.dynamodb.toListJson(List) : String`

Lo mismo que `$util.dynamodb.toList(List) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

Utilidades toMap

Lista de utilidades toMap

`$util.dynamodb.toMap(Map) : Map`

Convierte un mapa al formato de mapa de DynamoDB. Cada valor del mapa se convierte también al formato de DynamoDB correspondiente. Es algo inflexible en cuanto al modo en que representa algunos objetos anidados: por ejemplo, utiliza listas ("L") en lugar de conjuntos ("SS", "NS" "BS"). Esto devuelve un objeto que describe el valor del atributo de 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`

Lo mismo que `$util.dynamodb.toMap(Map) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

`$util.dynamodb.toMapValues(Map) : Map`

Crea una copia del mapa en la que cada valor se convierte al formato correspondiente de DynamoDB. Es algo inflexible en cuanto al modo en que representa algunos objetos anidados: por ejemplo, utiliza listas ("L") en lugar de conjuntos ("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**
>
> Esto es ligeramente diferente de `$util.dynamodb.toMap(Map) : Map`, ya que solo devuelve el contenido del valor de atributo de DynamoDB y no todo el valor de atributo en sí. Por ejemplo, las siguientes instrucciones son exactamente lo mismo:
>
> ```
> $util.dynamodb.toMapValues($map)
> $util.dynamodb.toMap($map).get("M")
> ```

`$util.dynamodb.toMapValuesJson(Map) : String`

Lo mismo que `$util.dynamodb.toMapValues(Map) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

Utilidades S3Object

Lista de utilidades S3Object

`$util.dynamodb.toS3Object(String key, String bucket, String region) : Map`

Convierte la clave, el bucket y la región a la representación de objeto de S3 de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de 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`

Lo mismo que `$util.dynamodb.toS3Object(String key, String bucket, String region) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

`$util.dynamodb.toS3Object(String key, String bucket, String region, String version) : Map`

Convierte la clave, el bucket, la región y la versión opcional a la representación de objeto de S3 de DynamoDB. Esto devuelve un objeto que describe el valor del atributo de 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`

Lo mismo que `$util.dynamodb.toS3Object(String key, String bucket, String region, String version) : Map`, pero devuelve un valor de atributo de DynamoDB como cadena con codificación JSON.

`$util.dynamodb.fromS3ObjectJson(String) : Map`

Acepta el valor de cadena de un objeto de S3 de DynamoDB y devuelve un mapa que contiene la clave, el bucket, la región y la versión opcional.

```
Input:      $util.dynamodb.fromS3ObjectJson({ "S" : "{ \"s3\" : { \"key\" : \"foo\",
 \"bucket\" : \"bar\", \"region\" : \"baz\", \"version\" = \"beep\" } }" })
Output:     { "key" : "foo", "bucket" : "bar", "region" : "baz", "version" :
 "beep" }
```

Auxiliares de Amazon RDS en $util.rds

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

`$util.rds` contiene métodos auxiliares que dan formato a las operaciones de Amazon RDS al eliminar los datos extraños de los resultados

Lista de utilidades $util.rds

`$util.rds.toJsonString(String serializedSQLResult): String`

Devuelve una `String` transformando el formato de resultado de la operación de API de datos de Amazon Relational Database Service (Amazon RDS) sin procesar stringified en una cadena más concisa. La cadena de devolución es una lista de registros de SQL en serie del conjunto de resultados. Cada registro se representa como un conjunto de pares clave-valor. Las claves son los nombres de columna correspondientes.

Si la instrucción correspondiente en la entrada era una consulta SQL que causa una mutación (por ejemplo INSERT, UPDATE, DELETE), se devolverá una lista vacía. Por ejemplo, la consulta `select * from Books limit 2` proporciona el resultado sin procesar de la operación de datos de 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"
                }
            ]
        }
    ]
}
```

El valor de `util.rds.toJsonString` es:

```
[
  {
    "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`

Es igual que `util.rds.toJsonString`, pero el resultado es un `Object` JSON.

Aplicaciones auxiliares para HTTP en $util.http

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

La utilidad `$util.http` proporciona métodos auxiliares que puede utilizar para gestionar parámetros de solicitud HTTP y añadir encabezados de respuesta.

Lista de utilidades $util.http

`$util.http.copyHeaders(Map) : Map`

Copia el encabezado del mapa sin el conjunto restringido de los encabezados HTTP. Se puede usar para reenviar encabezados de solicitud al siguiente punto de conexión HTTP.

```
{
    ...
    "params": {
        ...
        "headers": $util.http.copyHeaders($ctx.request.headers),
        ...
    },
    ...
}
```

`$util.http.addResponseHeader(String, Object)`

Añade un único encabezado personalizado con el nombre (`String`) y el valor (`Object`) de la respuesta. Se aplican las siguientes restricciones:

- Los nombres de los encabezados no pueden coincidir con ninguno de los AWS AppSync encabezados existentes AWS o restringidos.
- Los nombres del encabezado no pueden comenzar por prefijos restringidos, como `x-amzn-` o `x-amz-`.
- El tamaño de los encabezados de respuesta personalizada no puede superar los 4 KB. Esto incluye los nombres y valores del encabezado.
- Debe definir cada encabezado de respuesta una vez por operación de GraphQL. Sin embargo, si define un encabezado personalizado con el mismo nombre varias veces, la definición más reciente aparecerá en la respuesta. Todos los encabezados se contabilizan para el límite de tamaño del encabezado independientemente de los nombres.

```
...
$util.http.addResponseHeader("itemsCount", 7)
$util.http.addResponseHeader("render", $ctx.args.render)
...
```

`$util.http.addResponseHeaders(Map)`

Añade varios encabezados de respuesta a la respuesta desde el mapa de nombres (`String`) y valores (`Object`) especificado. Las mismas limitaciones enumeradas para el método `addResponseHeader(String, Object)` también se aplican a este método.

```
...
#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)
...
```

Aplicaciones auxiliares para XML en $util.xml

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

`$util.xml` contiene métodos auxiliares que facilitan la conversión de las respuestas XML a JSON o a un diccionario.

Lista de utilidades $util.xml

`$util.xml.toMap(String) : Map`

Convierte una cadena XML a un diccionario.

```
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`

Convierte una cadena XML en una cadena JSON. Esto es similar a toMap, salvo que el resultado es una cadena. Esto resulta útil si se desea convertir directamente y devolver la respuesta XML de un objeto HTTP a JSON.

`$util.xml.toJsonString(String, Boolean) : String`

Convierte una cadena XML en una cadena JSON con un parámetro booleano opcional para determinar si se desea codificar la cadena JSON.

Aplicaciones auxiliares de transformación en $util.transform

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

`$util.transform` contiene métodos auxiliares que facilitan las operaciones complejas sobre orígenes de datos, como las operaciones de filtro de Amazon DynamoDB.

Aplicaciones auxiliares de transformación

Lista de utilidades de aplicaciones auxiliares de transformación

`$util.transform.toDynamoDBFilterExpression(Map) : Map`

Convierte una cadena de entrada en una expresión de filtro que puede usarse en 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`

Convierte la entrada dada en su expresión DSL de OpenSearch consulta equivalente y la devuelve como una cadena 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"
                  }
                }
              ]
            }
          }
      ]
    }
}
```

Se entiende que el operador predeterminado es AND.

Filtros de suscripción de aplicaciones auxiliares de transformación

Lista de utilidades de filtros de suscripción de aplicaciones auxiliares de transformación

`$util.transform.toSubscriptionFilter(Map) : Map`

Convierte un objeto de entrada `Map` en un objeto de expresión `SubscriptionFilter`. El método `$util.transform.toSubscriptionFilter` se utiliza como entrada a la extensión `$extensions.setSubscriptionFilter()`. Para obtener más información, consulte el artículo sobre extensiones.

`$util.transform.toSubscriptionFilter(Map, List) : Map`

Convierte un objeto de entrada `Map` en un objeto de expresión `SubscriptionFilter`. El método `$util.transform.toSubscriptionFilter` se utiliza como entrada a la extensión `$extensions.setSubscriptionFilter()`. Para obtener más información, consulte el artículo sobre extensiones.

El primer argumento es el objeto de entrada `Map` que se convierte en el objeto de expresión `SubscriptionFilter`. El segundo argumento es una `List` de nombres de campo que se omiten en el primer objeto de entrada `Map` al construir el objeto de expresión `SubscriptionFilter`.

`$util.transform.toSubscriptionFilter(Map, List, Map) : Map`

Convierte un objeto de entrada `Map` en un objeto de expresión `SubscriptionFilter`. El método `$util.transform.toSubscriptionFilter` se utiliza como entrada a la extensión `$extensions.setSubscriptionFilter()`. Para obtener más información, consulte el artículo sobre extensiones.

El primer argumento es el objeto de entrada `Map` que se convierte en el objeto de expresión `SubscriptionFilter`, el segundo argumento es una `List` de nombres de campo que se omitirán en el primer objeto de entrada `Map` y el tercer argumento es un objeto de entrada `Map` de reglas estrictas que se incluye al construir el objeto de expresión `SubscriptionFilter`. Estas reglas estrictas se incluyen en el objeto de expresión `SubscriptionFilter` de tal forma que se cumpla al menos una de las reglas para pasar el filtro de suscripción.

Argumentos de filtro de suscripción

En la siguiente tabla se explica cómo se definen los argumentos de las siguientes utilidades:

- `$util.transform.toSubscriptionFilter(Map) : Map`
- `$util.transform.toSubscriptionFilter(Map, List) : Map`
- `$util.transform.toSubscriptionFilter(Map, List, Map) : Map`

Argument 1: Map

El argumento 1 es un objeto `Map` con los siguientes valores clave:

- nombres de los campos
- "and"
- "or"

En el caso de los nombres de campo como claves, las condiciones de las entradas de estos campos adoptan la forma de `"operator" : "value"`.

En el siguiente ejemplo se muestra cómo se pueden añadir entradas a `Map`:

```
"field_name" : {
                    "operator1" : value
               }

## We can have multiple conditions for the same field_name:

"field_name" : {
                    "operator1" : value
                    "operator2" : value
                    .
                    .
                    .
               }
```

Cuando un campo tiene dos o más condiciones, se considera que todas estas condiciones utilizan la operación OR.

La entrada `Map` también puede tener "and" y "or" como claves, lo que implica que todas las entradas que incluyen deben unirse con la lógica AND u OR, según la clave. Los valores clave "and" y "or" esperan una matriz de condiciones.

```
"and" : [
```

```
            {
                "field_name1" : {
                    "operator1" : value
                }
             },

             {
                "field_name2" : {
                    "operator1" : value
                }
             },
             .
             .
        ].
```

Tenga en cuenta que puede anidar "and" y "or". Es decir, puede tener "and" y "or" anidados dentro de otro bloque "and" y "or". Sin embargo, no funciona para campos simples.

```
"and" : [

            {
                "field_name1" : {
                    "operator" : value
                }
             },

             {
                "or" : [
                            {
                                "field_name2" : {
                                    "operator" : value
                                }
                            },

                            {
                                "field_name3" : {
                                    "operator" : value
                                }
                            }

                        ].
```

En el siguiente ejemplo se muestra una entrada de argumento 1 con `$util.transform.toSubscriptionFilter(Map) : Map`.

Entradas

Argumento 1: mapa:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 2000
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Salida

El resultado es un objeto `Map`:

```
{
```

```
"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

El argumento 2 contiene una `List` de nombres de campo que no deberían tenerse en cuenta en la entrada `Map` (argumento 1) al construir el objeto de expresión `SubscriptionFilter`. `List` también puede estar vacía.

En el siguiente ejemplo se muestran las entradas de argumento 1 y argumento 2 con `$util.transform.toSubscriptionFilter(Map, List) : Map`.

Entradas

Argumento 1: mapa:

```
{

  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 20
      }
    }
  ],
  "or": [
```

```
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Argumento 2: lista:

```
["percentageUp", "author"]
```

Salida

El resultado es un objeto `Map`:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    }
  ]
```

```
}
```

Argument 3: Map

El argumento 3 es un objeto `Map` que tiene nombres de campo como valores clave (no puede tener "and" u "or"). En el caso de los nombres de campo como claves, las condiciones de estos campos son entradas en forma de `"operator" : "value"`. A diferencia del argumento 1, el argumento 3 no puede tener varias condiciones en la misma clave. Además, el argumento 3 no tiene una cláusula "and" u "or", por lo que tampoco hay anidamiento involucrado.

El argumento 3 representa una lista de reglas estrictas, que se añaden al objeto de expresión `SubscriptionFilter` para que se cumpla al menos una de estas condiciones para pasar el filtro.

```
{
  "fieldname1": {
    "operator": value
  },
  "fieldname2": {
    "operator": value
  }
}
.
.
.
```

En el siguiente ejemplo se muestran las entradas de argumento 1, argumento 2 y argumento 3 con `$util.transform.toSubscriptionFilter(Map, List, Map) : Map`.

Entradas

Argumento 1: mapa:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
```

```
      }
    },
    {
      "downvotes": {
        "lt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Argumento 2: lista:

```
["percentageUp", "author"]
```

Argumento 3: mapa:

```
{
  "upvotes": {
    "gte": 250
  },
  "author": {
    "eq": "Person1"
  }
}
```

Salida

El resultado es un objeto `Map`:

```
{
  "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"
        }
      ]
    }
  ]
}
```

Aplicaciones auxiliares de matemáticas en $util.math

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

`$util.math` contiene métodos para ayudar con operaciones matemáticas comunes.

Lista de utilidades $util.math

`$util.math.roundNum(Double) : Integer`

Toma un valor doble y lo redondea al entero más cercano.

`$util.math.minVal(Double, Double) : Double`

Toma dos dobles y devuelve el valor mínimo entre los dos dobles.

`$util.math.maxVal(Double, Double) : Double`

Toma dos dobles y devuelve el valor máximo entre los dos dobles.

`$util.math.randomDouble() : Double`

Devuelve un doble aleatorio entre 0 y 1.

> **Important**
>
> Esta función no debe usarse para nada que requiera un alto grado de aleatoriedad de entropía (por ejemplo, criptografía).

`$util.math.randomWithinRange(Integer, Integer) : Integer`

Devuelve un valor entero aleatorio dentro del intervalo especificado. El primer argumento especifica el valor más bajo del intervalo y, el segundo argumento, el valor más alto.

> **Important**
>
> Esta función no debe usarse para nada que requiera un alto grado de aleatoriedad de entropía (por ejemplo, criptografía).

Aplicaciones auxiliares de cadena en $util.str

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

`$util.str` contiene métodos para ayudar con operaciones de cadena comunes.

Lista de utilidades $util.str

`$util.str.toUpper(String) : String`

Toma una cadena y la convierte para que esté completamente en mayúsculas.

`$util.str.toLower(String) : String`

Toma una cadena y la convierte para que esté completamente en minúsculas.

`$util.str.toReplace(String, String, String) : String`

Sustituye una subcadena dentro de una cadena por otra cadena. El primer argumento especifica la cadena en la que se va a realizar la operación de sustitución. El segundo argumento especifica la subcadena que se va a sustituir. El tercer argumento especifica la cadena por la que se va a sustituir el segundo argumento. A continuación se muestra un ejemplo del uso de esta utilidad:

```
INPUT:      $util.str.toReplace("hello world", "hello", "mellow")
OUTPUT:     "mellow world"
```

`$util.str.normalize(String, String) : String`

Normaliza una cadena con uno de los cuatro formatos de normalización de Unicode: NFC, NFD, NFKC o NFKD. El primer argumento es la cadena que se va a normalizar. El segundo argumento es "nfc", "nfd", "nfkc" o "nfkd" y especifica el tipo de normalización que se utilizará en el proceso de normalización.

Extensiones

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

`$extensions` contiene un conjunto de métodos para realizar acciones adicionales en sus solucionadores.

$extensiones. evictFromApiCaché (cadena, cadena, objeto): objeto

Expulsa un elemento de la caché del AWS AppSync servidor. El primer argumento es el nombre de tipo. El segundo argumento es el nombre de campo. El tercer argumento es un objeto que contiene elementos de pares clave-valor que especifican el valor clave de almacenamiento en caché. Debe colocar los elementos del objeto en el mismo orden que las claves de almacenamiento en caché del elemento `cachingKey` del solucionador almacenado en caché.

> **Note**
>
> Esta utilidad solo funciona para mutaciones, no para consultas.

$extensiones. setSubscriptionFilter() filterJsonObject

Define filtros de suscripción mejorados. Cada evento de notificación de suscripción se evalúa con respecto a los filtros de suscripción proporcionados y envía notificaciones a los clientes si todos los filtros se evalúan como `true`. El argumento es `filterJsonObject` como se describe a continuación.

> **Note**
>
> Puede utilizar este método de extensión solo en las plantillas de mapeo de respuestas de un solucionador de suscripción.

$extensiones. setSubscriptionInvalidationFiltro () filterJsonObject

Define los filtros de invalidación de suscripciones. Los filtros de suscripción se evalúan con respecto a la carga de invalidación y, a continuación, invalidan una suscripción determinada si los filtros se evalúan como `true`. El argumento es `filterJsonObject` como se describe a continuación.

> **Note**
>
> Puede utilizar este método de extensión solo en las plantillas de mapeo de respuestas de un solucionador de suscripción.

Argumento: filterJsonObject

El objeto JSON define los filtros de suscripción o invalidación. Es una matriz de filtros en un `filterGroup`. Cada filtro es una colección de filtros individuales.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                }
           ]

        },
        {
           "filters" : [
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
```

```
                }
            ]

        }
    ]
}
```

Cada filtro tiene tres atributos:

- `fieldName`: campo de esquema de GraphQL.
- `operator`: tipo de operador.
- `value`: valores que se van a comparar con el valor `fieldName` de notificación de suscripción.

A continuación se muestra un ejemplo de asignación de estos atributos:

```
{
 "fieldName" : "severity",
 "operator" : "le",
 "value" : $context.result.severity
}
```

Campo: fieldName

El tipo de cadena `fieldName` hace referencia a un campo definido en el esquema de GraphQL que coincide con el `fieldName` en la carga útil de notificaciones de suscripción. Cuando se encuentra una coincidencia, el `value` del campo de esquema de GraphQL se compara con el `value` del filtro de notificaciones de suscripción. En el siguiente ejemplo, el filtro `fieldName` coincide con el campo `service` definido en un tipo de GraphQL determinado. Si la carga útil de notificaciones contiene un campo `service` con un `value` equivalente a `AWS AppSync`, el filtro se evalúa como `true`:

```
{
 "fieldName" : "service",
 "operator" : "eq",
 "value" : "AWS AppSync"
}
```

Campo: valor

El valor puede ser de un tipo diferente según el operador:

- Un solo número o booleano
  - Ejemplos de cadena: `"test"` y `"service"`
  - Ejemplos de número: `1`, `2` y `45.75`
  - Ejemplos de booleano: `true` y `false`
- Pares de números o cadenas
  - Ejemplo de par de cadenas: `["test1","test2"]` y `["start","end"]`
  - Ejemplo de par de números: `[1,4]`, `[67,89]` y `[12.45, 95.45]`
- Matrices de números o cadenas
  - Ejemplo de matriz de cadenas: `["test1","test2","test3","test4","test5"]`
  - Ejemplo de matriz de números: `[1,2,3,4,5]` y `[12.11,46.13,45.09,12.54,13.89]`

Campo: operador

Una cadena que distingue entre mayúsculas y minúsculas con los siguientes valores posibles:

Operador	Descripción	Tipos de valor posibles
eq	Igualdad	entero, flotante, cadena, booleano
Uno	Desigualdad	entero, flotante, cadena, booleano
le	Menor que o igual a	entero, flotante, cadena
lt	Menor que	entero, flotante, cadena
edad	Mayor que o igual a	entero, flotante, cadena
gt	Mayor que	entero, flotante, cadena
contiene	Comprueba si hay una subsecuencia o un valor en el conjunto.	entero, flotante, cadena
no contiene	Comprueba la ausencia de una subsecuencia o la	entero, flotante, cadena

	ausencia de un valor en el conjunto.	
Empieza con	Comprueba si hay un prefijo.	cadena
in	Comprueba si hay elementos coincidentes en la lista.	Matriz de números enteros, flotantes o cadenas
notIn	Comprueba si hay elementos coincidentes que no están en la lista.	Matriz de números enteros, flotantes o cadenas
entre	Entre dos valores	entero, flotante, cadena
Contiene cualquier	Contiene elementos comunes	entero, flotante, cadena

En la siguiente tabla se describe la utilización de cada operador en la notificación de suscripción.

eq (equal)

El operador `eq` se evalúa como `true` si el valor del campo de notificación de suscripción coincide y es igual al valor del filtro de forma estricta. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `service` con el valor equivalente a `AWS AppSync`.

Tipos de valor posibles: entero, flotante, cadena y booleano

```
{
 "fieldName" : "service",
 "operator" : "eq",
 "value" : "AWS AppSync"
}
```

ne (not equal)

El operador `ne` se evalúa como `true` si el valor del campo de notificación de suscripción es diferente al valor del filtro. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `service` con un valor diferente a `AWS AppSync`.

Tipos de valor posibles: entero, flotante, cadena y booleano

```
{
 "fieldName" : "service",
 "operator" : "ne",
 "value" : "AWS AppSync"
}
```

le (less or equal)

El operador `le` se evalúa como `true` si el valor del campo de notificación de suscripción es inferior o igual al valor del filtro. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `size` con un valor inferior o igual a `5`.

Tipos de valor posibles: entero, flotante y cadena

```
{
 "fieldName" : "size",
 "operator" : "le",
 "value" : 5
}
```

lt (less than)

El operador `lt` se evalúa como `true` si el valor del campo de notificación de suscripción es inferior al valor del filtro. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `size` con un valor inferior a `5`.

Tipos de valor posibles: entero, flotante y cadena

```
{
 "fieldName" : "size",
 "operator" : "lt",
 "value" : 5
}
```

ge (greater or equal)

El operador ge se evalúa como `true` si el valor del campo de notificación de suscripción es superior o igual al valor del filtro. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `size` con un valor superior o igual a `5`.

Tipos de valor posibles: entero, flotante y cadena

```
{
 "fieldName" : "size",
 "operator" : "ge",
 "value" : 5
}
```

gt (greater than)

El operador `gt` se evalúa como `true` si el valor del campo de notificación de suscripción es superior al valor del filtro. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `size` con un valor superior a `5`.

Tipos de valor posibles: entero, flotante y cadena

```
{
 "fieldName" : "size",
 "operator" : "gt",
 "value" : 5
}
```

contains

El operador `contains` comprueba si hay una subcadena, una subsecuencia o un valor en un conjunto o un único elemento. Un filtro con el operador `contains` se evalúa como `true` si el valor del campo de notificación de suscripción contiene el valor del filtro. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `seats` con el valor de matriz que contiene el valor `10`.

Tipos de valor posibles: entero, flotante y cadena

```
{
 "fieldName" : "seats",
 "operator" : "contains",
 "value" : 10
}
```

En otro ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `event` con `launch` como subcadena.

```
{
 "fieldName" : "event",
```

```
 "operator" : "contains",
 "value" : "launch"
}
```

notContains

El operador `notContains` busca la ausencia de una subcadena, una subsecuencia o un valor en un conjunto o un único elemento. El filtro con el operador `notContains` se evalúa como `true` si el valor del campo de notificación de suscripción no contiene el valor del filtro. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `seats` con el valor de matriz que no contiene el valor `10`.

Tipos de valor posibles: entero, flotante y cadena

```
{
 "fieldName" : "seats",
 "operator" : "notContains",
 "value" : 10
}
```

En otro ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un valor de campo `event` sin `launch` como su subsecuencia.

```
{
 "fieldName" : "event",
 "operator" : "notContains",
 "value" : "launch"
}
```

beginsWith

El operador `beginsWith` comprueba si hay un prefijo en una cadena. El filtro que contiene el operador `beginsWith` se evalúa como `true` si el valor del campo de notificación de suscripción comienza con el valor del filtro. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `service` con un valor que comienza con `AWS`.

Tipo de valor posible: cadena

```
{
 "fieldName" : "service",
 "operator" : "beginsWith",
```

```
 "value" : "AWS"
}
```

in

El operador `in` comprueba si hay elementos coincidentes en una matriz. El filtro que contiene el operador `in` se evalúa como `true` si el valor del campo de notificación de suscripción existe en una matriz. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `severity` con uno de los valores presentes en la matriz: `[1,2,3]`.

Tipo de valor posible: matriz de enteros, flotantes o cadenas

```
{
 "fieldName" : "severity",
 "operator" : "in",
 "value" : [1,2,3]
}
```

notIn

El operador `notIn` comprueba si faltan elementos en una matriz. El filtro que contiene el operador `notIn` se evalúa como `true` si el valor del campo de notificación de suscripción no existe en la matriz. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `severity` con uno de los valores no presentes en la matriz: `[1,2,3]`.

Tipo de valor posible: matriz de enteros, flotantes o cadenas

```
{
 "fieldName" : "severity",
 "operator" : "notIn",
 "value" : [1,2,3]
}
```

between

El operador `between` comprueba si hay valores entre dos números o cadenas. El filtro que contiene el operador `between` se evalúa como `true` si el valor del campo de notificación de suscripción se encuentra entre el par de valores del filtro. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `severity` con los valores 2, 3 y 4.

Tipos de valor posibles: par de enteros, flotantes o cadenas

```
{
 "fieldName" : "severity",
 "operator" : "between",
 "value" : [1,5]
}
```

containsAny

El operador `containsAny` comprueba si hay elementos en matrices. Un filtro con el operador `containsAny` se evalúa como `true` si la intersección del valor de conjunto de campos de notificación de suscripción y el valor de conjunto de filtros no está vacía. En el siguiente ejemplo, el filtro se evalúa como `true` si la notificación de suscripción tiene un campo `seats` con un valor de matriz que contiene `10` o `15`. Esto significa que el filtro se evaluaría como `true` si la notificación de suscripción tuviera un valor de campo `seats` de `[10,11]` o `[15,20,30]`.

Tipos de valor posibles: entero, flotante o cadena

```
{
 "fieldName" : "seats",
 "operator" : "containsAny",
 "value" : [10, 15]
}
```

Lógica AND

Puede combinar varios filtros mediante la lógica AND definiendo varias entradas dentro del objeto `filters` de la matriz `filterGroup`. En el siguiente ejemplo, los filtros se evalúan como `true` si la notificación de suscripción tiene un campo `userId` con un valor equivalente a `1` Y un valor de campo `group` de `Admin` o `Developer`.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                },
```

```
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
            ]

        }
    ]
}
```

Lógica OR

Puede combinar varios filtros mediante la lógica OR definiendo varios objetos de filtro dentro de la matriz `filterGroup`. En el siguiente ejemplo, los filtros se evalúan como `true` si la notificación de suscripción tiene un campo `userId` con un valor equivalente a `1` O un valor de campo `group` de `Admin` o `Developer`.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                }
           ]

        },
        {
           "filters" : [
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
           ]

        }
    ]
}
```

Excepciones

Tenga en cuenta que existen varias restricciones del uso de filtros:

- En el objeto `filters`, puede haber un máximo de cinco elementos `fieldName` únicos por filtro. Esto significa que puede combinar un máximo de cinco objetos `fieldName` individuales mediante la lógica AND.
- Puede haber un máximo de veinte valores para el operador `containsAny`.
- Puede haber un máximo de cinco valores para los operadores `in` y `notIn`.
- Cada cadena puede tener un máximo de 256 caracteres.
- Cada comparación entre cadenas distingue entre mayúsculas y minúsculas.
- El filtrado de objetos anidados permite hasta cinco niveles anidados de filtrado.
- Cada `filterGroup` puede tener un máximo de 10 `filters`. Esto significa que puede combinar un máximo de 10 `filters` mediante la lógica OR.
  - El operador `in` es un caso especial de lógica OR. En el siguiente ejemplo, hay dos `filters`:

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                },
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
           ]
        }
    ]
}
```

El grupo de filtros anterior se evalúa de la siguiente manera y cuenta para el límite máximo de filtros:

```
{
```

```
    "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

Se utiliza para iniciar la invalidación de una suscripción a partir de una mutación. El argumento es `invalidationJsonObject` como se describe a continuación.

> **Note**
>
> Esta extensión solo se puede usar en las plantillas de mapeo de respuestas de los solucionadores de mutaciones.

Solo puede usar como máximo cinco llamadas al método `$extensions.invalidateSubscriptions()` únicas en una sola solicitud. Si supera este límite, recibirá un error de GraphQL.

Argumento: invalidationJsonObject

El `invalidationJsonObject` define lo siguiente:

- `subscriptionField`: suscripción del esquema de GraphQL a invalidar. Se baraja la posibilidad de invalidar una suscripción única, definida como cadena en el `subscriptionField`.
- `payload`: lista de pares clave-valor que se utiliza como entrada para la invalidación de suscripciones si el filtro de invalidación se evalúa como `true` en comparación con sus valores.

  En el siguiente ejemplo se invalida a los clientes suscritos y conectados que utilizan la suscripción `onUserDelete` cuando el filtro de invalidación definido en el solucionador de suscripción se evalúa como `true` en comparación con el valor `payload`.

```
$extensions.invalidateSubscriptions({
        "subscriptionField": "onUserDelete",
        "payload": {
                "group": "Developer"
                "type" : "Full-Time"
      }
    })
```

Referencia de las plantillas de mapeo de solucionador para DynamoDB

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

El solucionador de DynamoDB de AWS AppSync permite utilizar GraphQL para almacenar y recuperar datos de tablas de Amazon DynamoDB ya existentes en su cuenta. Para funcionar, este

solucionador permite mapear una solicitud de GraphQL entrante a una llamada de DynamoDB y, a continuación, mapear la respuesta de DynamoDB a GraphQL. En esta sección se describen las distintas plantillas de mapeo para las operaciones de DynamoDB admitidas.

GetItem

El documento de mapeo de solicitudes de `GetItem` le permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `GetItem` a DynamoDB, así como especificar lo siguiente:

- La clave del elemento de DynamoDB.
- Si se utiliza una lectura consistente o no.

El documento de mapeo de `GetItem` tiene la siguiente estructura:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "consistentRead" : true,
    "projection" : {
        ...
    }
}
```

Los campos se definen de la siguiente manera:

Campos GetItem

Lista de campos GetItem

`version`

La versión de la definición de plantilla `2017-02-28` y `2018-05-29` se admiten actualmente. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `GetItem`, este valor se debe establecer en `GetItem`. Este valor es obligatorio.

`key`

La clave del elemento de DynamoDB. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`consistentRead`

Indica si se realizará o no una lectura lectura altamente coherente con DynamoDB. Este valor es opcional y de forma predeterminada es `false`.

`projection`

Proyección que se utiliza para especificar los atributos que se devolverán de la operación de DynamoDB. Para obtener más información acerca de las proyecciones, consulte la sección Proyecciones. Este campo es opcional.

El elemento que se devuelve desde DynamoDB se convierte automáticamente a tipos primitivos de GraphQL y JSON, y se encuentra disponible en el contexto del mapeo (`$context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información acerca de las plantillas de mapeo de respuesta, consulte Información general sobre las plantillas de mapeo de solucionador.

Ejemplo

A continuación se muestra una plantilla de mapeo de una consulta de 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
}
```

Para obtener más información sobre la API `GetItem` de DynamoDB, consulte la documentación de la API de DynamoDB.

PutItem

El documento de mapeo de solicitudes de `PutItem` le permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `PutItem` a DynamoDB, así como especificar lo siguiente:

- La clave del elemento de DynamoDB.
- El contenido completo del elemento (compuesto de `key` y `attributeValues`).
- Condiciones para que la operación se lleve a cabo correctamente.

El documento de mapeo de `PutItem` tiene la siguiente estructura:

```
{
    "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
}
```

Los campos se definen de la siguiente manera:

Campos PutItem

Lista de campos PutItem

`version`

La versión de la definición de plantilla `2017-02-28` y `2018-05-29` se admiten actualmente. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `PutItem`, este valor se debe establecer en `PutItem`. Este valor es obligatorio.

`key`

La clave del elemento de DynamoDB. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un "valor con tipo", consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`attributeValues`

El resto de los atributos del elemento que debe colocarse en DynamoDB. Para obtener más información sobre cómo especificar un "valor con tipo", consulte Sistema de tipos (mapeo de solicitud). Este campo es opcional.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `PutItem` sobrescribe todas las entradas existentes para dicho elemento. Para obtener más información sobre las condiciones, consulte Expresiones de condición. Este valor es opcional.

`_version`

Valor numérico que representa la última versión conocida de un elemento. Este valor es opcional. Este campo se utiliza para detectar conflictos y solo se admite en orígenes de datos con control de versiones.

`customPartitionKey`

Cuando se habilita, este valor de cadena modifica el formato de los registros `ds_sk` y `ds_pk` que utiliza la tabla Delta Sync una vez habilitado el control de versiones (para obtener más

información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync). Cuando se habilita, también lo hace el procesamiento de la entrada `populateIndexFields`. Este campo es opcional.

`populateIndexFields`

Valor booleano que, cuando se habilita junto con la **`customPartitionKey`**, crea nuevas entradas para cada registro de la tabla Delta Sync, específicamente en las columnas `gsi_ds_pk` y `gsi_ds_sk`. Para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync. Este campo es opcional.

El elemento que se escribe en DynamoDB se convierte automáticamente a los tipos primitivos de GraphQL y JSON, y está disponible en el contexto de mapeo (`$context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información acerca de las plantillas de mapeo de respuesta, consulte Información general sobre las plantillas de mapeo de solucionador.

Ejemplo 1

El siguiente ejemplo muestra una plantilla de mapeo de una mutación de GraphQL `updateThing(foo: String!, bar: String!, name: String!, version: Int!)`.

Si no existe ningún elemento con la clave especificada, dicho elemento se crea. Si ya existe un elemento con la clave especificada, dicho elemento se sobrescribe.

```
{
    "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)
    }
}
```

Ejemplo 2

El siguiente ejemplo muestra una plantilla de mapeo de una mutación de GraphQL `updateThing(foo: String!, bar: String!, name: String!, expectedVersion: Int!)`.

En este ejemplo se comprueba que el elemento que actualmente se encuentra en DynamoDB tenga en el campo `version` definido con el valor `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)
        }
    }
}
```

Para obtener más información sobre la API `PutItem` de DynamoDB, consulte la documentación de la API de DynamoDB.

UpdateItem

El documento de mapeo de solicitudes de `UpdateItem` le permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `UpdateItem` a DynamoDB, así como especificar lo siguiente:

- La clave del elemento de DynamoDB.
- Una expresión de actualización que describe cómo se actualiza el elemento de DynamoDB.
- Condiciones para que la operación se lleve a cabo correctamente.

El documento de mapeo de `UpdateItem` tiene la siguiente estructura:

```
{
    "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
}
```

Los campos se definen de la siguiente manera:

Campos UpdateItem

Lista de campos UpdateItem

`version`

La versión de la definición de plantilla `2017-02-28` y `2018-05-29` se admiten actualmente. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `UpdateItem`, este valor se debe establecer en `UpdateItem`. Este valor es obligatorio.

`key`

La clave del elemento de DynamoDB. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`update`

La sección `update` permite especificar una expresión de actualización que describe cómo se actualiza el elemento en DynamoDB. Para obtener más información sobre el modo de escribir expresiones de actualización, consulte la documentación de DynamoDB UpdateExpressions. Esta sección es obligatoria.

La sección `update` tiene tres componentes:

`expression`

La expresión de actualización. Este valor es obligatorio.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre usado en la `expression` y el valor tiene que ser una cadena que corresponda al nombre de atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con las sustituciones de marcadores de posición de nombre de atributo de expresión que se usen en la `expression`.

`expressionValues`

Las sustituciones de los marcadores de posición de valor de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de valor usado en la `expression` y el valor tiene que ser un valor con tipo. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor debe especificarse. Este campo es opcional y solo debe rellenarse con las sustituciones de los marcadores de posición de valor de atributo de expresión que se usen en la `expression`.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `UpdateItem` actualiza todas las entradas existentes independientemente de su estado actual.

Para obtener más información sobre las condiciones, consulte Expresiones de condición. Este valor es opcional.

`_version`

Valor numérico que representa la última versión conocida de un elemento. Este valor es opcional. Este campo se utiliza para detectar conflictos y solo se admite en orígenes de datos con control de versiones.

`customPartitionKey`

Cuando se habilita, este valor de cadena modifica el formato de los registros `ds_sk` y `ds_pk` que utiliza la tabla Delta Sync una vez habilitado el control de versiones (para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync). Cuando se habilita, también lo hace el procesamiento de la entrada `populateIndexFields`. Este campo es opcional.

`populateIndexFields`

Valor booleano que, cuando se habilita junto con la **`customPartitionKey`**, crea nuevas entradas para cada registro de la tabla Delta Sync, específicamente en las columnas `gsi_ds_pk` y `gsi_ds_sk`. Para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync. Este campo es opcional.

El elemento que se actualiza en DynamoDB se convierte automáticamente a los tipos primitivos de GraphQL y JSON, y está disponible en el contexto de mapeo (`$context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información acerca de las plantillas de mapeo de respuesta, consulte Información general sobre las plantillas de mapeo de solucionador.

Ejemplo 1

El siguiente ejemplo muestra una plantilla de mapeo de la mutación de GraphQL `upvote(id: ID!)`.

En este ejemplo, se han incrementado en 1 los campos `upvotes` y `version` de un elemento de DynamoDB.

```
{
```

```
    "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 }
        }
    }
}
```

Ejemplo 2

El siguiente ejemplo muestra una plantilla de mapeo de una mutación de GraphQL `updateItem(id: ID!, title: String, author: String, expectedVersion: Int!)`.

Se trata de un ejemplo complejo que inspecciona los argumentos y genera de manera dinámica la expresión de actualización que solo incluye los argumentos que ha proporcionado el cliente. Por ejemplo, si se omiten `title` y `author`, no se actualizan. Si se especifica un argumento pero su valor es `null`, ese campo se elimina del objeto en DynamoDB. Por último, la operación tiene una condición que comprueba si el elemento que está actualmente en DynamoDB tiene el campo `version` establecido en `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)
        }
    }
}
```

Para obtener más información sobre la API `UpdateItem` de DynamoDB, consulte la documentación de la API de DynamoDB.

DeleteItem

El documento de mapeo de solicitudes de `DeleteItem` le permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `DeleteItem` a DynamoDB, así como especificar lo siguiente:

- La clave del elemento de DynamoDB.
- Condiciones para que la operación se lleve a cabo correctamente.

El documento de mapeo de `DeleteItem` tiene la siguiente estructura:

```
{
    "version" : "2018-05-29",
    "operation" : "DeleteItem",
    "customPartitionKey" : "foo",
    "populateIndexFields" : boolean value,
    "key": {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "condition" : {
        ...
    },
    "_version" : 1
}
```

Los campos se definen de la siguiente manera:

Campos DeleteItem

Lista de campos DeleteItem

`version`

La versión de la definición de plantilla `2017-02-28` y `2018-05-29` se admiten actualmente. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `DeleteItem`, este valor se debe establecer en `DeleteItem`. Este valor es obligatorio.

`key`

La clave del elemento de DynamoDB. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un "valor con tipo", consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `DeleteItem` elimina un elemento independientemente de su estado actual. Para obtener más información sobre las condiciones, consulte Expresiones de condición. Este valor es opcional.

`_version`

Valor numérico que representa la última versión conocida de un elemento. Este valor es opcional. Este campo se utiliza para detectar conflictos y solo se admite en orígenes de datos con control de versiones.

`customPartitionKey`

Cuando se habilita, este valor de cadena modifica el formato de los registros `ds_sk` y `ds_pk` que utiliza la tabla Delta Sync una vez habilitado el control de versiones (para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync). Cuando se habilita, también lo hace el procesamiento de la entrada `populateIndexFields`. Este campo es opcional.

`populateIndexFields`

Valor booleano que, cuando se habilita junto con la **`customPartitionKey`**, crea nuevas entradas para cada registro de la tabla Delta Sync, específicamente en las columnas `gsi_ds_pk` y `gsi_ds_sk`. Para obtener más información, consulte el artículo sobre detección de conflictos y sincronización en la Guía para desarrolladores de AWS AppSync. Este campo es opcional.

El elemento que se elimina de DynamoDB se convierte automáticamente a los tipos primitivos de GraphQL y JSON, y está disponible en el contexto de mapeo (`$context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información acerca de las plantillas de mapeo de respuesta, consulte Información general sobre las plantillas de mapeo de solucionador.

Ejemplo 1

El siguiente ejemplo muestra una plantilla de mapeo de una mutación de GraphQL `deleteItem(id: ID!)`. Si ya existe un elemento con este ID, se eliminará.

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

Ejemplo 2

El siguiente ejemplo muestra una plantilla de mapeo de una mutación de GraphQL `deleteItem(id: ID!, expectedVersion: Int!)`. Si ya existe un elemento con este ID, se eliminará, pero solo si su campo `version` está establecido en `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)
        }
    }
}
```

Para obtener más información sobre la API `DeleteItem` de DynamoDB, consulte la documentación de la API de DynamoDB.

Consulta

El documento de mapeo de solicitudes de `Query` le permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `Query` a DynamoDB, así como especificar lo siguiente:

- Expresión de clave.
- Qué índice utilizar.
- Todos los filtros adicionales.
- Cuántos elementos deben devolverse.
- Si se utilizarán lecturas coherentes.
- Dirección de consulta (hacia delante o hacia atrás).
- Token de paginación

El documento de mapeo de `Query` tiene la siguiente estructura:

```
{
    "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" : {
        ...
    }
```

```
}
```

Los campos se definen de la siguiente manera:

Campos de consulta

Lista de campos de consulta

`version`

La versión de la definición de plantilla `2017-02-28` y `2018-05-29` se admiten actualmente. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `Query`, este valor se debe establecer en `Query`. Este valor es obligatorio.

`query`

La sección `query` permite especificar una expresión de condición de clave que describe qué elementos deben recuperarse de DynamoDB. Para obtener más información sobre cómo escribir expresiones de condición de clave, consulte DynamoDB KeyConditions documentation. Esta sección debe especificarse.

`expression`

La expresión de la consulta. Este campo debe especificarse.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre usado en la `expression` y el valor tiene que ser una cadena que corresponda al nombre de atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con las sustituciones de marcadores de posición de nombre de atributo de expresión que se usen en la `expression`.

`expressionValues`

Las sustituciones de los marcadores de posición de valor de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de valor usado en la `expression` y el valor tiene que ser un valor con tipo. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor

es obligatorio. Este campo es opcional y solo debe rellenarse con las sustituciones de los marcadores de posición de valor de atributo de expresión que se usen en la `expression`.

`filter`

Un filtro adicional que se puede utilizar para filtrar los resultados de DynamoDB antes de que se devuelvan. Para obtener más información acerca de los filtros, consulte Filtros. Este campo es opcional.

`index`

El nombre del índice que se consultará. La operación de consulta de DynamoDB permite escanear en índices secundarios locales y globales, además de hacerlo en el índice de clave principal de una clave hash. Si se especifica, indica a DynamoDB que debe consultar el índice especificado. Si se omite, se consultará el índice de clave principal.

`nextToken`

El token de paginación para continuar una consulta anterior. Se debe obtener de una consulta anterior. Este campo es opcional.

`limit`

Número máximo de elementos que se van a evaluar, que no es necesariamente el número de elementos coincidentes. Este campo es opcional.

`scanIndexForward`

Valor booleano que indica si se debe consultar hacia delante o hacia atrás. Este campo es opcional y de forma predeterminada es `true`.

`consistentRead`

Valor booleano que indica si se utilizarán lecturas coherentes al consultar DynamoDB. Este campo es opcional y de forma predeterminada es `false`.

`select`

De forma predeterminada, el solucionador de DynamoDB de AWS AppSync solo devuelve los atributos que se proyectan en el índice. Si se necesitan más atributos, puede configurar este campo. Este campo es opcional. Los valores admitidos son:

`ALL_ATTRIBUTES`

Devuelve todos los atributos de elementos de la tabla o el índice especificados. Si consulta un índice secundario local, DynamoDB recupera todo el elemento de la tabla principal para cada elemento coincidente en el índice. Si el índice está configurado para proyectar todos los

atributos de los elementos, todos los datos se pueden obtener del índice secundario local y no es necesario efectuar una recuperación.

`ALL_PROJECTED_ATTRIBUTES`

Permitido solo al consultar un índice. Recupera todos los atributos que se han proyectado en el índice. Si el índice está configurado para proyectar todos los atributos, este valor de retorno equivale a especificar `ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Devuelve solo los atributos que aparecen en la `expression` de la `projection`. Este valor devuelto equivale a especificar la `expression` de la `projection` sin especificar ningún valor para `Select`.

`projection`

Proyección que se utiliza para especificar los atributos que se devolverán de la operación de DynamoDB. Para obtener más información acerca de las proyecciones, consulte la sección Proyecciones. Este campo es opcional.

Los resultados de DynamoDB se convierten automáticamente a los tipos primitivos de GraphQL y JSON, y están disponibles en el contexto de mapeo (`$context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información acerca de las plantillas de mapeo de respuesta, consulte Información general sobre las plantillas de mapeo de solucionador.

Los resultados tienen la estructura siguiente:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Los campos se definen de la siguiente manera:

`items`

Una lista que contiene los elementos que devuelve la consulta de DynamoDB.

`nextToken`

Si hubiera más resultados, `nextToken` contiene un token de paginación que puede usar en otra solicitud. Observe que AWS AppSync cifra y oculta el token de paginación devuelto de DynamoDB. Esto evita que los datos de las tablas se filtren accidentalmente al intermediario. Además, tenga en cuenta que estos tokens de paginación no se pueden utilizar en diferentes solucionadores.

`scannedCount`

El número de elementos que coincidían con la expresión de condición de consulta antes de aplicar una expresión de filtro (si la hay).

Ejemplo

El siguiente ejemplo muestra una plantilla de mapeo de una consulta de GraphQL `getPosts(owner: ID!)`.

En este ejemplo, se consulta un índice secundario global en una tabla para que devuelva todas las publicaciones que pertenecen al ID especificado.

```
{
    "version" : "2017-02-28",
    "operation" : "Query",
    "query" : {
        "expression" : "ownerId = :ownerId",
        "expressionValues" : {
            ":ownerId" : $util.dynamodb.toDynamoDBJson($context.arguments.owner)
        }
    },
    "index" : "owner-index"
}
```

Para obtener más información sobre la API `Query` de DynamoDB, consulte la documentación de la API de DynamoDB.

Examen

El documento de mapeo de solicitudes de `Scan` le permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `Scan` a DynamoDB, así como especificar lo siguiente:

- Un filtro para excluir resultados
- Qué índice utilizar.
- Cuántos elementos deben devolverse.
- Si se utilizarán lecturas coherentes.
- Token de paginación
- Exámenes en paralelo

El documento de mapeo de `Scan` tiene la siguiente estructura:

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "index" : "fooIndex",
    "limit" : 10,
    "consistentRead" : false,
    "nextToken" : "aPaginationToken",
    "totalSegments" : 10,
    "segment" : 1,
    "filter" : {
        ...
    },
    "projection" : {
        ...
    }
}
```

Los campos se definen de la siguiente manera:

Campos de Scan

Lista de campos de Scan

`version`

La versión de la definición de plantilla `2017-02-28` y `2018-05-29` se admiten actualmente. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `Scan`, este valor se debe establecer en `Scan`. Este valor es obligatorio.

`filter`

Un filtro que se puede utilizar para filtrar los resultados de DynamoDB antes de que se devuelvan. Para obtener más información acerca de los filtros, consulte Filtros. Este campo es opcional.

`index`

El nombre del índice que se consultará. La operación de consulta de DynamoDB permite escanear en índices secundarios locales y globales, además de hacerlo en el índice de clave principal de una clave hash. Si se especifica, indica a DynamoDB que debe consultar el índice especificado. Si se omite, se consultará el índice de clave principal.

`limit`

El número máximo de elementos que se evalúan en una sola vez. Este campo es opcional.

`consistentRead`

Valor booleano que indica si se utilizarán lecturas coherentes al consultar DynamoDB. Este campo es opcional y de forma predeterminada es `false`.

`nextToken`

El token de paginación para continuar una consulta anterior. Se debe obtener de una consulta anterior. Este campo es opcional.

`select`

De forma predeterminada, el solucionador de DynamoDB de AWS AppSync solo devuelve los atributos que se proyecten en el índice. Si se necesitan más atributos, este campo se puede configurar. Este campo es opcional. Los valores admitidos son:

`ALL_ATTRIBUTES`

Devuelve todos los atributos de elementos de la tabla o el índice especificados. Si consulta un índice secundario local, DynamoDB recupera todo el elemento de la tabla principal para cada elemento coincidente en el índice. Si el índice está configurado para proyectar todos los atributos de los elementos, todos los datos se pueden obtener del índice secundario local y no es necesario efectuar una recuperación.

`ALL_PROJECTED_ATTRIBUTES`

Permitido solo al consultar un índice. Recupera todos los atributos que se han proyectado en el índice. Si el índice está configurado para proyectar todos los atributos, este valor de retorno equivale a especificar `ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Devuelve solo los atributos que aparecen en la `expression` de la `projection`. Este valor devuelto equivale a especificar la `expression` de la `projection` sin especificar ningún valor para `Select`.

`totalSegments`

El número de segmentos para dividir en particiones la tabla al realizar un examen paralelo. Este campo es opcional, pero debe especificarse si se indica `segment`.

`segment`

El segmento de tabla de esta operación al realizar un examen en paralelo. Este campo es opcional, pero debe especificarse si se indica `totalSegments`.

`projection`

Proyección que se utiliza para especificar los atributos que se devolverán de la operación de DynamoDB. Para obtener más información acerca de las proyecciones, consulte la sección Proyecciones. Este campo es opcional.

Los resultados que devuelve el análisis de DynamoDB se convierten automáticamente a los tipos primitivos de GraphQL y JSON, y están disponibles en el contexto de mapeo (`$context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información acerca de las plantillas de mapeo de respuesta, consulte Información general sobre las plantillas de mapeo de solucionador.

Los resultados tienen la estructura siguiente:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Los campos se definen de la siguiente manera:

`items`

Una lista que contiene los elementos que devuelve el análisis de DynamoDB.

`nextToken`

Si hubiera más resultados, `nextToken` contiene un token de paginación que puede usar en otra solicitud. AWS AppSync cifra y oculta el token de paginación devuelto de DynamoDB. Esto evita que los datos de las tablas se filtren accidentalmente al intermediario. Además, estos tokens de paginación no se pueden utilizar en diferentes solucionadores.

`scannedCount`

El número de elementos que DynamoDB ha recuperado antes de aplicar una expresión de filtro (en caso de incluirse).

Ejemplo 1

El siguiente ejemplo muestra una plantilla de mapeo de una consulta de GraphQL: `allPosts`.

En este ejemplo, se devuelven todas las entradas de la tabla.

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

Ejemplo 2

El siguiente ejemplo muestra una plantilla de mapeo de una consulta de GraphQL: `postsMatching(title: String!)`.

En este ejemplo, todas las entradas de la tabla se devuelven donde el título comienza con el argumento `title`.

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "filter" : {
        "expression" : "begins_with(title, :title)",
```

```
        "expressionValues" : {
            ":title" : $util.dynamodb.toDynamoDBJson($context.arguments.title)
        },
    }
}
```

Para obtener más información sobre la API `Scan` de DynamoDB, consulte la documentación de la API de DynamoDB.

Sync (Sincronizar)

El documento de mapeo de solicitudes de `Sync` permite recuperar todos los resultados de una tabla de DynamoDB y, a continuación, recibir tan solo los datos alterados desde la última consulta (las actualizaciones delta). Únicamente se pueden realizar solicitudes `Sync` a orígenes de datos con control de versiones de DynamoDB. Puede especificar lo siguiente:

- Un filtro para excluir resultados
- Cuántos elementos deben devolverse.
- Token de paginación
- Cuando se inició la última operación `Sync`

El documento de mapeo de `Sync` tiene la siguiente estructura:

```
{
    "version" : "2018-05-29",
    "operation" : "Sync",
    "basePartitionKey": "Base Tables PartitionKey",
    "deltaIndexName": "delta-index-name",
    "limit" : 10,
    "nextToken" : "aPaginationToken",
    "lastSync" :  1550000000000,
    "filter" : {
        ...
    }
}
```

Los campos se definen de la siguiente manera:

Campos de Sync

Lista de campos de Sync

`version`

La versión de la definición de plantilla. Solo se admite `2018-05-29` actualmente. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación `Sync` de , en este valor debe establecerse `Sync`. Este valor es obligatorio.

`filter`

Un filtro que se puede utilizar para filtrar los resultados de DynamoDB antes de que se devuelvan. Para obtener más información acerca de los filtros, consulte Filtros. Este campo es opcional.

`limit`

El número máximo de elementos que se evalúan en una sola vez. Este campo es opcional. Si se omite, el límite predeterminado se establecerá en `100` elementos. El valor máximo de este campo son `1000` elementos.

`nextToken`

El token de paginación para continuar una consulta anterior. Se debe obtener de una consulta anterior. Este campo es opcional.

`lastSync`

El momento, en milisegundos transcurridos desde la fecha de inicio, en el que comenzó la última operación `Sync` que se ha realizado correctamente. Si se especifica, solo se devuelven los elementos que han cambiado después de `lastSync`. Este campo es opcional y solo debe rellenarse después de haber recuperado todas las páginas de una operación `Sync` inicial. Si se omite, se devolverán los resultados de la tabla Base; de lo contrario, se devolverán los resultados de la tabla Delta.

`basePartitionKey`

Clave de partición de la tabla Base utilizada al realizar una operación `Sync`. Este campo permite realizar una operación `Sync` cuando la tabla utiliza una clave de partición personalizada. Se trata de un campo opcional.

`deltaIndexName`

Índice utilizado para la operación `Sync`. Este índice es necesario para habilitar una operación `Sync` en toda la tabla de almacenamiento Delta cuando la tabla utiliza una clave de partición personalizada. La operación `Sync` se realizará en el GSI (creado en `gsi_ds_pk` y `gsi_ds_sk`). Este campo es opcional.

Los resultados que devuelve la sincronización de DynamoDB se convierten automáticamente a los tipos primitivos de GraphQL y JSON, y están disponibles en el contexto de mapeo (`$context.result`).

Para obtener más información sobre la conversión de tipos de DynamoDB, consulte la sección Sistema de tipos (mapeo de respuestas).

Para obtener más información acerca de las plantillas de mapeo de respuesta, consulte Información general sobre las plantillas de mapeo de solucionador.

Los resultados tienen la estructura siguiente:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10,
    startedAt = 1550000000000
}
```

Los campos se definen de la siguiente manera:

`items`

Una lista que contiene los elementos que devuelve la sincronización.

`nextToken`

Si hubiera más resultados, `nextToken` contiene un token de paginación que puede usar en otra solicitud. AWS AppSync cifra y oculta el token de paginación devuelto de DynamoDB. Esto evita que los datos de las tablas se filtren accidentalmente al intermediario. Además, estos tokens de paginación no se pueden utilizar en diferentes solucionadores.

`scannedCount`

El número de elementos que DynamoDB ha recuperado antes de aplicar una expresión de filtro (en caso de incluirse).

`startedAt`

El momento, en milisegundos transcurridos desde la fecha de inicio, en el que comenzó la operación de sincronización que puede almacenar localmente y usar en otra solicitud como argumento de `lastSync`. Si se incluyó un token de paginación en la solicitud, este valor será el mismo que el devuelto por la solicitud para la primera página de resultados.

Ejemplo 1

El siguiente ejemplo muestra una plantilla de mapeo de una consulta de GraphQL: `syncPosts(nextToken: String, lastSync: AWSTimestamp)`.

En este ejemplo, si se omite `lastSync`, se devuelven todas las entradas de la tabla base. Si se proporciona `lastSync`, solo se devuelven las entradas de la tabla Delta Sync que han cambiado desde `lastSync`.

```
{
    "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

El documento de mapeo de solicitudes de `BatchGetItem` permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `BatchGetItem` a DynamoDB para recuperar varios elementos, posiblemente de varias tablas. Para esta plantilla de solicitud, debe especificar lo siguiente:

- Los nombres de tabla de los que recuperar los elementos
- Las claves de los elementos que recuperar de cada tabla

Se aplican los límites de `BatchGetItem` de DynamoDB y no puede proporcionar ninguna expresión de condición.

El documento de mapeo de `BatchGetItem` tiene la siguiente estructura:

```
{
    "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" : {
                ...
            }
```

```
        }
    }
}
```

Los campos se definen de la siguiente manera:

Campos BatchGetItem

Lista de campos BatchGetItem

`version`

La versión de la definición de plantilla. Solo se admite `2018-05-29`. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `BatchGetItem`, este valor se debe establecer en `BatchGetItem`. Este valor es obligatorio.

`tables`

Las tablas de DynamoDB de las que recuperar los elementos. El valor es un mapa en el que se especifican los nombres de las tablas como claves del mapa. Al menos debe proporcionarse una tabla. Este valor `tables` es obligatorio.

`keys`

Lista de claves de DynamoDB que representan la clave principal de los elementos que se van a recuperar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud).

`consistentRead`

Si se utiliza una lectura consistente a la hora de ejecutar una operación GetItem. Este valor es opcional y de forma predeterminada es falso.

`projection`

Proyección que se utiliza para especificar los atributos que se devolverán de la operación de DynamoDB. Para obtener más información acerca de las proyecciones, consulte la sección Proyecciones. Este campo es opcional.

Cosas que tener en cuenta:

- Si un elemento no se ha recuperado de la tabla, aparece un elemento nulo en el bloque de datos para esa tabla.
- Los resultados de invocación se ordenan por tabla, según el orden en el que se hayan proporcionado dentro de la plantilla de mapeo de solicitud.
- Cada comando `Get` dentro de un `BatchGetItem` es atómico; sin embargo, un lote se puede procesar parcialmente. Si un lote se procesa parcialmente debido a un error, la claves sin procesar se devuelven como parte del resultado de invocación dentro del bloque unprocessedKeys.
- `BatchGetItem` está limitado a 100 claves.

Para la siguiente plantilla de mapeo de solicitud de ejemplo:

```
{
  "version": "2018-05-29",
  "operation": "BatchGetItem",
  "tables": {
    "authors": [
        {
          "author_id": {
            "S": "a1"
          }
        },
    ],
    "posts": [
        {
          "author_id": {
            "S": "a1"
          },
          "post_id": {
            "S": "p2"
          }
        }
    ],
  }
}
```

El resultado de invocación disponible en `$ctx.result` es el siguiente:

```
{
```

```
   "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": []
   }
}
```

El `$ctx.error` contiene detalles acerca del error. Está garantizado que los datos de claves, unprocessedKeys y cada clave de tabla que se le proporcionó en la plantilla de mapeo de solicitud estarán presentes en el resultado de invocación. Los elementos que se han eliminado aparecen en el bloque de datos. Los elementos que no se hayan procesado se marcan como null dentro del bloque de datos y se colocan dentro del bloque unprocessedKeys.

Para obtener un ejemplo más completo, siga el tutorial de lotes de DynamoDB con AppSync aquí Tutorial: Solucionadores por lotes de DynamoDB.

BatchDeleteItem

El documento de mapeo de solicitudes de `BatchDeleteItem` permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `BatchWriteItem` a DynamoDB para eliminar varios elementos, posiblemente en varias tablas. Para esta plantilla de solicitud, debe especificar lo siguiente:

- Los nombres de tabla de los que eliminar los elementos
- Las claves de los elementos que eliminar de cada tabla

Se aplican los límites de `BatchWriteItem` de DynamoDB y no puede proporcionar ninguna expresión de condición.

El documento de mapeo de `BatchDeleteItem` tiene la siguiente estructura:

```
{
    "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
        }],
    }
}
```

Los campos se definen de la siguiente manera:

Campos BatchDeleteItem

Lista de campos BatchDeleteItem

`version`

La versión de la definición de plantilla. Solo se admite `2018-05-29`. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `BatchDeleteItem`, este valor se debe establecer en `BatchDeleteItem`. Este valor es obligatorio.

`tables`

Las tablas de DynamoDB de las que eliminar los elementos. Cada tabla es una lista de claves de DynamoDB que representan la clave principal de los elementos que se van a eliminar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un "valor con tipo", consulte Sistema de tipos (mapeo de solicitud). Al menos debe proporcionarse una tabla. El valor `tables` es obligatorio.

Cosas que tener en cuenta:

- Al contrario que en la operación `DeleteItem`, el elemento completamente eliminado no se devuelve en la respuesta. Solo se devuelve la clave pasada.
- Si un elemento no se ha eliminado de la tabla, aparece un elemento nulo en el bloque de datos para esa tabla.
- Los resultados de invocación se ordenan por tabla, según el orden en el que se hayan proporcionado dentro de la plantilla de mapeo de solicitud.
- Cada comando `Delete` dentro de un `BatchDeleteItem` es atómico. Sin embargo, un lote puede procesarse parcialmente. Si un lote se procesa parcialmente debido a un error, la claves sin procesar se devuelven como parte del resultado de invocación dentro del bloque unprocessedKeys.
- `BatchDeleteItem` está limitado a 25 claves.

Para la siguiente plantilla de mapeo de solicitud de ejemplo:

```
{
  "version": "2018-05-29",
  "operation": "BatchDeleteItem",
  "tables": {
    "authors": [
        {
          "author_id": {
```

```
            "S": "a1"
          }
        },
    ],
    "posts": [
        {
          "author_id": {
            "S": "a1"
          },
          "post_id": {
            "S": "p2"
          }
        }
    ],
  }
}
```

El resultado de invocación disponible en `$ctx.result` es el siguiente:

```
{
   "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": []
   }
}
```

El `$ctx.error` contiene detalles acerca del error. Está garantizado que los datos de claves, unprocessedKeys y cada clave de tabla que se le proporcionó en la plantilla de mapeo de solicitud

estarán presentes en el resultado de invocación. Los elementos que se han eliminado están presentes en el bloque de datos. Los elementos que no se hayan procesado se marcan como null dentro del bloque de datos y se colocan dentro del bloque unprocessedKeys.

Para obtener un ejemplo más completo, siga el tutorial de lotes de DynamoDB con AppSync aquí Tutorial: Solucionadores por lotes de DynamoDB.

BatchPutItem

El documento de mapeo de solicitudes de `BatchPutItem` permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `BatchWriteItem` a DynamoDB para colocar varios elementos, posiblemente en varias tablas. Para esta plantilla de solicitud, debe especificar lo siguiente:

- Los nombres de tabla en los que poner los elementos
- Los elementos completos que poner en cada tabla

Se aplican los límites de `BatchWriteItem` de DynamoDB y no puede proporcionar ninguna expresión de condición.

El documento de mapeo de `BatchPutItem` tiene la siguiente estructura:

```
{
    "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
        }],
    }
}
```

Los campos se definen de la siguiente manera:

Campos BatchPutItem

Lista de campos BatchPutItem

`version`

La versión de la definición de plantilla. Solo se admite `2018-05-29`. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `BatchPutItem`, este valor se debe establecer en `BatchPutItem`. Este valor es obligatorio.

`tables`

Las tablas de DynamoDB en las que poner los elementos. Cada entrada de la tabla representa una lista de elementos de DynamoDB que insertar para esta tabla específica. Al menos debe proporcionarse una tabla. Este valor es obligatorio.

Cosas que tener en cuenta:

- Los elementos totalmente insertados se devuelven en la respuesta, si la operación se realiza correctamente.
- Si un elemento no se ha insertado en la tabla, aparece un elemento nulo en el bloque de datos para esa tabla.
- Los elementos insertados se ordenan por tabla, según el orden en el que se hayan proporcionado dentro de la plantilla de mapeo de solicitudes.

- Cada comando `Put` dentro de un `BatchPutItem` es atómico; sin embargo, un lote se puede procesar parcialmente. Si un lote se procesa parcialmente debido a un error, la claves sin procesar se devuelven como parte del resultado de invocación dentro del bloque unprocessedKeys.
- `BatchPutItem` está limitado a 25 elementos.

Para la siguiente plantilla de mapeo de solicitud de ejemplo:

```
{
  "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"
          }
        }
    ],
  }
}
```

El resultado de invocación disponible en `$ctx.result` es el siguiente:

```
{
   "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": []
  }
}
```

El `$ctx.error` contiene detalles acerca del error. Está garantizado que los datos de claves, unprocessedItems y cada clave de tabla que se le proporcionó en la plantilla de mapeo de solicitud estarán presentes en el resultado de invocación. Los elementos que se han insertado están en el bloque de datos. Los elementos que no se hayan procesado se marcan como null dentro del bloque de datos y se colocan dentro del bloque unprocessedItems.

Para obtener un ejemplo más completo, siga el tutorial de lotes de DynamoDB con AppSync aquí Tutorial: Solucionadores por lotes de DynamoDB.

TransactGetItems

El documento de mapeo de solicitudes de `TransactGetItems` permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `TransactGetItems` a DynamoDB para recuperar varios elementos, que pueden estar en varias tablas. Para esta plantilla de solicitud, debe especificar lo siguiente:

- El nombre de la tabla de cada elemento de solicitud de la que se va a recuperar el elemento

- La clave de cada elemento de solicitud que se va a recuperar de cada tabla

Se aplican los límites de `TransactGetItems` de DynamoDB y no puede proporcionar ninguna expresión de condición.

El documento de mapeo de `TransactGetItems` tiene la siguiente estructura:

```
{
    "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" : {
                ...
           }
       }
    ]
}
```

Los campos se definen de la siguiente manera:

Campos TransactGetItems

Lista de campos TransactGetItems

`version`

La versión de la definición de plantilla. Solo se admite `2018-05-29`. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `TransactGetItems`, este valor se debe establecer en `TransactGetItems`. Este valor es obligatorio.

`transactItems`

Los elementos de solicitud que se van a incluir. El valor es una matriz de elementos de solicitud. Se debe proporcionar al menos un elemento de solicitud. Este valor `transactItems` es obligatorio.

`table`

La tabla de DynamoDB de la que se va a recuperar el elemento. El valor es una cadena con el nombre de la tabla. Este valor `table` es obligatorio.

`key`

La clave de DynamoDB que representa la clave principal del elemento que se desea recuperar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud).

`projection`

Proyección que se utiliza para especificar los atributos que se devolverán de la operación de DynamoDB. Para obtener más información acerca de las proyecciones, consulte la sección Proyecciones. Este campo es opcional.

Cosas que tener en cuenta:

- Si una transacción se realiza correctamente, el orden de los elementos recuperados en el bloque `items` será el mismo que el orden de los elementos de solicitud.

- Las transacciones se realizan en régimen de todo o nada. Si algún elemento de solicitud causa un error, no se realizará la transacción completa y se devolverán los detalles del error.
- El hecho de no poder recuperar un elemento de solicitud no es un error. En su lugar, aparece un elemento null en el bloque items (elementos) en la posición correspondiente.
- Si el error de una transacción es TransactionCanceledException, se rellenará el bloque `cancellationReasons`. El orden de los motivos de cancelación en el bloque `cancellationReasons` será el mismo que el orden de los elementos de solicitud.
- `TransactGetItems` está limitado a 25 elementos de solicitud.

Para la siguiente plantilla de mapeo de solicitud de ejemplo:

```
{
    "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
               }
           }
       }
    ]
}
```

Si la transacción se realiza correctamente y solo se recupera el primer elemento solicitado, el resultado de la invocación disponible en `$ctx.result` es el siguiente:

```
{
```

```
    "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
}
```

Si la transacción no se realiza correctamente debido a la excepción TransactionCanceledException causada por el primer elemento de solicitud, el resultado de la invocación disponible en `$ctx.result` es el siguiente:

```
{
    "items": null,
    "cancellationReasons": [
       {
           "type":"Sample error type",
           "message":"Sample error message"
       },
       {
           "type":"None",
           "message":"None"
       }
    ]
}
```

El `$ctx.error` contiene detalles acerca del error. Los valores de items de claves y cancellationReasons estarán presentes sin duda en `$ctx.result`.

Para obtener un ejemplo más completo, siga el tutorial de transacciones de DynamoDB con AppSync aquí Tutorial: Solucionadores de transacciones de DynamoDB.

TransactWriteItems

El documento de mapeo de solicitudes de `TransactWriteItems` permite indicar al solucionador de DynamoDB de AWS AppSync que realice una solicitud `TransactWriteItems` a DynamoDB

para escribir varios elementos, posiblemente en varias tablas. Para esta plantilla de solicitud, debe especificar lo siguiente:

- El nombre de la tabla de destino de cada elemento de solicitud
- La operación de cada elemento de solicitud que se va a realizar. Hay cuatro tipos de operaciones compatibles: PutItem, UpdateItem, DeleteItem y ConditionCheck.
- La clave de cada elemento de solicitud que se va a escribir

Se aplican los límites de `TransactWriteItems` de DynamoDB.

El documento de mapeo de `TransactWriteItems` tiene la siguiente estructura:

```
{
    "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
            }
        }
    ]
}
```

Campos TransacTwriteItems

Lista de campos TransactWriteItems

Los campos se definen de la siguiente manera:

`version`

La versión de la definición de plantilla. Solo se admite `2018-05-29`. Este valor es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `TransactWriteItems`, este valor se debe establecer en `TransactWriteItems`. Este valor es obligatorio.

`transactItems`

Los elementos de solicitud que se van a incluir. El valor es una matriz de elementos de solicitud. Se debe proporcionar al menos un elemento de solicitud. Este valor `transactItems` es obligatorio.

Para `PutItem`, los campos se definen de la siguiente manera:

`table`

La tabla de DynamoDB de destino. El valor es una cadena con el nombre de la tabla. Este valor `table` es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `PutItem`, este valor se debe establecer en `PutItem`. Este valor es obligatorio.

`key`

La clave de DynamoDB que representa la clave principal del elemento que se desea colocar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`attributeValues`

El resto de los atributos del elemento que debe colocarse en DynamoDB. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este campo es opcional.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `PutItem` sobrescribe todas las entradas existentes para dicho elemento. Puede especificar si desea recuperar el elemento existente cuando se produzca un error en la comprobación de condiciones. Para obtener más información acerca de las condiciones transaccionales, consulte Expresiones de condición de transacción. Este valor es opcional.

Para `UpdateItem`, los campos se definen de la siguiente manera:

`table`

La tabla de DynamoDB que se va a actualizar. El valor es una cadena con el nombre de la tabla. Este valor `table` es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `UpdateItem`, este valor se debe establecer en `UpdateItem`. Este valor es obligatorio.

`key`

La clave de DynamoDB que representa la clave principal del elemento que se va a actualizar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`update`

La sección `update` permite especificar una expresión de actualización que describe cómo se actualiza el elemento en DynamoDB. Para obtener más información sobre el modo de escribir expresiones de actualización, consulte la documentación de DynamoDB UpdateExpressions. Esta sección es obligatoria.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `UpdateItem` actualiza todas las entradas existentes independientemente de su estado actual. Puede especificar si desea recuperar el elemento existente cuando se produzca un error en la comprobación de condiciones. Para obtener más información acerca de las condiciones transaccionales, consulte Expresiones de condición de transacción. Este valor es opcional.

Para `DeleteItem`, los campos se definen de la siguiente manera:

`table`

La tabla de DynamoDB en la que se elimina el elemento. El valor es una cadena con el nombre de la tabla. Este valor `table` es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `DeleteItem`, este valor se debe establecer en `DeleteItem`. Este valor es obligatorio.

`key`

La clave de DynamoDB que representa la clave principal del elemento que se desea eliminar. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más

información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Si no se especifica ninguna condición, la solicitud `DeleteItem` elimina un elemento independientemente de su estado actual. Puede especificar si desea recuperar el elemento existente cuando se produzca un error en la comprobación de condiciones. Para obtener más información acerca de las condiciones transaccionales, consulte Expresiones de condición de transacción. Este valor es opcional.

Para `ConditionCheck`, los campos se definen de la siguiente manera:

`table`

La tabla de DynamoDB en la que se comprueba la condición. El valor es una cadena con el nombre de la tabla. Este valor `table` es obligatorio.

`operation`

La operación de DynamoDB que se ha de realizar. Para ejecutar la operación de DynamoDB `ConditionCheck`, este valor se debe establecer en `ConditionCheck`. Este valor es obligatorio.

`key`

La clave de DynamoDB que representa la clave principal del elemento que hay que someter a una comprobación de condición. Los elementos de DynamoDB pueden tener solo una clave hash o una clave hash y una clave de clasificación, dependiendo de la estructura de la tabla. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor es obligatorio.

`condition`

Una condición para determinar si la solicitud debe realizarse correctamente o no, en función del estado del objeto ya incluido en DynamoDB. Puede especificar si desea recuperar el elemento existente cuando se produzca un error en la comprobación de condiciones. Para obtener más información acerca de las condiciones transaccionales, consulte Expresiones de condición de transacción. Este valor es obligatorio.

Cosas que tener en cuenta:

- Solo las claves de los elementos de solicitud se devuelven en la respuesta, si se realiza correctamente. El orden de las claves será el mismo que el orden de los elementos de solicitud.
- Las transacciones se realizan en régimen de todo o nada. Si algún elemento de solicitud causa un error, no se realizará la transacción completa y se devolverán los detalles del error.
- No se pueden dirigir dos elementos de solicitud al mismo elemento. De lo contrario, causarán un error de TransactionCanceledException.
- Si el error de una transacción es TransactionCanceledException, se rellenará el bloque `cancellationReasons`. Si se produce un error en la comprobación de condición de un elemento de solicitud y no se ha especificado que `returnValuesOnConditionCheckFailure` sea `false`, el elemento existente en la tabla se recuperará y almacenará en `item` en la posición correspondiente del bloque `cancellationReasons`.
- `TransactWriteItems` está limitado a 25 elementos de solicitud.

Para la siguiente plantilla de mapeo de solicitud de ejemplo:

```
{
    "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"
                    }
                }
            },
        }
    ]
}
```

Si la transacción se realiza correctamente, el resultado de la invocación disponible en `$ctx.result` es el siguiente:

```
{
    "keys": [
       // Key of the PutItem request
       {
           "post_id": "p1",
       },
       // Key of the UpdateItem request
       {
           "author_id": "a1"
       }
    ],
    "cancellationReasons": null
}
```

Si la transacción no se realiza correctamente debido a un error de comprobación de condición de la solicitud `PutItem`, el resultado de la invocación disponible en `$ctx.result` es el siguiente:

```
{
    "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"
       }
    ]
}
```

El `$ctx.error` contiene detalles acerca del error. Los valores de keys y cancellationReasons estarán presentes sin duda en `$ctx.result`.

Para obtener un ejemplo más completo, siga el tutorial de transacciones de DynamoDB con AppSync aquí Tutorial: Solucionadores de transacciones de DynamoDB.

Sistema de tipos (mapeo de solicitud)

Cuando se utiliza el solucionador de DynamoDB de AWS AppSync para llamar a sus tablas de DynamoDB, AWS AppSync debe conocer el tipo de cada valor que se va a utilizar en dicha llamada. Esto se debe a que DynamoDB admite más tipos primitivos que los disponibles en GraphQL o JSON (por ejemplo, conjuntos y datos binarios). AWS AppSync necesita indicaciones para hacer conversiones entre GraphQL y DynamoDB; de lo contrario, tendría que suponer cómo se estructuran los datos de la tabla.

Para obtener más información sobre los tipos de datos de DynamoDB, consulte la documentación relativa a los descriptores de tipos de datos y los tipos de datos de DynamoDB.

Un valor de DynamoDB se representa mediante un objeto JSON que contiene un único par de clave-valor. La clave especifica el tipo de DynamoDB y el valor especifica el valor en sí. En el siguiente ejemplo, la clave `S` indica que el valor es una cadena y el valor `identifier` es el valor de la cadena en sí.

```
{ "S" : "identifier" }
```

Tenga en cuenta que el objeto JSON no puede tener más de un par de clave-valor. Si se especifica más de un par de clave-valor, el documento de mapeo de solicitudes no se analizará.

Siempre que necesite especificar un valor en un documento de mapeo de solicitudes, deberá usar un valor de DynamoDB. Entre los lugares donde necesitará realizar esta operación figuran: las secciones `key` y `attributeValue`, y la sección `expressionValues` de secciones de expresión. En el siguiente ejemplo, el valor de cadena de DynamoDB `identifier` se asigna al campo `id` de una sección `key` (tal vez en un documento de mapeo de solicitudes `GetItem`).

```
"key" : {
   "id" : { "S" : "identifier" }
}
```

Tipos admitidos

AWS AppSync admite los siguientes tipos escalares, de documento y de conjunto de DynamoDB:

Tipo cadena **S**

Un valor de cadena único. Un valor de cadena de DynamoDB se indica de la siguiente manera:

```
{ "S" : "some string" }
```

Ejemplo de uso:

```
"key" : {
   "id" : { "S" : "some string" }
}
```

Tipo de conjunto de cadenas **SS**

Un conjunto de valores de cadena. Un valor de conjunto de cadenas de DynamoDB se indica de la siguiente manera:

```
{ "SS" : [ "first value", "second value", ... ] }
```

Ejemplo de uso:

```
"attributeValues" : {
   "phoneNumbers" : { "SS" : [ "+1 555 123 4567", "+1 555 234 5678" ] }
}
```

Tipo número **N**

Un valor numérico único. Un valor de número de DynamoDB se indica de la siguiente manera:

```
{ "N" : 1234 }
```

Ejemplo de uso:

```
"expressionValues" : {
   ":expectedVersion" : { "N" : 1 }
}
```

Tipo conjunto de números **NS**

Conjunto de valores de número. Un valor de conjunto de números de DynamoDB se indica de la siguiente manera:

```
{ "NS" : [ 1, 2.3, 4 ... ] }
```

Ejemplo de uso:

```
"attributeValues" : {
   "sensorReadings" : { "NS" : [ 67.8, 12.2, 70 ] }
}
```

Tipo binario **B**

Un valor binario. Un valor Binario de DynamoDB se indica de la siguiente manera:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

Observe que el valor es en realidad una cadena que contiene la representación codificada en base64 de los datos binarios. AWS AppSync descodifica esta cadena de nuevo a su valor binario antes de enviarlo a DynamoDB. AWS AppSync utiliza el esquema de descodificación base64

como se define en RFC 2045: cualquier carácter que no esté en el alfabeto base64 no se tiene en cuenta.

Ejemplo de uso:

```
"attributeValues" : {
   "binaryMessage" : { "B" : "SGVsbG8sIFdvcmxkIQo=" }
}
```

Tipo conjunto binario **BS**

Conjunto de valores binarios. Un valor de conjunto binario de DynamoDB se indica de la siguiente manera:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

Observe que el valor es en realidad una cadena que contiene la representación codificada en base64 de los datos binarios. AWS AppSync descodifica esta cadena de nuevo a su valor binario antes de enviarlo a DynamoDB. AWS AppSync utiliza el esquema de descodificación base64 como se define en RFC 2045: cualquier carácter que no esté en el alfabeto base64 no se tiene en cuenta.

Ejemplo de uso:

```
"attributeValues" : {
   "binaryMessages" : { "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ] }
}
```

Tipo booleano **BOOL**

Un valor booleano. Un valor Booleano de DynamoDB se indica de la siguiente manera:

```
{ "BOOL" : true }
```

Observe que solo los valores `true` y `false` son válidos.

Ejemplo de uso:

```
"attributeValues" : {
   "orderComplete" : { "BOOL" : false }
```

```
}
```

Tipo lista **L**

Lista del resto de valores de DynamoDB admitidos. Un valor de lista de DynamoDB se indica de la siguiente manera:

```
{ "L" : [ ... ] }
```

Observe que se trata de un valor compuesto, y que la lista puede contener cero o más de cualquiera de los valores de DynamoDB admitidos (incluidas otras listas). La lista también puede contener una combinación de diferentes tipos.

Ejemplo de uso:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

Tipo de mapa **M**

Representa una colección sin ordenar de pares de clave-valor de otros valores de DynamoDB admitidos. Un valor de mapa de DynamoDB se indica de la siguiente manera:

```
{ "M" : { ... } }
```

Observe que un mapa puede contener cero o más pares clave-valor. La clave tiene que ser una cadena y el valor puede ser cualquier valor de DynamoDB admitido (incluidos otros mapas). El mapa también puede contener una combinación de diferentes tipos.

Ejemplo de uso:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
```

```
}
```

Tipo nulo **`NULL`**

Un valor nulo. Un valor Nulo de DynamoDB se indica de la siguiente manera:

```
{ "NULL" : null }
```

Ejemplo de uso:

```
"attributeValues" : {
   "phoneNumbers" : { "NULL" : null }
}
```

Para obtener más información sobre cada tipo, consulte la documentación de DynamoDB.

Sistema de tipos (mapeo de respuestas)

Cuando recibe una respuesta de DynamoDB, AWS AppSync la convierte automáticamente a los tipos primitivos de GraphQL y JSON. Cada atributo de DynamoDB se descodifica y se devuelve en el contexto de mapeo de respuestas.

Por ejemplo, si DynamoDB devuelve lo siguiente:

```
{
    "id" : { "S" : "1234" },
    "name" : { "S" : "Nadia" },
    "age" : { "N" : 25 }
}
```

El solucionador de DynamoDB de AWS AppSync lo convierte a los tipos GraphQL y JSON:

```
{
    "id" : "1234",
    "name" : "Nadia",
    "age" : 25
}
```

En esta sección se explica cómo AWS AppSync convierte los tipos escalares, de documento y de conjunto de DynamoDB indicados:

Tipo cadena **S**

Un valor de cadena único. Se devuelve un valor de cadena de DynamoDB en forma de cadena.

Por ejemplo, si DynamoDB devuelve el siguiente valor de cadena de DynamoDB:

```
{ "S" : "some string" }
```

AWS AppSync lo convierte en una cadena:

```
"some string"
```

Tipo de conjunto de cadenas **SS**

Un conjunto de valores de cadena. Un valor de conjunto de cadenas de DynamoDB se devuelve como una lista de cadenas.

Por ejemplo, si DynamoDB devuelve el siguiente valor de conjunto de cadenas de DynamoDB:

```
{ "SS" : [ "first value", "second value", ... ] }
```

AWS AppSync lo convierte en una lista de cadenas:

```
[ "+1 555 123 4567", "+1 555 234 5678" ]
```

Tipo número **N**

Un valor numérico único. Un valor de número de DynamoDB se devuelve como número.

Por ejemplo, si DynamoDB devuelve el siguiente valor de número de DynamoDB:

```
{ "N" : 1234 }
```

AWS AppSync lo convierte en un número:

```
1234
```

Tipo conjunto de números **NS**

Conjunto de valores de número. Un valor de conjunto de números de DynamoDB se devuelve como una lista de números.

Por ejemplo, si DynamoDB devuelve el siguiente valor de conjunto de números de DynamoDB:

```
{ "NS" : [ 67.8, 12.2, 70 ] }
```

AWS AppSync lo convierte en una lista de números:

```
[ 67.8, 12.2, 70 ]
```

Tipo binario **B**

Un valor binario. Un valor Binario de DynamoDB se devuelve en forma de cadena que contiene la representación base64 de dicho valor.

Por ejemplo, si DynamoDB devuelve el siguiente valor binario de DynamoDB:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

AWS AppSync lo convierte en una cadena con la representación base64 del valor:

```
"SGVsbG8sIFdvcmxkIQo="
```

Observe que los datos binarios se codifican con el esquema base64 como se especifica en RFC 4648 y en RFC 2045.

Tipo conjunto binario **BS**

Conjunto de valores binarios. Un valor de conjunto binario de DynamoDB se devuelve en forma de una lista de cadenas con la representación base64 de los valores.

Por ejemplo, si DynamoDB devuelve el siguiente valor de conjunto binario de DynamoDB:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

AWS AppSync lo convierte en una lista de cadenas que contienen la representación base64 de los valores:

```
[ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ]
```

Observe que los datos binarios se codifican con el esquema base64 como se especifica en RFC 4648 y en RFC 2045.

Tipo booleano **BOOL**

Un valor booleano. Un valor Booleano de DynamoDB se devuelve en forma de valor booleano.

Por ejemplo, si DynamoDB devuelve el siguiente valor booleano de DynamoDB:

```
{ "BOOL" : true }
```

AWS AppSync lo convierte en un valor booleano:

```
true
```

Tipo lista **L**

Lista del resto de valores de DynamoDB admitidos. Un valor de lista de DynamoDB se devuelve en forma de lista de valores, donde el valor de cada elemento también se convierte.

Por ejemplo, si DynamoDB devuelve el siguiente valor de lista de DynamoDB:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

AWS AppSync lo convierte en una lista de valores convertidos:

```
[ "A string value", 1, [ "Another string value", "Even more string values!" ] ]
```

Tipo de mapa **M**

Colección de claves y valores de cualquier otro valor de DynamoDB admitido. Un valor de mapa de DynamoDB se devuelve en forma de objeto JSON en el que también se convierte cada clave y valor.

Por ejemplo, si DynamoDB devuelve el siguiente valor de mapa de DynamoDB:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
```

```
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

AWS AppSync lo convierte en un objeto JSON:

```
{
   "someString" : "A string value",
   "someNumber" : 1,
   "stringSet"  : [ "Another string value", "Even more string values!" ]
}
```

Tipo nulo **`NULL`**

Un valor nulo.

Por ejemplo, si DynamoDB devuelve el siguiente valor nulo de DynamoDB:

```
{ "NULL" : null }
```

AWS AppSync lo convierte en un valor nulo:

```
null
```

Filtros

Al consultar objetos de DynamoDB mediante las operaciones `Query` y `Scan`, tiene la posibilidad de especificar un `filter` para evaluar los resultados y devolver solo los valores deseados.

La sección de mapeo de filtros de un documento de mapeo `Query` o `Scan` tiene la siguiente estructura:

```
"filter" : {
    "expression" : "filter expression"
    "expressionNames" : {
        "#name" : "name",
    },
    "expressionValues" : {
```

```
        ":value" : ... typed value
    },
}
```

Los campos se definen de la siguiente manera:

`expression`

La expresión de la consulta. Para obtener más información sobre cómo escribir expresiones de filtro, consulte la documentación de DynamoDB QueryFilter y DynamoDB ScanFilter. Este campo debe especificarse.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre utilizado en la `expression`. El valor debe ser una cadena que corresponda al nombre del atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con las sustituciones de marcadores de posición de nombre de atributo de expresión que se usen en la `expression`.

`expressionValues`

Las sustituciones de los marcadores de posición de valor de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de valor usado en la `expression` y el valor tiene que ser un valor con tipo. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor debe especificarse. Este campo es opcional y solo debe rellenarse con las sustituciones de los marcadores de posición de valor de atributo de expresión que se usen en la `expression`.

Ejemplo

En el siguiente ejemplo, se muestra una sección de filtro para una plantilla de mapeo en la que las entradas obtenidas de DynamoDB solo se devuelven si el título comienza con el argumento `title`.

```
"filter" : {
    "expression" : "begins_with(#title, :title)",
    "expressionNames" : {
        "#title" : "title"
    },
    "expressionValues" : {
        ":title" : $util.dynamodb.toDynamoDBJson($context.arguments.title)
```

```
    }
}
```

Expresiones de condición

Al mutar objetos en DynamoDB utilizando las operaciones de DynamoDB `PutItem`, `UpdateItem` y `DeleteItem`, puede especificar opcionalmente una expresión de condición que determine si la solicitud se debe atender o no en función del estado del objeto que ya está en DynamoDB antes de ejecutar la operación.

El solucionador de DynamoDB de AWS AppSync permite especificar una expresión de condición en los documentos de mapeo de solicitudes `PutItem`, `UpdateItem` y `DeleteItem`, así como la estrategia que debe seguirse en caso de que la condición no se cumpla y el objeto no se actualice.

Ejemplo 1

El siguiente documento de mapeo `PutItem` no tiene una expresión de condición. Como resultado, pone un elemento en DynamoDB incluso si ya existe un elemento con la misma clave, lo que permite sobrescribir el elemento existente.

```
{
   "version" : "2017-02-28",
   "operation" : "PutItem",
   "key" : {
      "id" : { "S" : "1" }
   }
}
```

Ejemplo 2

El siguiente documento de mapeo `PutItem` tiene una expresión de condición que permitirá que la operación se complete solo si no existe un elemento con la misma clave en DynamoDB.

```
{
   "version" : "2017-02-28",
   "operation" : "PutItem",
   "key" : {
      "id" : { "S" : "1" }
   },
   "condition" : {
```

```
      "expression" : "attribute_not_exists(id)"
   }
}
```

De forma predeterminada, si la condición no se cumple, el solucionador de DynamoDB de AWS AppSync devolverá un error para la mutación y el valor actual del objeto en DynamoDB de un campo `data` de la sección `error` de la respuesta de GraphQL. Sin embargo, el solucionador de DynamoDB de AWS AppSync ofrece algunas características adicionales para ayudar a los desarrolladores a gestionar algunos casos límite habituales:

- Si el solucionador de DynamoDB de AWS AppSync puede determinar que el valor actual de DynamoDB coincide con el resultado deseado, trata la operación como si se hubiera realizado correctamente de todos modos.
- En lugar de devolver un error, puede configurar el solucionador para que invoque una función de Lambda personalizada que decida cómo debe gestionar el error AWS AppSync DynamoDB.

Estos casos se describen con más detalle en la sección de gestión de un error de comprobación de la condición.

Para obtener más información sobre las expresiones de condiciones de DynamoDB, consulte la documentación de expresiones de condición de DynamoDB.

Especificación de una condición

Todos los documentos de mapeo de solicitudes `PutItem`, `UpdateItem` y `DeleteItem` permiten especificar una sección `condition` opcional. Si se omite, no se comprobará ninguna condición. Si se especifica, la condición debe ser true para que la operación se lleve a cabo correctamente.

Las secciones `condition` tienen la siguiente estructura:

```
"condition" : {
    "expression" : "someExpression"
    "expressionNames" : {
        "#foo" : "foo"
    },
    "expressionValues" : {
        ":bar" : ... typed value
    },
    "equalsIgnore" : [ "version" ],
    "consistentRead" : true,
```

```
    "conditionalCheckFailedHandler" : {
        "strategy" : "Custom",
        "lambdaArn" : "arn:..."
    }
}
```

Los campos siguientes especifican la condición:

`expression`

La misma expresión de actualización. Para obtener más información sobre cómo escribir expresiones de condición, consulte la documentación de DynamoDB ContitionExpressions. Este campo debe especificarse.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre usado en la expresión, y el valor tiene que ser una cadena que corresponda al nombre de atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con las sustituciones de los marcadores de posición de nombre de atributo de expresión que se usen en la expresión.

`expressionValues`

Las sustituciones de los marcadores de posición de valor de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de valor usado en la expresión y el valor tiene que ser un valor con tipo. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor debe especificarse. Este campo es opcional y solo debe rellenarse con las sustituciones de los marcadores de posición de valor de atributo de expresión que se usen en la expresión.

El resto de los campos indican al solucionador de DynamoDB de AWS AppSync cómo gestionar los casos en que no se cumpla la condición:

`equalsIgnore`

Cuando no se cumple la condición para la operación `PutItem`, el solucionador de DynamoDB de AWS AppSync compara el elemento que hay actualmente en DynamoDB con el elemento que ha intentado escribir. Si son iguales, trata la operación como si se hubiera realizado correctamente. Puede utilizar el campo `equalsIgnore` para especificar una lista de atributos que AWS AppSync no debe tener en cuenta al realizar la comparación. Por ejemplo, si la única diferencia ha sido un

atributo `version`, trata la operación como si se hubiera realizado satisfactoriamente. Este campo es opcional.

`consistentRead`

Cuando una condición no se cumple, AWS AppSync obtiene el valor actual del elemento de DynamoDB mediante una lectura altamente coherente. Puede utilizar este campo para indicar al solucionador de DynamoDB de AWS AppSync que use una lectura coherente posterior en su lugar. Este campo es opcional y de forma predeterminada es `true`.

`conditionalCheckFailedHandler`

Esta sección le permite especificar la forma en que el solucionador de DynamoDB de AWS AppSync trata los casos en que no se cumpla la condición después de haber comparado el valor actual en DynamoDB con el resultado esperado. Esta sección es opcional. Si se omite, el valor predeterminado es una estrategia `Reject`.

`strategy`

La estrategia que el solucionador de DynamoDB de AWS AppSync sigue después de comparar el valor actual en DynamoDB con el resultado esperado. Este campo es obligatorio y tiene los siguientes valores posibles:

`Reject`

La mutación fracasa y se obtiene un error y un error para la mutación y el valor actual del objeto en DynamoDB en un campo `data` de la sección `error` de la respuesta de GraphQL.

`Custom`

El solucionador de DynamoDB de AWS AppSync invoca una función de Lambda personalizada para decidir cómo gestionar el incumplimiento de la condición. Cuando `strategy` tiene el valor `Custom`, el campo `lambdaArn` debe contener el ARN de la función Lambda que se va a invocar.

`lambdaArn`

El ARN de la función de Lambda que se invoca, que determina cómo debe gestionar el solucionador de DynamoDB de AWS AppSync el incumplimiento de la condición. Este campo solo tiene que especificarse cuando `strategy` tiene el valor `Custom`. Para obtener más información acerca de cómo utilizar esta característica, consulte la sección Gestión de los casos en que no se cumple la condición.

Gestión de un error de comprobación de la condición

De forma predeterminada, si la condición no se cumple, el solucionador de DynamoDB de AWS AppSync devolverá un error para la mutación y el valor actual del objeto en DynamoDB de un campo `data` de la sección `error` de la respuesta de GraphQL. Sin embargo, el solucionador de DynamoDB de AWS AppSync ofrece algunas características adicionales para ayudar a los desarrolladores a gestionar algunos casos límite habituales:

- Si el solucionador de DynamoDB de AWS AppSync puede determinar que el valor actual de DynamoDB coincide con el resultado deseado, trata la operación como si se hubiera realizado correctamente de todos modos.
- En lugar de devolver un error, puede configurar el solucionador para que invoque una función de Lambda personalizada que decida cómo debe gestionar el error AWS AppSync DynamoDB.

El diagrama de flujo de este proceso es:

Comprobación del resultado deseado

Cuando la condición no se cumple, el solucionador de DynamoDB de AWS AppSync realiza una solicitud de DynamoDB `GetItem` para obtener el valor actual del elemento de DynamoDB. De forma predeterminada, utiliza una lectura muy consistente; sin embargo, esto puede configurarse mediante el campo `consistentRead` en el bloque `condition` y compararlo con el resultado esperado:

- En la operación `PutItem`, el solucionador de DynamoDB de AWS AppSync compara el valor actual con el que intentó escribir, excluyendo de la comparación los atributos especificados en `equalsIgnore`. Si los elementos son los mismos, trata la operación como si se hubiera realizado y devuelve el elemento obtenido de DynamoDB. De lo contrario, sigue la estrategia configurada.

  Por ejemplo, si el documento de mapeo de `PutItem` tenía el siguiente aspecto:

```
{
   "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" ]
    }
}
```

Y el elemento que está actualmente en DynamoDB fuese de la siguiente manera:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

El solucionador de DynamoDB de AWS AppSync compararía el elemento que intentó escribir con el valor actual y vería que la única diferencia es el campo `version`, pero como la configuración pasa el campo `version` por alto, trataría la operación como correcta y devolvería el elemento obtenido de DynamoDB.

- En la operación `DeleteItem`, el solucionador de DynamoDB de AWS AppSync realiza una comprobación para verificar que se ha devuelto un elemento de DynamoDB. Si no se devuelve ningún elemento, trata la operación como si se hubiera realizado correctamente. De lo contrario, sigue la estrategia configurada.
- En la operación `UpdateItem`, el solucionador de DynamoDB de AWS AppSync no tiene suficiente información para determinar si el elemento que está actualmente en DynamoDB coincide con el resultado esperado y, por lo tanto, sigue la estrategia configurada.

Si el estado actual del objeto en DynamoDB es diferente del resultado esperado, el solucionador de DynamoDB de AWS AppSync sigue la estrategia configurada para rechazar la mutación o invocar una función de Lambda para determinar qué hacer a continuación.

Aplicación de la estrategia de rechazo

Si se sigue la estrategia `Reject`, el solucionador de DynamoDB de AWS AppSync devuelve un error para la mutación y el valor actual del objeto en DynamoDB también se devuelve en un campo `data` de la sección `error` de la respuesta de GraphQL. El elemento que se devuelve desde DynamoDB

pasa por la plantilla de mapeo de respuesta para convertirlo a un formato que el cliente espera y también se filtra según el conjunto de selección.

Por ejemplo, si se recibe la solicitud de mutación siguiente:

```
mutation {
    updatePerson(id: 1, name: "Steve", expectedVersion: 1) {
        Name
        theVersion
    }
}
```

Si el elemento devuelto de DynamoDB tiene un aspecto similar al siguiente:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

Y la plantilla de mapeo de respuesta es como se muestra a continuación:

```
{
   "id" : $util.toJson($context.result.id),
   "Name" : $util.toJson($context.result.name),
   "theVersion" : $util.toJson($context.result.version)
}
```

La respuesta GraphQL tiene este aspecto:

```
{
  "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
```

```
      },
      ...
    }
  ]
}
```

Además, si hay algún campo en el objeto devuelto que hayan cumplimentado otros solucionadores y si la mutación se ha efectuado satisfactoriamente, el objeto no se resuelve cuando se devuelve en la sección `error`.

Aplicación de la estrategia personalizada

Si se sigue la estrategia `Custom`, el solucionador de DynamoDB de AWS AppSync invoca una función de Lambda para decidir qué hacer a continuación. La función Lambda selecciona una de las siguientes opciones:

- `reject` la mutación. Esto le indica al solucionador de DynamoDB de AWS AppSync que se comporte como si la estrategia configurada fuera `Reject` y que devuelva un error para la mutación y el valor actual del objeto de DynamoDB, tal como se describe en la sección anterior.
- `discard` la mutación. Esto indica al solucionador de DynamoDB de AWS AppSync que no notifique el incumplimiento de la condición y que devuelva el valor en DynamoDB.
- `retry` la mutación. Esto le indica al solucionador de DynamoDB de AWS AppSync que vuelva a intentar la mutación con otro documento de mapeo de solicitud.

La solicitud de invocación Lambda

El solucionador de DynamoDB de AWS AppSync invoca la función de Lambda especificada en el `lambdaArn`. Se utiliza el mismo `service-role-arn` configurado en el origen de datos. La carga de la invocación tiene la siguiente estructura:

```
{
    "arguments": { ... },
    "requestMapping": {... },
    "currentValue": { ... },
    "resolver": { ... },
    "identity": { ... }
}
```

Los campos se definen de la siguiente manera:

`arguments`

Los argumentos de la mutación de GraphQL. Esto es lo mismo que los argumentos disponibles en el documento de mapeo de solicitudes en `$context.arguments`.

`requestMapping`

El documento de mapeo de solicitudes de esta operación.

`currentValue`

El valor actual del objeto en DynamoDB.

`resolver`

Información sobre el solucionador de AWS AppSync.

`identity`

Información sobre el intermediario. Se trata de la información de identidad disponible en el documento de mapeo de solicitudes en `$context.identity`.

Un ejemplo completo de la carga:

```
{
    "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"
    }
}
```

La respuesta de invocación Lambda

La función de Lambda puede inspeccionar la carga de invocación y aplicar cualquier lógica de negocio para decidir la forma en que el solucionador de DynamoDB de AWS AppSync debe gestionar el error. Existen tres opciones para gestionar el error de comprobación de condición:

- `reject` la mutación. La carga de respuesta de esta opción debe tener esta estructura:

  ```
  {
      "action": "reject"
  }
  ```

  Esto le indica al solucionador de DynamoDB de AWS AppSync que se comporte como si la estrategia configurada fuera `Reject` y que devuelva un error para la mutación y el valor actual del objeto de DynamoDB, tal como se describe en la sección anterior.
- `discard` la mutación. La carga de respuesta de esta opción debe tener esta estructura:

  ```
  {
  ```

```
    "action": "discard"
}
```

Esto indica al solucionador de DynamoDB de AWS AppSync que no notifique el incumplimiento de la condición y que devuelva el valor en DynamoDB.

- `retry` la mutación. La carga de respuesta de esta opción debe tener esta estructura:

```
{
    "action": "retry",
    "retryMapping": { ... }
}
```

Esto le indica al solucionador de DynamoDB de AWS AppSync que vuelva a intentar la mutación con otro documento de mapeo de solicitud. La estructura de la sección `retryMapping` depende de la operación de DynamoDB y es un subconjunto del documento de mapeo de solicitudes completo de esa operación.

Para `PutItem`, la sección `retryMapping` tiene la siguiente estructura. Para ver una descripción del campo `attributeValues`, consulte PutItem.

```
{
    "attributeValues": { ... },
    "condition": {
        "equalsIgnore" = [ ... ],
        "consistentRead" = true
    }
}
```

Para `UpdateItem`, la sección `retryMapping` tiene la siguiente estructura. Para ver una descripción de la sección `update`, consulte UpdateItem.

```
{
    "update" : {
        "expression" : "someExpression"
        "expressionNames" : {
            "#foo" : "foo"
        },
        "expressionValues" : {
            ":bar" : ... typed value
        }
```

```
    },
    "condition": {
        "consistentRead" = true
    }
}
```

Para `DeleteItem`, la sección `retryMapping` tiene la siguiente estructura.

```
{
    "condition": {
        "consistentRead" = true
    }
}
```

No hay forma de especificar otra operación o clave en la que trabajar. El solucionador de DynamoDB de AWS AppSync solo permite reintentos de la misma operación en el mismo objeto. Asimismo, la sección `condition` no permite especificar un `conditionalCheckFailedHandler`. Si el reintento fracasa, el solucionador de DynamoDB de AWS AppSync sigue la estrategia `Reject`.

A continuación se muestra un ejemplo de función Lambda para tratar una solicitud `PutItem` sin éxito. La lógica de negocio examina quién realizó la llamada. Si la realizó `jeffTheAdmin`, vuelve a intentar realizar la solicitud, actualizando `version` y `expectedVersion` desde el elemento actualmente en DynamoDB. De lo contrario, rechaza la mutación.

```
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)
};
```

Expresiones de condición de transacción

Las expresiones de condición de transacción están disponibles en las plantillas de mapeo de solicitudes de los cuatro tipos de operaciones en `TransactWriteItems`, a saber, `PutItem`, `DeleteItem`, `UpdateItem` y `ConditionCheck`.

Para `PutItem`, `DeleteItem` y `UpdateItem`, la expresión de condición de transacción es opcional. Para `ConditionCheck`, se requiere la expresión de condición de transacción.

Ejemplo 1

El siguiente documento de mapeo transaccional de `DeleteItem` no tiene una expresión de condición. Como resultado, elimina el elemento en DynamoDB.

```
{
   "version": "2018-05-29",
   "operation": "TransactWriteItems",
   "transactItems": [
      {
         "table": "posts",
         "operation": "DeleteItem",
         "key": {
            "id": { "S" : "1" }
         }
      }
   ]
```

```
}
```

Ejemplo 2

El siguiente documento de mapeo transaccional de `DeleteItem` tiene una expresión de condición de transacción que permite que la operación tenga éxito únicamente si el autor de esa publicación es igual a un nombre determinado.

```
{
   "version": "2018-05-29",
   "operation": "TransactWriteItems",
   "transactItems": [
      {
         "table": "posts",
         "operation": "DeleteItem",
         "key": {
            "id": { "S" : "1" }
         }
         "condition": {
            "expression": "author = :author",
            "expressionValues": {
               ":author": { "S" : "Chunyan" }
            }
         }
      }
   ]
}
```

Si la comprobación de condición no se realiza correctamente, provocará la excepción `TransactionCanceledException` y se devolverá el detalle del error en `$ctx.result.cancellationReasons`. Tenga en cuenta que, de forma predeterminada, el elemento anterior de DynamoDB que provocó el error en esa comprobación de condición se devolverá en `$ctx.result.cancellationReasons`.

Especificación de una condición

Todos los documentos de mapeo de solicitudes `PutItem`, `UpdateItem` y `DeleteItem` permiten especificar una sección `condition` opcional. Si se omite, no se comprobará ninguna condición. Si se especifica, la condición debe ser true para que la operación se lleve a cabo correctamente. `ConditionCheck` debe tener una sección `condition` sección para especificarla. La condición debe ser verdadera para que toda la transacción se realice correctamente.

Las secciones `condition` tienen la siguiente estructura:

```
"condition": {
    "expression": "someExpression",
    "expressionNames": {
        "#foo": "foo"
    },
    "expressionValues": {
        ":bar": ... typed value
    },
    "returnValuesOnConditionCheckFailure": false
}
```

Los campos siguientes especifican la condición:

`expression`

La misma expresión de actualización. Para obtener más información sobre cómo escribir expresiones de condición, consulte la documentación de DynamoDB ContitionExpressions. Este campo debe especificarse.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre usado en la expresión, y el valor tiene que ser una cadena que corresponda al nombre de atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con las sustituciones de los marcadores de posición de nombre de atributo de expresión que se usen en la expresión.

`expressionValues`

Las sustituciones de los marcadores de posición de valor de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de valor usado en la expresión y el valor tiene que ser un valor con tipo. Para obtener más información sobre cómo especificar un “valor con tipo”, consulte Sistema de tipos (mapeo de solicitud). Este valor debe especificarse. Este campo es opcional y solo debe rellenarse con las sustituciones de los marcadores de posición de valor de atributo de expresión que se usen en la expresión.

`returnValuesOnConditionCheckFailure`

Especifique si desea recuperar el elemento en DynamoDB cuando se produzca un error en la comprobación de condición. El elemento recuperado estará en `$ctx.result.cancellationReasons[$index].item`, donde `$index` es el índice del

elemento de solicitud que no ha superado la comprobación de condición. Este valor se establece de forma predeterminada en true.

Proyecciones

Al leer objetos de DynamoDB mediante las operaciones `GetItem`, `Scan`, `Query`, `BatchGetItem` y `TransactGetItems`, tiene la posibilidad de especificar una proyección para identificar los atributos deseados. La proyección tiene la siguiente estructura, que es similar a los filtros:

```
"projection" : {
    "expression" : "projection expression"
    "expressionNames" : {
        "#name" : "name",
    }
}
```

Los campos se definen de la siguiente manera:

`expression`

La expresión de proyección, que es una cadena. Para recuperar un solo atributo, especifique su nombre. Si desea obtener varios atributos, separe sus nombres mediante comas. Para obtener más información sobre la redacción de expresiones de proyección, consulte la documentación relativa a las expresiones de proyección de DynamoDB. Este campo es obligatorio.

`expressionNames`

Las sustituciones de los marcadores de posición de nombre de atributo de expresión, en forma de pares de clave-valor. La clave corresponde a un marcador de posición de nombre utilizado en la `expression`. El valor debe ser una cadena que corresponda al nombre del atributo del elemento en DynamoDB. Este campo es opcional y solo debe rellenarse con las sustituciones de marcadores de posición de nombre de atributo de expresión que se usen en la `expression`. Para obtener más información acerca de `expressionNames`, consulte la documentación de DynamoDB.

Ejemplo 1

El siguiente ejemplo es una sección de proyección para un mapeo de VTL en el que solo los atributos `author` y `id` se devuelven de DynamoDB:

```
"projection" : {
    "expression" : "#author, id",
    "expressionNames" : {
        "#author" : "author"
    }
}
```

> **Tip**
>
> Puede acceder a su conjunto de selección de solicitudes de GraphQL mediante $context.info.selectionSetList. Este campo permite enmarcar su expresión de proyección de forma dinámica según sus requisitos.

> **Note**
>
> Al utilizar expresiones de proyección con las operaciones `Query` y `Scan`, el valor de `select` debe ser `SPECIFIC_ATTRIBUTES`. Para obtener más información, consulte la documentación de DynamoDB.

Referencia de plantillas de mapeo de solucionador para RDS

Las plantillas de mapeo de solucionador de RDS de AWS AppSync permiten a los desarrolladores enviar consultas SQL a una API de datos para Amazon Aurora sin servidor y obtener el resultado de estas consultas.

Plantilla de mapeo de solicitudes

La plantilla de mapeo de solicitudes de RDS es bastante sencilla:

```
{
    "version": "2018-05-29",
    "statements": [],
    "variableMap": {},
    "variableTypeHintMap": {}
}
```

A continuación, se muestra el esquema JSON de la plantilla de mapeo de solicitudes de RDS una vez que se ha resuelto:

```
{
    "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"
        }
    }
}
```

A continuación se muestra un ejemplo de la plantilla de mapeo de solicitudes con una consulta estática:

```
{
    "version": "2018-05-29",
    "statements": [
        "select title, isbn13 from BOOKS where author = 'Mark Twain'"
    ]
}
```

Versión

El campo de versión es común a todas las plantillas de mapeo de solicitudes y define la versión utilizada por la plantilla. El campo version es obligatorio. El valor “2018-05-29” es la única versión admitida para las plantillas de mapeo de Amazon RDS.

```
"version": "2018-05-29"
```

Instrucciones y VariableMap

La matriz de instrucciones es un marcador de posición para las consultas que proporciona el desarrollador. En la actualidad, se admiten hasta dos consultas por plantilla de mapeo de solicitudes. El campo `variableMap` es opcional y contiene los alias que se pueden utilizar para que las instrucciones SQL sean más breves y legibles. Por ejemplo, lo siguiente es posible:

```
{
"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 AppSync utilizará los valores del mapa de variables para construir las consultas SqlParameterized que se enviarán a la API de datos de Amazon Aurora sin servidor. Las instrucciones SQL se ejecutan con parámetros proporcionados en el mapa de variables, lo que elimina el riesgo de inyección de código SQL.

VariableTypeHintMap

El `variableTypeHintMap` es un campo opcional que contiene tipos con alias que se pueden usar para enviar sugerencias de tipo de parámetros SQL. Estas sugerencias de tipo evitan la conversión explícita en las instrucciones SQL, lo que las hace más cortas. Por ejemplo, lo siguiente es posible:

```
{
    "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 AppSync utilizará el valor del mapa de variables para construir las consultas que se envían a la API de datos de Amazon Aurora sin servidor. También utiliza los datos de `variableTypeHintMap` y envía la información del tipo a RDS. `typeHints` compatibles con RDS se pueden encontrar aquí.

Referencia de plantillas de mapeo de solucionador para OpenSearch

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

El solucionador de AWS AppSync para Amazon OpenSearch Service permite utilizar GraphQL para almacenar y recuperar datos de dominios de OpenSearch Service ya existentes en su cuenta. Para funcionar, este solucionador permite mapear una solicitud de GraphQL entrante a una solicitud de OpenSearch Service y, a continuación, mapear la respuesta de OpenSearch Service a GraphQL. En esta sección se describen las plantillas de mapeo para las operaciones de OpenSearch Service admitidas.

Plantilla de mapeo de solicitudes

La mayoría de plantillas de mapeo de solicitudes de OpenSearch Service presentan una estructura común en la que tan solo cambian unos pocos elementos. En el siguiente ejemplo se ejecuta una búsqueda en un dominio de OpenSearch Service donde los documentos están organizados bajo un índice denominado `post`. Los parámetros de búsqueda se definen en la sección `body` y muchas de las cláusulas de consulta comunes se definen en el campo `query`. En este ejemplo se buscan documentos que contengan `"Nadia"`, `"Bailey"` o ambos en el campo `author` de un documento:

```
{
    "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" }}
                    ]
                }
            }
        }
    }
}
```

Plantilla de mapeo de respuestas

Al igual que ocurre con otros orígenes de datos, OpenSearch Service envía una respuesta a AWS AppSync que debe convertirse a GraphQL.

La mayoría de consultas de GraphQL buscan el campo `_source` de una respuesta de OpenSearch Service. Puesto que puede hacer búsquedas para devolver un documento individual o una lista de documentos, en OpenSearch Service se utilizan dos plantillas de mapeo de respuestas comunes:

Lista de resultados

```
[
    #foreach($entry in $context.result.hits.hits)
      #if( $velocityCount > 1 ) , #end
        $utils.toJson($entry.get("_source"))
    #end
]
```

Elemento individual

```
$utils.toJson($context.result.get("_source"))
```

Campo **`operation`**

(Solo plantilla de mapeo de SOLICITUDES)

Método o verbo HTTP (GET, POST, PUT, HEAD o DELETE) que envía AWS AppSync al dominio de OpenSearch Service. Tanto la clave como el valor deben ser cadenas.

```
"operation" : "PUT"
```

Campo **path**

(Solo plantilla de mapeo de SOLICITUDES)

Ruta de búsqueda de una solicitud de OpenSearch Service desde AWS AppSync. Esto constituye una URL para el verbo HTTP de la operación. Tanto la clave como el valor deben ser cadenas.

```
"path" : "/<indexname>/_doc/<_id>"
"path" : "/<indexname>/_doc"
"path" : "/<indexname>/_search"
"path" : "/<indexname>/_update/<_id>
```

Cuando se evalúa la plantilla de mapeo, esta ruta se envía como parte de la solicitud HTTP, incluido el dominio de OpenSearch Service. Por ejemplo, el ejemplo anterior puede convertirse como:

```
GET https://opensearch-domain-name.REGION.es.amazonaws.com/indexname/type/_search
```

Campo **params**

(Solo plantilla de mapeo de SOLICITUDES)

Se utiliza para especificar la acción que realiza la búsqueda, normalmente estableciendo el valor query dentro de body. Sin embargo, se pueden configurar otras funcionalidades, como, por ejemplo, el formato de las respuestas.

- headers

  Es la información del encabezado en forma de pares clave-valor. Tanto la clave como el valor deben ser cadenas. Por ejemplo:

  ```
  "headers" : {
      "Content-Type" : "application/json"
  }
  ```

  > **Note**
  >
  > Actualmente, AWS AppSync solo admite JSON como `Content-Type`.

- queryString

Son los pares clave-valor que especifican opciones comunes, como el formato de código de las respuestas JSON. Tanto la clave como el valor deben ser cadenas. Por ejemplo, si desea JSON con formato pretty, puede especificar:

```
"queryString" : {
    "pretty" : "true"
}
```

- body

  Esta es la parte principal de la solicitud, la que permite a AWS AppSync crear una petición de búsqueda bien formada dirigida al dominio de OpenSearch Service. La clave debe ser una cadena compuesta por un objeto. A continuación se muestran algunos ejemplos.

Ejemplo 1

Devuelve todos los documentos que incluyan la ciudad “seattle”:

```
"body":{
    "from":0,
    "size":50,
    "query" : {
        "match" : {
            "city" : "seattle"
        }
    }
}
```

Ejemplo 2

Devuelve todos los documentos que incluyan “washington” como ciudad o estado:

```
"body":{
    "from":0,
    "size":50,
    "query" : {
        "multi_match" : {
            "query" : "washington",
            "fields" : ["city", "state"]
        }
```

```
    }
}
```

Variables de transferencia

(Solo plantilla de mapeo de SOLICITUDES)

También puede transferir variables como parte de la evaluación a la instrucción en VTL. Por ejemplo, suponga que ha tenido la siguiente consulta de GraphQL:

```
query {
    searchForState(state: "washington"){
        ...
    }
}
```

La plantilla de mapeo podría tomar el estado como un argumento:

```
"body":{
    "from":0,
    "size":50,
    "query" : {
        "multi_match" : {
            "query" : "$context.arguments.state",
            "fields" : ["city", "state"]
        }
    }
}
```

Para obtener una lista de las funciones que puede incluir en el código VTL, consulte Acceso a los encabezados de consultas.

Referencia de plantillas de mapeo de solucionador para Lambda

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

Puede usar AWS AppSync funciones y resolutores para invocar las funciones de Lambda ubicadas en su cuenta. Puede configurar las cargas útiles de sus solicitudes y la respuesta de sus funciones Lambda antes de devolverlas a sus clientes. También puede utilizar plantillas de mapeo para dar pistas AWS AppSync sobre la naturaleza de la operación que se va a invocar. En esta sección se describen las distintas plantillas de mapeo para las operaciones de Lambda admitidas.

Plantilla de mapeo de solicitudes

La plantilla de mapeo de solicitudes de Lambda gestiona los campos relacionados con la función de Lambda:

```
{
  "version": string,
  "operation": Invoke|BatchInvoke,
  "payload": any type,
  "invocationType": RequestResponse|Event
}
```

Esta es la representación del esquema JSON de la plantilla de mapeo de solicitudes de Lambda cuando se resuelve:

```
{
  "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": {},
    "invocationType": {
      "$id": "/properties/invocationType",
      "type": "string",
      "enum": [
        "RequestResponse",
        "Event"
      ],
      "title": "The Mapping template invocation type.",
      "description": "What invocation type to execute.",
      "default": "RequestResponse"
    }
  },
  "required": [
    "version",
    "operation"
  ],
  "additionalProperties": false
}
```

Este es un ejemplo en el que se usa una `invoke` operación cuyos datos de carga útil son el `getPost` campo de un esquema de GraphQL junto con sus argumentos del contexto:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": $util.toJson($context.arguments)
  }
}
```

El documento de mapeo completo se pasa como entrada a la función Lambda, de modo que el ejemplo anterior ahora tiene este aspecto:

```
{
```

```
  "version": "2018-05-29",
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": {
      "id": "postId1"
    }
  }
}
```

Versión

Común a todas las plantillas de mapeo de solicitudes, `version` define la versión que utiliza la plantilla. `version`Es obligatorio y es un valor estático:

```
"version": "2018-05-29"
```

Operación

La fuente de datos Lambda le permite definir dos operaciones en el `operation` campo: `Invoke` y. `BatchInvoke` La `Invoke` operación permite llamar a AWS AppSync la función Lambda para cada solucionador de campos de GraphQL. `BatchInvoke`indica AWS AppSync que se agrupen las solicitudes para el campo GraphQL actual. El campo `operation` es obligatorio.

Para`Invoke`, la plantilla de mapeo de solicitudes resueltas coincide con la carga útil de entrada de la función Lambda. Modifiquemos el ejemplo anterior:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
    "payload": {
      "arguments": $util.toJson($context.arguments)
    }
}
```

Esto se resuelve y se pasa a la función Lambda, que podría tener un aspecto similar al siguiente:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
```

```
    "payload": {
      "arguments": {
        "id": "postId1"
      }
    }
}
```

`BatchInvoke`En efecto, la plantilla de mapeo se aplica a todos los solucionadores de campos del lote. Para mayor concisión, AWS AppSync fusiona todos los `payload` valores de la plantilla de mapeo resueltos en una lista bajo un único objeto que coincida con la plantilla de mapeo. La siguiente plantilla de ejemplo muestra esta combinación:

```
{
  "version": "2018-05-29",
  "operation": "BatchInvoke",
  "payload": $util.toJson($context)
}
```

Esta plantilla se resuelve para dar el siguiente documento de mapeo:

```
{
  "version": "2018-05-29",
  "operation": "BatchInvoke",
  "payload": [
    {...}, // context for batch item 1
    {...}, // context for batch item 2
    {...}  // context for batch item 3
  ]
}
```

Cada elemento de la `payload` lista corresponde a un único elemento del lote. También se espera que la función Lambda devuelva una respuesta en forma de lista que coincida con el orden de los elementos enviados en la solicitud:

```
[
  { "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
```

```
]
```

Carga

El `payload` campo es un contenedor que se utiliza para pasar cualquier JSON bien formado a la función Lambda. Si el `operation` campo está establecido en`BatchInvoke`, agrupa AWS AppSync los `payload` valores existentes en una lista. El campo `payload` es opcional.

Tipo de invocación

La fuente de datos Lambda le permite definir dos tipos de invocación: y. `RequestResponse` `Event` Los tipos de invocación son sinónimos de los tipos de invocación definidos en la API Lambda. El tipo `RequestResponse` de invocación permite AWS AppSync llamar a la función Lambda de forma sincrónica para esperar una respuesta. La `Event` invocación le permite invocar la función Lambda de forma asíncrona. Para obtener más información sobre cómo Lambda gestiona las solicitudes de tipo de `Event` invocación, consulte Invocación asíncrona. El campo `invocationType` es opcional. Si este campo no está incluido en la solicitud, se AWS AppSync utilizará de forma predeterminada el tipo de invocación. `RequestResponse`

Para cualquier `invocationType` campo, la solicitud resuelta coincide con la carga útil de entrada de la función Lambda. Modifiquemos el ejemplo anterior:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "invocationType": "Event"
  "payload": {
    "arguments": $util.toJson($context.arguments)
  }
}
```

Esto se resuelve y se pasa a la función Lambda, que podría tener un aspecto similar al siguiente:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "invocationType": "Event",
  "payload": {
    "arguments": {
      "id": "postId1"
```

```
    }
  }
}
```

Cuando la `BatchInvoke` operación se usa junto con el campo de tipo de `Event` invocación, AWS AppSync fusiona el solucionador de campos de la misma manera que se mencionó anteriormente y la solicitud se pasa a la función Lambda como un evento asíncrono, siendo una lista de valores. `payload` Le recomendamos que deshabilite el almacenamiento en caché de los resolutores de tipo `Event` invocación, ya que estos no se enviarían a Lambda si se produjera un error en la memoria caché.

Plantilla de mapeo de respuestas

Al igual que con otras fuentes de datos, la función Lambda envía una respuesta AWS AppSync que debe convertirse a un tipo GraphQL.

El resultado de la función de Lambda se define con el objeto `context` que está disponible a través de la propiedad `$context.result` de Velocity Template Language (VTL).

Si la forma de la respuesta de la función Lambda coincide exactamente con la forma del tipo de GraphQL, puede reenviar la respuesta mediante la siguiente plantilla de mapeo de respuesta:

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

No hay campos obligatorios ni restricciones de forma aplicables a la plantilla de mapeo de respuesta. Sin embargo, dado que los tipos de GraphQL son estrictos, la plantilla de mapeo resuelta debe coincidir con el tipo de GraphQL previsto.

Repuesta de la función de Lambda en lotes

Si el `operation` campo está establecido en`BatchInvoke`, AWS AppSync espera una lista de elementos de la función Lambda. Para volver AWS AppSync a asignar cada resultado al elemento de la solicitud original, la lista de respuestas debe coincidir en tamaño y orden. Es válido tener `null` elementos en la lista de respuestas; `$ctx.result` se establece en nulo en consecuencia.

Solucionadores de Lambda directos

Si desea evitar por completo el uso de plantillas de mapeo, AWS AppSync puede proporcionar una carga útil predeterminada a su función Lambda y una respuesta de función Lambda predeterminada

a un tipo GraphQL. Puedes elegir entre proporcionar una plantilla de solicitud, una plantilla de respuesta o ninguna de las dos, y gestionarla en consecuencia. AWS AppSync

Plantilla de mapeo de la solicitud Lambda directa

Si no se proporciona la plantilla de mapeo de solicitudes, AWS AppSync enviará el `Context` objeto directamente a la función Lambda como una `Invoke` operación. Para obtener más información sobre la estructura del objeto `Context`, consulte Referencia de contexto de las plantillas de mapeo del solucionador.

Plantilla de mapeo de la respuesta de Lambda directa

Cuando no se proporciona la plantilla de mapeo de respuestas, AWS AppSync realiza una de estas dos acciones al recibir la respuesta de la función Lambda. Si no proporcionó una plantilla de mapeo de solicitud o si proporcionó una plantilla de mapeo de solicitud con la versión`2018-05-29`, la respuesta será equivalente a la siguiente plantilla de mapeo de respuesta:

```
#if($ctx.error)
     $util.error($ctx.error.message, $ctx.error.type, $ctx.result)
 #end
 $util.toJson($ctx.result)
```

Si ha proporcionado una plantilla con la versión`2017-02-28`, la lógica de respuesta funciona de manera equivalente a la siguiente plantilla de mapeo de respuestas:

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

Superficialmente, la omisión de la plantilla de mapeo funciona de manera similar al uso de ciertas plantillas de mapeo, como se muestra en los ejemplos anteriores. Sin embargo, entre bastidores, se elude por completo la evaluación de las plantillas de mapeo. Como se omite el paso de evaluación de la plantilla, es posible que las aplicaciones experimenten menos sobrecarga y latencia durante la respuesta en algunos escenarios, en comparación con una función Lambda con una plantilla de mapeo de respuestas que deba evaluarse.

Gestión de errores personalizada en las respuestas de solucionador de Lambda directo

Puede personalizar las respuestas de error de las funciones de Lambda que invocan los solucionadores de Lambda directos mostrando una excepción personalizada. En el siguiente ejemplo, se muestra cómo crear una excepción personalizada mediante: JavaScript

```
class CustomException extends Error {
  constructor(message) {
    super(message);
    this.name = "CustomException";
  }
}

throw new CustomException("Custom message");
```

Cuando se muestran excepciones, `errorType` y `errorMessage` son el `name` y `message`, respectivamente, del error personalizado que se produce.

Si `errorType` es `UnauthorizedException` así, AWS AppSync devuelve el mensaje predeterminado (`"You are not authorized to make this call."`) en lugar de un mensaje personalizado.

El siguiente fragmento es un ejemplo de respuesta de GraphQL que muestra una personalización: `errorType`

```
{
  "data": {
    "query": null
  },
  "errors": [
    {
      "path": [
        "query"
      ],
      "data": null,
      "errorType": "CustomException",
      "errorInfo": null,
      "locations": [
        {
          "line": 5,
          "column": 10,
          "sourceName": null
        }
      ],
      "message": "Custom Message"
    }
  ]
}
```

Solucionadores de Lambda directos: agrupación en lotes habilitada

Puede habilitar la agrupación en lotes para el solucionador de Lambda directo mediante la configuración del `maxBatchSize` en el solucionador. Cuando `maxBatchSize` se establece en un valor superior `0` al de un solucionador de Direct Lambda, AWS AppSync envía las solicitudes en lotes a la función Lambda en tamaños de hasta. `maxBatchSize`

Si `maxBatchSize` se establece `0` en un solucionador Direct Lambda, se desactiva el procesamiento por lotes.

Para obtener más información sobre el funcionamiento de la agrupación en lotes con solucionadores de Lambda, consulte Caso de uso avanzado: agrupación en lotes.

Plantilla de mapeo de solicitudes

Cuando el procesamiento por lotes está activado y no se proporciona la plantilla de mapeo de solicitudes, AWS AppSync envía una lista de `Context` objetos como una `BatchInvoke` operación directamente a la función Lambda.

Plantilla de mapeo de respuestas

Si la agrupación en lotes está habilitada y no se proporciona la plantilla de mapeo de respuestas, la lógica de respuesta es equivalente a la siguiente plantilla de mapeo de respuestas:

```
#if( $context.result && $context.result.errorMessage )
      $utils.error($context.result.errorMessage, $context.result.errorType,
      $context.result.data)
#else
      $utils.toJson($context.result.data)
#end
```

La función de Lambda debe devolver una lista de resultados en el mismo orden que la lista de objetos `Context` que se han enviado. Puede devolver errores individuales proporcionando un `errorMessage` y `errorType` para un resultado específico. Cada resultado de la lista se indica con el formato siguiente:

```
{
   "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**
>
> Actualmente, se ignoran otros campos del objeto de resultado.

Gestión de errores de Lambda

Puede devolver un error para todos los resultados produciendo una excepción o un error en la función de Lambda. Si el tamaño de respuesta o solicitud de carga de la solicitud por lote es demasiado grande, Lambda devolverá un error. En ese caso, debería considerar la posibilidad de reducir el `maxBatchSize` o el tamaño de la carga de la respuesta.

Para obtener información sobre la gestión de errores individuales, consulte Devolución de errores individuales.

Funciones de Lambda de ejemplo

Con el siguiente esquema, puede crear un Resolver Lambda Directo para el solucionador de `Post.relatedPosts` campo y habilitar el procesamiento por lotes mediante la configuración anterior: `maxBatchSize 0`

```
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]
}
```

En la siguiente consulta, se llamará a la función de Lambda con lotes de solicitudes para resolver `relatedPosts`:

```
query getAllPosts {
  allPosts {
    id
    relatedPosts {
      id
    }
  }
}
```

A continuación, se proporciona una implementación sencilla de una función de Lambda:

```
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' }
    }
  })
}
```

Referencia de plantilla de mapeo de Resolver para EventBridge

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

La AWS AppSync plantilla de mapeo de resolución utilizada con la fuente de EventBridge datos le permite enviar eventos personalizados al EventBridge bus de Amazon.

Plantilla de mapeo de solicitudes

La plantilla de mapeo de `PutEvents` solicitudes le permite enviar varios eventos personalizados a un bus de EventBridge eventos. El documento de mapeo tiene la siguiente estructura:

```
{
    "version" : "2018-05-29",
    "operation" : "PutEvents",
    "events" : [{}]
}
```

El siguiente es un ejemplo de una plantilla de mapeo de solicitudes para 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"
    }

    ]
}
```

Plantilla de mapeo de respuestas

Si la `PutEvents` operación se realiza correctamente, la respuesta de EventBridge se incluye en`$ctx.result`:

```
#if($ctx.error)
  $util.error($ctx.error.message, $ctx.error.type, $ctx.result)
#end
  $util.toJson($ctx.result)
```

Los errores que se produzcan al realizar operaciones `PutEvents` como `InternalExceptions` o `Timeouts` aparecerán en `$ctx.error`. Para obtener una lista EventBridge de los errores más comunes, consulta la referencia de errores EventBridge comunes.

El `result` tendrá el siguiente formato:

```
{
    "Entries" [
        {
            "ErrorCode" : String,
            "ErrorMessage" : String,
            "EventId" : String
        }
    ],
    "FailedEntryCount" : number
}
```

- Entradas

Resultados de los eventos ingeridos, tanto correctos como incorrectos. Si la ingesta se realizó correctamente, la entrada contiene el `EventID`. De lo contrario, puede usar `ErrorCode` y `ErrorMessage` para identificar el problema con la entrada.

Para cada registro, el índice del elemento de respuesta es el mismo que el de la matriz de solicitudes.

- FailedEntryCount

  Número de entradas con error. Este valor se representa como un entero.

Para obtener más información sobre la respuesta de`PutEvents`, consulte PutEvents.

Ejemplo de respuesta de muestra 1

El siguiente ejemplo es una operación `PutEvents` con dos eventos correctos:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        }
    ],
    "FailedEntryCount" : 0
}
```

Ejemplo de respuesta de muestra 2

El siguiente ejemplo es una operación `PutEvents` con tres eventos, dos correctos y uno incorrecto:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        },
        {
```

```
            "ErrorCode" : "SampleErrorCode",
            "ErrorMessage" : "Sample Error Message"
        }
    ],
    "FailedEntryCount" : 1
}
```

Campo **PutEvents**

- Versión

  El campo `version` es común a todas las plantillas de mapeo de solicitudes y define la versión utilizada por la plantilla. Este campo es obligatorio. El valor `2018-05-29` es la única versión compatible con las plantillas de EventBridge mapeo.
- Operación

  La única operación admitida es `PutEvents`. Esta operación permite añadir eventos personalizados a su bus de eventos.
- Eventos

  Una matriz de eventos que se añadirán al bus de eventos. Esta matriz debe tener una asignación de entre 1 y 10 elementos.

  El objeto `Event` es un objeto JSON válido que tiene los siguientes campos:
  - `"source"`: cadena que define el origen del evento.
  - `"detail"`: objeto JSON que puede usar para asociar información sobre el evento. Este campo puede ser un mapa vacío (`{ }`).
  - `"detailType`: cadena que identifica el tipo de evento
  - `"resources"`: matriz JSON de cadenas que identifica los recursos involucrados en el evento. Este campo puede ser una matriz vacía.
  - `"time"`: marca temporal del evento proporcionada como cadena. Debe seguir el formato de marca temporal RFC3339.

Los siguientes fragmentos de código son algunos ejemplos de objetos `Event` válidos:

Ejemplo 1

```
{
```

```
    "source" : "source1",
    "detail" : {
        "key1" : [1,2,3,4],
        "key2" : "strval"
    },
    "detailType" : "sampleDetailType",
    "resources" : ["Resouce1", "Resource2"],
    "time" : "2022-01-10T05:00:10Z"
}
```

Ejemplo 2

```
{
    "source" : "source1",
    "detail" : {},
    "detailType" : "sampleDetailType"
}
```

Ejemplo 3

```
{
    "source" : "source1",
    "detail" : {
        "key1" : 1200
    },
    "detailType" : "sampleDetailType",
    "resources" : []
}
```

Referencia de plantillas de mapeo de solucionador para el origen de datos None

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

La plantilla de mapeo de solucionador de AWS AppSync que se utiliza con el origen de datos de tipo None permite dar forma a las solicitudes de operaciones locales de AWS AppSync.

Plantilla de mapeo de solicitudes

La plantilla de mapeo es sencilla y le permite transferir toda la información de contexto posible a través del campo `payload`.

```
{
   "version": string,
   "payload": any type
}
```

A continuación, se muestra el esquema JSON de la plantilla de mapeo de la solicitud una vez resuelta:

```
{
    "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
}
```

En el siguiente ejemplo se transfieren los argumentos del campo a través de la propiedad del contexto de VTL `$context.arguments`:

```
{
    "version": "2018-05-29",
```

```
    "payload": $util.toJson($context.arguments)
}
```

El valor del campo `payload` se reenviará a la plantilla de mapeo de respuesta y estará disponible en la propiedad del contexto de VTL (`$context.result`).

Este es un ejemplo que representa el valor interpolado del campo `payload`:

```
{
    "id": "postId1"
}
```

Versión

El campo `version` es común a todas las plantillas de mapeo de solicitudes y define la versión utilizada por la plantilla.

El campo `version` es obligatorio.

Ejemplo:

```
"version": "2018-05-29"
```

Carga

El campo `payload` es un contenedor que se puede utilizar para transferir cualquier formato JSON correcto a la plantilla de mapeo de respuesta.

El campo `payload` es opcional.

Plantilla de mapeo de respuestas

Dado que no hay ningún origen de datos, el valor del campo `payload` se reenviará a la plantilla de mapeo de respuesta y se establecerá en el objeto `context` que está disponible a través de la propiedad `$context.result` de VTL.

Si la forma del valor del campo `payload` coincide exactamente con la forma del tipo de GraphQL, puede reenviar la respuesta mediante la siguiente plantilla de mapeo de respuesta:

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

No hay campos obligatorios ni restricciones de forma aplicables a la plantilla de mapeo de respuesta. Sin embargo, dado que los tipos de GraphQL son estrictos, la plantilla de mapeo resuelta debe coincidir con el tipo de GraphQL previsto.

Referencia de plantillas de mapeo de solucionador para HTTP

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

Las plantillas de mapeo de solucionador de HTTP para AWS AppSync permiten enviar las solicitudes de AWS AppSync dirigidas a cualquier punto de enlace HTTP y también a las respuestas de este hacia AWS AppSync. El uso de plantillas de mapeo le permite indicar a AWS AppSync la naturaleza de la operación que se va a invocar. En esta sección se describen las distintas plantillas de mapeo para el solucionador de HTTP admitido.

Plantilla de mapeo de solicitudes

```
{
    "version": "2018-05-29",
    "method": "PUT|POST|GET|DELETE|PATCH",
    "params": {
        "query": Map,
        "headers": Map,
        "body": any
    },
    "resourcePath": string
}
```

Una vez resuelta la plantilla de mapeo de solicitud HTTP, la representación del esquema JSON de la plantilla de mapeo de solicitud debe ser similar a la siguiente:

```
{
    "$id": "https://aws.amazon.com/appsync/request-mapping-template.json",
    "type": "object",
    "properties": {
        "version": {
        "$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"
    ]
}
```

El siguiente es un ejemplo de una solicitud HTTP POST con cuerpo `text/plain`:

```
{
    "version": "2018-05-29",
    "method": "POST",
    "params": {
        "headers":{
        "Content-Type":"text/plain"
        },
        "body":"this is an example of text body"
    },
    "resourcePath": "/"
}
```

Versión

Solo plantilla de mapeo de solicitudes

Define la versión que utiliza la plantilla. `version` es común a todas las plantillas de mapeo de solicitud y es obligatoria.

```
"version": "2018-05-29"
```

Método

Solo plantilla de mapeo de solicitudes

Método o verbo HTTP (GET, POST, PUT, PATCH o DELETE) que envía AWS AppSync al punto de enlace HTTP.

```
"method": "PUT"
```

ResourcePath

Solo plantilla de mapeo de solicitudes

La ruta de recurso a la que desea acceso. Junto con el punto de enlace del origen de datos HTTP, la ruta del recurso forma la URL a la que el servicio AWS AppSync envía la solicitud.

```
"resourcePath": "/v1/users"
```

Cuando se evalúa la plantilla de mapeo, esta ruta se envía como parte de la solicitud HTTP, incluido el punto de conexión HTTP. Por ejemplo, el ejemplo anterior puede convertirse como:

```
PUT <endpoint>/v1/users
```

Campo params

Solo plantilla de mapeo de solicitudes

Se utiliza para especificar la acción que realiza la búsqueda, normalmente estableciendo el valor query dentro de body. Sin embargo, se pueden configurar otras funcionalidades, como, por ejemplo, el formato de las respuestas.

headers

Es la información del encabezado en forma de pares clave-valor. Tanto la clave como el valor deben ser cadenas.

Por ejemplo:

```
"headers" : {
    "Content-Type" : "application/json"
}
```

Los encabezados `Content-Type` admitidos actualmente son:

```
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
```

Nota: No puede definir los siguientes encabezados HTTP:

```
HOST
CONNECTION
USER-AGENT
EXPECTATION
TRANSFER_ENCODING
CONTENT_LENGTH
```

consulta

Son los pares clave-valor que especifican opciones comunes, como el formato de código de las respuestas JSON. Tanto la clave como el valor deben ser cadenas. En el siguiente ejemplo se muestra el modo de enviar una cadena de consulta como `?type=json`:

```
"query" : {
    "type" : "json"
}
```

body

La sección body contiene el cuerpo de la solicitud HTTP que defina. El cuerpo de la solicitud siempre es una cadena con codificación UTF-8, a menos que el tipo de contenido especifique un conjunto de caracteres.

```
"body":"body string"
```

Entidades de certificación (CA) reconocidas por AWS AppSync para los puntos de conexión HTTPS

> **Note**
>
> Let's Encrypt se acepta mediante los certificados identrust e isrgrootx1. Si utiliza Let's Encrypt, no es necesario que realice ninguna acción.

En este momento, los solucionadores de HTTP no admiten certificados autofirmados al utilizar HTTPS. AWS AppSync reconoce las siguientes autoridades de certificación durante la resolución de certificados SSL/TLS para HTTPS:

Certificados raíz conocidos en AWS AppSync

Nombre	Fecha	Huella digital SHA1
`digicertassuredidrootca`	21 de abril de 2018	`05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43`
`trustcenterclass2caii`	21 de abril de 2018	`AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E`
`thawtepremiumserverca`	21 de abril de 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 de noviembre de 2016	`96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09`
`swisssignplatinumg2ca`	21 de abril de 2018	`56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66`
`swisssignsilverg2ca`	21 de abril de 2018	`9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB`

Nombre	Fecha	Huella digital SHA1
thawteserverca	21 de abril de 2018	9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
equifaxsecureebusinessca1	21 de abril de 2018	AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
securetrustca	21 de abril de 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
utnuserfirstclientauthemailca	21 de abril de 2018	B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
thawtepersonalfreemailca	21 de abril de 2018	E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
affirmtrustnetworkingca	21 de abril de 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
entrustevca	21 de abril de 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
utnuserfirsthardwareca	21 de abril de 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
certumca	21 de abril de 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
addtrustclass1ca	21 de abril de 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
entrustrootcag2	21 de abril de 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
equifaxsecureca	21 de abril de 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
quovadisrootca3	21 de abril de 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85

Nombre	Fecha	Huella digital SHA1
quovadisrootca2	21 de abril de 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
digicertglobalrootg2	21 de abril de 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
digicerthighassuranceevrootca	21 de abril de 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
secomvalicertclass1ca	21 de abril de 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
equifaxsecureglobalebusinessca1	21 de abril de 2018	3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36
geotrustuniversalca	21 de abril de 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
deprecateditsecca	27 de enero de 2012	12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
verisignclass3ca	21 de abril de 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
thawteprimaryrootcag3	21 de abril de 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
thawteprimaryrootcag2	21 de abril de 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
deutschetelekomrootca2	21 de abril de 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
buypassclass3ca	21 de abril de 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57

Nombre	Fecha	Huella digital SHA1
utnuserfirstobject ca	21 de abril de 2018	E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B: AC:1D:81:D8:38:5E:2D:46
geotrustprimaryca	21 de abril de 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2: 10:0D:D6:02:90:37:F0:96
buypassclass2ca	21 de abril de 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6: C7:6B:EB:C6:0B:12:40:99
baltimorecodesigni ngca	21 de abril de 2018	30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD: 49:27:08:7C:60:56:7B:0D
verisignclass1ca	21 de abril de 2018	CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45: 3E:64:09:EA:E8:7D:60:F1
baltimorecybertrus tca	21 de abril de 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88: 2C:78:DB:28:52:CA:E4:74
starfieldclass2ca	21 de abril de 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20: 14:C3:D0:E3:37:0E:B5:8A
camerfirmachambers commerceca	21 de abril de 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0: DB:72:2E:31:30:61:F0:B1
ttelesecglobalroot class3ca	21 de abril de 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70: 19:9D:2A:BE:11:E3:81:D1
verisignclass3g5ca	21 de abril de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD: 56:BE:3D:9B:67:44:A5:E5
ttelesecglobalroot class2ca	21 de abril de 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62: 32:17:65:CF:17:D8:94:E9
trustcenteruniversalcai	21 de abril de 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C: CE:BB:9D:D9:4F:4E:39:F3
verisignclass3g4ca	21 de abril de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7: CF:8A:2D:64:C9:3F:6C:3A

Nombre	Fecha	Huella digital SHA1
verisignclass3g3ca	21 de abril de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
xrampglobalca	21 de abril de 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
amzninternalrootca	12 de diciembre de 2008	A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
certplusclass3primaryca	21 de abril de 2018	21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
certumtrustednetworkca	21 de abril de 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
verisignclass3g2ca	21 de abril de 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
globalsignr3ca	21 de abril de 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
utndatacorpsgcca	21 de abril de 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
secomscrootca2	21 de abril de 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
gtecybertrustgloballca	21 de abril de 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
secomscrootca1	21 de abril de 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
affirmtrustcommercialca	21 de abril de 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7

Nombre	Fecha	Huella digital SHA1
trustcenterclass4caii	21 de abril de 2018	A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
verisignuniversalrootca	21 de abril de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
globalsignr2ca	21 de abril de 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
certplusclass2primaryca	21 de abril de 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
digicertglobalrootca	21 de abril de 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
globalsignca	21 de abril de 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
thawteprimaryrootca	21 de abril de 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
starfieldrootg2ca	21 de abril de 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
geotrustglobalca	21 de abril de 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
soneraclass2ca	21 de abril de 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
verisigntsaca	21 de abril de 2018	20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
soneraclass1ca	21 de abril de 2018	07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
quovadisrootca	21 de abril de 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9

Nombre	Fecha	Huella digital SHA1
affirmtrustpremiumeccca	21 de abril de 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
starfieldservicesrootg2ca	21 de abril de 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
valicertclass2ca	21 de abril de 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
comodoaaaca	21 de abril de 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
aolrootca2	21 de abril de 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
keynectisrootca	21 de abril de 2018	9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
addtrustqualifiedca	21 de abril de 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
aolrootca1	21 de abril de 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
verisignclass2g3ca	21 de abril de 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
addtrustexternalca	21 de abril de 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
verisignclass2g2ca	21 de abril de 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
geotrustprimarycag3	21 de abril de 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycag2	21 de abril de 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0

Nombre	Fecha	Huella digital SHA1
swisssigngoldg2ca	21 de abril de 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
entrust2048ca	21 de abril de 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
chunghwaepkirootca	21 de abril de 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
camerfirmachambersignca	21 de abril de 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
camerfirmachambersca	21 de abril de 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
godaddyclass2ca	21 de abril de 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
affirmtrustpremiumca	21 de abril de 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
verisignclass1g3ca	21 de abril de 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
secomevrootca1	21 de abril de 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
verisignclass1g2ca	21 de abril de 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
amzninternalinfoseccag3	27 de febrero de 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 de noviembre de 2016	2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2

Nombre	Fecha	Huella digital SHA1
godaddyrootg2ca	21 de abril de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
digicertassuredidrootca	21 de abril de 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
microseceszignorootca2009	21 de abril de 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
affirmtrustcommercial	21 de abril de 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
comodoecccertificationauthority	21 de abril de 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
cadisigrootr2	21 de abril de 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
swisssignsilvercag2	21 de abril de 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
securetrustca	21 de abril de 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
cadisigrootr1	21 de abril de 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
accvraiz1	21 de abril de 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
entrustrootcertificationauthority	21 de abril de 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
camerfirmaglobalchambersignroot	21 de abril de 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
dstacescax6	21 de abril de 2018	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D

Nombre	Fecha	Huella digital SHA1
identrustpublicsectorrootca1	21 de abril de 2018	BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
starfieldrootcertificateauthorityg2	21 de abril de 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
secureglobalca	21 de abril de 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
eecertificationcentrerootca	21 de abril de 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
opentrustrootcag3	21 de abril de 2018	6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6
teliasonerarootcav1	21 de abril de 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
autoridaddecertificacionfirmaprofesionalcifa62634068	21 de abril de 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
opentrustrootcag2	21 de abril de 2018	79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
opentrustrootcag1	21 de abril de 2018	79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E
globalsigneccrootcar5	21 de abril de 2018	1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
globalsigneccrootcar4	21 de abril de 2018	69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
izenpecom	21 de abril de 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19

Nombre	Fecha	Huella digital SHA1
turktrustelektroniksertifikahizmetsaglayicisih5	21 de abril de 2018	C4:18:F6:4D:46:D1:DF:00:3D:27:30:13:72:43:A9:12:11:C6:75:FB
gdcatrustauthr5root	21 de abril de 2018	0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
dtrustrootclass3ca22009	21 de abril de 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
quovadisrootca3	21 de abril de 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
quovadisrootca2	21 de abril de 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
geotrustprimarycertificationauthorityg3	21 de abril de 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycertificationauthorityg2	21 de abril de 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
oistewisekeyglobalrootgbca	21 de abril de 2018	0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
addtrustexternalroot	21 de abril de 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
chambersofcommerceroot2008	21 de abril de 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
digicertglobalrootg3	21 de abril de 2018	7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E

Nombre	Fecha	Huella digital SHA1
comodoaaaservicesroot	21 de abril de 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
digicertglobalrootg2	21 de abril de 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
certinomisrootca	21 de abril de 2018	9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C:01:B9:32:C5:34:E7:88:A8
oistewisekeyglobalrootgaca	21 de abril de 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
dstrootcax3	21 de abril de 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
certigna	21 de abril de 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
digicerthighassuranceevrootca	21 de abril de 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
soneraclass2rootca	21 de abril de 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
trustcorrootcertca2	21 de abril de 2018	B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
usertrustrsacertificationauthority	21 de abril de 2018	2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
trustcorrootcertca1	21 de abril de 2018	FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
geotrustuniversalca	21 de abril de 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
certsignrootca	21 de abril de 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B

Nombre	Fecha	Huella digital SHA1
amazonrootca4	21 de abril de 2018	F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
amazonrootca3	21 de abril de 2018	0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
amazonrootca2	21 de abril de 2018	5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
verisignuniversalrootcertificationauthority	21 de abril de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
amazonrootca1	21 de abril de 2018	8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
networksolutionscertificate authority	21 de abril de 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
thawteprimaryrootcag3	21 de abril de 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
affirmtrustnetworking	21 de abril de 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
thawteprimaryrootcag2	21 de abril de 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
trustcoreca1	21 de abril de 2018	58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
deutschetelekomrootca2	21 de abril de 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
godaddyrootcertificateauthorityg2	21 de abril de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B

Nombre	Fecha	Huella digital SHA1
entrustrootcertificationauthorityec1	21 de abril de 2018	20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
szafirrootca2	21 de abril de 2018	E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
tubitakkamusmsslkoksertifikasisurum1	21 de abril de 2018	31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA
buypassclass3rootca	21 de abril de 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
comodorsacertificationauthority	21 de abril de 2018	AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
netlockaranyclassgoldfotanusitvany	21 de abril de 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
securitycommunicationrootca2	21 de abril de 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
dtrustrootclass3ca2ev2009	21 de abril de 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
starfieldclass2ca	21 de abril de 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
pscprocert	21 de abril de 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
actalisauthenticationrootca	21 de abril de 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
staatdernederlandenrootcag3	21 de abril de 2018	D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC

Nombre	Fecha	Huella digital SHA1
cfcaevroot	21 de abril de 2018	E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
digicerttrustedrootg4	21 de abril de 2018	DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F:C8:3A:4D:7D:77:5D:05:E4
staatdernederlandenrootcag2	21 de abril de 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
securitycommunicationevrootca1	21 de abril de 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
globalsignrootcar3	21 de abril de 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
globalsignrootcar2	21 de abril de 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
certumtrustednetworkca2	21 de abril de 2018	D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
acraizfnmtrcm	21 de abril de 2018	EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
hellenicacademicandresearch instituti onseccrootca2015	21 de abril de 2018	9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
certplusrootcag2	21 de abril de 2018	4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A
twcarootcertificationauthority	21 de abril de 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
twcaglobalrootca	21 de abril de 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65

Nombre	Fecha	Huella digital SHA1
certplusrootcag1	21 de abril de 2018	22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
geotrustuniversalca2	21 de abril de 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
baltimorecybertrustroot	21 de abril de 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
buypassclass2rootca	21 de abril de 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
certumtrustednetworkca	21 de abril de 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
digicertassuredidrootg3	21 de abril de 2018	F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
digicertassuredidrootg2	21 de abril de 2018	A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F
isrgrootx1	21 de abril de 2018	CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
entrustnetpremium2048secureserverca	21 de abril de 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
certplusclass2primaryca	21 de abril de 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
digicertglobalrootca	21 de abril de 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
entrustrootcertificationauthorityg2	21 de abril de 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4

Nombre	Fecha	Huella digital SHA1
starfieldservicesrootcertificateauthorityg2	21 de abril de 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
thawteprimaryrootca	21 de abril de 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
atostrustedroot2011	21 de abril de 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
geotrustglobalca	21 de abril de 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
luxtrustglobalroot2	21 de abril de 2018	1E:0E:56:19:0A:D1:8B:25:98:B2:04:44:FF:66:8A:04:17:99:5F:3F
etugracertificationauthority	21 de abril de 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
visaecommerceroot	21 de abril de 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
quovadisrootca	21 de abril de 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
identrustcommercialrootca1	21 de abril de 2018	DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
staatdernederlandenevrootca	21 de abril de 2018	76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
ttelesecglobalrootclass3	21 de abril de 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
ttelesecglobalrootclass2	21 de abril de 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9

Nombre	Fecha	Huella digital SHA1
comodocertificationauthority	21 de abril de 2018	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
securitycommunicationrootca	21 de abril de 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
quovadisrootca3g3	21 de abril de 2018	48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
xrampglobalcaroot	21 de abril de 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
securesignrootca11	21 de abril de 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
affirmtrustpremium	21 de abril de 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
globalsignrootca	21 de abril de 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
swisssigngoldcag2	21 de abril de 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
quovadisrootca2g3	21 de abril de 2018	09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
affirmtrustpremiumecc	21 de abril de 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
geotrustprimarycertificationauthority	21 de abril de 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
quovadisrootca1g3	21 de abril de 2018	1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67

Nombre	Fecha	Huella digital SHA1
hongkongpostrootca1	21 de abril de 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
usertrustecccertificationauthority	21 de abril de 2018	D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
cybertrustglobalroot	21 de abril de 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
godaddyclass2ca	21 de abril de 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
hellenicacademicandresearchinstitutionsrootca2015	21 de abril de 2018	01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
ecacc	21 de abril de 2018	28:90:3A:63:5B:52:80:FA:E6:77:4C:0B:6D:A7:D6:BA:A6:4A:F2:E8
hellenicacademicandresearchinstitutionsrootca2011	21 de abril de 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
verisignclass3publicprimarycertificationauthorityg5	21 de abril de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
verisignclass3publicprimarycertificationauthorityg4	21 de abril de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A

Nombre	Fecha	Huella digital SHA1
verisignclass3publicprimary certificationauthorityg3	21 de abril de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
trustisfpsrootca	21 de abril de 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
epkirootcertificationauthority	21 de abril de 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
globalchambersignroot2008	21 de abril de 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
camerfirmachambersofcommerceroot	21 de abril de 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
mozillacert81.pem	13 de marzo de 2014	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
mozillacert99.pem	13 de marzo de 2014	F1:7F:6F:B6:31:DC:99:E3:A3:C8:7F:FE:1C:F1:81:10:88:D9:60:33
mozillacert145.pem	13 de marzo de 2014	10:1D:FA:3F:D5:0B:CB:BB:9B:B5:60:0C:19:55:A4:1A:F4:73:3A:04
mozillacert37.pem	13 de marzo de 2014	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert4.pem	13 de marzo de 2014	E3:92:51:2F:0A:CF:F5:05:DF:F6:DE:06:7F:75:37:E1:65:EA:57:4B

Nombre	Fecha	Huella digital SHA1
mozillacert70.pem	13 de marzo de 2014	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
mozillacert88.pem	13 de marzo de 2014	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
mozillacert134.pem	13 de marzo de 2014	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
mozillacert26.pem	13 de marzo de 2014	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
mozillacert77.pem	13 de marzo de 2014	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert123.pem	13 de marzo de 2014	2A:B6:28:48:5E:78:FB:F3:AD:9E:79:10:DD:6B:DF:99:72:2C:96:E5
mozillacert15.pem	13 de marzo de 2014	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
mozillacert66.pem	13 de marzo de 2014	DD:E1:D2:A9:01:80:2E:1D:87:5E:84:B3:80:7E:4B:B1:FD:99:41:34
mozillacert112.pem	13 de marzo de 2014	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37

Nombre	Fecha	Huella digital SHA1
mozillacert55.pem	13 de marzo de 2014	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
mozillacert101.pem	13 de marzo de 2014	99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39
mozillacert119.pem	13 de marzo de 2014	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
mozillacert44.pem	13 de marzo de 2014	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
mozillacert108.pem	13 de marzo de 2014	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
mozillacert95.pem	13 de marzo de 2014	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
mozillacert141.pem	13 de marzo de 2014	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
mozillacert33.pem	13 de marzo de 2014	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
mozillacert0.pem	13 de marzo de 2014	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74

Nombre	Fecha	Huella digital SHA1
mozillacert84.pem	13 de marzo de 2014	D3:C0:63:F2:19:ED:07:3E:34:AD:5D:75:0B:32:76:29:FF:D5:9A:F2
mozillacert130.pem	13 de marzo de 2014	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
mozillacert148.pem	13 de marzo de 2014	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
mozillacert22.pem	13 de marzo de 2014	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
mozillacert7.pem	13 de marzo de 2014	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
mozillacert73.pem	13 de marzo de 2014	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
mozillacert137.pem	13 de marzo de 2014	4A:65:D5:F4:1D:EF:39:B8:B8:90:4A:4A:D3:64:81:33:CF:C7:A1:D1
mozillacert11.pem	13 de marzo de 2014	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
mozillacert29.pem	13 de marzo de 2014	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE

Nombre	Fecha	Huella digital SHA1
mozillacert62.pem	13 de marzo de 2014	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert126.pem	13 de marzo de 2014	25:01:90:19:CF:FB:D9:99:1C:B7:68:25:74:8D:94:5F:30:93:95:42
mozillacert18.pem	13 de marzo de 2014	79:98:A3:08:E1:4D:65:85:E6:C2:1E:15:3A:71:9F:BA:5A:D3:4A:D9
mozillacert51.pem	13 de marzo de 2014	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
mozillacert69.pem	13 de marzo de 2014	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
mozillacert115.pem	13 de marzo de 2014	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
mozillacert40.pem	13 de marzo de 2014	80:25:EF:F4:6E:70:C8:D4:72:24:65:84:FE:40:3B:8A:8D:6A:DB:F5
mozillacert58.pem	13 de marzo de 2014	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
mozillacert104.pem	13 de marzo de 2014	4F:99:AA:93:FB:2B:D1:37:26:A1:99:4A:CE:7F:F0:05:F2:93:5D:1E

Nombre	Fecha	Huella digital SHA1
mozillacert91.pem	13 de marzo de 2014	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
mozillacert47.pem	13 de marzo de 2014	1B:4B:39:61:26:27:6B:64:91:A2:68:6D:D7:02:43:21:2D:1F:1D:96
mozillacert80.pem	13 de marzo de 2014	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
mozillacert98.pem	13 de marzo de 2014	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
mozillacert144.pem	13 de marzo de 2014	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
mozillacert36.pem	13 de marzo de 2014	23:88:C9:D3:71:CC:9E:96:3D:FF:7D:3C:A7:CE:FC:D6:25:EC:19:0D
mozillacert3.pem	13 de marzo de 2014	87:9F:4B:EE:05:DF:98:58:3B:E3:60:D6:33:E7:0D:3F:FE:98:71:AF
mozillacert87.pem	13 de marzo de 2014	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
mozillacert133.pem	13 de marzo de 2014	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84

Nombre	Fecha	Huella digital SHA1
mozillacert25.pem	13 de marzo de 2014	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
mozillacert76.pem	13 de marzo de 2014	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert122.pem	13 de marzo de 2014	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
mozillacert14.pem	13 de marzo de 2014	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
mozillacert65.pem	13 de marzo de 2014	69:BD:8C:F4:9C:D3:00:FB:59:2E:17:93:CA:55:6A:F3:EC:AA:35:FB
mozillacert111.pem	13 de marzo de 2014	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
mozillacert129.pem	13 de marzo de 2014	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
mozillacert54.pem	13 de marzo de 2014	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
mozillacert100.pem	13 de marzo de 2014	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0

Nombre	Fecha	Huella digital SHA1
mozillacert118.pem	13 de marzo de 2014	7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
mozillacert151.pem	13 de marzo de 2014	AC:ED:5F:65:53:FD:25:CE:01:5F:1F:7A:48:3B:6A:74:9F:61:78:C6
mozillacert43.pem	13 de marzo de 2014	F9:CD:0E:2C:DA:76:24:C1:8F:BD:F0:F0:AB:B6:45:B8:F7:FE:D5:7A
mozillacert107.pem	13 de marzo de 2014	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
mozillacert94.pem	13 de marzo de 2014	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
mozillacert140.pem	13 de marzo de 2014	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
mozillacert32.pem	13 de marzo de 2014	60:D6:89:74:B5:C2:65:9E:8A:0F:C1:88:7C:88:D2:46:69:1B:18:2C
mozillacert83.pem	13 de marzo de 2014	A0:73:E5:C5:BD:43:61:0D:86:4C:21:13:0A:85:58:57:CC:9C:EA:46
mozillacert147.pem	13 de marzo de 2014	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4

Nombre	Fecha	Huella digital SHA1
mozillacert21.pem	13 de marzo de 2014	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
mozillacert39.pem	13 de marzo de 2014	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
mozillacert6.pem	13 de marzo de 2014	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
mozillacert72.pem	13 de marzo de 2014	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
mozillacert136.pem	13 de marzo de 2014	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert10.pem	13 de marzo de 2014	5F:3A:FC:0A:8B:64:F6:86:67:34:74:DF:7E:A9:A2:FE:F9:FA:7A:51
mozillacert28.pem	13 de marzo de 2014	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
mozillacert61.pem	13 de marzo de 2014	E0:B4:32:2E:B2:F6:A5:68:B6:54:53:84:48:18:4A:50:36:87:43:84
mozillacert79.pem	13 de marzo de 2014	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27

Nombre	Fecha	Huella digital SHA1
mozillacert125.pem	13 de marzo de 2014	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert17.pem	13 de marzo de 2014	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
mozillacert50.pem	13 de marzo de 2014	8C:96:BA:EB:DD:2B:07:07:48:EE:30:32:66:A0:F3:98:6E:7C:AE:58
mozillacert68.pem	13 de marzo de 2014	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
mozillacert114.pem	13 de marzo de 2014	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert57.pem	13 de marzo de 2014	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
mozillacert103.pem	13 de marzo de 2014	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
mozillacert90.pem	13 de marzo de 2014	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
mozillacert46.pem	13 de marzo de 2014	40:9D:4B:D9:17:B5:5C:27:B6:9B:64:CB:98:22:44:0D:CD:09:B8:89

Nombre	Fecha	Huella digital SHA1
mozillacert97.pem	13 de marzo de 2014	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
mozillacert143.pem	13 de marzo de 2014	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
mozillacert35.pem	13 de marzo de 2014	2A:C8:D5:8B:57:CE:BF:2F:49:AF:F2:FC:76:8F:51:14:62:90:7A:41
mozillacert2.pem	13 de marzo de 2014	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
mozillacert86.pem	13 de marzo de 2014	74:2C:31:92:E6:07:E4:24:EB:45:49:54:2B:E1:BB:C5:3E:61:74:E2
mozillacert132.pem	13 de marzo de 2014	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
mozillacert24.pem	13 de marzo de 2014	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
mozillacert9.pem	13 de marzo de 2014	F4:8B:11:BF:DE:AB:BE:94:54:20:71:E6:41:DE:6B:BE:88:2B:40:B9
mozillacert75.pem	13 de marzo de 2014	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A

Nombre	Fecha	Huella digital SHA1
mozillacert121.pem	13 de marzo de 2014	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
mozillacert139.pem	13 de marzo de 2014	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
mozillacert13.pem	13 de marzo de 2014	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
mozillacert64.pem	13 de marzo de 2014	62:7F:8D:78:27:65:63:99:D2:7D:7F:90:44:C9:FE:B3:F3:3E:FA:9A
mozillacert110.pem	13 de marzo de 2014	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
mozillacert128.pem	13 de marzo de 2014	A9:E9:78:08:14:37:58:88:F2:05:19:B0:6D:2B:0D:2B:60:16:90:7D
mozillacert53.pem	13 de marzo de 2014	7F:8A:B0:CF:D0:51:87:6A:66:F3:36:0F:47:C8:8D:8C:D3:35:FC:74
mozillacert117.pem	13 de marzo de 2014	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
mozillacert150.pem	13 de marzo de 2014	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9

Nombre	Fecha	Huella digital SHA1
mozillacert42.pem	13 de marzo de 2014	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
mozillacert106.pem	13 de marzo de 2014	E7:A1:90:29:D3:D5:52:DC:0D:0F:C6:92:D3:EA:88:0D:15:2E:1A:6B
mozillacert93.pem	13 de marzo de 2014	31:F1:FD:68:22:63:20:EE:C6:3B:3F:9D:EA:4A:3E:53:7C:7C:39:17
mozillacert31.pem	13 de marzo de 2014	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
mozillacert49.pem	13 de marzo de 2014	61:57:3A:11:DF:0E:D8:7E:D5:92:65:22:EA:D0:56:D7:44:B3:23:71
mozillacert82.pem	13 de marzo de 2014	2E:14:DA:EC:28:F0:FA:1E:8E:38:9A:4E:AB:EB:26:C0:0A:D3:83:C3
mozillacert146.pem	13 de marzo de 2014	21:FC:BD:8E:7F:6C:AF:05:1B:D1:B3:43:EC:A8:E7:61:47:F2:0F:8A
mozillacert20.pem	13 de marzo de 2014	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert38.pem	13 de marzo de 2014	CB:A1:C5:F8:B0:E3:5E:B8:B9:45:12:D3:F9:34:A2:E9:06:10:D3:36

Nombre	Fecha	Huella digital SHA1
mozillacert5.pem	13 de marzo de 2014	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
mozillacert71.pem	13 de marzo de 2014	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
mozillacert89.pem	13 de marzo de 2014	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
mozillacert135.pem	13 de marzo de 2014	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
mozillacert27.pem	13 de marzo de 2014	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
mozillacert60.pem	13 de marzo de 2014	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
mozillacert78.pem	13 de marzo de 2014	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
mozillacert124.pem	13 de marzo de 2014	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert16.pem	13 de marzo de 2014	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13

Nombre	Fecha	Huella digital SHA1
mozillacert67.pem	13 de marzo de 2014	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
mozillacert113.pem	13 de marzo de 2014	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
mozillacert56.pem	13 de marzo de 2014	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
mozillacert102.pem	13 de marzo de 2014	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
mozillacert45.pem	13 de marzo de 2014	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert109.pem	13 de marzo de 2014	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
mozillacert96.pem	13 de marzo de 2014	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
mozillacert142.pem	13 de marzo de 2014	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
mozillacert34.pem	13 de marzo de 2014	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9

Nombre	Fecha	Huella digital SHA1
mozillacert1.pem	13 de marzo de 2014	23:E5:94:94:51:95:F2:41:48:03:B4:D5:64:D2:A3:A3:F5:D8:8B:8C
mozillacert85.pem	13 de marzo de 2014	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
mozillacert131.pem	13 de marzo de 2014	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
mozillacert149.pem	13 de marzo de 2014	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
mozillacert23.pem	13 de marzo de 2014	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
mozillacert8.pem	13 de marzo de 2014	3E:2B:F7:F2:03:1B:96:F3:8C:E6:C4:D8:A8:5D:3E:2D:58:47:6A:0F
mozillacert74.pem	13 de marzo de 2014	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
mozillacert120.pem	13 de marzo de 2014	DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97:FE:2F:9D:F5:B7:D1:8A:41
mozillacert138.pem	13 de marzo de 2014	E1:9F:E3:0E:8B:84:60:9E:80:9B:17:0D:72:A8:C5:BA:6E:14:09:BD

Nombre	Fecha	Huella digital SHA1
mozillacert12.pem	13 de marzo de 2014	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
mozillacert63.pem	13 de marzo de 2014	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
mozillacert127.pem	13 de marzo de 2014	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
mozillacert19.pem	13 de marzo de 2014	B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB:B3:94:BD:63:7B:A7:82:B7
mozillacert52.pem	13 de marzo de 2014	8B:AF:4C:9B:1D:F0:2A:92:F7:DA:12:8E:B9:1B:AC:F4:98:60:4B:6F
mozillacert116.pem	13 de marzo de 2014	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
mozillacert41.pem	13 de marzo de 2014	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
mozillacert59.pem	13 de marzo de 2014	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
mozillacert105.pem	13 de marzo de 2014	77:47:4F:C6:30:E4:0F:4C:47:64:3F:84:BA:B8:C6:95:4A:8A:41:EC

Nombre	Fecha	Huella digital SHA1
mozillacert92.pem	13 de marzo de 2014	A3:F1:33:3F:E2:42:BF:CF:C5:D1:4E:8F:39:42:98:40:68:10:D1:A0
mozillacert30.pem	13 de marzo de 2014	E7:B4:F6:9D:61:EC:90:69:DB:7E:90:A7:40:1A:3C:F4:7D:4F:E8:EE
mozillacert48.pem	13 de marzo de 2014	A0:A1:AB:90:C9:FC:84:7B:3B:12:61:E8:97:7D:5F:D3:22:61:D3:CC
verisignc4g2.pem	20 de marzo de 2014	0B:77:BE:BB:CB:7A:A2:47:05:DE:CC:0F:BD:6A:02:FC:7A:BD:9B:52
verisignc2g3.pem	20 de marzo de 2014	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
verisignc1g6.pem	31 de diciembre de 2014	51:7F:61:1E:29:91:6B:53:82:FB:72:E7:44:D9:8D:C3:CC:53:6D:64
verisignc2g2.pem	20 de marzo de 2014	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
verisignroot.pem	20 de marzo de 2014	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
verisignc2g1.pem	20 de marzo de 2014	67:82:AA:E0:ED:EE:E2:1A:58:39:D3:C0:CD:14:68:0A:4F:60:14:2A

Nombre	Fecha	Huella digital SHA1
verisignc3g5.pem	20 de marzo de 2014	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
verisignc1g3.pem	20 de marzo de 2014	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
verisignc3g4.pem	20 de marzo de 2014	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
verisignc1g2.pem	20 de marzo de 2014	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
verisignc3g3.pem	20 de marzo de 2014	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
verisignc1g1.pem	20 de marzo de 2014	90:AE:A2:69:85:FF:14:80:4C:43:49:52:EC:E9:60:84:77:AF:55:6F
verisignc3g2.pem	20 de marzo de 2014	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
verisignc3g1.pem	20 de marzo de 2014	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
verisignc2g6.pem	31 de diciembre de 2014	40:B3:31:A0:E9:BF:E8:55:BC:39:93:CA:70:4F:4E:C2:51:D4:1D:8F

Nombre	Fecha	Huella digital SHA1
verisignc4g3.pem	20 de marzo de 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 de diciembre de 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 de diciembre de 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 de diciembre de 2014	27:AC:93:69:FA:F2:52:07:BB:26:27:CE:FA:CC:BE:4E:F9:C3:19:B8
dstacescax6	18 de junio de 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 de junio de 2018	27:AC:93:69:FA:F2:52:07:BB:26:27:CE:FA:CC:BE:4E:F9:C3:19:B8
verisignc4g3.pem	18 de junio de 2018	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
swisssignplatinumg2ca	21 de abril de 2018	56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66
geotrustprimarycertificatio nauthorityg3	18 de junio de 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycertificatio nauthorityg2	18 de junio de 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0

Nombre	Fecha	Huella digital SHA1
buypassclass2rootca	18 de junio de 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
camerfirmachambersofcommerceroot	18 de junio de 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
mozillacert20.pem	18 de junio de 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert12.pem	18 de junio de 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
mozillacert90.pem	18 de junio de 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
mozillacert82.pem	18 de junio de 2018	2E:14:DA:EC:28:F0:FA:1E:8E:38:9A:4E:AB:EB:26:C0:0A:D3:83:C3
mozillacert140.pem	18 de junio de 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
mozillacert74.pem	18 de junio de 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
mozillacert132.pem	18 de junio de 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
mozillacert66.pem	18 de junio de 2018	DD:E1:D2:A9:01:80:2E:1D:87:5E:84:B3:80:7E:4B:B1:FD:99:41:34
mozillacert124.pem	18 de junio de 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert58.pem	18 de junio de 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
securitycommunicationrootca2	18 de junio de 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74

Nombre	Fecha	Huella digital SHA1
mozillacert116.pem	18 de junio de 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
mozillacert108.pem	18 de junio de 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
certigna	18 de junio de 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert3.pem	18 de junio de 2018	87:9F:4B:EE:05:DF:98:58:3B:E3:60:D6:33:E7:0D:3F:FE:98:71:AF
verisignc1g1.pem	18 de junio de 2018	90:AE:A2:69:85:FF:14:80:4C:43:49:52:EC:E9:60:84:77:AF:55:6F
verisignc4g2.pem	18 de junio de 2018	0B:77:BE:BB:CB:7A:A2:47:05:DE:CC:0F:BD:6A:02:FC:7A:BD:9B:52
deutschetelekomrootca2	18 de junio de 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
starfieldrootg2ca	21 de abril de 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
comodoecccertificationauthority	18 de junio de 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
digicertglobalrootg3	18 de junio de 2018	7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E
digicertglobalrootg2	18 de junio de 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
mozillacert11.pem	18 de junio de 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
mozillacert81.pem	18 de junio de 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E

Nombre	Fecha	Huella digital SHA1
mozillacert73.pem	18 de junio de 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
szafirrootca2	18 de junio de 2018	E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
mozillacert131.pem	18 de junio de 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
ecacc	18 de junio de 2018	28:90:3A:63:5B:52:80:FA:E6:77:4C:0B:6D:A7:D6:BA:A6:4A:F2:E8
mozillacert65.pem	18 de junio de 2018	69:BD:8C:F4:9C:D3:00:FB:59:2E:17:93:CA:55:6A:F3:EC:AA:35:FB
turktrustelektroniksertifikahizmetsaglayicisih5	18 de junio de 2018	C4:18:F6:4D:46:D1:DF:00:3D:27:30:13:72:43:A9:12:11:C6:75:FB
mozillacert123.pem	18 de junio de 2018	2A:B6:28:48:5E:78:FB:F3:AD:9E:79:10:DD:6B:DF:99:72:2C:96:E5
mozillacert57.pem	18 de junio de 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
mozillacert115.pem	18 de junio de 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
mozillacert49.pem	18 de junio de 2018	61:57:3A:11:DF:0E:D8:7E:D5:92:65:22:EA:D0:56:D7:44:B3:23:71
mozillacert107.pem	18 de junio de 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
verisignclass3g4ca	21 de abril de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A

Nombre	Fecha	Huella digital SHA1
securetrustca	18 de junio de 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
mozillacert2.pem	18 de junio de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
buypassclass2ca	21 de abril de 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
secomscrootca2	21 de abril de 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
secomscrootca1	21 de abril de 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
trustisfpsrootca	18 de junio de 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
hongkongpostrootca1	18 de junio de 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
certsignrootca	18 de junio de 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
geotrustprimaryca	21 de abril de 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
twcaglobalrootca	18 de junio de 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
camerfirmachambersca	21 de abril de 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
mozillacert10.pem	18 de junio de 2018	5F:3A:FC:0A:8B:64:F6:86:67:34:74:DF:7E:A9:A2:FE:F9:FA:7A:51
mozillacert80.pem	18 de junio de 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB

Nombre	Fecha	Huella digital SHA1
mozillacert72.pem	18 de junio de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
comodoaaaca	21 de abril de 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert130.pem	18 de junio de 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
mozillacert64.pem	18 de junio de 2018	62:7F:8D:78:27:65:63:99:D2:7D:7F:90:44:C9:FE:B3:F3:3E:FA:9A
mozillacert122.pem	18 de junio de 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
mozillacert56.pem	18 de junio de 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
equifaxsecureebusinessca1	21 de abril de 2018	AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
camerfirmachambersignca	21 de abril de 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
mozillacert114.pem	18 de junio de 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert48.pem	18 de junio de 2018	A0:A1:AB:90:C9:FC:84:7B:3B:12:61:E8:97:7D:5F:D3:22:61:D3:CC
pscprocert	18 de junio de 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
mozillacert106.pem	18 de junio de 2018	E7:A1:90:29:D3:D5:52:DC:0D:0F:C6:92:D3:EA:88:0D:15:2E:1A:6B
mozillacert1.pem	18 de junio de 2018	23:E5:94:94:51:95:F2:41:48:03:B4:D5:64:D2:A3:A3:F5:D8:8B:8C

Nombre	Fecha	Huella digital SHA1
eecertificationcentrerootca	18 de junio de 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
digicertglobalrootca	18 de junio de 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
thawteprimaryrootcag3	18 de junio de 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
thawteprimaryrootcag2	18 de junio de 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
entrustrootcertificationauthorityec1	18 de junio de 2018	20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
valicertclass2ca	21 de abril de 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
globalchambersignroot2008	18 de junio de 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
amazonrootca4	18 de junio de 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 de junio de 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
amazonrootca3	18 de junio de 2018	0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
amazonrootca2	18 de junio de 2018	5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
securitycommunicationrootca	18 de junio de 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7

Nombre	Fecha	Huella digital SHA1
amazonrootca1	18 de junio de 2018	8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
acraizfnmtrcm	18 de junio de 2018	EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
quovadisrootca3g3	18 de junio de 2018	48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
certplusrootcag2	18 de junio de 2018	4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A
certplusrootcag1	18 de junio de 2018	22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
mozillacert71.pem	18 de junio de 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
mozillacert63.pem	18 de junio de 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
mozillacert121.pem	18 de junio de 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
ttelesecglobalrootclass3ca	21 de abril de 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
mozillacert55.pem	18 de junio de 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
mozillacert113.pem	18 de junio de 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
baltimorecybertrustca	21 de abril de 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
mozillacert47.pem	18 de junio de 2018	1B:4B:39:61:26:27:6B:64:91:A2:68:6D:D7:02:43:21:2D:1F:1D:96

Nombre	Fecha	Huella digital SHA1
mozillacert105.pem	18 de junio de 2018	77:47:4F:C6:30:E4:0F:4C:47:64:3F:84:BA:B8:C6:95:4A:8A:41:EC
mozillacert39.pem	18 de junio de 2018	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
usertrustecccertificationauthority	18 de junio de 2018	D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
mozillacert0.pem	18 de junio de 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
securitycommunicationevrootca1	18 de junio de 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
verisignc3g5.pem	18 de junio de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
globalsignr3ca	21 de abril de 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
trustcoreca1	18 de junio de 2018	58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
equifaxsecureglobalebusinessca1	21 de abril de 2018	3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36
geotrustuniversalca	18 de junio de 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
affirmtrustpremiumca	21 de abril de 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
staatdernederlandenrootcag3	18 de junio de 2018	D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC
staatdernederlandenrootcag2	18 de junio de 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16

Nombre	Fecha	Huella digital SHA1
mozillacert70.pem	18 de junio de 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
secomevrootca1	21 de abril de 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
geotrustglobalca	18 de junio de 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
mozillacert62.pem	18 de junio de 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert120.pem	18 de junio de 2018	DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97:FE:2F:9D:F5:B7:D1:8A:41
mozillacert54.pem	18 de junio de 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
mozillacert112.pem	18 de junio de 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
mozillacert46.pem	18 de junio de 2018	40:9D:4B:D9:17:B5:5C:27:B6:9B:64:CB:98:22:44:0D:CD:09:B8:89
swisssigngoldcag2	18 de junio de 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert104.pem	18 de junio de 2018	4F:99:AA:93:FB:2B:D1:37:26:A1:99:4A:CE:7F:F0:05:F2:93:5D:1E
mozillacert38.pem	18 de junio de 2018	CB:A1:C5:F8:B0:E3:5E:B8:B9:45:12:D3:F9:34:A2:E9:06:10:D3:36
certplusclass3pprimaryca	21 de abril de 2018	21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
entrustrootcertificationauthorityg2	18 de junio de 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4

Nombre	Fecha	Huella digital SHA1
godaddyrootg2ca	21 de abril de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
cfcaevroot	18 de junio de 2018	E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
verisignc3g4.pem	18 de junio de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
geotrustuniversalca2	18 de junio de 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
starfieldservicesrootg2ca	21 de abril de 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
digicerthighassuranceevrootca	18 de junio de 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
entrustnetpremium2048secureserverca	18 de junio de 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
camerfirmaglobalchambersignroot	18 de junio de 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
verisignclass3g3ca	21 de abril de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
godaddyclass2ca	18 de junio de 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
mozillacert61.pem	18 de junio de 2018	E0:B4:32:2E:B2:F6:A5:68:B6:54:53:84:48:18:4A:50:36:87:43:84
mozillacert53.pem	18 de junio de 2018	7F:8A:B0:CF:D0:51:87:6A:66:F3:36:0F:47:C8:8D:8C:D3:35:FC:74
atostrustedroot2011	18 de junio de 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21

Nombre	Fecha	Huella digital SHA1
mozillacert111.pem	18 de junio de 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
staatdernederlandenevrootca	18 de junio de 2018	76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
mozillacert45.pem	18 de junio de 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert103.pem	18 de junio de 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
mozillacert37.pem	18 de junio de 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert29.pem	18 de junio de 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
izenpecom	18 de junio de 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
comodorsacertificationauthority	18 de junio de 2018	AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
mozillacert99.pem	18 de junio de 2018	F1:7F:6F:B6:31:DC:99:E3:A3:C8:7F:FE:1C:F1:81:10:88:D9:60:33
mozillacert149.pem	18 de junio de 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
utnuserfirstobjectca	21 de abril de 2018	E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
verisignc3g3.pem	18 de junio de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
dstrootcax3	18 de junio de 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13

Nombre	Fecha	Huella digital SHA1
addtrustexternalroot	18 de junio de 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
certumtrustednetworkca	18 de junio de 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
affirmtrustpremiumecc	18 de junio de 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
starfieldclass2ca	18 de junio de 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
actalisauthenticationrootca	18 de junio de 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
verisignclass2g3ca	21 de abril de 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
isrgrootx1	18 de junio de 2018	CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
godaddyrootcertificateauthorityg2	18 de junio de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
mozillacert60.pem	18 de junio de 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
chunghwaepkirootca	21 de abril de 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert52.pem	18 de junio de 2018	8B:AF:4C:9B:1D:F0:2A:92:F7:DA:12:8E:B9:1B:AC:F4:98:60:4B:6F
microseceszignorootca2009	18 de junio de 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
securesignrootca11	18 de junio de 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3

Nombre	Fecha	Huella digital SHA1
mozillacert110.pem	18 de junio de 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
mozillacert44.pem	18 de junio de 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
mozillacert102.pem	18 de junio de 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
mozillacert36.pem	18 de junio de 2018	23:88:C9:D3:71:CC:9E:96:3D:FF:7D:3C:A7:CE:FC:D6:25:EC:19:0D
mozillacert28.pem	18 de junio de 2018	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
baltimorecybertrustroot	18 de junio de 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
amzninternalrootca	12 de diciembre de 2008	A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
mozillacert98.pem	18 de junio de 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
mozillacert148.pem	18 de junio de 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
verisignc3g2.pem	18 de junio de 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
quovadisrootca2g3	18 de junio de 2018	09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
geotrustprimarycertificatioauthority	18 de junio de 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96

Nombre	Fecha	Huella digital SHA1
opentrustrootcag3	18 de junio de 2018	6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6
opentrustrootcag2	18 de junio de 2018	79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
opentrustrootcag1	18 de junio de 2018	79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E
verisignclass3ca	21 de abril de 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
globalsignca	21 de abril de 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
ttelesecglobalrootclass2ca	21 de abril de 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
verisignclass1g3ca	21 de abril de 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
verisignuniversalrootca	21 de abril de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
soneraclass2ca	21 de abril de 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
starfieldservicesrootcertificateauthorityg2	18 de junio de 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
mozillacert51.pem	18 de junio de 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
mozillacert43.pem	18 de junio de 2018	F9:CD:0E:2C:DA:76:24:C1:8F:BD:F0:F0:AB:B6:45:B8:F7:FE:D5:7A

Nombre	Fecha	Huella digital SHA1
mozillacert101.pem	18 de junio de 2018	99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39
mozillacert35.pem	18 de junio de 2018	2A:C8:D5:8B:57:CE:BF:2F:49:AF:F2:FC:76:8F:51:14:62:90:7A:41
globalsignr2ca	21 de abril de 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
mozillacert27.pem	18 de junio de 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
affirmtrustpremium	18 de junio de 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
mozillacert19.pem	18 de junio de 2018	B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB:B3:94:BD:63:7B:A7:82:B7
mozillacert97.pem	18 de junio de 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
netlockaranyclassgoldfotanusitvany	18 de junio de 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
mozillacert89.pem	18 de junio de 2018	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
verisignroot.pem	18 de junio de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
mozillacert147.pem	18 de junio de 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
aolrootca2	21 de abril de 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84

Nombre	Fecha	Huella digital SHA1
cia-crt-g3-01-ca	23 de noviembre de 2016	2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
aolrootca1	21 de abril de 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
verisignc3g1.pem	18 de junio de 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert139.pem	18 de junio de 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
soneraclass2rootca	18 de junio de 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
swisssignsilverg2ca	21 de abril de 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
thawteprimaryrootca	18 de junio de 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
gdcatrustauthr5root	18 de junio de 2018	0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
trustcenterclass4caii	21 de abril de 2018	A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
usertrustrsacertificationauthority	18 de junio de 2018	2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
digicertassuredidrootg3	18 de junio de 2018	F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
digicertassuredidrootg2	18 de junio de 2018	A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F

Nombre	Fecha	Huella digital SHA1
mozillacert50.pem	18 de junio de 2018	8C:96:BA:EB:DD:2B:07:07:48:EE:30:32:66:A0:F3:98:6E:7C:AE:58
mozillacert42.pem	18 de junio de 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
mozillacert100.pem	18 de junio de 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
mozillacert34.pem	18 de junio de 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
affirmtrustcommercialca	21 de abril de 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert26.pem	18 de junio de 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
globalsigneccrootcar5	18 de junio de 2018	1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
globalsigneccrootcar4	18 de junio de 2018	69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
buypassclass3rootca	18 de junio de 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
mozillacert18.pem	18 de junio de 2018	79:98:A3:08:E1:4D:65:85:E6:C2:1E:15:3A:71:9F:BA:5A:D3:4A:D9
mozillacert96.pem	18 de junio de 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
verisignc2g6.pem	18 de junio de 2018	40:B3:31:A0:E9:BF:E8:55:BC:39:93:CA:70:4F:4E:C2:51:D4:1D:8F
secomvalicertclass1ca	21 de abril de 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E

Nombre	Fecha	Huella digital SHA1
mozillacert88.pem	18 de junio de 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
accvraiz1	18 de junio de 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
mozillacert146.pem	18 de junio de 2018	21:FC:BD:8E:7F:6C:AF:05:1B:D1:B3:43:EC:A8:E7:61:47:F2:0F:8A
mozillacert138.pem	18 de junio de 2018	E1:9F:E3:0E:8B:84:60:9E:80:9B:17:0D:72:A8:C5:BA:6E:14:09:BD
verisignclass3g2ca	21 de abril de 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
dtrustrootclass3ca2ev2009	18 de junio de 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
xrampglobalca	21 de abril de 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
mozillacert9.pem	18 de junio de 2018	F4:8B:11:BF:DE:AB:BE:94:54:20:71:E6:41:DE:6B:BE:88:2B:40:B9
verisignuniversalrootcertificationauthority	18 de junio de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
tubitakkamusmsslkoksertifikasisurum1	18 de junio de 2018	31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA
mozillacert41.pem	18 de junio de 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
mozillacert33.pem	18 de junio de 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D

Nombre	Fecha	Huella digital SHA1
mozillacert25.pem	18 de junio de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
mozillacert17.pem	18 de junio de 2018	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
mozillacert95.pem	18 de junio de 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
affirmtrustpremiumeccca	21 de abril de 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
mozillacert87.pem	18 de junio de 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
mozillacert145.pem	18 de junio de 2018	10:1D:FA:3F:D5:0B:CB:BB:9B:B5:60:0C:19:55:A4:1A:F4:73:3A:04
mozillacert79.pem	18 de junio de 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
mozillacert137.pem	18 de junio de 2018	4A:65:D5:F4:1D:EF:39:B8:B8:90:4A:4A:D3:64:81:33:CF:C7:A1:D1
digicertassuredidrootca	18 de junio de 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
addtrustqualifiedca	21 de abril de 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert129.pem	18 de junio de 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
verisignclass2g2ca	21 de abril de 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
baltimorecodesigningca	21 de abril de 2018	30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD:49:27:08:7C:60:56:7B:0D

Nombre	Fecha	Huella digital SHA1
luxtrustglobalroot2	18 de junio de 2018	1E:0E:56:19:0A:D1:8B:25:98:B2:04:44:FF:66:8A:04:17:99:5F:3F
visaecommerceroot	18 de junio de 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
oistewisekeyglobalrootgbca	18 de junio de 2018	0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
mozillacert8.pem	18 de junio de 2018	3E:2B:F7:F2:03:1B:96:F3:8C:E6:C4:D8:A8:5D:3E:2D:58:47:6A:0F
comodocertificationauthority	18 de junio de 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 de noviembre de 2016	96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09
verisignc1g6.pem	18 de junio de 2018	51:7F:61:1E:29:91:6B:53:82:FB:72:E7:44:D9:8D:C3:CC:53:6D:64
trustcenterclass2caii	21 de abril de 2018	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
quovadisrootca1g3	18 de junio de 2018	1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67
mozillacert40.pem	18 de junio de 2018	80:25:EF:F4:6E:70:C8:D4:72:24:65:84:FE:40:3B:8A:8D:6A:DB:F5
cadisigrootr2	18 de junio de 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
cadisigrootr1	18 de junio de 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6

Nombre	Fecha	Huella digital SHA1
mozillacert32.pem	18 de junio de 2018	60:D6:89:74:B5:C2:65:9E:8A:0F:C1:88:7C:88:D2:46:69:1B:18:2C
utndatacorpsgcca	21 de abril de 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
mozillacert24.pem	18 de junio de 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
addtrustclass1ca	21 de abril de 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
mozillacert16.pem	18 de junio de 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
affirmtrustnetworkingca	21 de abril de 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
mozillacert94.pem	18 de junio de 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
mozillacert86.pem	18 de junio de 2018	74:2C:31:92:E6:07:E4:24:EB:45:49:54:2B:E1:BB:C5:3E:61:74:E2
mozillacert144.pem	18 de junio de 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
mozillacert78.pem	18 de junio de 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
mozillacert136.pem	18 de junio de 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert128.pem	18 de junio de 2018	A9:E9:78:08:14:37:58:88:F2:05:19:B0:6D:2B:0D:2B:60:16:90:7D
verisignclass1g2ca	21 de abril de 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47

Nombre	Fecha	Huella digital SHA1
hellenicacademicandresearchinstitutionsrootca2015	18 de junio de 2018	01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
soneraclass1ca	21 de abril de 2018	07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
hellenicacademicandresearchinstitutionsrootca2011	18 de junio de 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
certumtrustednetworkca2	18 de junio de 2018	D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
equifaxsecureca	21 de abril de 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
thawteserverca	21 de abril de 2018	9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
mozillacert7.pem	18 de junio de 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
affirmtrustnetworking	18 de junio de 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
deprecateditsecca	27 de enero de 2012	12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
globalsignrootcar3	18 de junio de 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
globalsignrootcar2	18 de junio de 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE

Nombre	Fecha	Huella digital SHA1
quovadisrootca	18 de junio de 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
mozillacert31.pem	18 de junio de 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
entrustrootcertificationauthority	18 de junio de 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert23.pem	18 de junio de 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
mozillacert15.pem	18 de junio de 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
verisignc2g3.pem	18 de junio de 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
mozillacert93.pem	18 de junio de 2018	31:F1:FD:68:22:63:20:EE:C6:3B:3F:9D:EA:4A:3E:53:7C:7C:39:17
mozillacert151.pem	18 de junio de 2018	AC:ED:5F:65:53:FD:25:CE:01:5F:1F:7A:48:3B:6A:74:9F:61:78:C6
mozillacert85.pem	18 de junio de 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
certplusclass2primaryca	18 de junio de 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
mozillacert143.pem	18 de junio de 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
mozillacert77.pem	18 de junio de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert135.pem	18 de junio de 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18

Nombre	Fecha	Huella digital SHA1
mozillacert69.pem	18 de junio de 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
mozillacert127.pem	18 de junio de 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
mozillacert119.pem	18 de junio de 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
geotrustprimarycag3	21 de abril de 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
identrustpublicsectorrootca1	18 de junio de 2018	BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
geotrustprimarycag2	21 de abril de 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
trustcorrootcertca2	18 de junio de 2018	B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
mozillacert6.pem	18 de junio de 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
trustcorrootcertca1	18 de junio de 2018	FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
networksolutionscertificate authority	18 de junio de 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
twcarootcertificationauthority	18 de junio de 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
addtrustexternalca	21 de abril de 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68

Nombre	Fecha	Huella digital SHA1
verisignclass3g5ca	21 de abril de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
autoridaddecertificacionfirmaprofesionalcifa62634068	18 de junio de 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
hellenicacademicandresearchinstitutionseccrootca2015	18 de junio de 2018	9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
verisigntsaca	21 de abril de 2018	20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
utnuserfirsthardwareca	21 de abril de 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
identrustcommercialrootca1	18 de junio de 2018	DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
dtrustrootclass3ca22009	18 de junio de 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
epkirootcertificationauthority	18 de junio de 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert30.pem	18 de junio de 2018	E7:B4:F6:9D:61:EC:90:69:DB:7E:90:A7:40:1A:3C:F4:7D:4F:E8:EE
teliasonerarootcav1	18 de junio de 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
buypassclass3ca	21 de abril de 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57

Nombre	Fecha	Huella digital SHA1
mozillacert22.pem	18 de junio de 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
mozillacert14.pem	18 de junio de 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
verisignc2g2.pem	18 de junio de 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
certumca	21 de abril de 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
mozillacert92.pem	18 de junio de 2018	A3:F1:33:3F:E2:42:BF:CF:C5:D1:4E:8F:39:42:98:40:68:10:D1:A0
mozillacert150.pem	18 de junio de 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
mozillacert84.pem	18 de junio de 2018	D3:C0:63:F2:19:ED:07:3E:34:AD:5D:75:0B:32:76:29:FF:D5:9A:F2
ttelesecglobalroot class3	18 de junio de 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
globalsignrootca	18 de junio de 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
ttelesecglobalroot class2	18 de junio de 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
mozillacert142.pem	18 de junio de 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
mozillacert76.pem	18 de junio de 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert134.pem	18 de junio de 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62

Nombre	Fecha	Huella digital SHA1
mozillacert68.pem	18 de junio de 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
etugracertificationauthority	18 de junio de 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert126.pem	18 de junio de 2018	25:01:90:19:CF:FB:D9:99:1C:B7:68:25:74:8D:94:5F:30:93:95:42
keynectisrootca	21 de abril de 2018	9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
mozillacert118.pem	18 de junio de 2018	7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
quovadisrootca3	18 de junio de 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
quovadisrootca2	18 de junio de 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
mozillacert5.pem	18 de junio de 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
verisignc1g3.pem	18 de junio de 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
cybertrustglobalroot	18 de junio de 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
amzninternalinfosecag3	27 de febrero de 2015	B9:B1:CA:38:F7:BF:9C:D2:D4:95:E7:B6:5E:75:32:9B:A8:78:2E:F6
starfieldrootcertificateauthorityg2	18 de junio de 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E

Nombre	Fecha	Huella digital SHA1
entrust2048ca	21 de abril de 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
swisssignsilvercag2	18 de junio de 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
affirmtrustcommercial	18 de junio de 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
certinomisrootca	18 de junio de 2018	9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C:01:B9:32:C5:34:E7:88:A8
xrampglobalcaroot	18 de junio de 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
secureglobalca	18 de junio de 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
swisssigngoldg2ca	21 de abril de 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert21.pem	18 de junio de 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
mozillacert13.pem	18 de junio de 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
verisignc2g1.pem	18 de junio de 2018	67:82:AA:E0:ED:EE:E2:1A:58:39:D3:C0:CD:14:68:0A:4F:60:14:2A
mozillacert91.pem	18 de junio de 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
oistewisekeyglobalrootgaca	18 de junio de 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
mozillacert83.pem	18 de junio de 2018	A0:73:E5:C5:BD:43:61:0D:86:4C:21:13:0A:85:58:57:CC:9C:EA:46

Nombre	Fecha	Huella digital SHA1
entrustevca	21 de abril de 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert141.pem	18 de junio de 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
mozillacert75.pem	18 de junio de 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
mozillacert133.pem	18 de junio de 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
mozillacert67.pem	18 de junio de 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
mozillacert125.pem	18 de junio de 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert59.pem	18 de junio de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
thawtepremiumserverca	21 de abril de 2018	E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66
mozillacert117.pem	18 de junio de 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
utnuserfirstclientauthemailca	21 de abril de 2018	B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
entrustrootcag2	21 de abril de 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
mozillacert109.pem	18 de junio de 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
digicerttrustedrootg4	18 de junio de 2018	DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F:C8:3A:4D:7D:77:5D:05:E4

Nombre	Fecha	Huella digital SHA1
gdroot-g2.pem	18 de junio de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
comodoaaaservicesroot	18 de junio de 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert4.pem	18 de junio de 2018	E3:92:51:2F:0A:CF:F5:05:DF:F6:DE:06:7F:75:37:E1:65:EA:57:4B
verisignclass3publicprimary certificationauthorityg5	18 de junio de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
chambersofcommerceroot2008	18 de junio de 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
verisignclass3publicprimary certificationauthorityg4	18 de junio de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
verisignclass3publicprimary certificationauthorityg3	18 de junio de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
thawtepersonalfreemailca	21 de abril de 2018	E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
verisignc1g2.pem	18 de junio de 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
gtecybertrustglobalca	21 de abril de 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74

Nombre	Fecha	Huella digital SHA1
`trustcenteruniversalcai`	21 de abril de 2018	`6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3`
`camerfirmachamberscommerceca`	21 de abril de 2018	`6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1`
`verisignclass1ca`	21 de abril de 2018	`CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45:3E:64:09:EA:E8:7D:60:F1`

Registro de cambios de plantillas de mapeo de solucionador

> **Note**
>
> Ahora admitimos de forma básica el tiempo de ejecución APPSYNC_JS y su documentación. Considere la opción de utilizar el tiempo de ejecución APPSYNC_JS y sus guías aquí.

El solucionador y las plantillas de mapeo de función tienen varias versiones. La versión de la plantilla de mapeo (por ejemplo, `2018-05-29`) establece lo siguiente: *La forma esperada de la configuración de solicitud de origen de datos proporcionada por la plantilla de solicitud. *El comportamiento de ejecución de la plantilla de mapeo de solicitud y la plantilla de asignación de respuesta.

Las versiones se representan con el formato AAAA-MM-DD, una fecha posterior corresponde a una versión más reciente. En esta página se muestran las diferencias entre las versiones de la plantilla de mapeo admitidas actualmente en AWS AppSync.

Temas

- Disponibilidad de operación de un origen de datos por matriz de la versión
- Cambio de la versión en una plantilla de mapeo de solucionador de unidad
- Cambio de la versión en una función
- 2018-05-29
- 2017-02-28

Disponibilidad de operación de un origen de datos por matriz de la versión

Operación/versión admitida	2017-02-28	2018-05-29
AWS Lambda Invoke	Sí	Sí
AWS Lambda BatchInvoke	Sí	Sí
None Datasource	Sí	Sí
Amazon OpenSearch GET	Sí	Sí
Amazon OpenSearch POST	Sí	Sí
Amazon OpenSearch PUT	Sí	Sí
Amazon OpenSearch DELETE	Sí	Sí
Amazon OpenSearch GET	Sí	Sí
DynamoDB GetItem	Sí	Sí
DynamoDB Scan	Sí	Sí
DynamoDB Query	Sí	Sí
DynamoDB DeleteItem	Sí	Sí
DynamoDB PutItem	Sí	Sí
DynamoDB BatchGetItem	No	Sí
DynamoDB BatchPutItem	No	Sí
DynamoDB BatchDeleteItem	No	Sí
HTTP	No	Sí
Amazon RDS	No	Sí

Nota: Solo la versión 2018-05-29 es compatible actualmente en las funciones.

Cambio de la versión en una plantilla de mapeo de solucionador de unidad

Para los solucionadores de unidad, la versión se especifica como parte del cuerpo de la plantilla de mapeo de solicitud. Para actualizar la versión, solo tiene que actualizar el campo `version` a la nueva versión.

Por ejemplo, para actualizar la versión en la plantilla de AWS Lambda:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments": $utils.toJson($context.arguments)
    }
}
```

Debe actualizar el campo de versión de `2017-02-28` a `2018-05-29`, tal y como se indica a continuación:

```
{
    "version": "2018-05-29", ## Note the version
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments": $utils.toJson($context.arguments)
    }
}
```

Cambio de la versión en una función

Para funciones, la versión se especifica como el campo `functionVersion` en el objeto de función. Para actualizar la versión, basta con actualizar `functionVersion`. Nota: Actualmente, solo `2018-05-29` es compatible con las funciones.

Lo siguiente es un ejemplo de comando CLI para actualizar una versión de función existente:

```
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)"
```

Nota: Se recomienda omitir el campo de la versión de la plantilla de mapeo de solicitudes de función, ya que no será atendido. Si especifica una versión dentro de una plantilla de mapeo de solicitudes de función, el valor de la versión se anulará por el valor del campo `functionVersion`.

2018-05-29

Cambio de comportamiento

- Si el resultado de la invocación del origen de datos es `null`, la plantilla de mapeo de respuesta se ejecuta.
- Si la invocación de origen de datos genera un error, le corresponde a usted gestionar el error, el resultado evaluado de la plantilla de mapeo de respuesta siempre se colocará dentro del bloque `data` de la respuesta GraphQL.

Razonamiento

- Un resultado de invocación `null` tiene un significado y en algunos casos de uso de aplicaciones es posible que desee gestionar los resultados `null` de un modo personalizado. Por ejemplo, una aplicación podría comprobar si existe un registro en una tabla de Amazon DynamoDB para realizar alguna comprobación de autorización. En este caso, un resultado de invocación `null` supondría que el usuario podría no estar autorizado. Ejecutar la plantilla de mapeo de respuesta ahora ofrece la posibilidad de generar un error no autorizado. Este comportamiento proporciona un mayor control al diseñador de la API.

En el caso de la siguiente plantilla de mapeo de respuesta:

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

Anteriormente con `2017-02-28`, si `$ctx.result` volvía nulo, la plantilla de mapeo de respuesta no se ejecutaba. Con `2018-05-29`, ya podemos gestionar este escenario. Por ejemplo, podemos optar por generar un error de autorización tal como se indica a continuación:

```
# throw an unauthorized error if the result is null
```

```
#if ( $util.isNull($ctx.result) )
    $util.unauthorized()
#end
$util.toJson($ctx.result)
```

Nota: Los errores que vuelven de un origen de datos en ocasiones no son graves o incluso previstos, por eso a la plantilla de mapeo de respuesta se le debe dar la flexibilidad de gestionar el error de invocación y decidir si pasarlo por alto, volver a generarlo o generar un error diferente.

En el caso de la siguiente plantilla de mapeo de respuesta:

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

Anteriormente, con `2017-02-28`, en caso de que se produzca un error de invocación, la plantilla de mapeo de respuesta se evaluaba y el resultado se colocaba automáticamente en el bloque `errors` de la respuesta GraphQL. Con `2018-05-29`, ya podemos elegir qué hacer con el error, volver a generarlo, generar otro error o añadir el error al devolver los datos.

Volver a generar un error de invocación

En la siguiente plantilla de respuesta, generamos el mismo error que volvió del origen de datos.

```
#if ( $ctx.error )
    $util.error($ctx.error.message, $ctx.error.type)
#end
$util.toJson($ctx.result)
```

En caso de un error de invocación (por ejemplo, si `$ctx.error` está presente), la respuesta es como la del siguiente ejemplo:

```
{
    "data": {
        "getPost": null
    },
    "errors": [
        {
            "path": [
                "getPost"
            ],
            "errorType": "DynamoDB:ConditionalCheckFailedException",
            "message": "Conditional check failed exception...",
```

```
            "locations": [
                {
                    "line": 5,
                    "column": 5
                }
            ]
        }
    ]
}
```

Generar un error diferente

En la siguiente plantilla de respuesta, generamos nuestro propio error personalizado después de procesar error que volvió del origen de datos.

```
#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)
```

En caso de un error de invocación (por ejemplo, si `$ctx.error` está presente), la respuesta es como la del siguiente ejemplo:

```
{
    "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
                }
            ]
        }
    ]
}
```

Añadir un error a los datos de retorno

En la siguiente plantilla de respuesta, añadimos el mismo error que volvió del origen de datos al devolver los datos dentro de la respuesta. Esto también se conoce como una respuesta parcial.

```
#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
```

En caso de un error de invocación (por ejemplo, si `$ctx.error` está presente), la respuesta es como la del siguiente ejemplo:

```
{
    "data": {
        "getPost": {
            "id": "1",
            "title: "A post"
        }
    },
    "errors": [
        {
            "path": [
                "getPost"
            ],
            "errorType": "ConditionalCheckFailedException",
            "message": "Conditional check failed exception...",
            "locations": [
                {
```

```
                    "line": 5,
                    "column": 5
                }
            ]
        }
    ]
}
```

Migración de 2017-02-28 a 2018-05-29

La migración de 2017-02-28 a 2018-05-29 es sencilla. Cambie el campo de versión en la plantilla de mapeo de solicitudes de solucionador o en el objeto de la versión de función. No obstante, tenga en cuenta que la ejecución de 2018-05-29 tiene un comportamiento diferente al de 2017-02-28; los cambios se describen aquí.

Mantener el mismo comportamiento de ejecución de 2017-02-28 en 2018-05-29

En algunos casos, es posible mantener el mismo comportamiento de ejecución que en la versión 2017-02-28 al ejecutar una plantilla con la versión 2018-05-29.

Ejemplo: DynamoDB PutItem

Con la siguiente plantilla de solicitud DynamoDB PutItem 2017-02-28:

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key": {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "attributeValues" : {
        "baz" : ... typed value
    },
    "condition" : {
       ...
    }
}
```

Y la siguiente plantilla de respuesta:

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

La migración a 2018-05-29 estas plantillas tal y como se indica a continuación:

```
{
    "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" : {
       ...
    }
}
```

Y cambia la plantilla de respuesta tal y como se indica a continuación:

```
## 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)
```

Ahora que es su responsabilidad gestionar errores, optamos por generar el mismo error utilizando `$util.error()` que se obtuvo de DynamoDB. Puede adaptar este fragmento de código para convertir su plantilla de mapeo a 2018-05-29; tenga en cuenta que si su plantilla de respuesta es diferente tendrá que tener en cuenta los cambios del comportamiento de ejecución.

Ejemplo: DynamoDB GetItem

Con la siguiente plantilla de solicitud de DynamoDB GetItem 2017-02-28:

```
{
```

```
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "consistentRead" : true
}
```

Y la siguiente plantilla de respuesta:

```
## map table attribute postId to field Post.id
$util.qr($ctx.result.put("id", $ctx.result.get("postId")))

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

La migración a 2018-05-29 estas plantillas tal y como se indica a continuación:

```
{
    "version" : "2018-05-29", ## Note the new 2018-05-29 version
    "operation" : "GetItem",
    "key" : {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "consistentRead" : true
}
```

Y cambia la plantilla de respuesta tal y como se indica a continuación:

```
## 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)
```

En la versión 2017-02-28, si la invocación del origen de datos fuera `null`, lo que significa que no hay ningún elemento en la tabla de DynamoDB que coincida con la clave, la plantilla de mapeo de respuesta no se ejecutaría. Podría estar bien para la mayoría de los casos, pero si espera que `$ctx.result` no sea `null`, ahora tiene que gestionar dicho escenario.

2017-02-28

Características

- Si el resultado de la invocación del origen de datos es `null`, la plantilla de mapeo de respuesta no se ejecuta.
- Si la invocación de origen de datos genera un error, se ejecuta la plantilla de mapeo de respuesta y el resultado evaluado se coloca dentro del bloque `errors.data` de la respuesta GraphQL.

Referencia de tipos

Esta sección se utiliza como referencia de los tipos de esquemas.

Tipos escalares en AWS AppSync

Un tipo de objeto de GraphQL tiene un nombre y campos, y esos campos pueden tener subcampos. En última instancia, los campos de un tipo de objeto deben resolverse en tipos escalares, que representan las hojas de la consulta. Para obtener más información sobre los tipos de objetos y los escalares, consulte Schemas and types en el sitio web de GraphQL.

Además del conjunto predeterminado de escalares de GraphQL, AWS AppSync también le permite usar escalares definidos por el servicio que comienzan con el prefijo AWS. AWS AppSync no admite la creación de escalares definidos por el usuario (personalizados). Debe usar los escalares predeterminados o de AWS.

No puede utilizar AWS como prefijo para tipos de objetos personalizados.

La siguiente sección es una referencia para de los tipos de esquemas.

Escalares predeterminados

GraphQL define los siguientes escalares predeterminados:

Lista de escalares predeterminados

`ID`

Identificador único de un objeto. Este escalar está serializado como una `String` pero no está destinado a ser legible por humanos.

`String`

Secuencia de caracteres UTF-8.

`Int`

Valor entero entre $-(2^{31})$ y $2^{31}-1$.

`Float`

Valor de coma flotante según la norma IEEE 754.

`Boolean`

Un valor booleano, ya sea `true` o `false`.

Escalares AWS AppSync

AWS AppSync define los siguientes escalares:

Lista de escalares de AWS AppSync

`AWSDate`

Cadena de fecha ISO 8601 extendida en el formato `YYYY-MM-DD`.

`AWSTime`

Cadena de hora ISO 8601 extendida en el formato `hh:mm:ss.sss`.

`AWSDateTime`

Cadena de fecha y hora ISO 8601 extendida en el formato `YYYY-MM-DDThh:mm:ss.sssZ`.

> **Note**
>
> Los escalares `AWSDate`, `AWSTime` y `AWSDateTime` pueden incluir opcionalmente un desfase de zona horaria. Por ejemplo, los valores `1970-01-01Z`, `1970-01-01-07:00` y `1970-01-01+05:30` son todos válidos para `AWSDate`. El desfase de zona horaria debe ser `Z` (UTC) o un desfase en horas y minutos (y, opcionalmente, en segundos). Por ejemplo, `±hh:mm:ss`. El campo de segundos del desfase horario se considera válido aunque no forme parte de la norma ISO 8601.

`AWSTimestamp`

Valor entero que representa el número de segundos anteriores o posteriores a `1970-01-01-T00:00Z`.

`AWSEmail`

Dirección de correo electrónico con el formato `local-part@domain-part` definido en el RFC 822.

`AWSJSON`

Cadena JSON. Cualquier construcción JSON válida se analiza y carga automáticamente en el código de resolución como mapas, listas o valores escalares en lugar de como cadenas de entrada de literales. Las cadenas sin comillas o un JSON no válido provocan un error de validación de GraphQL.

`AWSPhone`

Número de teléfono. Este valor se almacena como una cadena. Los números de teléfono pueden contener espacios o guiones para separar grupos de dígitos. Se supone que los números de teléfono sin código de país son números de Estados Unidos o Norteamérica que cumplen el plan de numeración de Norteamérica (NANP).

`AWSURL`

URL tal como se define en el RFC 1738. Por ejemplo, `https://www.amazon.com/dp/B000NZW3KC/` o `mailto:example@example.com`. Las direcciones URL deben contener un esquema (`http`, `mailto`) y no pueden contener dos barras diagonales (`//`) en la parte de la ruta.

`AWSIPAddress`

Dirección IPv4 o IPv6 válida. Las direcciones IPv4 se esperan en la notación de puntos cuádruples (`123.12.34.56`). Se espera que las direcciones IPv6 estén en un formato sin corchetes y separadas por dos puntos (`1a2b:3c4b::1234:4567`). Puede incluir un sufijo CIDR opcional (`123.45.67.89/16`) para indicar la máscara de subred.

Ejemplo de uso de esquema

En el siguiente ejemplo de esquema, GraphQL utiliza todos los escalares personalizados como un "objeto" y muestra las plantillas de solicitud y respuesta del solucionador para las operaciones básicas put, get y list. Por último, en el ejemplo se muestra cómo se puede utilizar esta información al ejecutar consultas y mutaciones.

```
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
}
```

Este es el aspecto que podría tener una plantilla de solicitud de `putObject`. Un `putObject` utiliza una operación `PutItem` para crear o actualizar un elemento en la tabla de Amazon DynamoDB. Tenga en cuenta que este fragmento de código no tiene una tabla de Amazon DynamoDB configurada como origen de datos. Se utiliza únicamente como ejemplo:

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

```
}
```

La plantilla de respuesta para `putObject` devuelve los resultados:

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

Este es el aspecto que podría tener una plantilla de solicitud de `getObject`. Un `getObject` utiliza una operación `GetItem` para devolver un conjunto de atributos del elemento dada la clave principal. Tenga en cuenta que este fragmento de código no tiene una tabla de Amazon DynamoDB configurada como origen de datos. Se utiliza únicamente como ejemplo:

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

La plantilla de respuesta para `getObject` devuelve los resultados:

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

Este es el aspecto que podría tener una plantilla de solicitud de `listObjects`. Un `listObjects` utiliza una operación `Scan` para devolver uno o más elementos y atributos. Tenga en cuenta que este fragmento de código no tiene una tabla de Amazon DynamoDB configurada como origen de datos. Se utiliza únicamente como ejemplo:

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

La plantilla de respuesta para `listObjects` devuelve los resultados:

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

Estos son algunos ejemplos de uso de este esquema con consultas de 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
    }
}
```

Interfaces y uniones en GraphQL

El sistema de tipos de GraphQL admite las interfaces. Una interfaz expone un determinado conjunto de campos que un tipo debe incluir para implementar la interfaz.

El sistema de tipos GraphQL también admite las uniones. Las uniones son idénticas a las interfaces, con la excepción de que no definen un conjunto de campos común. En general, las uniones suelen ser la opción preferida frente a las interfaces cuando los tipos posibles no comparten una jerarquía lógica.

La siguiente sección es una referencia de los tipos de esquemas.

Ejemplos de interfaces

Podríamos representar una interfaz `Event` que represente cualquier tipo de actividad o reunión de personas. Algunos tipos de eventos posibles son `Concert`, `Conference` y `Festival`. Todos estos tipos comparten características comunes, incluidos un nombre, un lugar donde se celebra el evento, así como una fecha de inicio y de fin. Pero también tienen diferencias. Por ejemplo, una `Conference` incluye una lista de ponentes y talleres mientras que un `Concert` incluye una banda.

En el lenguaje de definición de esquema (SDL), la interfaz `Event` se define de la manera siguiente:

```
interface Event {
        id: ID!
        name : String!
        startsAt: String
        endsAt: String
        venue: Venue
        minAgeRestriction: Int
}
```

Y cada tipo implementa la interfaz `Event` del modo siguiente:

```
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]
}
```

Las interfaces son útiles para representar elementos, que pueden ser de varios tipos. Por ejemplo, podríamos buscar todos los eventos que ocurran en una ubicación específica. Agreguemos un campo `findEventsByVenue` al esquema de la siguiente manera:

```
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]
}
```

`findEventsByVenue` devuelve una lista de `Event`. Puesto que los campos de la interfaz GraphQL son comunes a todos los tipos de implementación, se puede seleccionar cualquier campo en la interfaz (`Event`, `id`, `name`, `startsAt`, `endsAt`, `venue` y `minAgeRestriction`). Además,

puede obtener acceso a los campos en cualquier tipo de implementación utilizando fragmentos de GraphQL, siempre que especifique el tipo.

Observemos un ejemplo de consulta de GraphQL que utiliza la interfaz.

```
query {
  findEventsAtVenue(venueId: "Madison Square Garden") {
    id
    name
    minAgeRestriction
    startsAt

    ... on Festival {
      performers
    }

    ... on Concert {
      performingBand
    }

    ... on Conference {
      speakers
      workshops
    }
  }
}
```

La consulta anterior da una sola lista de resultados y el servidor podría ordenar los eventos por fecha de inicio de forma predeterminada.

```
{
  "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"
        ]
      }
    ]
  }
}
```

Dado que los resultados se devuelven como un conjunto único de eventos, el uso de interfaces para representar características comunes resulta muy útil para ordenar los resultados.

Ejemplos de uniones

Como se indicó anteriormente, las uniones no definen conjuntos de campos comunes. Un resultado de búsqueda puede representar muchos tipos diferentes. Con el esquema `Event`, puede definir una unión `SearchResult` de la siguiente manera:

```
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
```

En este caso, para consultar cualquier campo en nuestra unión `SearchResult`, debe utilizar fragmentos.

```
query {
  search(query: "Madison") {
    ... on Venue {
      id
      name
      address
    }

    ... on Festival {
      id
      name
      performers
    }

    ... on Concert {
      id
      name
      performingBand
    }

    ... on Conference {
      speakers
      workshops
    }
  }
}
```

Resolución Type en AWS AppSync

La resolución Type es el mecanismo por el que el motor GraphQL identifica un valor resuelto como un tipo de objeto determinado.

Volvamos al ejemplo de búsqueda de unión. Siempre y cuando nuestra consulta haya dado resultados, cada elemento de la lista de resultados debe presentarse como uno de los tipos posibles que ha definido la unión `SearchResult` (es decir, `Conference`, `Festival`, `Concert` o `Venue`).

Dado que la lógica para identificar un `Festival` a partir de un `Venue` o una `Conference` depende de los requisitos de la aplicación, debemos darle una pista al motor de GraphQL para que identifique nuestros tipos posibles a partir de los resultados sin procesar.

Con AWS AppSync, esta pista se representa con un metacampo llamado `__typename`, cuyo valor se corresponde con el nombre del tipo de objeto identificado. `__typename` es necesario para los tipos de retorno que son interfaces o uniones.

Ejemplo de resolución Type

Utilicemos el esquema anterior. Puede seguir el ejemplo navegando a la consola y añadiendo los siguientes datos en la página Schema (Esquema):

```
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
}
```

A continuación, asociaremos un solucionador al campo `Query.search`. En la sección `Resolvers`, elija Asociar solucionador, cree un nuevo origen de datos del tipo NONE y, a continuación, asígnele el nombre StubDataSource. En este ejemplo, vamos a imaginar que hemos tomado los resultados de una fuente externa y que hemos codificado los resultados obtenidos en la plantilla de mapeo de solicitud.

En el panel de la plantilla de mapeo de solicitud, escriba lo siguiente:

```
{
    "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"]
        }
    ]
}
```

Si la aplicación devuelve el nombre del tipo como parte del campo `id`, la lógica de resolución de tipos debe analizar el campo `id` para extraer el nombre del tipo y, a continuación, añadir el campo `__typename` a cada uno de los resultados. Puede aplicar dicha lógica en la plantilla de mapeo de respuesta de la siguiente manera:

> **Note**
>
> También puede realizar esta tarea como parte de su función de Lambda, si usa el origen de datos de 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)
```

Ejecute la siguiente consulta:

```
query {
  search(query: "Madison") {
    ... on Venue {
      id
      name
      address
    }

    ... on Festival {
        id
      name
      performers
    }

    ... on Concert {
      id
      name
      performingBand
    }

    ... on Conference {
      speakers
      workshops
    }
  }
}
```

La consulta genera los siguientes resultados:

```
{
  "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"
        ]
      }
    ]
  }
}
```

La lógica de la resolución Type varía en función de cada aplicación. Por ejemplo, podría tener una lógica de identificación distinta que compruebe la existencia de determinados campos o incluso una combinación de campos. Es decir, podría detectar la presencia del campo `performers` para identificar un `Festival` o la combinación de los campos `speakers` y `workshops` para identificar una `Conference`. En definitiva, le corresponde a usted definir la lógica que desea usar.

Solución de problemas y errores comunes

En esta sección se explican algunos errores comunes y cómo solucionarlos.

Mapeo de clave de DynamoDB incorrecto

Si una operación de GraphQL devuelve el siguiente mensaje de error, puede deberse a que la estructura de la plantilla de mapeo de solicitudes no coincide con la estructura de clave de Amazon DynamoDB:

```
The provided key element does not match the schema (Service: AmazonDynamoDBv2; Status
 Code: 400; Error Code
```

Por ejemplo, si la tabla de DynamoDB tiene una clave hash llamada `"id"` y la plantilla dice `"PostID"`, como en el ejemplo siguiente, se produce el error anterior, ya que `"id"` no coincide con `"PostID"`.

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

Falta el solucionador

Si ejecuta una operación de GraphQL, por ejemplo una consulta, y obtiene una respuesta nula, puede ser debido a que no tiene un solucionador configurado.

Por ejemplo, si importa un esquema que define un campo `getCustomer(userId: ID!):` y no ha configurado un solucionador para este campo, cuando se ejecuta una consulta, por ejemplo `getCustomer(userId:"ID123"){...}`, recibirá una respuesta como la siguiente:

```
{
    "data": {
    "getCustomer": null
```

```
    }
}
```

Errores en la plantilla de mapeo

Si la plantilla de mapeo no está configurada correctamente, recibirá una respuesta de GraphQL cuyo valor de `errorType` será `MappingTemplate`. El campo `message` debe indicar dónde se encuentra el problema de la plantilla de mapeo.

Por ejemplo, si la plantilla de mapeo de solicitud no tiene el campo `operation` o si el nombre del campo `operation` es incorrecto, obtendrá una respuesta similar a la siguiente:

```
{
    "data": {
        "searchPosts": null
    },
    "errors": [
        {
        "path": [
            "searchPosts"
        ],
        "errorType": "MappingTemplate",
        "locations": [
            {
            "line": 2,
            "column": 3
            }
        ],
        "message": "Value for field '$[operation]' not found."
        }
    ]
}
```

Tipos de retorno incorrectos

El tipo de retorno del origen de datos debe coincidir con el tipo definido para un objeto en el esquema; de lo contrario, es posible que se produzca un error de GraphQL como este:

```
"errors": [
    {
```

```
    "path": [
        "posts"
    ],
    "locations": null,
    "message": "Can't resolve value (/posts) : type mismatch error, expected type LIST,
 got OBJECT"
    }
]
```

Por ejemplo, esto podría ocurrir con la siguiente definición de consulta:

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

que espera una lista de objetos `[Posts]`. Por ejemplo, si tuviera una función Lambda en Node.JS con algo parecido a lo siguiente:

```
const result = { data: data.Items.map(item => { return item ; }) };
callback(err, result);
```

Se generaría un error, ya que `result` es un objeto. Tendría que cambiar la devolución de llamada a `result.data` o modificar el esquema para que no devuelva una lista.

Procesamiento de solicitudes no válidas

Si no AWS AppSync es posible procesar y enviar una solicitud (debido a datos incorrectos, como una sintaxis no válida) al solucionador de campos, la carga útil de respuesta devolverá los datos del campo con los valores establecidos en `null` y cualquier error relevante.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.