Entwicklerhandbuch

AWS AppSync GraphQL

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS AppSync GraphQL: Entwicklerhandbuch

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Marken und Handelsmarken von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, die geeignet ist, Kunden irrezuführen oder Amazon in irgendeiner Weise herabzusetzen oder zu diskreditieren. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist AWS AppSync ... 1
AWS AppSync features ... 1
AWS AppSync Benutzen Sie es zum ersten Mal? ... 2
Zugehörige Services ... 3
Preisgestaltung für AWS AppSync ... 3
GraphQL und Architektur AWS AppSync ... 4
Was ist eine API ... 5
Clients ... 5
Ressourcen ... 5
Was ist REST ... 6
Einheitliche Oberfläche ... 6
Staatenlosigkeit ... 7
Mehrschichtiges System ... 7
Cachefähigkeit ... 7
Was ist eine API? RESTful ... 8
Wie RESTful APIs arbeiten sie? ... 8
Was ist GraphQL ... 8
Vergleich von REST und GraphQL ... 9
Warum GraphQL gegenüber REST verwenden ... 11
Komponenten einer GraphQL-API ... 13
GraphQL-Schemas ... 14
Datenquellen ... 33
Resolver ... 49
Zusätzliche Eigenschaften von GraphQL ... 59
Deklarativ ... 59
Hierarchical (Hierarchisch) ... 60
Introspektiv ... 61
Starkes Tippen ... 62
Erste Schritte: Erstellen Sie Ihre erste GraphQL-API ... 63
Ein Schema starten ... 64
Ein Rundgang durch die Konsole AWS AppSync ... 68
Schema-Designer ... 69
Datenquellen ... 70
Abfragen ... 71

Einstellungen ... 71
Verwenden von GraphQL-Mutationen zum Hinzufügen von Daten zu einer DynamoDB-Tabelle ... 72
Verwenden von GraphQL-Abfragen zum Abrufen von Daten aus einer DynamoDB-Tabelle ... 77
Zusätzliche Abschnitte ... 80
Integration ... 81
Zusätzliche Lektüre ... 82
GraphQL entwerfen APIs ... 83
Strukturierung einer GraphQL-API (leer oder importiert APIs) ... 83
Entwerfen Ihres GraphQL-Schemas ... 84
Eine Datenquelle anhängen ... 113
Resolver konfigurieren AWS AppSync ... 126
Verwendung APIs mit dem CDK ... 184
Verwendung von Abonnements für Echtzeit-Datenanwendungen ... 203
GraphQL-Schemaabonnementanweisungen ... 203
Abonnementargumente verwenden ... 206
Generisches pub/sub APIs Powered by Serverless erstellen WebSockets ... 210
Definition erweiterter Abonnementfilter ... 214
Verbindungen mithilfe von Filtern abbestellen WebSocket ... 225
Aufbau eines WebSocket Echtzeit-Clients ... 230
Zusammenführen APIs ... 252
Zusammengeführt APIs und Verbund ... 254
Konfliktlösung zusammengeführter APIs ... 256
Schemas konfigurieren ... 264
Autorisierungsmodi konfigurieren ... 264
Konfiguration von Ausführungsrollen ... 265
Konfiguration von Cross-Account-Zusammengeführt mit APIs AWS RAM ... 267
Zusammenführen ... 269
Zusätzliche Unterstützung für Merged APIs ... 270
Einschränkungen der zusammengeführten APIs ... 271
Merged erstellen APIs ... 272
GraphQL APIs mit RDS-Introspektion erstellen ... 274
Verwenden der Introspektionsfunktion (Konsole) ... 275
Verwendung der Introspektionsfunktion (API) ... 278
Erstellen einer Client-Anwendung ... 282
JavaScript Resolver-Tutorials ... 285

Erstellen einer einfachen Post-Anwendung mit DynamoDB-Resolvern JavaScript 285
Erstellen Sie Ihre GraphQL-API 286
Definition einer grundlegenden Post-API 286
Einrichtung Ihrer Amazon DynamoDB-Tabelle 287
Einen AddPost-Resolver einrichten (Amazon DynamoDB) PutItem 288
Einrichtung des GetPost-Resolvers (Amazon DynamoDB) GetItem 292
Eine UpdatePost-Mutation erstellen (Amazon DynamoDB) UpdateItem 294
Stimmenmutationen erstellen (Amazon DynamoDB UpdateItem) 299
Einen DeletePost-Resolver einrichten (Amazon DynamoDB) DeleteItem 302
Einen AllPost-Resolver einrichten (Amazon DynamoDB Scan) 309
Einen allPostsBy Author-Resolver einrichten (Amazon DynamoDB Query) 313
Verwenden von Sets 318
Schlussfolgerung 325
Verwenden von Resolvern AWS Lambda 326
Erstellen einer Lambda-Funktion 326
Eine Datenquelle für Lambda konfigurieren 328
Erstellen Sie ein GraphQL-Schema 329
Resolver konfigurieren 127
Testen Sie Ihre GraphQL-API 331
Zurückkehrende Fehler 332
Anwendungsfall für Fortgeschrittene: Batching 335
Verwenden von lokalen Resolvern 345
Die Pub/Sub-App erstellen 345
Nachrichten senden und abonnieren 346
Kombinieren von GraphQL-Resolvern 347
Beispielschema 348
Änderung von Daten mithilfe von Resolvern 349
DynamoDB und Service OpenSearch 349
Verwenden von OpenSearch Service-Resolvern 352
Erstellen Sie eine neue OpenSearch Dienstdomäne 352
Konfigurieren Sie eine Datenquelle für Service OpenSearch 353
Einen Resolver anschließen 355
Ihre Suchanfragen ändern 357
Daten zum OpenSearch Service hinzufügen 358
Ein einzelnes Dokument wird abgerufen 359
Führen Sie Abfragen und Mutationen durch 360

Bewährte Methoden ... 361
DynamoDB-Transaktionen durchführen ... 361
Berechtigungen ... 361
Datenquelle ... 362
Transaktionen ... 364
DynamoDB-Batchoperationen verwenden ... 371
Batches aus einzelnen Tabellen ... 372
Batch mit mehreren Tabellen ... 377
Fehlerbehandlung ... 385
Verwendung von HTTP-Resolvern ... 391
Erstellen einer REST-API ... 392
Erstellen Sie Ihre GraphQL-API ... 392
Erstellen eines GraphQL-Schemas ... 393
Konfigurieren Sie Ihre HTTP-Datenquelle ... 394
Konfigurieren von Resolvern ... 161
Dienste werden aufgerufen AWS ... 397
Verwenden von Aurora PostgreSQL mit Daten-API ... 399
Richten Sie Ihre Aurora PostgreSQL-Datenbank ein ... 400
Datenbank und Tabelle erstellen ... 401
Erstellen eines GraphQL-Schemas ... 402
Resolver für RDS ... 404
Löschen Sie Ihren Cluster ... 412
Tutorials zum VTL-Resolver ... 414
Erstellen einer einfachen Post-Anwendung mit DynamoDB-Resolvern ... 415
DynamoDB-Tabellen einrichten ... 415
Erstellen Sie Ihre GraphQL-API ... 392
Definition einer grundlegenden Post-API ... 417
Konfiguration der Datenquelle für die DynamoDB-Tabellen ... 418
Den AddPost-Resolver einrichten (DynamoDB) PutItem ... 419
Einrichtung des GetPost-Resolvers (DynamoDB) GetItem ... 425
Eine UpdatePost-Mutation erstellen (DynamoDB) UpdateItem ... 428
Den updatePost Resolver (DynamoDB) ändern UpdateItem ... 431
UpvotePost- und DownvotePost-Mutationen erstellen (DynamoDB) UpdateItem ... 438
Den DeletePost-Resolver einrichten (DynamoDB) DeleteItem ... 442
Einrichten des allPost-Resolvers (DynamoDB Scan) ... 449
Den allPostsBy Author Resolver einrichten (DynamoDB Query) ... 454

Verwenden von Sätzen ... 318
Verwenden von Listen und Zuordnungen ... 467
Schlussfolgerung ... 471
Verwenden von Resolvern AWS Lambda ... 471
Erstellen einer Lambda-Funktion ... 472
Eine Datenquelle für Lambda konfigurieren ... 474
Erstellen Sie ein GraphQL-Schema ... 393
Resolver konfigurieren ... 161
Testen Sie Ihre GraphQL-API ... 478
Zurückkehrende Fehler ... 480
Anwendungsfall für Fortgeschrittene: Batching ... 482
Verwenden von OpenSearch Service-Resolvern ... 493
One-Click-Setup ... 493
Erstellen Sie eine neue OpenSearch Dienstdomäne ... 494
Datenquelle für den OpenSearch Service konfigurieren ... 494
Verbinden eines Resolvers ... 496
Ändern Ihrer Suchvorgänge ... 498
Daten zum OpenSearch Dienst hinzufügen ... 499
Abrufen eines einzelnen Dokuments ... 500
Ausführen von Abfragen und Mutationen ... 501
Bewährte Methoden ... 502
Verwendung lokaler Resolver ... 502
Erstellen der Paging-Anwendung ... 503
Senden an und Abonnieren von Seiten ... 504
Kombinieren von GraphQL-Resolvern ... 505
Beispielschema ... 505
Ändern von Daten mithilfe von Resolvern ... 506
DynamoDB und Service OpenSearch ... 507
DynamoDB-Batchoperationen verwenden ... 511
Berechtigungen ... 512
Datenquelle ... 513
Stapelvorgang mit einer Tabelle ... 514
Stapelvorgang mit mehreren Tabellen ... 518
Fehlerbehandlung ... 526
DynamoDB-Transaktionen durchführen ... 532
Berechtigungen ... 512

Datenquelle ... 513
Transaktionen ... 534
Verwendung von HTTP-Resolvern ... 544
One-Click-Setup ... 493
Erstellen einer REST-API ... 392
Erstellen der GraphQL-API ... 392
Erstellen eines GraphQL-Schemas ... 393
Konfigurieren der HTTP-Datenquelle ... 394
Konfigurieren von Resolvern ... 161
Dienste aufrufen AWS ... 550
Verwenden von Aurora Serverless v2-Resolvern ... 552
Richten Sie Ihren Datenbankcluster ein ... 553
Aktivieren der Daten-API ... 555
Datenbank und Tabelle erstellen ... 555
GraphQL-Schema ... 556
Connect Sie Ihre API mit Datenbankoperationen ... 161
Ändern Sie Ihre Daten über die API ... 565
Rufen Sie Ihre Daten ab ... 566
Schützen Sie Ihren Datenzugriff ... 567
Verwendung von Pipeline-Resolvern ... 570
One-Click-Setup ... 493
Manuelle Einrichtung ... 571
Testen der GraphQL-API ... 478
Verwenden von Delta Sync-Vorgängen für versionierte Datenquellen ... 585
One-Click-Setup ... 493
Schema ... 586
Mutationen ... 589
Sync-Abfragen ... 589
Beispiel ... 589
Konfiguration und Einstellungen ... 596
Konfiguration von serverseitigem Caching und API-Nutzlastkomprimierung ... 597
Instance-Typen ... 597
Caching-Verhalten ... 598
Cache-Verschlüsselung ... 601
Räumung des Caches ... 601
Einen Cache-Eintrag löschen ... 601

Löschen eines Cache-Eintrags auf der Grundlage seiner Identität ... 603
API-Antworten komprimieren ... 605
Konfiguration benutzerdefinierter Domainnamen für GraphQL und Echtzeit APIs ... 606
Registrierung und Konfiguration eines Domainnamens ... 607
Einen benutzerdefinierten Domainnamen erstellen in AWS AppSync ... 607
Benutzerdefinierte Wildcard-Domainnamen in AWS AppSync ... 608
Versionierung, Konflikterkennung und Synchronisierungsvorgänge für DynamoDB ... 609
Versionierung von DynamoDB-Datenquellen ... 610
Erkennung und Lösung von Konflikten ... 614
Verwenden von DynamoDB-Synchronisierungsvorgängen für versionierte Datenquellen ... 625
Verwendung CloudWatch zur Überwachung und Protokollierung von GraphQL-API-Daten ... 625
Einrichtung und Konfiguration ... 626
CloudWatch Metriken ... 628
CloudWatch Protokolle ... 639
Referenz zum Protokolltyp ... 645
Analysieren Sie Ihre Logs mit Logs Insights CloudWatch ... 647
Analysieren Sie Ihre Logs mit Service OpenSearch ... 649
Migration des Protokollformats ... 649
Anfragen verfolgen in AWS X-Ray ... 650
Einrichtung und Konfiguration ... 626
Verfolgen Sie Ihre API mit X-Ray ... 651
Protokollierung von AWS CloudTrail-API-Aufrufen mit ... 653
AWS AppSync Informationen in CloudTrail ... 653
AWS AppSync Datenereignisse in CloudTrail ... 654
Grundlegendes zu AWS AppSync Einträgen in Protokolldateien ... 656
AWS AppSync Private verwenden APIs ... 663
Private erstellen AWS AppSync APIs ... 665
Erstellen eines Schnittstellenendpunkts für AWS AppSync ... 666
Fortschrittliche -Beispiele ... 667
Verwendung von IAM-Richtlinien zur Beschränkung der Erstellung öffentlicher APIs ... 671
GraphQL teilen APIs ... 672
Voraussetzungen für die gemeinsame Nutzung von AWS AppSync GraphQL APIs ... 673
AWS AppSync GraphQL teilen APIs ... 673
AWS AppSync GraphQL nicht mehr teilen APIs ... 676
Kontoübergreifende Ereignisse ... 676
Konfiguration von GraphQL-Laufkomplexität, Abfragetiefe und Introspektion ... 677

Verwenden der Introspektionsfunktion ... 677
Konfiguration von Grenzwerten für die Abfragetiefe ... 679
Konfiguration der Grenzwerte für die Anzahl der Resolver ... 680
Verwenden von -Umgebungsvariablen ... 682
Konfiguration von Umgebungsvariablen (Konsole) ... 682
Konfiguration von Umgebungsvariablen (API) ... 683
Konfiguration von Umgebungsvariablen (CFN) ... 685
Umgebungsvariablen und zusammengeführt APIs ... 685
Abrufen von Umgebungsvariablen ... 685
GraphQL autorisieren und authentifizieren APIs ... 687
Autorisierungstypen ... 687
API_KEY Autorisierung ... 688
AWS_LAMBDA Autorisierung ... 691
Umgehung der Autorisierungsbeschränkungen für SigV4- und OIDC-Tokens ... 696
AWS_IAM Autorisierung ... 697
OPENID_CONNECT Autorisierung ... 700
Autorisierung durch AMAZON_COGNITO_USER_POOLS ... 701
Verwenden zusätzlicher Autorisierungsmodi ... 703
Differenzierte Zugriffskontrolle ... 705
Informationen filtern ... 708
Datenquellenzugriff ... 709
Anwendungsfälle der Zugriffskontrolle zur Sicherung von Anfragen und Antworten ... 710
Übersicht ... 710
Lesen von Daten ... 711
Daten schreiben ... 715
Öffentliche und private Aufzeichnungen ... 717
Daten in Echtzeit ... 718
Verwenden AWS WAF zum Schutz APIs ... 722
Integrieren Sie eine API mit AppSync AWS WAF ... 723
Regeln für eine Web-ACL erstellen ... 725
Sicherheit ... 729
Datenschutz ... 730
Verschlüsselung in Bewegung ... 731
Compliance-Validierung ... 731
Sicherheit der Infrastruktur ... 732
Ausfallsicherheit ... 733

Identity and Access Management ... 733
Zielgruppe ... 734
Authentifizierung mit Identitäten ... 735
Verwalten des Zugriffs mit Richtlinien ... 739
Wie AWS AppSync funktioniert mit IAM ... 742
Identitätsbasierte Richtlinien ... 749
Fehlerbehebung ... 762
Protokollieren von AWS AppSync API-Aufrufen mit AWS CloudTrail ... 765
AWS AppSync Informationen in CloudTrail ... 765
AWS AppSync Logdateieinträge verstehen ... 766
Bewährte Methoden ... 502
Machen Sie sich mit Authentifizierungsmethoden vertraut ... 768
Verstehen Sie, wie sich Änderungen an der API-Konfiguration ausbreiten ... 769
Verwenden Sie TLS für HTTP-Resolver ... 769
Verwenden Sie Rollen mit den geringstmöglichen Berechtigungen ... 769
Bewährte Methoden für IAM-Richtlinien ... 769
Resolver-Referenz () JavaScript ... 772
JavaScript Übersicht über Resolver ... 772
Unterstützte Runtime-Funktionen ... 773
Resolver für Einheiten ... 773
Aufbau eines Pipeline-Resolvers JavaScript ... 773
Zusätzliche Themen ... 778
Beispiel für einen Pipeline-Resolver mit Amazon DynamoDB ... 778
Dienstprogramme für die `APPSYNC_JS` Laufzeit konfigurieren ... 781
Bündelung TypeScript, und Quellzuordnungen für die Laufzeit `APPSYNC_JS` ... 784
Testen Sie Ihren Resolver und Ihre Funktionshandler ... 791
Migration von VTL zu JavaScript ... 794
Wahl zwischen direktem Datenquellenzugriff und Proxying über eine Lambda-Datenquelle . 797
JavaScript Objektreferenz für den Resolver-Kontext ... 799
Zugreifen auf den `context` ... 799
JavaScript Laufzeitfunktionen für Resolver und Funktionen ... 810
Unterstützte Runtime-Funktionen ... 811
Integrierte Dienstprogramme ... 818
Integrierten Module ... 821
Laufzeit-Dienstprogramme ... 846
Zeithelfer in util.time ... 847

DynamoDB-Helfer in util.dynamodb ... 848
HTTP-Helfer in util.http ... 855
Transformationshelfer in util.transform ... 857
Zeichenketten-Helfer in util.str ... 870
Erweiterungen ... 871
XML-Hilfsprogramme in util.xml ... 875
JavaScript Resolver-Funktionsreferenz für DynamoDB ... 877
GetItem ... 879
PutItem ... 881
UpdateItem ... 884
DeleteItem ... 889
Abfrage ... 892
Scan ... 897
Synchronisierung ... 901
BatchGetItem ... 904
BatchDeleteItem ... 907
BatchPutItem ... 910
TransactGetItems ... 912
TransactWriteItems ... 916
Geben Sie System ein (Zuordnung der Anfrage) ... 923
Geben Sie System ein (Antwortzuordnung) ... 928
Filter ... 932
Bedingungsausdrücke ... 934
Ausdrücke für Transaktionsbedingungen ... 948
Projektionen ... 950
JavaScript Resolver-Funktionsreferenz für OpenSearch ... 952
Anforderung ... 952
Antwort ... 953
`operation` field ... 954
`path` field ... 954
`params` field ... 954
Variablen übergeben ... 956
JavaScript Resolver-Funktionsreferenz für Lambda ... 957
Objekt anfordern ... 957
Antwortobjekt ... 961
Batch-Antwort mit Lambda-Funktion ... 962

JavaScript Resolver-Funktionsreferenz für die Datenquelle EventBridge 962
Anforderung 952
Antwort 963
PutEvents-Felder 964
JavaScript Resolver-Funktionsreferenz für die Datenquelle None 966
Anforderung 952
Nutzlast 960
Antwort 963
JavaScript Resolver-Funktionsreferenz für HTTP 967
Anforderung 952
Methode 968
ResourcePath 968
Params-Felder 969
Antwort 963
JavaScript Resolver-Funktionsreferenz für Amazon RDS 970
Mit SQL markierte Vorlage 971
Aussagen erstellen 972
Abrufen von Daten 973
Hilfsfunktionen 974
Umwandlung 982
JavaScript Resolver-Funktionsreferenz für Amazon Bedrock 983
Objekt anfordern 984
Antwortobjekt 991
Lang andauernde Aufrufe 991
Geben Sie eine Referenz ein 992
Referenz zur Resolver-Mapping-Vorlage (VTL) 999
Übersicht über die Resolver-Mapping-Vorlage 1000
Resolver für Einheiten 1000
Pipeline-Resolver 180
-Beispielvorlage 1005
Evaluierte Deserialisierungsregeln für Mapping-Vorlagen 1007
Programmierleitfaden für Resolver-Mapping-Vorlagen 1009
Aufstellen 1010
Variablen 1011
Methoden aufrufen 1013
Zeichenfolgen 1014

Loops ... 1015
Arrays ... 1016
Bedingte Prüfungen ... 1017
Operatoren ... 1018
Kontext ... 1020
Filtern ... 1020
Kontextreferenz für die Resolver-Mapping-Vorlage ... 1026
Zugreifen auf den `$context` ... 1026
Bereinigung von Eingängen ... 1036
Referenz zum Hilfsprogramm für Resolver-Mapping-Vorlagen ... 1037
Hilfsprogramme in $util ... 1039
AWS AppSync Direktiven ... 1052
Zeithelfer in $util.time ... 1052
Listet Helfer in $util.list auf ... 1055
Ordnen Sie Helfer in $util.map zu ... 1056
DynamoDB-Helferobjekte in $util.dynamodb ... 1057
Amazon RDS-Helfer in $util.rds ... 1067
HTTP-Helfer in $util.http ... 1070
XML-Helfer in $util.xml ... 1073
Transformationshelfer in $util.transform ... 1075
Mathematische Helfer in $util.math ... 1089
Zeichenketten-Helfer in $util.str ... 1090
Erweiterungen ... 1091
Referenz zur Resolver-Mapping-Vorlage für DynamoDB ... 1105
GetItem ... 1107
PutItem ... 1109
UpdateItem ... 1113
DeleteItem ... 1119
Abfrage ... 1122
Scan ... 1127
Synchronisierung ... 1132
BatchGetItem ... 1135
BatchDeleteItem ... 1139
BatchPutItem ... 1142
TransactGetItems ... 1146
TransactWriteItems ... 1150

Geben Sie System ein (Zuordnung anfordern) 1159
Geben Sie System ein (Antwortzuordnung) 1164
Filter 1168
Bedingungsausdrücke 1170
Ausdrücke für Transaktionsbedingungen 1183
Projektionen 1186
Referenz zur Resolver-Mapping-Vorlage für RDS 1188
Vorlage für die Zuordnung anfordern 1188
Version 1190
Aussagen und VariableMap 1190
VariableTypeHintMap 1191
Referenz zur Resolver-Mapping-Vorlage für OpenSearch 1191
Zuordnungsvorlage anfordern 1188
Vorlage für die Zuordnung von Antworten 953
`operation` field 954
`path` field 954
`params` field 954
Variablen übergeben 956
Referenz zur Resolver-Mapping-Vorlage für Lambda 1197
Mapping-Vorlage anfordern 1188
Vorlage für die Zuordnung von Antworten 953
Batch-Antwort mit Lambda-Funktion 1203
Direkte Lambda-Resolver 1203
Referenz zur Resolver-Mapping-Vorlage für EventBridge 1209
Vorlage für die Zuordnung anfordern 1188
Vorlage für die Zuordnung von Antworten 953
`PutEvents`-Felder 964
Referenz zur Resolver-Mapping-Vorlage für die Datenquelle None 1214
Vorlage für die Anforderungszuordnung 1188
Version 1190
Nutzlast 1201
Vorlage für die Zuordnung von Antworten 953
Referenz zur Resolver-Mapping-Vorlage für HTTP 1216
Zuordnungsvorlage anfordern 1188
Version 1190
Methode 1219

ResourcePath 1220
Params-Felder 954
Antwort 1221
Von vier HTTPS-Endpunkten anerkannte AWS AppSync Zertifizierungsstellen (CA) 1222
Changelog für die Resolver-Mapping-Vorlage 1289
Matrix der Verfügbarkeit von Datenquellenoperationen pro Version 1290
Änderung der Version in einer Unit-Resolver-Zuweisungsvorlage 1291
Änderung der Version in einer Funktion 1292
2018-05-29 1292
2017-02-28 1299
GraphQL-Typreferenz 1300
Skalare Typen in GraphQL 1301
Standard-Skalare 1301
AWS AppSync Skalare 1302
Beispiel für die Verwendung eines Schemas 1303
Schnittstellen und Unions in GraphQL 1307
Beispiele für Benutzeroberflächen 1307
Beispiele für Vereinigungen 1311
Geben Sie Auflösung ein AWS AppSync 1312
Beispiel für die Typauflösung 1313
Problembehebung und häufige Fehler 1318
Falsche DynamoDB-Schlüsselzuweisung 1318
Fehlender Resolver 1318
Fehler bei der Zuordnungsvorlage 1319
Falsche Rückgabetypen 1319
Ungültige Anfragen werden verarbeitet 1320
..... mcccxxi

Was ist AWS AppSync?

AWS AppSync ermöglicht es Entwicklern, ihre Anwendungen und Dienste mit Daten und Ereignissen mit sicherem, serverlosem und leistungsstarkem GraphQL und Pub/Sub zu verbinden. APIs Sie können Folgendes tun mit: AWS AppSync

- Greifen Sie von einem einzigen GraphQL-API-Endpunkt aus auf Daten aus einer oder mehreren Datenquellen zu.
- Kombinieren Sie mehrere GraphQL-Quellen APIs zu einer einzigen, zusammengeführten GraphQL-API.
- Veröffentlichen Sie Datenaktualisierungen in Echtzeit für Ihre Anwendungen.
- Nutzen Sie die integrierten Sicherheits-, Überwachungs-, Protokollierungs- und Ablaufverfolgungsfunktionen mit optionalem Caching für geringe Latenz.
- Zahlen Sie nur für API-Anfragen und alle Nachrichten, die in Echtzeit zugestellt werden.

> **Important**
>
> Ab dem 13. März 2025 können Sie WebSockets mithilfe von AWS AppSync Events eine PubSub Echtzeit-API erstellen. Weitere Informationen finden Sie unter Ereignisse veröffentlichen über WebSocket im AWS AppSync Events Developer Guide.

Themen

- AWS AppSync features
- AWS AppSync Benutzen Sie es zum ersten Mal?
- Zugehörige Services
- Preisgestaltung für AWS AppSync

AWS AppSync features

- Vereinfachter Datenzugriff und -abfrage, unterstützt von GraphQL
- Serverlos WebSockets für GraphQL-Abonnements und Pub/Sub-Kanäle

- Serverseitiges Caching zur Bereitstellung von Daten in Hochgeschwindigkeits-In-Memory-Caches mit geringer Latenz
- JavaScript und TypeScript Unterstützung beim Schreiben von Geschäftslogik
- Unternehmenssicherheit mit Private APIs zur Einschränkung des API-Zugriffs und der API-Integration AWS WAF
- Integrierte Autorisierungskontrollen mit Unterstützung für API-Schlüssel, IAM, Amazon Cognito, OpenID Connect-Anbieter und Lambda-Autorisierung für benutzerdefinierte Logik.
- Zusammengeführt, APIs um föderierte Anwendungsfälle zu unterstützen

Weitere Informationen zu den einzelnen Funktionen finden Sie unter AWS AppSync Funktionen.

AWS AppSync Benutzen Sie es zum ersten Mal?

Wir empfehlen AWS AppSync Erstbenutzern, zunächst die folgenden Abschnitte zu lesen:

- Wenn Sie mit GraphQL nicht vertraut sind, lesen Sie den. Erste Schritte: Erstellen Sie Ihre erste GraphQL-API
- Wenn Sie Anwendungen erstellen, die GraphQL verwenden APIs, finden Sie weitere Informationen unter Erstellen einer Client-Anwendung undthe section called “Verwendung von Abonnements für Echtzeit-Datenanwendungen”.
- Wenn Sie nach Informationen zum GraphQL-Resolver suchen, lesen Sie die folgenden Informationen:

  JavaScript/TypeScript

  - Anleitungen zum Resolver () JavaScript
  - Resolver-Referenz () JavaScript

  VTL

  - Resolver-Tutorials (VTL)
  - Referenz zur Resolver-Mapping-Vorlage (VTL)
- Wenn Sie AWS AppSync beispielsweise nach Projekten, Updates und mehr suchen, schauen Sie sich den AppSync Blog an.

Zugehörige Services

Wenn Sie eine Web- oder Mobil-App von Grund auf neu entwickeln, sollten Sie die Verwendung in Betracht ziehen AWS Amplify. Amplify nutzt AWS AppSync und andere AWS Dienste, mit denen Sie mit weniger Aufwand robustere und leistungsfähigere Web- und Mobilanwendungen erstellen können.

Preisgestaltung für AWS AppSync

AWS AppSync basiert auf Millionen von Anfragen und Updates. Das Caching kostet eine zusätzliche Gebühr. Weitere Informationen finden Sie unter AWS AppSync Preise.

Im Folgenden sind die Ausnahmen von der allgemeinen AWS AppSync Preisgestaltung aufgeführt:

- Bei Anfragen mit Autorisierungs- und Authentifizierungsfehlern entstehen keine Gebühren.
- Bei Aufrufmethoden, die einen API-Schlüssel benötigen, entstehen keine Gebühren, wenn API-Schlüssel fehlen oder ungültig sind.

GraphQL und Architektur AWS AppSync

> **Note**
>
> In diesem Handbuch wird davon ausgegangen, dass der Benutzer über fundierte Kenntnisse des REST-Architekturstils verfügt. Wir empfehlen, dieses und andere Frontend-Themen zu lesen, bevor Sie mit GraphQL arbeiten und. AWS AppSync

GraphQL ist eine Abfrage- und Manipulationssprache für APIs. GraphQL bietet eine flexible und intuitive Syntax zur Beschreibung von Datenanforderungen und Interaktionen. Es ermöglicht Entwicklern, genau nach dem zu fragen, was benötigt wird, und vorhersehbare Ergebnisse zu erhalten. Es ermöglicht auch den Zugriff auf viele Quellen in einer einzigen Anfrage, wodurch die Anzahl der Netzwerkaufrufe und die Bandbreitenanforderungen reduziert werden, wodurch die Akkulaufzeit und die CPU-Zyklen, die von Anwendungen verbraucht werden, reduziert werden.

Das Aktualisieren von Daten wird durch Mutationen vereinfacht, sodass Entwickler beschreiben können, wie sich die Daten ändern sollten. GraphQL ermöglicht auch die schnelle Einrichtung von Echtzeitlösungen über Abonnements. All diese Funktionen in Kombination mit leistungsstarken Entwicklertools machen GraphQL für die Verwaltung von Anwendungsdaten unverzichtbar.

GraphQL ist eine Alternative zu REST. RESTful Architektur ist derzeit eine der beliebtesten Lösungen für die Client-Server-Kommunikation. Im Mittelpunkt steht das Konzept, dass Ihre Ressourcen (Daten) über eine URL verfügbar gemacht werden. Diese URLs können verwendet werden, um über CRUD-Operationen (Create, Read, Update, Delete) in Form von HTTP-Methoden wie `GETPOST`, und auf die Daten zuzugreifen und `DELETE` sie zu bearbeiten. Der Vorteil von REST besteht darin, dass es relativ einfach zu erlernen und zu implementieren ist. Sie können schnell einrichten RESTful APIs , um eine Vielzahl von Diensten anzurufen.

Die Technologie wird jedoch immer komplizierter. Da Anwendungen, Tools und Dienste zunehmend für ein weltweites Publikum skaliert werden, ist der Bedarf an schnellen, skalierbaren Architekturen von größter Bedeutung. REST weist viele Mängel auf, wenn es um skalierbare Abläufe geht. In diesem Anwendungsfall finden Sie ein Beispiel.

In den folgenden Abschnitten werden wir uns mit einigen der damit verbundenen Konzepte befassen RESTful APIs. Anschließend stellen wir GraphQL vor und wie es funktioniert.

Weitere Informationen zu GraphQL und den Vorteilen der Migration zu AWS finden Sie im Entscheidungsleitfaden für GraphQL-Implementierungen.

Themen

- Was ist eine API?
- Was ist REST?
- Was ist GraphQL?
- Vergleich von REST und GraphQL
- Warum GraphQL über REST verwenden?
- Komponenten einer GraphQL-API
- Zusätzliche Eigenschaften von GraphQL

Was ist eine API?

Eine Anwendungsprogrammierschnittstelle (API) definiert die Regeln, die Sie befolgen müssen, um mit anderen Softwaresystemen zu kommunizieren. Entwickler veröffentlichen oder erstellen Inhalte, APIs sodass andere Anwendungen programmgesteuert mit ihren Anwendungen kommunizieren können. Die Timesheet-Anwendung stellt beispielsweise eine API zur Verfügung, die nach dem vollständigen Namen eines Mitarbeiters und einem Datumsbereich fragt. Wenn sie diese Informationen erhält, verarbeitet sie intern die Arbeitszeittabelle des Mitarbeiters und gibt die Anzahl der Arbeitsstunden in diesem Zeitraum zurück.

Sie können sich eine Web-API als ein Gateway zwischen Clients und Ressourcen im Internet vorstellen.

Clients

Kunden sind Benutzer, die auf Informationen aus dem Internet zugreifen möchten. Der Client kann eine Person oder ein Softwaresystem sein, das die API verwendet. Entwickler können beispielsweise Programme schreiben, die auf Wetterdaten von einem Wettersystem zugreifen. Oder Sie können über Ihren Browser auf dieselben Daten zugreifen, wenn Sie die Wetter-Website direkt besuchen.

Ressourcen

Ressourcen sind die Informationen, die verschiedene Anwendungen ihren Kunden zur Verfügung stellen. Ressourcen können Bilder, Videos, Text, Zahlen oder jede Art von Daten sein. Der Computer,

der die Ressource an den Client weitergibt, wird auch als Server bezeichnet. Organizations nutzen APIs Ressourcen gemeinsam und stellen Webdienste bereit, ohne dabei Sicherheit, Kontrolle und Authentifizierung zu vernachlässigen. APIs Helfen Sie ihnen außerdem dabei, festzustellen, welche Kunden Zugriff auf bestimmte interne Ressourcen erhalten.

Was ist REST?

Auf einer höheren Ebene ist Representational State Transfer (REST) eine Softwarearchitektur, die Bedingungen dafür festlegt, wie eine API funktionieren sollte. REST wurde ursprünglich als Richtlinie für die Verwaltung der Kommunikation in einem komplexen Netzwerk wie dem Internet entwickelt. Sie können eine REST-basierte Architektur verwenden, um leistungsstarke und zuverlässige Kommunikation in großem Maßstab zu unterstützen. Sie können sie einfach implementieren und modifizieren und so Transparenz und plattformübergreifende Portabilität für jedes API-System schaffen.

API-Entwickler können APIs mit verschiedenen Architekturen entwerfen. APIs die dem REST-Architekturstil folgen, werden REST APIs genannt. Webdienste, die die REST-Architektur implementieren, werden als RESTful Webdienste bezeichnet. Der Begriff RESTful API bezieht sich im Allgemeinen auf RESTful Web APIs. Sie können die Begriffe REST-API und RESTful API jedoch synonym verwenden.

Im Folgenden sind einige der Prinzipien des REST-Architekturstils aufgeführt:

Einheitliche Oberfläche

Die einheitliche Oberfläche ist grundlegend für das Design eines jeden RESTful Webservices. Es zeigt an, dass der Server Informationen in einem Standardformat überträgt. Die formatierte Ressource wird in REST als Repräsentation bezeichnet. Dieses Format kann sich von der internen Darstellung der Ressource in der Serveranwendung unterscheiden. Beispielsweise kann der Server Daten als Text speichern, sie aber in einem HTML-Darstellungsformat senden.

Eine einheitliche Benutzeroberfläche unterliegt vier architektonischen Einschränkungen:

1. Anfragen sollten Ressourcen identifizieren. Sie tun dies, indem sie eine einheitliche Ressourcenkennung verwenden.
2. Die Ressourcendarstellung der Clients enthält genügend Informationen, um die Ressource bei Bedarf zu ändern oder zu löschen. Der Server erfüllt diese Bedingung, indem er Metadaten sendet, die die Ressource näher beschreiben.

3. Kunden erhalten Informationen darüber, wie die Darstellung weiter verarbeitet werden kann. Der Server erreicht dies, indem er selbstbeschreibende Nachrichten sendet, die Metadaten darüber enthalten, wie der Client sie am besten verwenden kann.
4. Die Clients erhalten Informationen über alle anderen zugehörigen Ressourcen, die sie zum Erledigen einer Aufgabe benötigen. Der Server erreicht dies, indem er Hyperlinks in der Darstellung sendet, sodass Clients dynamisch mehr Ressourcen ermitteln können.

Staatenlosigkeit

In der REST-Architektur bezieht sich Zustandslosigkeit auf eine Kommunikationsmethode, bei der der Server jede Client-Anfrage unabhängig von allen vorherigen Anfragen abschließt. Clients können Ressourcen in beliebiger Reihenfolge anfordern, und jede Anfrage ist zustandslos oder isoliert von anderen Anfragen. Diese Rest-API-Designeinschränkung impliziert, dass der Server die Anfrage jedes Mal vollständig verstehen und erfüllen kann.

Mehrschichtiges System

In einer mehrschichtigen Systemarchitektur kann der Client eine Verbindung zu anderen autorisierten Vermittlern zwischen dem Client und dem Server herstellen und erhält weiterhin Antworten vom Server. Server können Anfragen auch an andere Server weiterleiten. Sie können Ihren RESTful Webservice so gestalten, dass er auf mehreren Servern mit mehreren Ebenen wie Sicherheits-, Anwendungs- und Geschäftslogik ausgeführt wird, die zusammenarbeiten, um Kundenanfragen zu erfüllen. Diese Ebenen bleiben für den Client unsichtbar.

Cachefähigkeit

RESTful Webdienste unterstützen das Zwischenspeichern, bei dem einige Antworten auf dem Client oder auf einem Zwischenspeicher gespeichert werden, um die Antwortzeit des Servers zu verbessern. Nehmen wir zum Beispiel an, Sie besuchen eine Website, die auf jeder Seite gemeinsame Kopf- und Fußzeilenbilder enthält. Jedes Mal, wenn Sie eine neue Website besuchen, muss der Server dieselben Bilder erneut senden. Um dies zu vermeiden, speichert der Client diese Bilder nach der ersten Antwort im Cache oder speichert sie und verwendet die Bilder dann direkt aus dem Cache. RESTful Webdienste steuern das Caching mithilfe von API-Antworten, die sich selbst als zwischenspeicherbar oder nicht zwischenspeicherbar definieren.

Was ist eine API? RESTful

RESTful API ist eine Schnittstelle, über die zwei Computersysteme Informationen sicher über das Internet austauschen. Die meisten Geschäftsanwendungen müssen mit anderen internen Anwendungen und Anwendungen von Drittanbietern kommunizieren, um verschiedene Aufgaben ausführen zu können. Um beispielsweise monatliche Gehaltsabrechnungen zu erstellen, muss Ihr internes Buchhaltungssystem Daten mit dem Banksystem Ihres Kunden teilen, um die Rechnungsstellung zu automatisieren und mit einer internen Arbeitszeiterfassungsanwendung zu kommunizieren. RESTful APIs unterstützen diesen Informationsaustausch, weil sie sicheren, zuverlässigen und effizienten Software-Kommunikationsstandards folgen.

Wie RESTful APIs arbeiten sie?

Die grundlegende Funktion einer RESTful API entspricht dem Surfen im Internet. Der Client kontaktiert den Server über die API, wenn er eine Ressource benötigt. API-Entwickler erklären in der API-Dokumentation der Serveranwendung, wie der Client die REST-API verwenden sollte. Dies sind die allgemeinen Schritte für jeden REST-API-Aufruf:

1. Der Client sendet eine Anfrage an den Server. Der Client folgt der API-Dokumentation, um die Anfrage so zu formatieren, dass der Server sie versteht.
2. Der Server authentifiziert den Client und bestätigt, dass der Client berechtigt ist, diese Anfrage zu stellen.
3. Der Server empfängt die Anfrage und verarbeitet sie intern.
4. Der Server gibt eine Antwort an den Client zurück. Die Antwort enthält Informationen, die dem Client mitteilen, ob die Anfrage erfolgreich war. Die Antwort enthält auch alle Informationen, die der Client angefordert hat.

Die Details der REST-API-Anfrage und der Antwort variieren geringfügig, je nachdem, wie die API-Entwickler die API entwerfen.

Was ist GraphQL?

GraphQL ist sowohl eine Abfragesprache für APIs als auch eine Laufzeit für die Ausführung dieser Abfragen. GraphQL ermöglicht es Kunden, genau die Daten anzufordern, die sie benötigen, was in vielen Szenarien eine flexiblere und effizientere Alternative zu REST darstellt. Im Gegensatz zu REST, das auf vordefinierten Endpunkten basiert, verwendet GraphQL einen einzigen Endpunkt, an dem Clients ihre Datenanforderungen in Form von Abfragen und Mutationen spezifizieren können.

Weitere Informationen zur Struktur von GraphQL finden Sie unter Komponenten einer APIs GraphQL-API.

Vergleich von REST und GraphQL

APIs (Anwendungsprogrammierschnittstellen) spielen eine entscheidende Rolle bei der Erleichterung des Datenaustauschs zwischen Anwendungen. Wie bereits erwähnt, APIs haben sich zwei herausragende Designansätze herauskristallisiert: GraphQL und REST. Beide dienen zwar dem grundlegenden Zweck, die Client-Server-Kommunikation zu ermöglichen, unterscheiden sich jedoch erheblich in ihrer Implementierung und ihren Anwendungsfällen.

GraphQL und REST haben mehrere Hauptmerkmale gemeinsam:

1. Client-Server-Modell: Beide verwenden eine Client-Server-Architektur für den Datenaustausch.
2. Staatenlosigkeit: Keiner von beiden verwaltet Informationen zur Clientsitzung zwischen Anfragen.
3. HTTP-basiert: Beide verwenden in der Regel HTTP als zugrundeliegendes Kommunikationsprotokoll.
4. Ressourcenorientiertes Design: Beide gestalten ihren Datenaustausch auf der Grundlage von Ressourcen, die sich auf alle Daten oder Objekte beziehen, auf die der Client über die API zugreifen und sie bearbeiten kann.
5. Flexibilität des Datenformats: JSON ist in beiden Fällen das am häufigsten verwendete Datenaustauschformat, obwohl auch andere Formate wie XML und HTML unterstützt werden.
6. Sprach- und datenbankunabhängig: Beide können mit jeder Programmiersprache oder Datenbankstruktur arbeiten, wodurch sie in hohem Maße interoperabel sind.
7. Caching-Unterstützung: Beide unterstützen Caching, sodass Clients und Server häufig aufgerufene Daten speichern können, um die Leistung zu verbessern.

GraphQL und REST teilen zwar einige grundlegende Prinzipien, unterscheiden sich jedoch erheblich in ihrem Ansatz für API-Design und Datenabruf:

1. Anforderungsstruktur und Datenabruf

   REST verwendet verschiedene HTTP-Methoden (GET, POST, PUT, DELETE), um Operationen mit Ressourcen durchzuführen. Dies erfordert häufig mehrere Endpunkte für unterschiedliche Ressourcen, was zu Ineffizienzen beim Datenabruf führen kann. Beispielsweise kann das Ausführen eines GET-Vorgangs zum Abrufen der Daten eines Benutzers dazu führen, dass Daten

zu viel oder zu wenig abgerufen werden. Um die richtigen Daten zu erhalten, können Kürzung oder mehrere Operationen aufgerufen werden.

GraphQL verwendet einen einzigen Endpunkt für alle Operationen. Es stützt sich auf Abfragen zum Abrufen von Daten und Mutationen zum Ändern von Daten. Clients können Abfragen verwenden, um mit einer einzigen Anfrage genau die Daten abzurufen, die sie benötigen, wodurch der Netzwerkaufwand durch Minimierung der Datenübertragung reduziert wird.

2. Serverseitiges Schema

   REST benötigt kein serverseitiges Schema, obwohl eines optional definiert werden kann, um ein effizientes API-Design und eine effiziente API-Dokumentation zu gewährleisten.

   GraphQL verwendet ein stark typisiertes serverseitiges Schema, um Daten und Datendienste zu definieren. Das in GraphQL Schema Definition Language (SDL) geschriebene Schema umfasst Objekttypen und Felder für jedes Objekt sowie serverseitige Resolverfunktionen, die Operationen für jedes Feld definieren.

3. Versioning

   REST beinhaltet häufig die Versionierung in der URL, was dazu führen kann, dass mehrere API-Versionen gleichzeitig verwaltet werden. Die Versionierung ist nicht verpflichtend, kann aber dazu beitragen, grundlegende Änderungen zu verhindern.

   GraphQL fördert eine kontinuierliche Weiterentwicklung der API ohne explizite Versionierung, indem es Abwärtskompatibilität erfordert. Gelöschte Felder geben Fehlermeldungen zurück, während veraltete Tags alte Felder auslaufen lassen und Warnmeldungen zurückgeben.

4. Behandlung von Fehlern

   REST ist schwach typisiert, sodass die Fehlerbehandlung in den umgebenden Code integriert werden muss. Dadurch werden typbezogene Fehler (z. B. das Parsen einer Zahl als Text) möglicherweise nicht automatisch erkannt.

   Im Gegensatz dazu ist GraphQL stark typisiert und erfordert eine umfassende Schemadefinition. Auf diese Weise kann Ihr Service viele Anforderungsfehler automatisch und mit einem hohen Detaillierungsgrad identifizieren.

5. Anwendungsfälle

   REST ist besser geeignet für:

   - Kleinere Anwendungen mit weniger komplexen Datenanforderungen.

- Szenarien, in denen Daten und Operationen von allen Clients auf ähnliche Weise verwendet werden.
- Anwendungen ohne komplexe Datenabfrageanforderungen.

GraphQL ist besser geeignet für:

- Szenarien mit begrenzter Bandbreite, in denen die Minimierung von Anfragen und Antworten entscheidend ist.
- Anwendungen mit mehreren Datenquellen, die an einem einzigen Endpunkt kombiniert werden müssen.
- Fälle, in denen Kundenanfragen stark variieren und unterschiedliche Antwortstrukturen erwartet werden.

Beachten Sie, dass es möglich ist, sowohl GraphQL als auch REST APIs in einer einzigen Anwendung für verschiedene Funktionsbereiche zu verwenden. Darüber hinaus können Sie eine RESTful API so aktualisieren, dass sie GraphQL-Funktionen enthält, ohne sie komplett neu schreiben zu müssen. Ein Beispiel finden Sie unter So erstellen Sie GraphQL-Resolver für AWS Datenquellen.

Warum GraphQL über REST verwenden?

REST ist einer der Eckpfeiler der Architektur des Webs. APIs Da die Welt jedoch immer stärker vernetzt wird, wird die Notwendigkeit, robuste und skalierbare Anwendungen zu entwickeln, zu einem immer dringlicheren Problem werden. REST wird zwar häufig zur Erstellung von Websites verwendet APIs, es wurden jedoch mehrere wiederkehrende Nachteile bei RESTful Implementierungen festgestellt:

1. Datenanfragen: Mit dieser Methode RESTful APIs würden Sie die benötigten Daten in der Regel über Endgeräte anfordern. Das Problem tritt auf, wenn Sie Daten haben, die möglicherweise nicht so übersichtlich verpackt sind. Die Daten, die Sie benötigen, befinden sich möglicherweise hinter mehreren Abstraktionsebenen, und die einzige Möglichkeit, die Daten abzurufen, besteht darin, mehrere Endpunkte zu verwenden, was bedeutet, dass mehrere Anfragen zum Extrahieren aller Daten gestellt werden müssen.
2. Zu viel und zu wenig abrufen: Um die Probleme mehrerer Anfragen noch zu verschärfen, sind die Daten von jedem Endpunkt genau definiert, was bedeutet, dass Sie alle Daten zurückgeben, die für diese API definiert wurden, auch wenn Sie sie technisch nicht wollten.

Dies kann zu übermäßigem Abrufen führen, was bedeutet, dass unsere Anfragen überflüssige Daten zurückgeben. Nehmen wir zum Beispiel an, Sie fordern Personaldaten des Unternehmens an und möchten die Namen der Mitarbeiter in einer bestimmten Abteilung wissen. Der Endpunkt, der die Daten zurückgibt, enthält die Namen, kann aber auch andere Daten wie Berufsbezeichnung oder Geburtsdatum enthalten. Da die API fest ist, können Sie nicht einfach nur die Namen anfordern. Der Rest der Daten wird mitgeliefert.

Die gegenteilige Situation, in der wir nicht genügend Daten zurückgeben, wird als Unterabruf bezeichnet. Um alle angeforderten Daten zu erhalten, müssen Sie möglicherweise mehrere Anfragen an den Dienst stellen. Je nachdem, wie die Daten strukturiert waren, könnten Sie auf ineffiziente Abfragen stoßen, die zu Problemen wie dem gefürchteten n+1-Problem führten.

3. Langsame Entwicklungsiterationen: Viele Entwickler passen sie RESTful APIs an den Ablauf ihrer Anwendungen an. Wenn ihre Anwendungen jedoch wachsen, können sowohl die Front- als auch die Backends umfangreiche Änderungen erfordern. Infolgedessen passen sie sich APIs möglicherweise nicht mehr effizient oder wirkungsvoll der Form der Daten an. Dies führt aufgrund der Notwendigkeit von API-Änderungen zu langsameren Produktiterationen.
4. Skalierbare Leistung: Aufgrund dieser sich verschärfenden Probleme gibt es viele Bereiche, in denen die Skalierbarkeit beeinträchtigt wird. Die Leistung auf der Anwendungsseite kann beeinträchtigt werden, weil Ihre Anfragen zu viele oder zu wenig Daten zurückgeben (was zu mehr Anfragen führt). In beiden Situationen wird das Netzwerk unnötig belastet, was zu einer schlechten Leistung führt. Auf Seiten der Entwickler kann die Geschwindigkeit der Entwicklung reduziert werden, weil Sie nicht mehr richtig APIs sind und nicht mehr zu den Daten passen, die sie anfordern.

Das Verkaufsargument von GraphQL besteht darin, die Nachteile von REST zu überwinden. Hier sind einige der wichtigsten Lösungen, die GraphQL Entwicklern bietet:

1. Einzelne Endpunkte: GraphQL verwendet einen einzigen Endpunkt, um Daten abzufragen. Es ist nicht erforderlich, mehrere zu erstellen, APIs um sie an die Form der Daten anzupassen. Dies führt dazu, dass weniger Anfragen über das Netzwerk gesendet werden.
2. Abrufen: GraphQL löst die ständigen Probleme des Über- und Unterabrufs, indem es einfach die Daten definiert, die Sie benötigen. Mit GraphQL können Sie die Daten an Ihre Bedürfnisse anpassen, sodass Sie nur das erhalten, wonach Sie gefragt haben.
3. Abstraktion: GraphQL APIs enthält einige Komponenten und Systeme, die die Daten mithilfe eines sprachunabhängigen Standards beschreiben. Mit anderen Worten, Form und Struktur der Daten

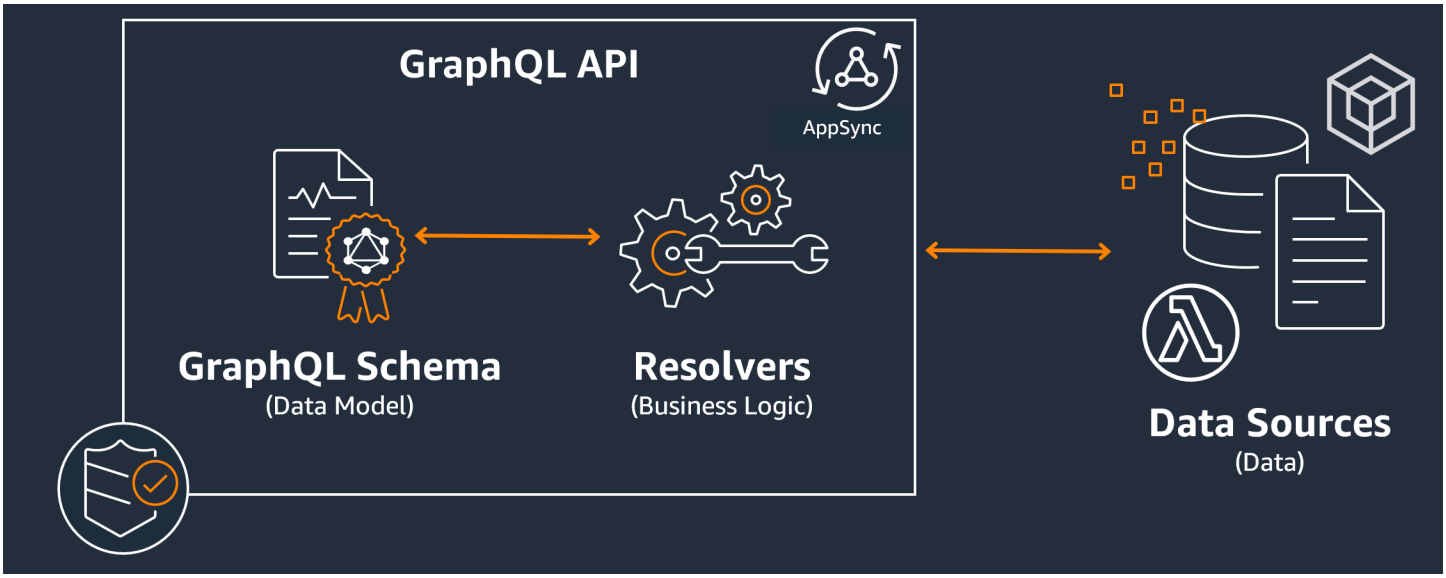
sind standardisiert, sodass sowohl das Front- als auch das Backend wissen, wie sie über das Netzwerk gesendet werden. Dies ermöglicht es Entwicklern an beiden Enden, mit den Systemen von GraphQL zu arbeiten und nicht um sie herum.

4. Schnelle Iterationen: Aufgrund der Standardisierung der Daten sind Änderungen an einem Ende der Entwicklung am anderen Ende möglicherweise nicht erforderlich. Beispielsweise führen Änderungen an der Frontend-Präsentation möglicherweise nicht zu umfangreichen Änderungen am Backend, da GraphQL die einfache Änderung der Datenspezifikation ermöglicht. Sie können einfach die Form der Daten definieren oder ändern, um sie an die Anforderungen der Anwendung anzupassen, wenn sie wächst. Dies führt zu weniger potenziellem Entwicklungsaufwand.

Dies sind nur einige der Vorteile von GraphQL. In den nächsten Abschnitten erfahren Sie, wie GraphQL strukturiert ist und welche Eigenschaften es zu einer einzigartigen Alternative zu REST machen.

Komponenten einer GraphQL-API

Eine Standard-GraphQL-API besteht aus einem einzigen Schema, das die Form der abgefragten Daten behandelt. Ihr Schema ist mit einer oder mehreren Ihrer Datenquellen wie einer Datenbank oder einer Lambda-Funktion verknüpft. Dazwischen befinden sich ein oder mehrere Resolver, die die Geschäftslogik für Ihre Anfragen verwalten. Jede Komponente spielt eine wichtige Rolle in Ihrer GraphQL-Implementierung. In den folgenden Abschnitten werden diese drei Komponenten und ihre Rolle im GraphQL-Dienst vorgestellt.



Themen

- GraphQL-Schemas
- Datenquellen
- Resolver

GraphQL-Schemas

Das GraphQL-Schema ist die Grundlage einer GraphQL-API. Es dient als Blaupause, die die Form Ihrer Daten definiert. Es ist auch ein Vertrag zwischen Ihrem Client und Ihrem Server, der festlegt, wie Ihre Daten abgerufen und/oder geändert werden.

GraphQL-Schemas sind in der Schema Definition Language (SDL) geschrieben. SDL besteht aus Typen und Feldern mit einer etablierten Struktur:

- Typen: Mit Typen definiert GraphQL die Form und das Verhalten der Daten. GraphQL unterstützt eine Vielzahl von Typen, die später in diesem Abschnitt erklärt werden. Jeder Typ, der in Ihrem Schema definiert ist, enthält seinen eigenen Bereich. Innerhalb des Bereichs befinden sich ein oder mehrere Felder, die einen Wert oder eine Logik enthalten können, die in Ihrem GraphQL-Dienst verwendet werden. Typen erfüllen viele verschiedene Rollen, am häufigsten sind Objekte oder Skalare (primitive Werttypen).
- Felder: Felder existieren innerhalb des Gültigkeitsbereichs eines Typs und enthalten den Wert, der vom GraphQL-Dienst angefordert wird. Diese sind Variablen in anderen Programmiersprachen sehr ähnlich. Die Form der Daten, die Sie in Ihren Feldern definieren, bestimmt, wie die Daten in einem Anforderungs-/Antwortvorgang strukturiert werden. Auf diese Weise können Entwickler vorhersagen, was zurückgegeben wird, ohne zu wissen, wie das Backend des Dienstes implementiert ist.

Um zu visualisieren, wie ein Schema aussehen würde, schauen wir uns den Inhalt eines einfachen GraphQL-Schemas an. Im Produktionscode befindet sich Ihr Schema normalerweise in einer Datei namens `schema.graphql` oder`schema.json`. Nehmen wir an, wir schauen uns ein Projekt an, das einen GraphQL-Dienst implementiert. In diesem Projekt werden Personaldaten des Unternehmens gespeichert, und die `schema.graphql` Datei wird verwendet, um Personaldaten abzurufen und neues Personal zu einer Datenbank hinzuzufügen. Der Code könnte so aussehen:

schema.graphql

```
type Person {
   id: ID!
```

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

Wir können sehen, dass im Schema drei Typen definiert sind: `Person`, `Query` und `Mutation`. Wenn wir uns das an sehen `Person`, können wir vermuten, dass dies die Blaupause für eine Instanz eines Unternehmensmitarbeiters ist, was diesen Typ zu einem Objekt machen würde. In seinem Geltungsbereich sehen wir, `id` `name`, und `age`. Dies sind die Felder, die die Eigenschaften von a definieren `Person`. Das bedeutet, dass unsere Datenquelle jeden `Person` Typ `name` als `String` skalaren (primitiven) Typ und `age` als `Int` skalaren (primitiven) Typ speichert. Der `id` fungiert als besonderer, eindeutiger Bezeichner für jeden. `Person` Es ist auch ein erforderlicher Wert, wie durch das `!` Symbol gekennzeichnet.

Die nächsten beiden Objekttypen verhalten sich unterschiedlich. GraphQL reserviert einige Schlüsselwörter für spezielle Objekttypen, die definieren, wie die Daten im Schema aufgefüllt werden. Ein `Query` Typ ruft Daten aus der Quelle ab. In unserem Beispiel könnte unsere Abfrage `Person` Objekte aus einer Datenbank abrufen. Dies erinnert Sie möglicherweise an `GET` Operationen in der RESTful Terminologie. A `Mutation` wird Daten ändern. In unserem Beispiel kann unsere Mutation der Datenbank weitere `Person` Objekte hinzufügen. Das erinnert Sie vielleicht an Operationen zur Änderung des Zustands wie `PUT` oder `POST`. Das Verhalten aller speziellen Objekttypen wird später in diesem Abschnitt erklärt.

Nehmen wir an, dass `Query` in unserem Beispiel etwas aus der Datenbank abgerufen wird. Wenn wir uns die Felder von an sehen `Query`, sehen wir ein Feld namens `people`. Sein Feldwert ist `[Person]`. Das bedeutet, dass wir eine Instanz von `Person` in der Datenbank abrufen möchten. Das Hinzufügen von Klammern bedeutet jedoch, dass wir eine Liste aller `Person` Instanzen zurückgeben möchten und nicht nur eine bestimmte.

Der `Mutation` Typ ist für die Ausführung von Zustandsänderungsoperationen wie Datenänderungen verantwortlich. Eine Mutation ist dafür verantwortlich, dass eine Operation zur Änderung des Zustands an der Datenquelle ausgeführt wird. In unserem Beispiel enthält unsere Mutation eine

aufgerufene OperationaddPerson, die der Datenbank ein neues Person Objekt hinzufügt. Die Mutation verwendet ein Person und erwartet eine Eingabe für die age Felder idname, und.

An dieser Stelle wundern Sie sich vielleicht, wie Operationen wie ohne eine Code-Implementierung addPerson funktionieren, da sie angeblich ein gewisses Verhalten ausführt und einer Funktion mit einem Funktionsnamen und Parametern sehr ähnlich sieht. Derzeit funktioniert es nicht, da ein Schema nur als Deklaration dient. Um das Verhalten von zu implementierenaddPerson, müssten wir ihm einen Resolver hinzufügen. Ein Resolver ist eine Codeeinheit, die immer dann ausgeführt wird, wenn das zugehörige Feld (in diesem Fall die addPerson Operation) aufgerufen wird. Wenn Sie eine Operation verwenden möchten, müssen Sie die Resolver-Implementierung irgendwann hinzufügen. In gewisser Weise können Sie sich die Schemaoperation als Funktionsdeklaration und den Resolver als Definition vorstellen. Resolver werden in einem anderen Abschnitt erklärt.

Dieses Beispiel zeigt nur die einfachsten Möglichkeiten, wie ein Schema Daten manipulieren kann. Sie erstellen komplexe, robuste und skalierbare Anwendungen, indem Sie die Funktionen von GraphQL nutzen und. AWS AppSync Im nächsten Abschnitt definieren wir all die verschiedenen Typen und Feldverhalten, die Sie in Ihrem Schema verwenden können.

GraphQL-Typen

GraphQL unterstützt viele verschiedene Typen. Wie Sie im vorherigen Abschnitt gesehen haben, definieren Typen die Form oder das Verhalten Ihrer Daten. Sie sind die grundlegenden Bausteine eines GraphQL-Schemas.

Typen können in Eingaben und Ausgaben eingeteilt werden. Eingaben sind Typen, die als Argument für die speziellen Objekttypen (Query, usw.) übergeben werden dürfenMutation, wohingegen Ausgabetypen ausschließlich zum Speichern und Zurückgeben von Daten verwendet werden. Eine Liste der Typen und ihrer Kategorisierungen ist unten aufgeführt:

- Objekte: Ein Objekt enthält Felder, die eine Entität beschreiben. Zum Beispiel könnte ein Objekt so etwas wie ein sein, book mit Feldern, die seine Eigenschaften beschreiben authorNamepublishingYear, wie, usw. Es handelt sich ausschließlich um Ausgabetypen.
- Skalare: Dies sind primitive Typen wie int, string usw. Sie werden normalerweise Feldern zugewiesen. Anhand des authorName Felds als Beispiel könnte ihm der String Skalar zugewiesen werden, um einen Namen wie „John Smith“ zu speichern. Skalare können sowohl Eingabe- als auch Ausgabetypen sein.
- Eingaben: Eingaben ermöglichen es Ihnen, eine Gruppe von Feldern als Argument zu übergeben. Sie sind sehr ähnlich strukturiert wie Objekte, können aber als Argumente an spezielle Objekte

übergeben werden. Mithilfe von Eingaben können Sie Skalare, Aufzählungen und andere Eingaben in ihrem Gültigkeitsbereich definieren. Eingaben können nur Eingabetypen sein.

- Spezialobjekte: Spezielle Objekte führen zustandsverändernde Operationen durch und übernehmen den Großteil der Schwerstarbeit des Dienstes. Es gibt drei spezielle Objekttypen: Abfrage, Mutation und Abonnement. Abfragen rufen in der Regel Daten ab; Mutationen manipulieren Daten; Abonnements öffnen und halten eine bidirektionale Verbindung zwischen Clients und Servern aufrecht, sodass eine ständige Kommunikation gewährleistet ist. Spezielle Objekte können aufgrund ihrer Funktionalität weder eingegeben noch ausgegeben werden.
- Enums: Enums sind vordefinierte Listen zulässiger Werte. Wenn Sie eine Aufzählung aufrufen, können ihre Werte nur den Werten entsprechen, die in ihrem Gültigkeitsbereich definiert sind. Wenn Sie beispielsweise eine Aufzählung mit dem Namen „`trafficLights`Darstellung einer Liste von Verkehrssignalen“ haben, könnte sie Werte wie `redLight` und `greenLight` aber nicht haben. `purpleLight` Eine echte Ampel hat nur eine begrenzte Anzahl von Signalen, sodass Sie diese anhand der Aufzählung definieren und sie bei der Referenzierung als einzig zulässige Werte erzwingen könnten. `trafficLight` Aufzählungen können sowohl Eingabe- als auch Ausgabetypen sein.
- Unions/Interfaces: Unions ermöglichen es Ihnen, ein oder mehrere Dinge in einer Anfrage zurückzugeben, abhängig von den Daten, die vom Client angefordert wurden. Wenn Sie beispielsweise einen Typ mit einem `title` Feld und einen `Book` Typ mit einem Feld hätten, könnten Sie eine `name` Vereinigung zwischen beiden `Author` Typen erstellen. Wenn Ihr Kunde eine Datenbank nach dem Ausdruck „Julius Caesar“ abfragen möchte, könnte die Vereinigung Julius Caesar (das Stück von William Shakespeare) aus dem `Book` `title` und Julius Caesar (den Autor von Commentarii de Bello Gallico) aus dem zurückgeben. `Author` `name` Bei Unions kann es sich nur um Ausgabetypen handeln.

  Schnittstellen sind Gruppen von Feldern, die Objekte implementieren müssen. Dies ist ein bisschen vergleichbar mit Schnittstellen in Programmiersprachen wie Java, wo Sie die in der Schnittstelle definierten Felder implementieren müssen. Nehmen wir zum Beispiel an, Sie haben eine Schnittstelle namens erstelltBook, die ein `title` Feld enthält. Nehmen wir an, Sie haben später einen Typ namens `Novel` Implemented erstelltBook. Sie `Novel` müssten ein `title` Feld einschließen. Sie `Novel` könnten jedoch auch andere Felder hinzufügen, die nicht in der Benutzeroberfläche enthalten sind, wie `pageCount` oder`ISBN`. Schnittstellen können nur Ausgabetypen sein.

In den folgenden Abschnitten wird erklärt, wie die einzelnen Typen in GraphQL funktionieren.

Objekte

GraphQL-Objekte sind der Haupttyp, den Sie im Produktionscode sehen werden. In GraphQL können Sie sich ein Objekt als eine Gruppierung verschiedener Felder vorstellen (ähnlich wie Variablen in anderen Sprachen), wobei jedes Feld durch einen Typ (normalerweise ein Skalar oder ein anderes Objekt) definiert wird, der einen Wert enthalten kann. Objekte stellen eine Dateneinheit dar, die aus Ihrer Serviceimplementierung abgerufen oder bearbeitet werden kann.

Objekttypen werden mit dem Schlüsselwort deklariert. `Type` Lassen Sie uns unser Schemabeispiel leicht modifizieren:

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

type Occupation {
  title: String
}
```

Die Objekttypen hier sind `Person` und`Occupation`. Jedes Objekt hat seine eigenen Felder mit eigenen Typen. Eine Funktion von GraphQL ist die Möglichkeit, Felder auf andere Typen festzulegen. Sie können sehen, dass das `occupation` Feld in einen `Occupation` Objekttyp `Person` enthält. Wir können diese Assoziation herstellen, weil GraphQL nur die Daten beschreibt und nicht die Implementierung des Dienstes.

Skalare

Skalare sind im Wesentlichen primitive Typen, die Werte enthalten. In gibt AWS AppSync es zwei Arten von Skalaren: die standardmäßigen GraphQL-Skalare und Skalare. AWS AppSync Skalare werden normalerweise verwendet, um Feldwerte innerhalb von Objekttypen zu speichern. Zu den standardmäßigen GraphQL-Typen gehören `Int`,`Float`,`String`,`Boolean`, und`ID`. Lassen Sie uns das vorherige Beispiel noch einmal verwenden:

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

```
}

type Occupation {
  title: String
}
```

Wenn wir die `title` Felder `name` und herausgreifen, enthalten beide einen `String` Skalar. `Name`könnte einen Zeichenkettenwert wie "`John Smith`" zurückgeben und der Titel könnte etwas wie "" `firefighter` zurückgeben. Einige GraphQL-Implementierungen unterstützen auch benutzerdefinierte Skalare, die das `Scalar` Schlüsselwort verwenden und das Verhalten des Typs implementieren. Unterstützt AWS AppSync derzeit jedoch keine benutzerdefinierten Skalare. Eine Liste der Skalare finden Sie unter Skalartypen in. AWS AppSync

Eingaben

Aufgrund des Konzepts der Eingabe- und Ausgabetypen gelten bestimmte Einschränkungen bei der Übergabe von Argumenten. Typen, die üblicherweise übergeben werden müssen, insbesondere Objekte, sind eingeschränkt. Sie können den Eingabetyp verwenden, um diese Regel zu umgehen. Eingaben sind Typen, die Skalare, Aufzählungen und andere Eingabetypen enthalten.

Eingaben werden mit dem Schlüsselwort definiert: `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
```

```
}
```

Wie Sie sehen können, können wir separate Eingaben haben, die den ursprünglichen Typ nachahmen. Diese Eingaben werden häufig in Ihren Feldoperationen wie folgt verwendet:

```
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
}
```

Beachten Sie, dass wir immer noch übergebenoccupationInput, anstatt eine Occupation zu erstellenPerson.

Dies ist nur ein Szenario für Eingaben. Sie müssen Objekte nicht unbedingt 1:1 kopieren, und im Produktionscode werden Sie ihn höchstwahrscheinlich nicht so verwenden. Es empfiehlt sich, GraphQL-Schemas zu nutzen, indem Sie nur das definieren, was Sie als Argumente eingeben müssen.

Dieselben Eingaben können auch in mehreren Operationen verwendet werden, aber wir empfehlen, dies nicht zu tun. Jede Operation sollte idealerweise eine eigene eindeutige Kopie der Eingaben enthalten, falls sich die Anforderungen des Schemas ändern.

Besondere Objekte

GraphQL reserviert einige Schlüsselwörter für spezielle Objekte, die einen Teil der Geschäftslogik dafür definieren, wie Ihr Schema Daten abruft/manipuliert. In einem Schema kann höchstens eines dieser Schlüsselwörter vorkommen. Sie dienen als Einstiegspunkte für alle angeforderten Daten, die Ihre Clients für Ihren GraphQL-Dienst ausführen.

Spezielle Objekte werden ebenfalls mit dem `type` Schlüsselwort definiert. Obwohl sie anders als normale Objekttypen verwendet werden, ist ihre Implementierung sehr ähnlich.

Queries

Abfragen sind `GET` Operationen insofern sehr ähnlich, als sie einen schreibgeschützten Abruf durchführen, um Daten aus Ihrer Quelle abzurufen. In GraphQL `Query` definiert der alle Einstiegspunkte für Clients, die Anfragen an Ihren Server stellen. In Ihrer GraphQL-Implementierung wird es immer eine `Query` geben.

Hier sind die `Query` und die modifizierten Objekttypen, die wir in unserem vorherigen Schemabeispiel verwendet haben:

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

Unser `Query` enthält ein Feld namens`people`, das eine Liste von `Person` Instanzen aus der Datenquelle zurückgibt. Nehmen wir an, wir müssen das Verhalten unserer Anwendung ändern, und jetzt müssen wir eine Liste nur der `Occupation` Instanzen für einen bestimmten Zweck zurückgeben. Wir könnten es einfach zur Abfrage hinzufügen:

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

In GraphQL können wir unsere Abfrage als einzige Quelle für Anfragen behandeln. Wie Sie sehen können, ist dies potenziell viel einfacher als RESTful Implementierungen, die möglicherweise unterschiedliche Endpunkte verwenden, um dasselbe zu erreichen (`.../api/1/people`und). `.../api/1/occupations`

Angenommen, wir haben eine Resolver-Implementierung für diese Abfrage, können wir jetzt eine tatsächliche Abfrage durchführen. Obwohl der `Query` Typ existiert, müssen wir ihn explizit aufrufen, damit er im Code der Anwendung ausgeführt werden kann. Dies kann mit dem `query` Schlüsselwort geschehen:

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

Wie Sie sehen können, wird diese Abfrage aufgerufen `getItems` und gibt `people` (eine Liste von `Person` Objekten) und `occupations` (eine Liste von `Occupation` Objekten) zurück. `people`In geben wir nur das `name` Feld von jedem zurück`Person`, während wir jeweils das `title` Feld zurückgeben`Occupation`. Die Antwort könnte so aussehen:

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

```
    ]
  }
}
```

Die Beispielantwort zeigt, wie die Daten der Form der Abfrage folgen. Jeder abgerufene Eintrag wird im Bereich des Felds aufgeführt. `people`und geben `occupations` Dinge als separate Listen zurück. Obwohl nützlich, könnte es bequemer sein, die Abfrage so zu ändern, dass sie eine Liste mit Namen und Berufen von Personen zurückgibt:

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

Dies ist eine legale Änderung, da unser `Person` Typ ein `occupation` Feld vom Typ `Occupation` enthält. Wenn es im Geltungsbereich von aufgeführt `Person` ist`people`, geben wir jedes Feld `name` zusammen mit dem zugehörigen Wert `Occupation` von zurück`title`. Die Antwort könnte so aussehen:

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

```
}
```

Mutations

Mutationen ähneln zustandsverändernden Operationen wie `PUT` oder`POST`. Sie führen einen Schreibvorgang durch, um Daten in der Quelle zu ändern, und rufen dann die Antwort ab. Sie definieren Ihre Einstiegspunkte für Anfragen zur Datenänderung. Im Gegensatz zu Abfragen kann eine Mutation je nach den Anforderungen des Projekts in das Schema aufgenommen werden oder auch nicht. Hier ist die Mutation aus dem Schemabeispiel:

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

Das `addPerson` Feld stellt einen Einstiegspunkt dar, der der Datenquelle a `Person` hinzufügt. `addPerson`ist der Feldname;`id`,`name`, und `age` sind die Parameter; und `Person` ist der Rückgabetyp. Rückblick auf den `Person` Typ:

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

Wir haben das `occupation` Feld hinzugefügt. Wir können dieses Feld jedoch nicht `Occupation` direkt auf setzen, da Objekte nicht als Argumente übergeben werden können; es handelt sich ausschließlich um Ausgabetypen. Wir sollten stattdessen eine Eingabe mit denselben Feldern als Argument übergeben:

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

Wir können unsere auch einfach aktualisieren`addPerson`, um dies als Parameter einzubeziehen, wenn wir neue `Person` Instanzen erstellen:

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

Hier ist das aktualisierte Schema:

```
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
}
```

Beachten Sie, dass das `title` Feld von übergeben `occupation` wird`occupationInput`, um die Erstellung des Objekts `Person` anstelle des ursprünglichen `Occupation` Objekts abzuschließen. Unter der Annahme, dass wir eine Resolver-Implementierung für haben`addPerson`, können wir jetzt eine tatsächliche Mutation durchführen. Obwohl der `Mutation` Typ existiert, müssen wir ihn explizit aufrufen, damit er im Code der Anwendung ausgeführt wird. Dies kann mit dem `mutation` Schlüsselwort geschehen:

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

Diese Mutation wird als Operation bezeichnet `createPerson` und `addPerson` ist die Operation. Um eine neue zu erstellen`Person`, können wir die Argumente für`id`, `name``age`, und eingeben`occupation`. Im Rahmen von `addPerson` können wir auch andere Felder wie

nameage, usw. sehen. Dies ist Ihre Antwort. Dies sind die Felder, die nach Abschluss des addPerson Vorgangs zurückgegeben werden. Hier ist der letzte Teil des Beispiels:

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

Mit dieser Mutation könnte ein Ergebnis so aussehen:

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

Wie Sie sehen können, hat die Antwort die von uns angeforderten Werte in demselben Format zurückgegeben, das in unserer Mutation definiert war. Es empfiehlt sich, alle Werte zurückzugeben, die geändert wurden, um Verwirrung zu vermeiden und die Notwendigkeit weiterer Abfragen in der future zu verringern. Mutationen ermöglichen es Ihnen, mehrere Operationen in ihren Geltungsbereich einzubeziehen. Sie werden nacheinander in der Reihenfolge ausgeführt, die in der Mutation angegeben ist. Wenn wir beispielsweise eine weitere Operation mit dem Namen erstellenadd0ccupation, die Berufsbezeichnungen zur Datenquelle hinzufügt, können wir dies in der nachfolgenden Mutation aufrufen. addPerson addPersonwird zuerst bearbeitet, gefolgt vonadd0ccupation.

Subscriptions

Abonnements dienen WebSocketsdazu, eine dauerhafte bidirektionale Verbindung zwischen dem Server und seinen Clients herzustellen. In der Regel abonniert ein Client den Server oder hört ihn ab. Immer wenn der Server eine serverseitige Änderung vornimmt oder ein Ereignis ausführt, erhält der abonnierte Client die Updates. Dieser Protokolltyp ist nützlich, wenn mehrere Clients abonniert sind und über Änderungen auf dem Server oder anderen Clients informiert werden müssen. Abonnements können beispielsweise verwendet werden, um Social-Media-Feeds zu aktualisieren. Es könnte zwei Benutzer geben, Benutzer A und Benutzer B, die beide automatische Benachrichtigungen abonniert haben, wenn sie Direktnachrichten erhalten. Benutzer A auf Client A könnte eine Direktnachricht an Benutzer B auf Client B senden. Der Client von Benutzer A würde die Direktnachricht senden, die vom Server verarbeitet würde. Der Server würde dann die Direktnachricht an das Konto von Benutzer B senden und gleichzeitig eine automatische Benachrichtigung an Client B senden.

Hier ist ein Beispiel für a`Subscription`, das wir dem Schemabeispiel hinzufügen könnten:

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

Das `personAdded` Feld sendet eine Nachricht an abonnierte Kunden, wenn der Datenquelle eine neue hinzugefügt `Person` wird. Vorausgesetzt, wir haben eine Resolver-Implementierung für`personAdded`, können wir jetzt das Abonnement verwenden. Obwohl der `Subscription` Typ existiert, müssen wir ihn explizit aufrufen, damit er im Code der Anwendung ausgeführt wird. Dies kann mit dem `subscription` Schlüsselwort geschehen:

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

Das Abonnement wird aufgerufen`personAddedOperation`, und der Vorgang ist`personAdded`. `personAdded`gibt die `name` Felder `id` und und neuer `Person` Instanzen zurück. Wenn wir uns das Mutationsbeispiel ansehen, haben wir `Person` mit dieser Operation eine hinzugefügt:

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

Wenn unsere Kunden Updates für die neu hinzugefügten Apps abonniert hättenPerson, könnten sie nach dem Start Folgendes sehenaddPerson:

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

Im Folgenden finden Sie eine Zusammenfassung dessen, was Abonnements bieten:

Abonnements sind bidirektionale Kanäle, die es dem Client und dem Server ermöglichen, schnelle, aber stetige Updates zu erhalten. Sie verwenden in der Regel das WebSocket Protokoll, das standardisierte und sichere Verbindungen herstellt.

Abonnements sind insofern flexibel, als sie den Aufwand für den Verbindungsaufbau reduzieren. Einmal abonniert, kann ein Kunde dieses Abonnement einfach für längere Zeit nutzen. Sie nutzen Computerressourcen in der Regel effizient, indem sie es Entwicklern ermöglichen, die Laufzeit des Abonnements individuell zu gestalten und zu konfigurieren, welche Informationen angefordert werden.

Im Allgemeinen ermöglichen Abonnements dem Kunden, mehrere Abonnements gleichzeitig abzuschließen. Apropos AWS AppSync, Abonnements werden nur für den Empfang von Echtzeit-Updates vom AWS AppSync Dienst verwendet. Sie können nicht zur Durchführung von Abfragen oder Mutationen verwendet werden.

Die wichtigste Alternative zu Abonnements ist das Polling, bei dem Abfragen in festgelegten Intervallen gesendet werden, um Daten anzufordern. Dieser Prozess ist in der Regel weniger effizient als Abonnements und belastet sowohl den Client als auch das Backend stark.

Eine Sache, die in unserem Schemabeispiel nicht erwähnt wurde, war die Tatsache, dass Ihre speziellen Objekttypen auch in einem `schema` Root-Verzeichnis definiert werden müssen. Wenn Sie also ein Schema exportieren AWS AppSync, könnte es so aussehen:

schema.graphql

```
schema {
```

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

.
.
.

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

Aufzählungen

Aufzählungen oder Aufzählungen sind spezielle Skalare, die die zulässigen Argumente einschränken, die ein Typ oder Feld haben kann. Das bedeutet, dass immer dann, wenn eine Aufzählung im Schema definiert wird, der zugehörige Typ oder das zugehörige Feld auf die Werte in der Aufzählung beschränkt wird. Aufzählungen werden als Zeichenkettenskalare serialisiert. Beachten Sie, dass verschiedene Programmiersprachen GraphQL-Enums möglicherweise unterschiedlich handhaben. JavaScript Hat beispielsweise keine native Enum-Unterstützung, sodass die Enum-Werte stattdessen Int-Werten zugeordnet werden können.

Aufzählungen werden mit dem Schlüsselwort definiert. `enum` Ein Beispiel:

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

Beim Aufrufen der `trafficLights` Enumeration können die Argumente nur`solidRed`, `solidYellow``solidGreen`, usw. sein. Es ist üblich, Aufzählungen zu verwenden, um Dinge darzustellen, für die es eine bestimmte, aber begrenzte Anzahl von Auswahlmöglichkeiten gibt.

Unionen/Schnittstellen

Siehe Schnittstellen und Unions in GraphQL.

GraphQL-Felder

Felder existieren innerhalb des Gültigkeitsbereichs eines Typs und enthalten den Wert, der vom GraphQL-Dienst angefordert wird. Diese sind Variablen in anderen Programmiersprachen sehr ähnlich. Hier ist zum Beispiel ein `Person` Objekttyp:

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

Die Felder sind in diesem Fall `name` und `age` und enthalten jeweils einen `String` `Int` Und-Wert. Objektfelder wie die oben gezeigten können als Eingaben in den Feldern (Operationen) Ihrer Abfragen und Mutationen verwendet werden. Sehen Sie sich zum Beispiel Folgendes `Query` an:

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

Das `people` Feld fordert alle Instanzen von `Person` aus der Datenquelle an. Wenn Sie a `Person` in Ihrem GraphQL-Server hinzufügen oder abrufen, können Sie davon ausgehen, dass die Daten dem Format Ihrer Typen und Felder folgen, das heißt, die Struktur Ihrer Daten im Schema bestimmt, wie sie in Ihrer Antwort strukturiert werden:

```
}
  "data": {
    "people": [
      {
        "name": "John Smith",
        "age": "50"
      },
      {
```

```
        "name": "Andrew Miller",
        "age": "60"
      },
      .
      .
      .
    ]
  }
}
```

Felder spielen eine wichtige Rolle bei der Strukturierung von Daten. Im Folgenden werden einige zusätzliche Eigenschaften erläutert, die zur weiteren Anpassung auf Felder angewendet werden können.

Listen

Listen geben alle Elemente eines bestimmten Typs zurück. Mit Hilfe von Klammern kann dem Typ eines Felds eine Liste hinzugefügt werden`[]`:

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

In `Query` `Person` geben die umgebenden Klammern an, dass Sie alle Instanzen von `Person` aus der Datenquelle als Array zurückgeben möchten. In der Antwort `Person` werden die `age` Werte `name` und jeweils als eine einzige, durch Trennzeichen getrennte Liste zurückgegeben:

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

```
      .                              # Data of Person N
      .
      .
    ]
  }
}
```

Sie sind nicht auf spezielle Objekttypen beschränkt. Sie können Listen auch in den Feldern regulärer Objekttypen verwenden.

Werte, die nicht NULL sind

Werte, die nicht NULL sind, geben ein Feld an, das in der Antwort nicht Null sein darf. Sie können ein Feld mit dem folgenden Symbol auf einen Wert ungleich Null setzen: !

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

Das name Feld kann nicht explizit Null sein. Wenn Sie die Datenquelle abfragen und eine Nulleingabe für dieses Feld angeben würden, würde ein Fehler ausgegeben.

Sie können Listen und Nicht-Null-Werte kombinieren. Vergleichen Sie diese Abfragen:

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

.
.
.

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

.
.
```

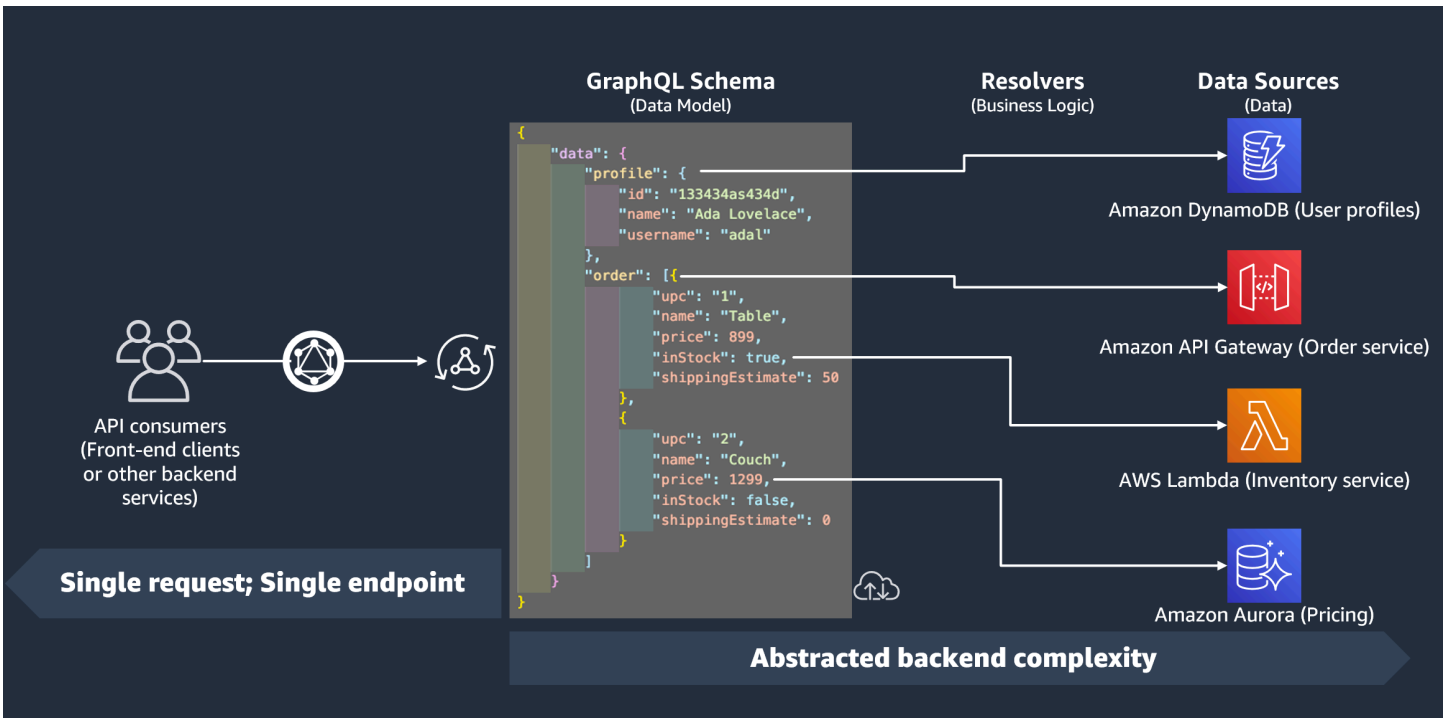
```
.

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

In Anwendungsfall 1 darf die Liste keine Nullelemente enthalten. Im Anwendungsfall 2 kann die Liste selbst nicht auf Null gesetzt werden. Im Anwendungsfall 3 dürfen die Liste und ihre Elemente nicht Null sein. In jedem Fall können Sie jedoch immer noch leere Listen zurückgeben.

Wie Sie sehen können, gibt es in GraphQL viele bewegliche Komponenten. In diesem Abschnitt haben wir die Struktur eines einfachen Schemas und die verschiedenen Typen und Felder gezeigt, die ein Schema unterstützt. Im folgenden Abschnitt erfahren Sie mehr über die anderen Komponenten einer GraphQL-API und wie sie mit dem Schema funktionieren.

Datenquellen

Im vorherigen Abschnitt haben wir gelernt, dass ein Schema die Form Ihrer Daten definiert. Wir haben jedoch nie erklärt, woher diese Daten stammen. In echten Projekten ist Ihr Schema wie ein Gateway, das alle Anfragen an den Server verarbeitet. Wenn eine Anfrage gestellt wird, fungiert das Schema als einziger Endpunkt, der mit dem Client verbunden ist. Das Schema greift auf Daten aus der Datenquelle zu, verarbeitet sie und leitet sie zurück an den Client weiter. Sehen Sie sich die folgende Infografik an:



AWS AppSync und GraphQL implementieren hervorragend Backend For Frontend (BFF) -Lösungen. Sie arbeiten zusammen, um die Komplexität skalierbar zu reduzieren, indem sie das Backend abstrahieren. Wenn Ihr Service unterschiedliche Datenquellen und/oder Microservices verwendet, können Sie im Wesentlichen einen Teil der Komplexität abstrahieren, indem Sie die Form der Daten jeder Quelle (Untergraph) in einem einzigen Schema (Supergraph) definieren. Das bedeutet, dass Ihre GraphQL-API nicht auf die Verwendung einer Datenquelle beschränkt ist. Sie können Ihrer GraphQL-API eine beliebige Anzahl von Datenquellen zuordnen und in Ihrem Code angeben, wie sie mit dem Dienst interagieren sollen.

Wie Sie in der Infografik sehen können, enthält das GraphQL-Schema alle Informationen, die Clients benötigen, um Daten anzufordern. Das bedeutet, dass alles in einer einzigen Anfrage verarbeitet werden kann und nicht in mehreren Anfragen, wie es bei REST der Fall ist. Diese Anfragen durchlaufen das Schema, das der einzige Endpunkt des Dienstes ist. Wenn Anfragen verarbeitet werden, führt ein Resolver (im nächsten Abschnitt erklärt) seinen Code aus, um die Daten aus der entsprechenden Datenquelle zu verarbeiten. Wenn die Antwort zurückgegeben wird, wird der mit der Datenquelle verknüpfte Untergraph mit den Daten im Schema gefüllt.

AWS AppSync unterstützt viele verschiedene Datenquellentypen. In der folgenden Tabelle werden wir jeden Typ beschreiben, einige der Vorteile der einzelnen Typen auflisten und nützliche Links für zusätzlichen Kontext bereitstellen.

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen

Amazon-DynamoDB	„Amazon DynamoDB ist ein vollständig verwalteter NoSQL-Datenbankservice, der schnelle und vorhersehbare Leistung mit nahtloser Skalierbarkeit bietet. Mit DynamoDB können Sie den Verwaltungsaufwand für den Betrieb und die Skalierung	• Leistung im großen Maßstab: DynamoDB ist auf konsisten te Leistung in jeder Größenord nung ausgelegt. Dies ist durch die Verwendung von Partitionen möglich. DynamoDB partition iert Ihre Tabellen automatisch in	• Offizielle DynamoDB-Dokumentation
			  • Partitionen
			  • Auto-Scaling
			  • Fehlertoleranz
			  • Überwachung
			  • Sicherheit
			• GraphQL und DynamoDB
			• Resolver-Operation en für DynamoDB

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
	verteilter Datenbanken verringern, sodass Sie sich nicht um Hardwarebereitstellung, Einrichtung und Konfiguration, Replikation, Software-Patching oder Cluster-Skalierung kümmern müssen. DynamoDB bietet auch Verschlüsselung im Ruhezustand, wodurch der betriebliche Aufwand und die Komplexität, die mit dem Schutz sensibler Daten verbunden sind, entfallen.“	mehrere Zuordnungen, die in mehreren SSDs auf mehreren Knoten gespeichert werden. Dies erhöht im Allgemeinen den Netzwerkdurchsatz und reduziert die Latenz. • Skalierbare Kapazität: DynamoDB überwacht Ihren Datenverkehr und ermöglicht es Ihnen, Ihren Durchsatz automatisch zu skalieren, falls das Netzwerk über einen längeren Zeitraum überlastet bleibt. • Verfügbarkeit und Fehlertoleranz: DynamoDB wird von mehreren physisch isolierten Regionen unterstützt, die jeweils mehrere physisch isolierte Availability Zones enthalten	• Preismodell

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
		. DynamoDB wechselt im Falle einer Dienstunterbrechung automatisch in eine Backup-Zone. Zur Datensicherheit können Sie Ihre Daten auch manuell sichern und replizieren. • Protokollierung und Überwachung: DynamoDB bietet verschiedene Analysetools für Ihre Tabellen. Sie können die Leistung Ihrer Tabelle überwachen und Alarme einrichten, um Sie über drastische Änderungen am Service zu informieren. • Sicherheit: DynamoDB befolgt strenge Protokolle, um sicherzustellen, dass Ihre Daten den Sicherheitsanforderungen	

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
		Ihres Unternehmens entsprechen. • Integration mit AWS AppSync: DynamoDB ist nahtlos in unseren Service integriert. Sie können neue DynamoDB-Tabellen erstellen und daraus automatisch ein Schema generieren, um Ihren Entwicklungsprozess zu optimieren. Wir bieten auch eine ganze Sammlung von Vorgängen, mit denen Sie auf einfache Weise Daten aus vorhandenen DynamoDB-Tabellen in Ihrem Konto in Ihrem Resolver anfordern können.	

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen

AWS Lambda	"AWS Lambda ist ein Rechendienst, mit dem Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen.
	
	Lambda führt Ihren Code auf einer hochverfügbaren Recheninfrastruktur aus und führt die gesamte Verwaltung der Rechenressourcen durch, einschließlich Server- und Betriebssystemwartung, Kapazitätsbereitstellung und automatischer Skalierung sowie Protokollierung. Mit Lambda müssen Sie lediglich Ihren Code in einer der von Lambda unterstützten Sprachlaufzeiten bereitstellen.“
		• Pay-as-you-use Modell: Lambda berechnet Ihnen nur Gebühren, wenn Sie ihre Ressourcen nutzen. Sie ermöglichen es Ihnen auch, die Menge der verwendeten Ressourcen entsprechend Ihren Anwendungsanforderungen zu skalieren.
		• Automatische Skalierung: Manchmal benötigt Ihre Anwendung zusätzliche Rechenleistung für einen bestimmten Prozess. Lambda ermöglicht es Ihnen, Rechenressourcen automatisch an die Anforderungen Ihrer Anwendung anzupassen.
		• Schnellere Bereitstellungszeiten: Mit einem Bereitste
			• Offizielle Dokumentation
			  • Skalierung
			  • Einsatz
			  • Laufzeiten
			• Anleitung zum Lambda-Resolver
			• Preismodell

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
		llungspaket können Sie Ihren Entwicklungsprozess rationalisieren. Verwenden Sie ein Paket, um Ihren Funktionscode in den Lambda-Service hochzuladen. Sie können dann ihre Laufzeitumgebungen verwenden, um Ihre Funktionen zu testen und auszuführen. • Vielseitigkeit: Lambda kann in einer Vielzahl von Anwendungsfällen eingesetzt werden. Sie können Lambda nahtlos in Dienste und AWS Dienste von Drittanbietern integrieren. Einige Beispiele hierfür sind CI/CD-Pipelines und Massenmailing-Dienste. • Integration mit AWS AppSync:	

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
		Sie können Ihre Lambda-Funktionen einfach in Ihrem Resolver aufrufen, um Anfragen zu bearbeiten. Unser Service bietet einen optimierten Anforderungsvorgang zur Durchführung von Lambda-Aufrufen. Wir erlauben sowohl einzelne als auch gebündelte Anrufe.	

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
OpenSearch	„Amazon OpenSearch Service ist ein verwalteter Service, der es einfach macht, OpenSearch Cluster in der AWS Cloud bereitzustellen, zu betreiben und zu skalieren. Amazon OpenSearch Service unterstützt OpenSearch ältere Elasticsearch-Betriebssysteme (bis zu 7.10, die endgültige Open-Source-Version der Software). Wenn Sie einen Cluster erstellen, haben Sie die Option, die Suchmaschine zu verwenden. OpenSearchist eine vollständig quelloffene Such- und Analyse-Engine für Anwendungsfälle wie Protokollanalysen, Anwendungsüberwachung in Echtzeit und Clickstream-Analyse. Weitere Informationen finden	• Skalierung: Mit OpenSearch Serverless können Sie den Service ganz einfach an Ihre Servicean forderungen anpassen. • Datenaufnahme: Sie können Ingestion verwenden OpenSearch , um Daten zu importieren, zu verarbeiten und zu analysieren. Es gibt viele Anwendungen für die Datenaufnahme, die Sie hier finden. • Sicherheit: OpenSearch kann Ihre AWS Sicherheitskonfiguration einschließlich IAM,, CloudTrail VPCs, Authentifizierung usw. verwalten. • Verfügbarkeit: Unterstützt in seinem Service OpenSearch auch	• Offizielle Dokumentation • Serverless • Preismodell

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
	Sie in der OpenSearch-Dokumentation. Amazon OpenSearch Service stellt alle Ressourcen für Ihren OpenSearch Cluster bereit und startet ihn. Außerdem werden ausgefallene OpenSearch Serviceknoten automatisch erkannt und ersetzt, wodurch der mit selbstverwalteten Infrastrukturen verbundene Aufwand reduziert wird. Sie können Ihren Cluster mit einem einzigen API-Aufruf oder mit ein paar Klicks in der Konsole skalieren.“	verschiedene Regionen und Availability Zones. • Integration mit AWS AppSync: In können Sie GraphQL verwenden AWS AppSync, APIs um Daten aus vorhandenen OpenSearch Dienstdomänen in Ihrem Konto zu speichern und abzurufen.	

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
HTTP-Endpunkte	Sie können HTTP-Endpunkte als Datenquellen verwenden. AWS AppSync kann Anfragen mit den relevanten Informationen wie Parametern und Nutzdaten an die Endpunkte senden. Die HTTP-Antwort wird dem Resolver zugänglich gemacht, der die endgültige Antwort zurückgibt, nachdem er seine Operation(en) abgeschlossen hat.	• Nützlich für einfache Anwendungen, die nicht so stark in Dienste wie Lambda integriert sind.	• Referenz zum Resolver

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen

Amazon EventBridge	"EventBridge ist ein serverloser Service, der Ereignisse verwendet, um Anwendungskomponenten miteinander zu verbinden, sodass Sie leichter skalierbare, ereignisgesteuerte Anwendungen erstellen können. Verwenden Sie ihn, um Ereignisse aus Quellen wie selbst entwickelten Anwendungen, AWS Diensten und Software von Drittanbietern an Verbrauchseranwendungen in Ihrem Unternehmen weiterzuleiten. EventBridge bietet eine einfache und konsistente Methode zum Erfassen, Filtern, Transformieren und Bereitstellen von Ereignissen, sodass Sie schnell neue Anwendungen erstellen können.“	• Ereignisgesteuerte Architektur: Sie können die Vorteile einer ereignisgesteuerten Architektur nutzen.
		• Planung: Sie können den EventBridge Scheduler verwenden, um Ihre Aufgaben und Regeln mithilfe von Cron-Ausdrücken zu automatisieren oder Zeitintervalle als Alternative zu Ereignismustern festzulegen.
		• Pipes: Mithilfe von EventBridge Pipes können Sie den Event-Bus durch eine Pipe ersetzen, die zusätzliche Filterereignismuster und eine Anreicherung durch Datentransformationen beinhaltet, bevor Sie das Ereignis an das Ziel senden.
			• Offizielle Dokumentation
			  • Rohre
			  • Scheduler
			• Resolver-Referenz
			• Preismodell

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
		• Integration mit AWS AppSync: AWS AppSync ermöglicht es Ihnen, Ereignisse mithilfe Ihres Resolvers an Event-Busse zu senden.	

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen

Relationale Datenbanken	„Amazon Relational Database Service (Amazon RDS) ist ein Webservice, der die Einrichtung, den Betrieb und die Skalierung einer relationalen Datenbank in der AWS Cloud erleichtert. Er bietet kosteneffiziente, anpassbare Kapazität für eine relationale Datenbank nach Industriestandard und verwaltet allgemeine Datenbankverwaltungsaufgaben.“	• Verwaltung leicht gemacht: RDS führt regelmäßig Wartungsarbeiten an seinen Ressourcen durch. Die Wartung umfasst in den meisten Fällen Aktualisierungen der der DB-Instance zugrunde liegenden Hardware, des zugrunde liegenden Betriebssystems (OS) oder der Version der Datenbank-Engine. Unter normalen Umständen können Sie entscheiden, wann Updates durchgeführt werden sollen (Ausnahmen umfassen Sicherheitspatches).
		• Empfehlungen: Die Empfehlungsfunktion von RDS bietet automatisierte Vorschläge zur Behebung potenziel
			• Offizielle Dokumentation
			  • Funktionen
			  • Wartung
			  • Empfehlungen
			  • Speicheroptionen
			  • Verfügbarkeit
			  • Sicherheit
			• Preismodell

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
		ler Probleme in Ihrer Instanz. • Verfügbarkeit: RDS ist in verschiedenen physischen Regionen auf der ganzen Welt verfügbar. Sie können Ihre Datenbankanforderungen problemlos auf verschiedene Knoten verteilen, um Ihren Kunden einen besseren Service zu bieten. • Anpassung: RDS ist auf die Anforderungen großer Unternehmen zugeschnitten. RDS bietet verschiedene Optionen für Datenverarbeitung, schnelle Bereitstellung, Skalierbarkeit und Speicherung. • Sicherheit: RDS ist in mehrere Tools und Dienste integriert, um die Datenbanksicherhei	

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
		t auf Benutzer-, Datenbank- und Netzwerkebene aufrechtzuerhalten. • Integration mit AWS AppSync: Wenn Sie nach einer ausgereif ten Backend-L ösung suchen, AWS AppSync können Sie mit Ihrer Instance als Datenquelle Daten senden, verarbeit en, speichern und zurückgeben.	

Datenquelle	Beschreibung	Vorteile	Zusätzliche Informationen
Keine Datenquelle	Wenn Sie nicht vorhaben, einen Datenquellendienst zu verwenden, können Sie ihn auf einstellen none. Eine none Datenquelle ist zwar immer noch explizit als Datenquelle kategorisiert, aber kein Speicherm edium. Trotzdem ist es in bestimmten Fällen immer noch nützlich, um Daten zu manipulieren und weiterzuleiten.	• Potenziell nützlich für Dinge wie Datenkonvertierung • Nützlich, wenn Sie etwas lokal lösen	• Resolver-Referenz

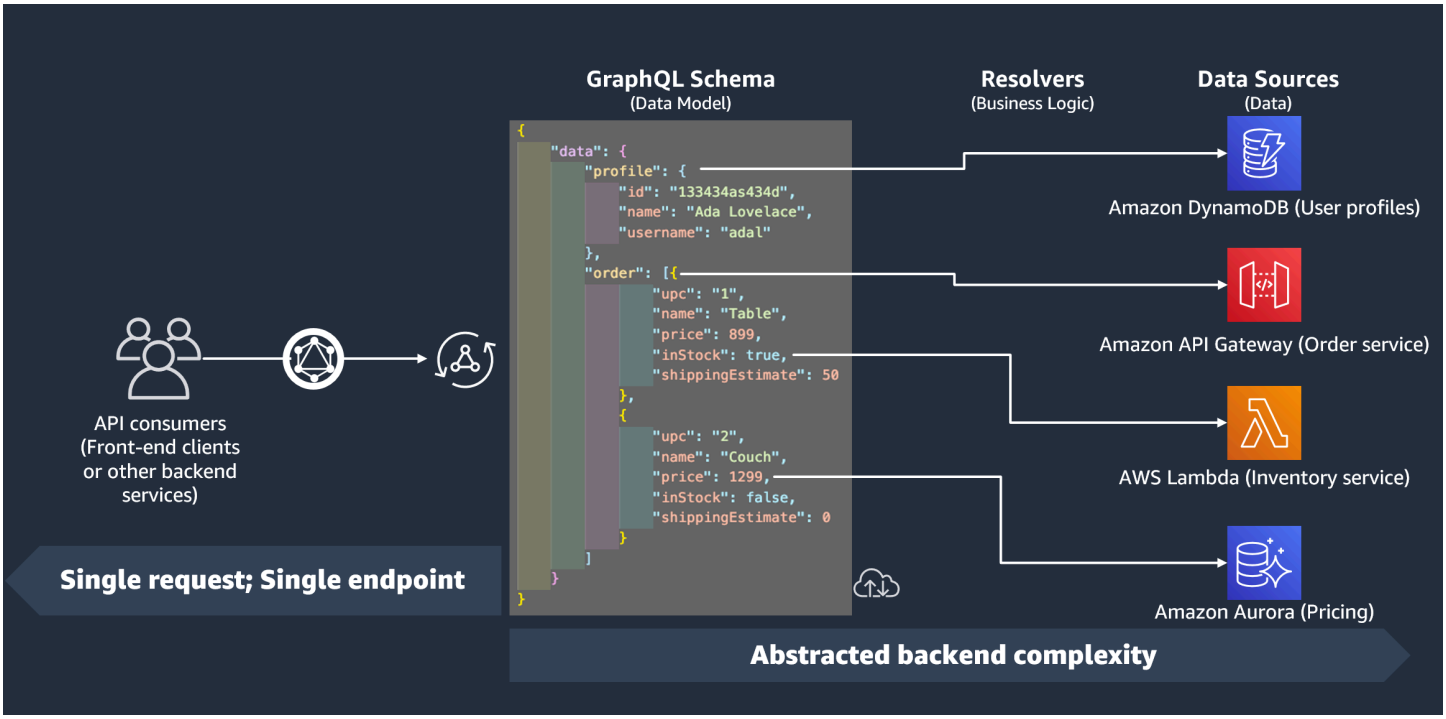
> **Tip**
>
> Weitere Informationen darüber, wie Datenquellen miteinander interagieren AWS AppSync, finden Sie unter Anhängen einer Datenquelle.

Resolver

In den vorherigen Abschnitten haben Sie sich mit den Komponenten des Schemas und der Datenquelle vertraut gemacht. Jetzt müssen wir uns damit befassen, wie das Schema und die Datenquellen interagieren. Alles beginnt mit dem Resolver.

Ein Resolver ist eine Codeeinheit, die regelt, wie die Daten dieses Felds aufgelöst werden, wenn eine Anfrage an den Dienst gestellt wird. Resolver sind an bestimmte Felder innerhalb Ihrer Typen in Ihrem Schema angehängt. Sie werden am häufigsten verwendet, um die

Zustandsänderungsoperationen für Ihre Abfrage-, Mutations- und Abonnement-Feldoperationen zu implementieren. Der Resolver verarbeitet die Anfrage eines Clients und gibt dann das Ergebnis zurück. Dabei kann es sich um eine Gruppe von Ausgabetypen wie Objekten oder Skalaren handeln:



Laufzeit des Resolvers

AWS AppSync In müssen Sie zunächst eine Laufzeit für Ihren Resolver angeben. Eine Resolver-Runtime gibt die Umgebung an, in der ein Resolver ausgeführt wird. Sie bestimmt auch die Sprache, in der Ihre Resolver geschrieben werden. AWS AppSync unterstützt derzeit APPSYNC_JS für JavaScript und Velocity Template Language (VTL). Weitere Informationen finden Sie unter JavaScript Laufzeitfunktionen für Resolver und Funktionen für JavaScript oder Referenz zum Resolver Mapping Template Utility für VTL.

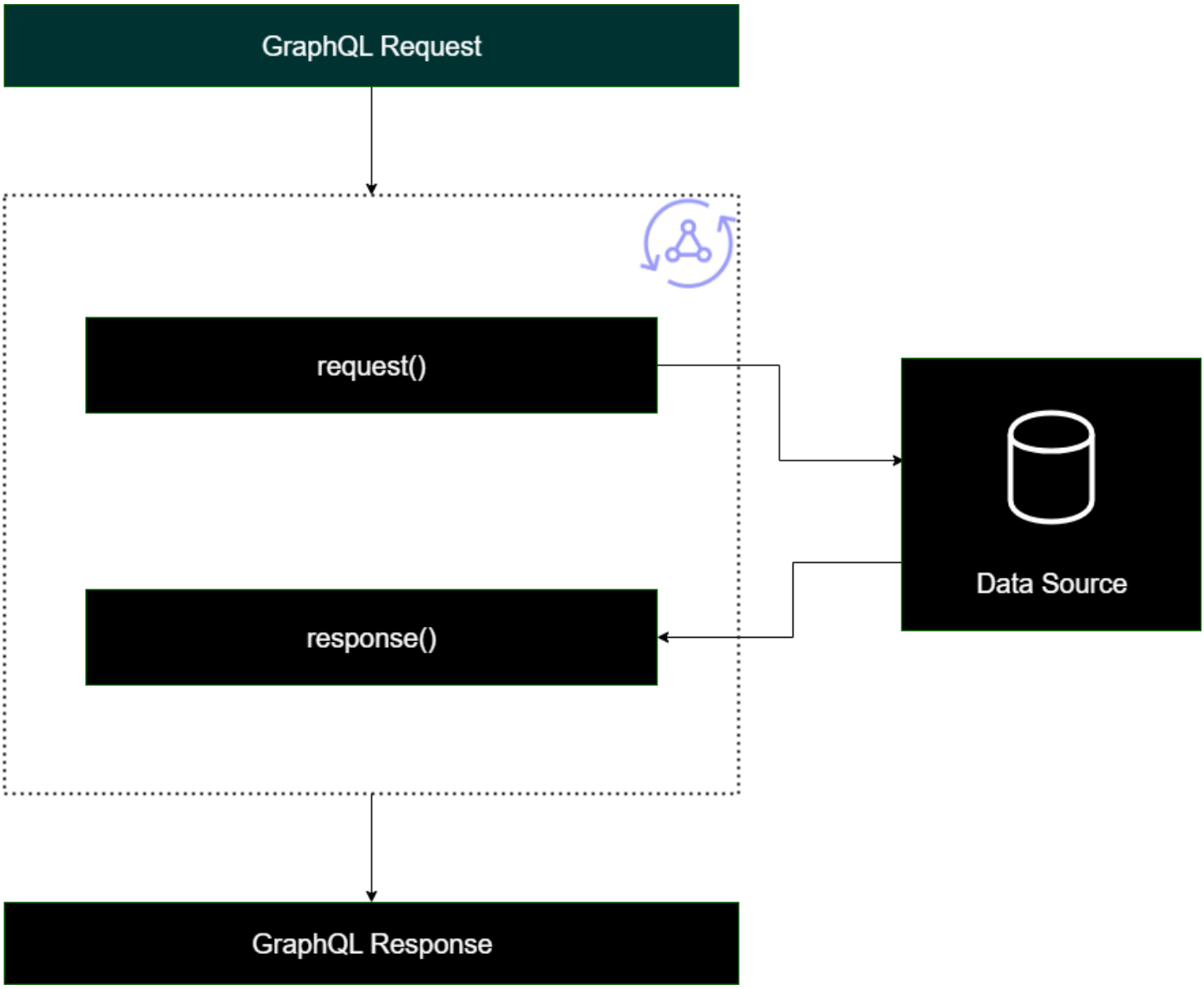
Resolver-Struktur

Was den Code angeht, können Resolver auf verschiedene Arten strukturiert werden. Es gibt Unit - und Pipeline-Resolver.

Resolver für Einheiten

Ein Unit-Resolver besteht aus Code, der einen einzelnen Anforderungs- und Antworthandler definiert, die für eine Datenquelle ausgeführt werden. Der Anforderungshandler verwendet ein Kontextobjekt als Argument und gibt die Anforderungsnutzdaten zurück, die zum Aufrufen Ihrer Datenquelle verwendet wurden. Der Antworthandler erhält von der Datenquelle eine Nutzlast mit dem Ergebnis

der ausgeführten Anfrage zurück. Der Response-Handler wandelt die Nutzlast in eine GraphQL-Antwort um, um das GraphQL-Feld aufzulösen.

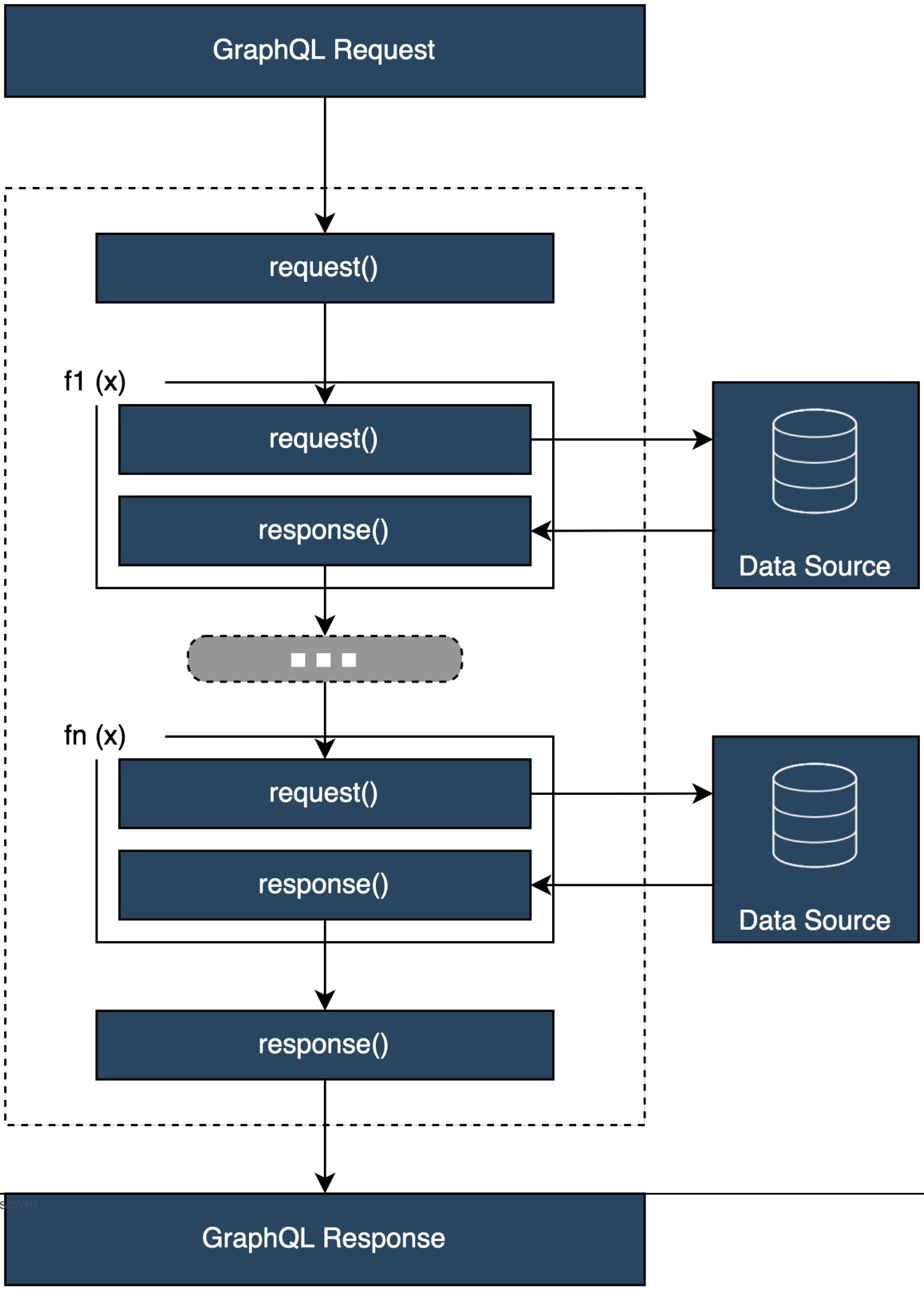


Pipeline-Resolver

Bei der Implementierung von Pipeline-Resolvern gibt es eine allgemeine Struktur, der sie folgen:

- Schritt vor dem: Wenn eine Anfrage vom Client gestellt wird, werden den Resolvern für die verwendeten Schemafelder (normalerweise Ihre Abfragen, Mutationen, Abonnements) die Anforderungsdaten übergeben. Der Resolver beginnt mit der Verarbeitung der Anforderungsdaten mit einem Before-Step-Handler, der es ermöglicht, einige Vorverarbeitungsvorgänge durchzuführen, bevor die Daten den Resolver passieren.

- Funktion (en): Nachdem der vorherige Schritt ausgeführt wurde, wird die Anforderung an die Funktionsliste übergeben. Die erste Funktion in der Liste wird für die Datenquelle ausgeführt. Eine Funktion ist eine Teilmenge des Codes Ihres Resolvers, die einen eigenen Anfrage- und Antworthandler enthält. Ein Request-Handler nimmt die Anforderungsdaten und führt Operationen an der Datenquelle durch. Der Antworthandler verarbeitet die Antwort der Datenquelle, bevor er sie an die Liste zurückgibt. Wenn es mehr als eine Funktion gibt, werden die Anforderungsdaten zur Ausführung an die nächste Funktion in der Liste gesendet. Die Funktionen in der Liste werden seriell in der vom Entwickler festgelegten Reihenfolge ausgeführt. Sobald alle Funktionen ausgeführt wurden, wird das Endergebnis an den nächsten Schritt übergeben.
- Nach dem Schritt: Der Nachschritt ist eine Handler-Funktion, mit der Sie einige letzte Operationen an der Antwort der endgültigen Funktion ausführen können, bevor Sie sie an die GraphQL-Antwort übergeben.


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

Struktur des Resolver-Handlers

Handler sind in der Regel Funktionen, die aufgerufen `Request` werden und: `Response`

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

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

In einem Unit-Resolver wird es nur einen Satz dieser Funktionen geben. In einem Pipeline-Resolver wird es einen Satz dieser Funktionen für den Vorher-Nachher-Schritt und einen zusätzlichen Satz pro Funktion geben. Um zu veranschaulichen, wie das aussehen könnte, schauen wir uns einen einfachen `Query` Typ an:

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

Dies ist eine einfache Abfrage mit einem Feld namens `helloWorld` Typ`String`. Nehmen wir an, wir möchten immer, dass dieses Feld die Zeichenfolge „Hello World“ zurückgibt. Um dieses Verhalten zu implementieren, müssen wir den Resolver zu diesem Feld hinzufügen. In einem Unit-Resolver könnten wir so etwas hinzufügen:

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

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

Das `request` kann einfach leer gelassen werden, weil wir keine Daten anfordern oder verarbeiten. Wir können auch davon ausgehen, dass unsere Datenquelle dies ist`None`, was bedeutet, dass dieser Code keine Aufrufe ausführen muss. Die Antwort gibt einfach „Hello World“ zurück. Um diesen Resolver zu testen, müssen wir eine Anfrage mit dem Abfragetyp stellen:

```
query helloWorldTest {
```

```
  helloWorld
}
```

Dies ist eine Abfrage namens`helloWorldTest`, die das `helloWorld` Feld zurückgibt. Bei der Ausführung wird der `helloWorld` Field Resolver ebenfalls ausgeführt und gibt die Antwort zurück:

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

Konstanten wie diese zurückzugeben, ist das Einfachste, was Sie tun können. In Wirklichkeit werden Sie Eingaben, Listen und mehr zurückgeben. Hier ist ein komplizierteres Beispiel:

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

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

Hier geben wir eine Liste von zurück`Books`. Nehmen wir an, wir verwenden eine DynamoDB-Tabelle zum Speichern von Buchdaten. Unsere Handler könnten so aussehen:

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

Unsere Anfrage verwendete einen integrierten Scanvorgang, um nach allen Einträgen in der Tabelle zu suchen, speicherte die Ergebnisse im Kontext und übergab sie dann an die Antwort. In der Antwort wurden die Ergebniselemente übernommen und in der Antwort zurückgegeben:

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

        ...

      ]
    }
  }
}
```

Resolver-Kontext

In einem Resolver muss jeder Schritt in der Kette von Handlern den Status der Daten aus den vorherigen Schritten kennen. Das Ergebnis eines Handlers kann gespeichert und als Argument an einen anderen übergeben werden. GraphQL definiert vier grundlegende Resolver-Argumente:

Resolver-Basisargumente	Beschreibung
`obj`, `root`, `parent`, usw.	Das Ergebnis des übergeordneten Elements.
`args`	Die Argumente, die dem Feld in der GraphQL-Abfrage zur Verfügung gestellt werden.
`context`	Ein Wert, der jedem Resolver zur Verfügung gestellt wird und wichtige Kontextinformationen

Resolver-Basisargumente	Beschreibung
	wie den aktuell angemeldeten Benutzer oder den Zugriff auf eine Datenbank enthält.
`info`	Ein Wert, der feldspezifische Informationen enthält, die für die aktuelle Abfrage relevant sind, sowie die Schemadetails.

AWS AppSync In kann das Argument `context` (ctx) alle oben genannten Daten enthalten. Es ist ein Objekt, das pro Anfrage erstellt wird und Daten wie Autorisierungsdaten, Ergebnisdaten, Fehler, Anforderungsmetadaten usw. enthält. Der Kontext ist eine einfache Möglichkeit für Programmierer, Daten zu manipulieren, die aus anderen Teilen der Anfrage stammen. Nimm diesen Ausschnitt noch einmal:

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

Der Anfrage wird der Kontext (ctx) als Argument übergeben; dies ist der Status der Anfrage. Sie führt einen Scan für alle Elemente in einer Tabelle durch und speichert das Ergebnis anschließend wieder im `result` Kontext unter. Der Kontext wird dann an das Antwortargument übergeben, das auf den zugreift `result` und seinen Inhalt zurückgibt.

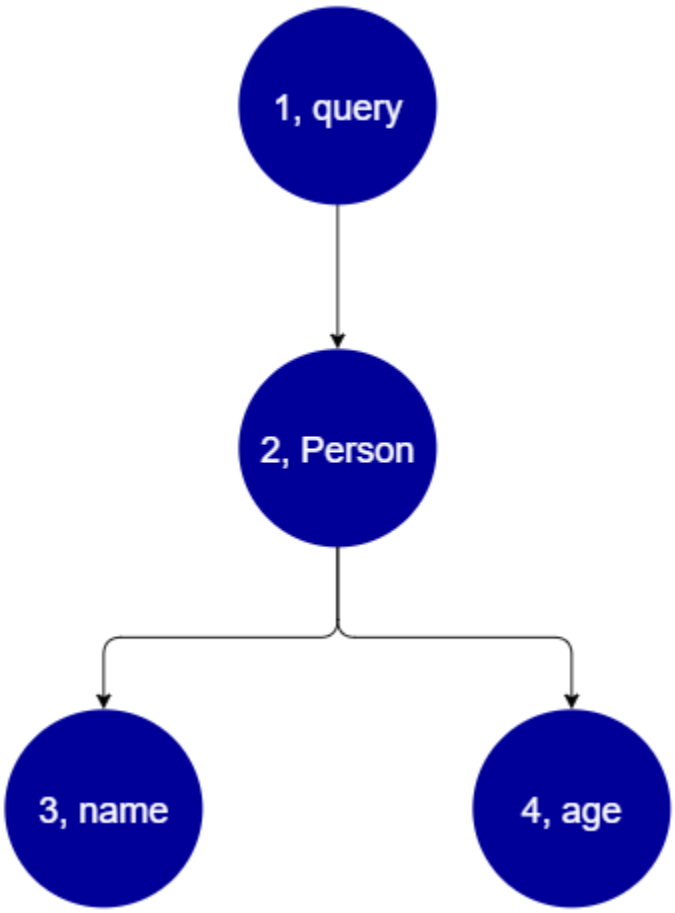
Anfragen und Analyse

Wenn Sie eine Abfrage an Ihren GraphQL-Dienst stellen, muss dieser vor der Ausführung einen Parsing- und Validierungsprozess durchlaufen. Ihre Anfrage wird analysiert und in einen abstrakten Syntaxbaum übersetzt. Der Inhalt des Baums wird validiert, indem mehrere Validierungsalgorithmen

anhand Ihres Schemas ausgeführt werden. Nach dem Validierungsschritt werden die Knoten des Baums durchsucht und verarbeitet. Resolver werden aufgerufen, die Ergebnisse werden im Kontext gespeichert und die Antwort wird zurückgegeben. Betrachten Sie beispielsweise diese Abfrage:

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

Wir kehren `Person` mit den Feldern A `name` und `age` zurück. Wenn Sie diese Abfrage ausführen, sieht der Baum ungefähr so aus:



Aus dem Baum geht hervor, dass diese Anfrage den Stamm nach dem `Query` im Schema durchsucht. Innerhalb der Abfrage wird das `Person` Feld aufgelöst. Aus früheren Beispielen wissen wir, dass dies eine Eingabe des Benutzers sein könnte, eine Werteliste usw., die höchstwahrscheinlich an einen Objekttyp gebunden `Person` ist, der die Felder enthält, die wir benötigen (`nameundage`). Sobald diese beiden untergeordneten Felder gefunden wurden, werden

sie in der angegebenen Reihenfolge (`name`gefolgt von`age`) aufgelöst. Sobald der Baum vollständig aufgelöst ist, ist die Anfrage abgeschlossen und wird an den Client zurückgesendet.

Zusätzliche Eigenschaften von GraphQL

GraphQL besteht aus mehreren Designprinzipien, um Einfachheit und Robustheit im großen Maßstab zu gewährleisten.

Deklarativ

GraphQL ist deklarativ, was bedeutet, dass der Benutzer die Daten beschreibt (formt), indem er nur die Felder deklariert, die er abfragen möchte. Die Antwort gibt nur die Daten für diese Eigenschaften zurück. Hier ist zum Beispiel eine Operation, die ein `Book` Objekt in einer DynamoDB-Tabelle mit dem ISBN 13-Wert von abruft: `id` *9780199536061*

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

Die Antwort gibt nur die Felder in der Payload (`name``year`, und) zurück und `author` sonst nichts:

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

Aufgrund dieses Entwurfsprinzips beseitigt GraphQL die ständigen Probleme des Über- und Unterabrufs, mit denen REST in komplexen Systemen zu APIs kämpfen hat. Dies führt zu einer effizienteren Datenerfassung und einer verbesserten Netzwerkleistung.

Hierarchical (Hierarchisch)

GraphQL ist insofern flexibel, als die angeforderten Daten vom Benutzer an die Bedürfnisse der Anwendung angepasst werden können. Die angeforderten Daten folgen immer den Typen und der Syntax der in Ihrer GraphQL-API definierten Eigenschaften. Der folgende Ausschnitt zeigt beispielsweise den `getBook` Vorgang mit einem neuen Feldbereich namens, der alle gespeicherten Anführungszeichenfolgen und Seiten zurückgibt`quotes`, die mit dem verknüpft sind: Book *9780199536061*

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

Das Ausführen dieser Abfrage gibt das folgende Ergebnis zurück:

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

Wie Sie sehen können, wurden die mit dem angeforderten Buch verknüpften `quotes` Felder als Array im gleichen Format zurückgegeben, das in unserer Abfrage beschrieben wurde. Obwohl es hier nicht gezeigt wurde, hat GraphQL den zusätzlichen Vorteil, dass es nicht genau weiß, wo sich die abgerufenen Daten befinden. `Books`und `quotes` könnte separat gespeichert werden, aber GraphQL ruft die Informationen trotzdem ab, solange die Zuordnung besteht. Das bedeutet, dass Ihre Abfrage eine Vielzahl von eigenständigen Daten in einer einzigen Anfrage abrufen kann.

Introspektiv

GraphQL ist selbstdokumentierend oder introspektiv. Es unterstützt mehrere integrierte Operationen, mit denen Benutzer die zugrunde liegenden Typen und Felder innerhalb des Schemas anzeigen können. Hier ist zum Beispiel ein `Foo` Typ mit einem `date` `description` Und-Feld:

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

Wir könnten die `_type` Operation verwenden, um die Typisierungsmetadaten unter dem Schema zu finden:

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

Dies wird eine Antwort zurückgeben:

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

Diese Funktion kann verwendet werden, um herauszufinden, welche Typen und Felder ein bestimmtes GraphQL-Schema unterstützt. GraphQL unterstützt eine Vielzahl dieser introspektiven Operationen. Weitere Informationen finden Sie unter Introspektion.

Starkes Tippen

GraphQL unterstützt starke Typisierung durch sein Typen- und Feldsystem. Wenn Sie etwas in Ihrem Schema definieren, muss es einen Typ haben, der vor der Laufzeit validiert werden kann. Es muss auch der Syntaxspezifikation von GraphQL entsprechen. Dieses Konzept unterscheidet sich nicht von der Programmierung in anderen Sprachen. Hier ist zum Beispiel der `Foo` Typ von früher:

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

Wir können sehen, dass das das Objekt `Foo` ist, das erstellt wird. In einer Instanz von `Foo` wird es ein `date` `description` Und-Feld geben, beide vom `String` primitiven Typ (Skalar). Syntaktisch gesehen sehen wir, dass das deklariert `Foo` wurde und dass seine Felder innerhalb seines Gültigkeitsbereichs existieren. Diese Kombination aus Typprüfung und logischer Syntax stellt sicher, dass Ihre GraphQL-API übersichtlich und selbstverständlich ist. Die Typisierungs- und Syntaxspezifikation von GraphQL finden Sie hier.

Erste Schritte: Erstellen Sie Ihre erste GraphQL-API in AWS AppSync

Sie können die AWS AppSync Konsole verwenden, um eine GraphQL-API zu konfigurieren und zu starten. GraphQL benötigt APIs im Allgemeinen drei Komponenten:

1. GraphQL-Schema — Ihr GraphQL-Schema ist die Blaupause der API. Es definiert die Typen und Felder, die Sie anfordern können, wenn eine Operation ausgeführt wird. Um das Schema mit Daten zu füllen, müssen Sie Datenquellen mit der GraphQL-API verbinden. In dieser Schnellstartanleitung erstellen wir ein Schema mit einem vordefinierten Modell.
2. Datenquellen — Dies sind die Ressourcen, die die Daten zum Auffüllen Ihrer GraphQL-API enthalten. Dies kann eine DynamoDB-Tabelle, eine Lambda-Funktion usw. sein, die eine Vielzahl von Datenquellen AWS AppSync unterstützt, um robustes und skalierbares GraphQL zu erstellen. APIs Datenquellen sind mit Feldern im Schema verknüpft. Immer wenn eine Anforderung für ein Feld ausgeführt wird, wird das Feld mit den Daten aus der Quelle gefüllt. Dieser Mechanismus wird vom Resolver gesteuert. In dieser Schnellstartanleitung erstellen wir eine Datenquelle, die neben dem Schema ein vordefiniertes Modell verwendet.
3. Resolver — Resolver sind dafür verantwortlich, das Schemafeld mit der Datenquelle zu verknüpfen. Sie rufen die Daten aus der Quelle ab und geben dann das Ergebnis auf der Grundlage dessen zurück, was durch das Feld definiert wurde. AWS AppSync unterstützt JavaScript sowohl VTL als auch VTL zum Schreiben von Resolvern für Ihr GraphQL. APIs In dieser Schnellstartanleitung werden die Resolver automatisch auf der Grundlage des Schemas und der Datenquelle generiert. Wir werden uns in diesem Abschnitt nicht damit befassen.

AWS AppSync unterstützt die Erstellung und Konfiguration aller GraphQL-Komponenten. Wenn Sie die Konsole öffnen, können Sie die folgenden Methoden verwenden, um Ihre API zu erstellen:

1. Entwerfen einer benutzerdefinierten GraphQL-API, indem sie über ein vordefiniertes Modell generiert und eine neue DynamoDB-Tabelle (Datenquelle) zur Unterstützung eingerichtet wird.
2. Entwerfen einer GraphQL-API mit einem leeren Schema und ohne Datenquellen oder Resolver.
3. Verwenden Sie eine DynamoDB-Tabelle, um Daten zu importieren und die Typen und Felder Ihres Schemas zu generieren.
4. Nutzung AWS AppSync der WebSocket Funktionen und der Pub/Sub-Architektur zur Entwicklung in Echtzeit. APIs

5. Verwenden von vorhandenem GraphQL APIs (Quelle APIs), um eine Verknüpfung zu einer zusammengeführten API herzustellen.

> **Note**
>
> Wir empfehlen, den Abschnitt Ein Schema entwerfen zu lesen, bevor Sie mit fortgeschritteneren Tools arbeiten. In diesen Leitfäden werden einfachere Beispiele erläutert, die Sie konzeptionell verwenden können, um komplexere Anwendungen zu erstellen. AWS AppSync

AWS AppSync unterstützt auch mehrere Optionen, die keine Konsole sind, um APIs GraphQL zu erstellen. Dazu zählen:

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

Das folgende Beispiel zeigt Ihnen, wie Sie die grundlegenden Komponenten einer GraphQL-API mithilfe vordefinierter Modelle und DynamoDB erstellen.

Themen

- Ein Schema in der Konsole starten AWS AppSync
- Ein Rundgang durch die Konsole AWS AppSync
- Verwenden von GraphQL-Mutationen zum Hinzufügen von Daten zu einer DynamoDB-Tabelle in der Konsole AWS AppSync
- Verwenden von GraphQL-Abfragen zum Abrufen von Daten aus einer DynamoDB-Tabelle in der Konsole AWS AppSync
- Zusätzliche Abschnitte für die Konsole AWS AppSync

Ein Schema in der Konsole starten AWS AppSync

In diesem Beispiel erstellen Sie eine Todo API, mit der Benutzer Todo Elemente für tägliche Arbeitserinnerungen wie `Finish task` oder erstellen können. `Pick up groceries` Diese

API demonstriert, wie GraphQL-Operationen verwendet werden, bei denen der Status in einer DynamoDB-Tabelle persistiert.

Konzeptionell gibt es drei Hauptschritte, um Ihre erste GraphQL-API zu erstellen. Sie müssen das Schema (Typen und Felder) definieren, Ihre Datenquelle (n) an Ihre Felder anhängen und dann den Resolver schreiben, der die Geschäftslogik verarbeitet. Die Konsolenerfahrung ändert jedoch die Reihenfolge. Wir definieren zunächst, wie unsere Datenquelle mit unserem Schema interagieren soll, und definieren dann später das Schema und den Resolver.

Um Ihre GraphQL-API zu erstellen

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
2. Wählen Sie im Dashboard Create API (API erstellen) aus.
3. Wenn GraphQL ausgewählt APIs ist, wählen Sie Design from scratch. Wählen Sie anschließend Weiter.
4. Ändern Sie für den API-Namen den vorausgefüllten Namen in **`Todo API`** und wählen Sie dann Weiter.

   > **Note**
   >
   > Es gibt hier auch andere Optionen, aber wir werden sie für dieses Beispiel nicht verwenden.

5. Gehen Sie im Abschnitt GraphQL-Ressourcen angeben wie folgt vor:

   a. Wählen Sie jetzt Typ erstellen, der von einer DynamoDB-Tabelle unterstützt wird.

      > **Note**
      >
      > Das bedeutet, dass wir eine neue DynamoDB-Tabelle erstellen werden, die als Datenquelle angehängt wird.

   b. Geben Sie im Feld Modellname ein. **`Todo`**

      > **Note**
      >
      > Unsere erste Anforderung besteht darin, unser Schema zu definieren. Dieser Modellname wird der Typname sein. Was Sie also wirklich tun, ist einen `type` Namen zu erstellen`Todo`, der im Schema vorhanden sein wird:

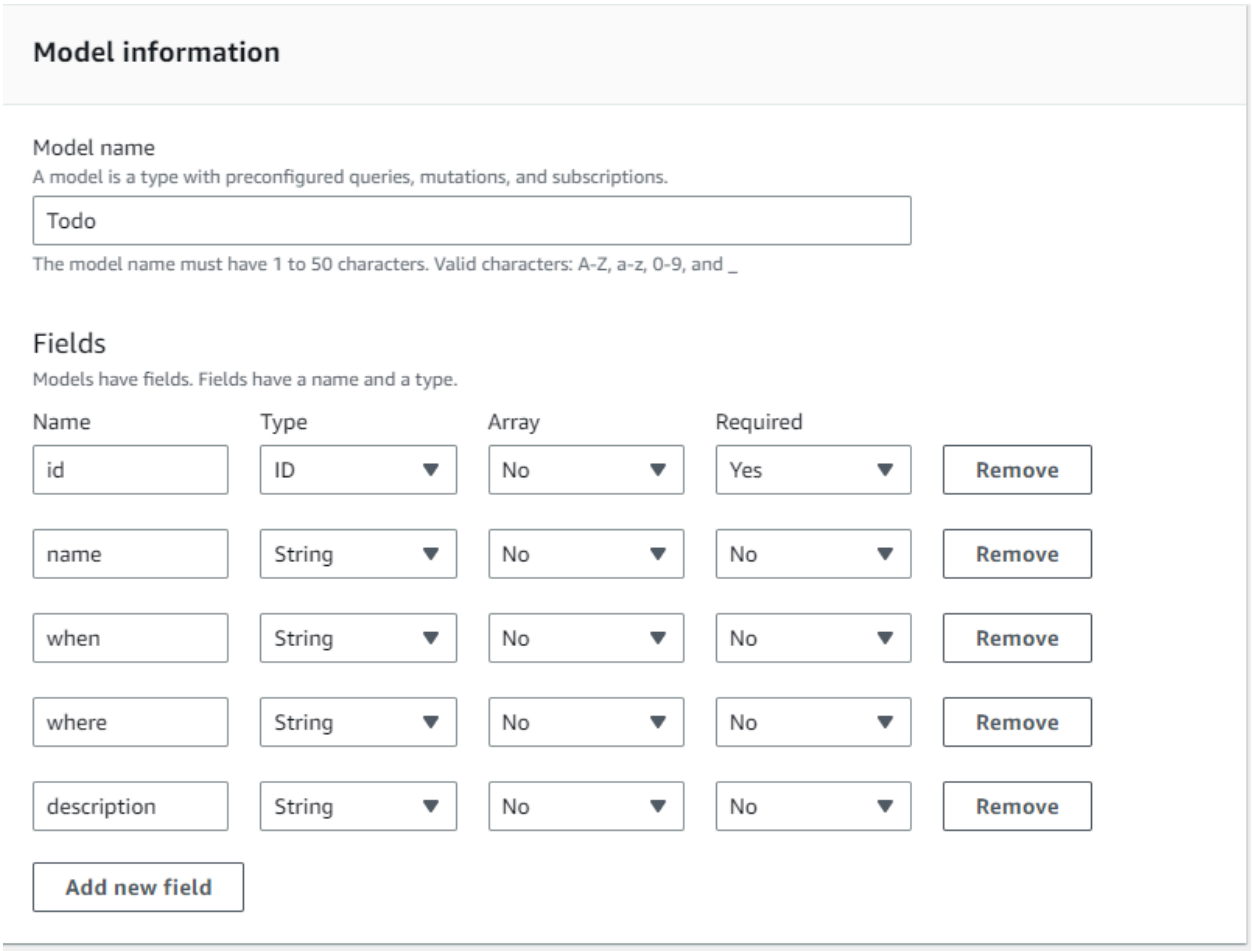
```
type Todo {}
```

c. Gehen Sie unter Felder wie folgt vor:

i. Erstellen Sie ein Feld mit dem Namen**`id`**, dem Typ `ID` und der erforderlichen Einstellung auf`Yes`.

> **Note**
>
> Dies sind die Felder, die innerhalb des Gültigkeitsbereichs Ihres `Todo` Typs existieren werden. Ihr Feldname wird hier `id` mit dem folgenden Typ aufgerufen`ID!`:
>
> ```
> type Todo {
>  id: ID!
> }
> ```
>
> AWS AppSync unterstützt mehrere Skalarwerte für verschiedene Anwendungsfälle.

ii. Erstellen Sie mit „Neues Feld hinzufügen“ vier zusätzliche Felder mit den `Name` Werten**`name`**, **`whenwhere`**, und**`description`**. Ihre `Type` Werte werden sein`String`, `Array` und die `Required` Werte und werden beide auf gesetzt`No`. Sie sieht wie folgt aus:



Note

Der vollständige Typ und seine Felder werden wie folgt aussehen:

```
type Todo {
 id: ID!
 name: String
 when: String
 where: String
 description: String
}
```

Da wir ein Schema unter Verwendung dieses vordefinierten Modells erstellen, wird es auch mit mehreren Standardmutationen gefüllt, die auf dem Typ basieren, z. B. `createdelete`, und `update` um Ihnen das einfache Auffüllen Ihrer Datenquelle zu erleichtern.

d. Geben Sie unter Modelltabelle konfigurieren einen Tabellennamen ein, z. B. **`TodoAPITable`** Stellen Sie den Primärschlüssel auf ein`id`.

> **Note**
>
> Wir erstellen im Wesentlichen eine neue DynamoDB-Tabelle namens*`TodoAPITable`*, die an die API als unsere primäre Datenquelle angehängt wird. Unser Primärschlüssel ist auf das erforderliche `id` Feld gesetzt, das wir zuvor definiert haben. Beachten Sie, dass diese neue Tabelle leer ist und außer dem Partitionsschlüssel nichts enthält.

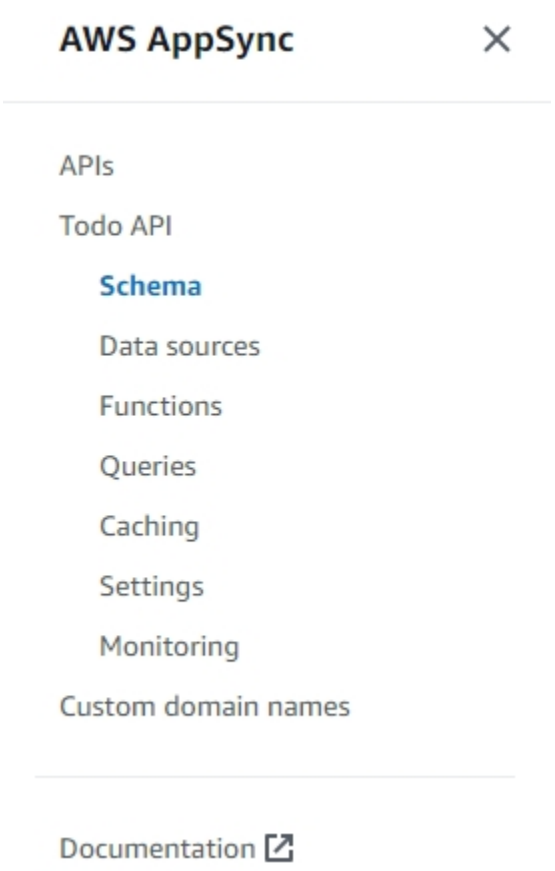
e. Wählen Sie Weiter.

6. Überprüfen Sie Ihre Änderungen und wählen Sie Create API. Warten Sie einen Moment, bis der AWS AppSync Service die Erstellung Ihrer API abgeschlossen hat.

Sie haben erfolgreich eine GraphQL-API mit ihrem Schema und der DynamoDB-Datenquelle erstellt. Um die obigen Schritte zusammenzufassen, haben wir uns entschieden, eine völlig neue GraphQL-API zu erstellen. Wir haben den Namen der API definiert und dann unsere Schemadefinition hinzugefügt, indem wir unseren ersten Typ hinzugefügt haben. Wir haben den Typ und seine Felder definiert und dann entschieden, eine Datenquelle an eines der Felder anzuhängen, indem wir eine neue DynamoDB-Tabelle ohne Daten erstellt haben.

Ein Rundgang durch die Konsole AWS AppSync

Bevor wir Daten zu unserer DynamoDB-Tabelle hinzufügen, sollten wir uns die grundlegenden Funktionen der AWS AppSync Konsolenerfahrung ansehen. Über die AWS AppSync Konsolenregisterkarte auf der linken Seite der Seite können Benutzer einfach zu den wichtigsten Komponenten oder Konfigurationsoptionen navigieren, die Folgendes bieten: AWS AppSync



Schema-Designer

Wählen Sie Schema, um das Schema anzuzeigen, das Sie gerade erstellt haben. Wenn Sie den Inhalt des Schemas überprüfen, werden Sie feststellen, dass es bereits mit einer Reihe von Hilfsoperationen geladen wurde, um den Entwicklungsprozess zu optimieren. Wenn Sie im Schema-Editor durch den Code blättern, gelangen Sie schließlich zu dem Modell, das Sie im vorherigen Abschnitt definiert haben:

```
type Todo {
 id: ID!
 name: String
 when: String
 where: String
 description: String
}
```

Ihr Modell wurde zum Basistyp, der in Ihrem gesamten Schema verwendet wurde. Wir werden damit beginnen, unserer Datenquelle Daten hinzuzufügen, indem wir Mutationen verwenden, die automatisch anhand dieses Typs generiert wurden.

Hier sind einige zusätzliche Tipps und Fakten zum Schema-Editor:

1. Der Code-Editor verfügt über Funktionen zur Linting- und Fehlerüberprüfung, die Sie beim Schreiben Ihrer eigenen Apps verwenden können.
2. Die rechte Seite der Konsole zeigt die erstellten GraphQL-Typen und -Resolver verschiedener Typen der höchsten Ebene an, z. B. Abfragen.
3. Wenn Sie einem Schema neue Typen hinzufügen (z. B.`type User {...}`), können Sie DynamoDB-Ressourcen für Sie AWS AppSync bereitstellen lassen. Dazu gehören der richtige Primärschlüssel, Sortierschlüssel und das Index-Design für Ihr GraphQL-Datenzugriffsmuster. Wenn Sie oben Create Resources (Ressourcen erstellen) und einen der benutzerdefinierten Typen im Menü wählen, können Sie verschiedene Feldoptionen im Schemadesign auswählen. Wir werden dies im Abschnitt „Ein Schema entwerfen“ behandeln.

Resolver-Konfiguration

Im Schema-Designer enthält der Abschnitt Resolvers alle Typen und Felder in Ihrem Schema. Wenn Sie durch die Liste der Felder blättern, werden Sie feststellen, dass Sie Resolver an bestimmte Felder anhängen können, indem Sie Anhängen wählen. Dadurch wird ein Code-Editor geöffnet, in dem Sie Ihren Resolver-Code schreiben können. AWS AppSync unterstützt sowohl VTL als auch JavaScript Runtimes. Diese können oben auf der Seite geändert werden, indem Sie Actions und dann Update Runtime wählen. Unten auf der Seite können Sie auch Funktionen erstellen, die mehrere Operationen nacheinander ausführen. Resolver sind jedoch ein fortgeschrittenes Thema, auf das wir in diesem Abschnitt nicht eingehen werden.

Datenquellen

Wählen Sie Datenquellen aus, um Ihre DynamoDB-Tabelle anzuzeigen. Wenn Sie die `Resource` Option auswählen (falls verfügbar), können Sie die Konfiguration Ihrer Datenquelle anzeigen. In unserem Beispiel führt dies zur DynamoDB-Konsole. Von dort aus können Sie Ihre Daten bearbeiten. Sie können einige Daten auch direkt bearbeiten, indem Sie die Datenquelle und dann Bearbeiten auswählen. Wenn Sie Ihre Datenquelle jemals löschen müssen, können Sie Ihre Datenquelle auswählen und dann Löschen auswählen. Schließlich können Sie neue Datenquellen erstellen, indem Sie Datenquelle erstellen auswählen und dann den Namen und den Typ konfigurieren.

Beachten Sie, dass diese Option dazu dient, den AWS AppSync Dienst mit einer vorhandenen Ressource zu verknüpfen. Sie müssen die Ressource dennoch mit dem entsprechenden Dienst in Ihrem Konto erstellen, bevor sie AWS AppSync erkannt wird.

Abfragen

Wählen Sie Abfragen, um Ihre Abfragen und Mutationen anzuzeigen. Als wir unsere GraphQL-API mit unserem Modell erstellt haben, wurden AWS AppSync automatisch einige Hilfsmutationen und Abfragen zu Testzwecken generiert. Im Abfrage-Editor befindet sich auf der linken Seite der Explorer. Dies ist eine Liste mit all Ihren Mutationen und Abfragen. Sie können die Operationen und Felder, die Sie hier verwenden möchten, ganz einfach aktivieren, indem Sie auf deren Namenswerte klicken. Dadurch wird der Code automatisch in der Mitte des Editors angezeigt. Hier können Sie Ihre Mutationen und Abfragen bearbeiten, indem Sie Werte ändern. Am unteren Rand des Editors befindet sich der Editor für Abfragevariablen, mit dem Sie die Feldwerte für die Eingabevariablen Ihrer Operationen eingeben können. Wenn Sie oben im Editor Ausführen wählen, wird eine Dropdownliste angezeigt, in der Sie die auszuführende Abfrage/Mutation auswählen können. Die Ausgabe für diesen Lauf wird auf der rechten Seite der Seite angezeigt. Zurück im Explorer-Bereich oben können Sie eine Operation (Abfrage, Mutation, Abonnement) auswählen und dann das Pluszeichen wählen, um eine neue Instanz dieser bestimmten Operation hinzuzufügen. Oben auf der Seite befindet sich eine weitere Dropdownliste, die den Autorisierungsmodus für Ihre Abfrageläufe enthält. Wir werden diese Funktion in diesem Abschnitt jedoch nicht behandeln (weitere Informationen finden Sie unter Sicherheit.).

Einstellungen

Wählen Sie Einstellungen, um einige Konfigurationsoptionen für Ihre GraphQL-API anzuzeigen. Hier können Sie einige Optionen wie Protokollierung, Tracing und Firewall-Funktionen für Webanwendungen aktivieren. Sie können auch neue Autorisierungsmodi hinzufügen, um Ihre Daten vor unerwünschten Datenlecks zu schützen. Diese Optionen sind jedoch weiter fortgeschritten und werden in diesem Abschnitt nicht behandelt.

> **Note**
>
> Der Standardautorisierungsmodus verwendet einen API-Schlüssel`API_KEY`, um die Anwendung zu testen. Dies ist die Basisautorisierung, die allen neu erstellten GraphQL APIs erteilt wird. Wir empfehlen, dass Sie eine andere Methode für die Produktion verwenden.

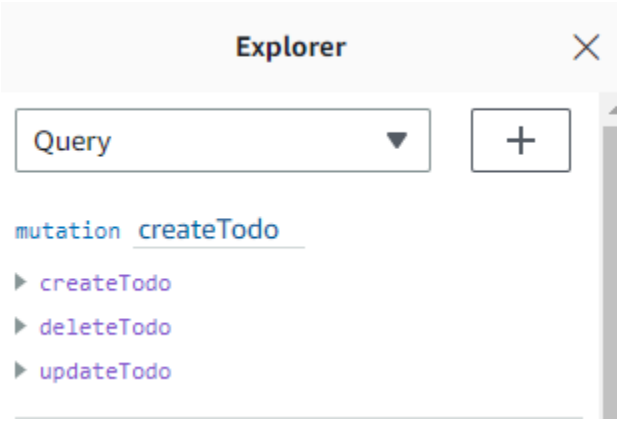
Für das Beispiel in diesem Abschnitt verwenden wir nur den API-Schlüssel. Weitere Informationen zu den unterstützten Autorisierungsmethoden finden Sie unter Sicherheit.

Verwenden von GraphQL-Mutationen zum Hinzufügen von Daten zu einer DynamoDB-Tabelle in der Konsole AWS AppSync

Ihr nächster Schritt besteht darin, Ihrer leeren DynamoDB-Tabelle mithilfe einer GraphQL-Mutation Daten hinzuzufügen. Mutationen sind einer der grundlegenden Operationstypen in GraphQL. Sie sind im Schema definiert und ermöglichen es Ihnen, Daten in Ihrer Datenquelle zu manipulieren. In Bezug auf REST APIs sind diese Operationen wie `PUT` oder sehr ähnlich`POST`.

Um Daten zu Ihrer Datenquelle hinzuzufügen

1. Falls Sie dies noch nicht getan haben, melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
2. Wählen Sie Ihre API aus der Tabelle aus.
3. Wählen Sie in der Registerkarte auf der linken Seite Abfragen aus.
4. Auf der Explorer-Registerkarte links neben der Tabelle sehen Sie möglicherweise mehrere Mutationen und Abfragen, die bereits im Abfrage-Editor definiert wurden:



> **Note**
>
> Diese Mutation ist tatsächlich als `Mutation` Typ in Ihrem Schema enthalten. Es hat den Code:
>
> ```
> type Mutation {
>  createTodo(input: CreateTodoInput!): Todo
> ```

```
 updateTodo(input: UpdateTodoInput!): Todo
 deleteTodo(input: DeleteTodoInput!): Todo
}
```

Wie Sie sehen können, ähneln die Operationen hier denen im Abfrage-Editor.

AWS AppSync hat diese automatisch anhand des zuvor definierten Modells generiert. In diesem Beispiel wird die `createTodo` Mutation verwendet, um Einträge zu unserer *`TodoAPITable`* Tabelle hinzuzufügen.

5. Wählen Sie die `createTodo` Operation aus, indem Sie sie unter der `createTodo` Mutation erweitern:

Aktivieren Sie die Kontrollkästchen für alle Felder wie im Bild oben.

> **Note**
>
> Die Attribute, die Sie hier sehen, sind die verschiedenen modifizierbaren Elemente der Mutation. Sie `input` können sich als Parameter von `createTodo` vorstellen. Die verschiedenen Optionen mit Checkboxen sind die Felder, die in der Antwort zurückgegeben werden, sobald ein Vorgang ausgeführt wird.

6. Im Code-Editor in der Mitte des Bildschirms werden Sie feststellen, dass die Operation unter der `createTodo` Mutation angezeigt wird:

```
mutation createTodo($createtodoinput: CreateTodoInput!) {
```

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

Note

Um dieses Snippet richtig zu erklären, müssen wir uns auch den Schemacode ansehen. Die Deklaration `mutation createTodo($createtodoinput: CreateTodoInput!){}` ist die Mutation mit einer ihrer Operationen,. `createTodo` Die vollständige Mutation befindet sich im Schema:

```
type Mutation {
 createTodo(input: CreateTodoInput!): Todo
 updateTodo(input: UpdateTodoInput!): Todo
 deleteTodo(input: DeleteTodoInput!): Todo
}
```

Zurück zur Mutationsdeklaration aus dem Editor: Der Parameter ist ein Objekt, das `$createtodoinput` mit dem erforderlichen Eingabetyp aufgerufen wird`CreateTodoInput`. Beachten Sie, dass `CreateTodoInput` (und alle Eingaben in der Mutation) auch im Schema definiert sind. Hier ist zum Beispiel der Boilerplate-Code für: `CreateTodoInput`

```
input CreateTodoInput {
 name: String
 when: String
 where: String
 description: String
}
```

Er enthält die Felder, die wir in unserem Modell definiert haben, nämlich`name`, `when``where`, und. `description`
Um zum Editor-Code zurückzukehren, deklarieren wir die Eingabe als`$createtodoinput`, die auch in der Mutationsdeklaration verwendet wurde.

`createTodo(input: $createtodoinput) {}` Wir tun dies, weil GraphQL so unsere Eingaben anhand der bereitgestellten Typen validieren und sicherstellen kann, dass sie mit den richtigen Eingaben verwendet werden.
Der letzte Teil des Editor-Codes zeigt die Felder, die in der Antwort zurückgegeben werden, nachdem eine Operation ausgeführt wurde:

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

Auf der Registerkarte Abfragevariablen unter diesem Editor wird es ein generisches `createtodoinput` Objekt geben, das die folgenden Daten enthalten kann:

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

Note

Hier weisen wir die Werte für die zuvor erwähnte Eingabe zu:

```
input CreateTodoInput {
 name: String
 when: String
 where: String
 description: String
}
```

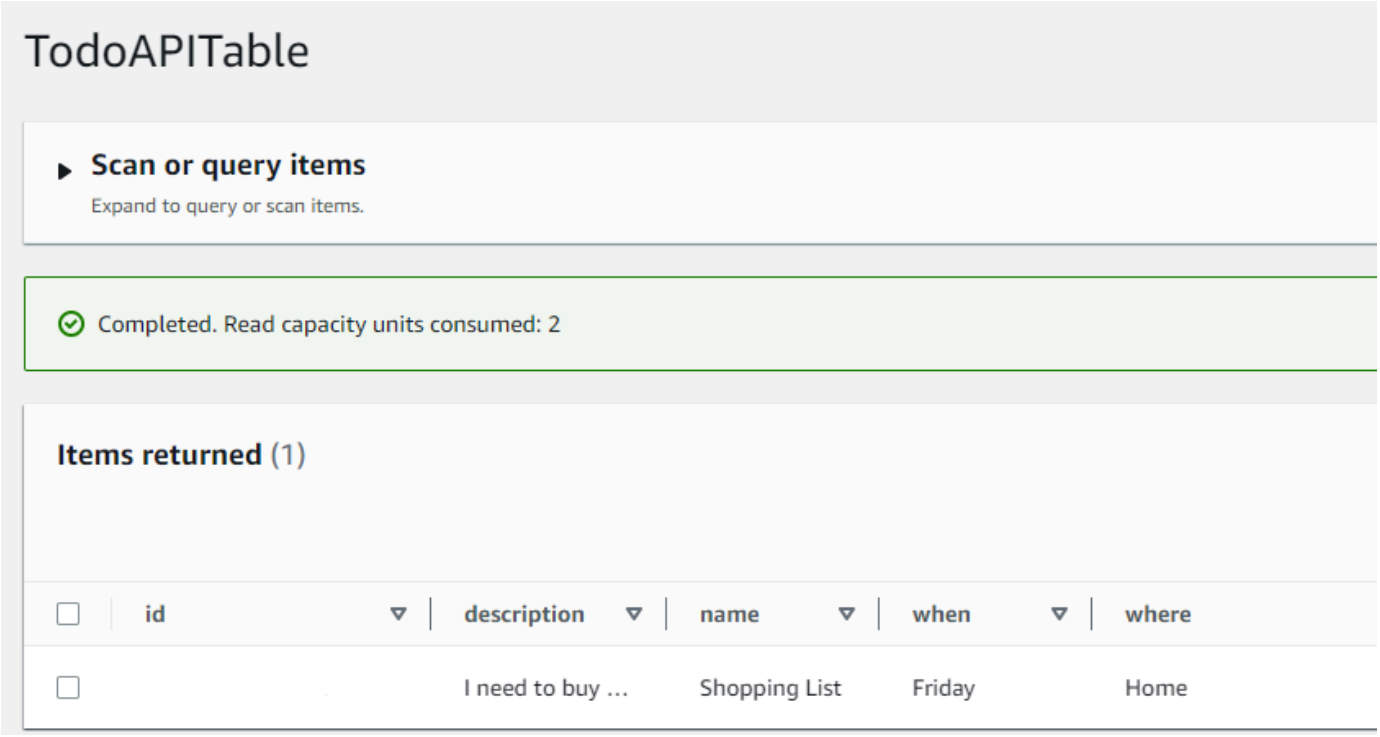
Ändern Sie das, `createtodoinput` indem Sie Informationen hinzufügen, die wir in unsere DynamoDB-Tabelle aufnehmen möchten. In diesem Fall wollten wir einige `Todo` Elemente als Erinnerung erstellen:

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

7. Wählen Sie oben im Editor die Option Ausführen aus. Wählen Sie in der Drop-down-Liste CreateTodo aus. Auf der rechten Seite des Editors sollten Sie die Antwort sehen. Dies kann etwa wie folgt aussehen:

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

Wenn Sie zum DynamoDB-Dienst navigieren, sehen Sie jetzt einen Eintrag in Ihrer Datenquelle mit diesen Informationen:



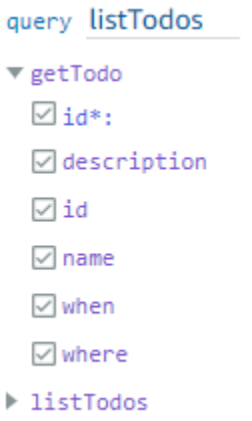
Um den Vorgang zusammenzufassen: Die GraphQL-Engine analysierte den Datensatz und ein Resolver fügte ihn in Ihre Amazon DynamoDB-Tabelle ein. Auch dies können Sie in der DynamoDB-Konsole überprüfen. Beachten Sie, dass Sie keinen Wert übergeben müssen. `id` Ein `id` wird generiert und in den Ergebnissen zurückgegeben. Das liegt daran, dass in dem Beispiel eine `autoId()` Funktion in einem GraphQL-Resolver für den Partitionsschlüssel verwendet wurde, der auf Ihren DynamoDB-Ressourcen festgelegt wurde. Wir werden in einem anderen Abschnitt behandeln, wie Sie Resolver erstellen können. Notieren Sie sich den zurückgegebenen `id` Wert. Sie werden ihn im nächsten Abschnitt verwenden, um Daten mit einer GraphQL-Abfrage abzurufen.

Verwenden von GraphQL-Abfragen zum Abrufen von Daten aus einer DynamoDB-Tabelle in der Konsole AWS AppSync

Da nun ein Datensatz in Ihrer Datenbank vorhanden ist, erhalten Sie Ergebnisse, wenn Sie eine Abfrage ausführen. Eine Abfrage ist eine der anderen grundlegenden Operationen von GraphQL. Es wird verwendet, um Informationen aus Ihrer Datenquelle zu analysieren und abzurufen. In Bezug auf REST APIs ähnelt dies der `GET` Operation. Der Hauptvorteil von GraphQL-Abfragen ist die Möglichkeit, die genauen Datenanforderungen Ihrer Anwendung zu spezifizieren, sodass Sie die relevanten Daten zur richtigen Zeit abrufen.

Um Ihre Datenquelle abzufragen

1. Falls Sie dies noch nicht getan haben, melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
2. Wählen Sie Ihre API aus der Tabelle aus.
3. Wählen Sie in der Registerkarte auf der linken Seite Abfragen aus.
4. Erweitern Sie auf der Explorer-Registerkarte links neben der Tabelle unter `query` `listTodos` den `getTodo` Vorgang:



5. Im Code-Editor sollten Sie den Operationscode sehen:

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

Geben Sie im Feld den Wert ein, den Sie im Ergebnis der Mutationsoperation gespeichert haben. `(id:"")` In unserem Beispiel wäre das:

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

```
  }
```

6. Wählen Sie Run und dann ListToDos. Das Ergebnis wird rechts neben dem Editor angezeigt. Unser Beispiel sah so aus:

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

> **Note**
>
> Abfragen geben nur die von Ihnen angegebenen Felder zurück. Sie können die Felder, die Sie nicht benötigen, abwählen, indem Sie sie aus dem Rückgabefeld löschen:
>
> ```
> {
>     description
>     id
>     name
>     when
>     where
>   }
> ```
>
> Sie können auch das Kästchen auf der Registerkarte Explorer neben dem Feld, das Sie löschen möchten, deaktivieren.

7. Sie können den `listTodos` Vorgang auch ausprobieren, indem Sie die Schritte zum Erstellen eines Eintrags in Ihrer Datenquelle wiederholen und dann die Abfrageschritte mit dem `listTodos` Vorgang wiederholen. Hier ist ein Beispiel, in dem wir eine zweite Aufgabe hinzugefügt haben:

```
{
  "createtodoinput": {
```

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

Durch den Aufruf der `listTodos` Operation wurden sowohl die alten als auch die neuen Einträge zurückgegeben:

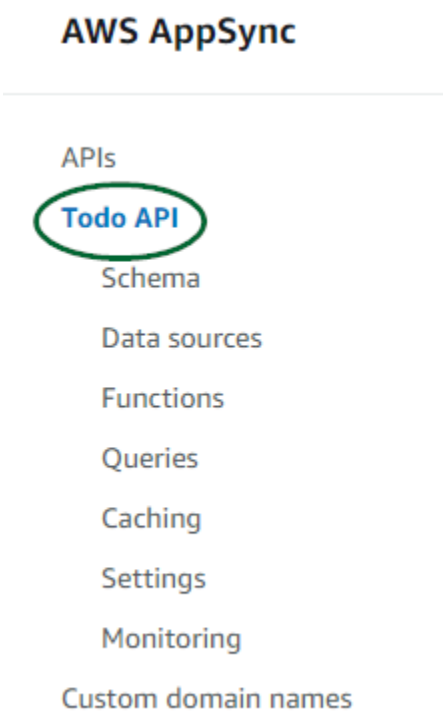
```
{
  "data": {
    "listTodos": {
      "items": [
        {
          "id": "abcdefgh-1234-1234-1234-abcdefghijkl",
          "name": "Shopping List",
          "when": "Friday",
          "where": "Home",
          "description": "I need to buy eggs"
        },
        {
          "id": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",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

Zusätzliche Abschnitte für die Konsole AWS AppSync

Diese Abschnitte dienen als Referenz für weiterführende AWS AppSync Themen. Wir empfehlen, den Abschnitt Zusätzliche Lektüre zu lesen, bevor Sie etwas anderes tun.

Integration

Wenn Sie auf der Registerkarte „Konsole“ den Namen Ihrer API wählen, wird die Integrationsseite angezeigt:



Sie fasst die Schritte zur Einrichtung Ihrer API zusammen und beschreibt die nächsten Schritte zum Erstellen einer Client-Anwendung. Der Abschnitt In Ihre App integrieren enthält Informationen zur Verwendung der AWS Amplify-Toolchain, um den Prozess der Verbindung Ihrer API mit iOS, Android und JavaScript Anwendungen durch Konfiguration und Codegenerierung zu automatisieren. Die Amplify-Toolchain bietet volle Unterstützung für die Erstellung von Projekten von Ihrer lokalen Workstation aus, einschließlich GraphQL-Bereitstellung und Workflows für CI/CD.

Im Abschnitt Client-Beispiele sind auch Beispiel-Clientanwendungen (z. B. iOS JavaScript, Android) zum Testen einer end-to-end Erfahrung aufgeführt. Sie können diese Beispiele klonen und herunterladen. Die Konfigurationsdatei enthält die erforderlichen Informationen (z. B. Ihre Endpunkt-URL), die Sie für den Einstieg benötigen. Folgen Sie den Anweisungen auf der AWS Amplify Toolchain-Seite, um Ihre App auszuführen.

Zusätzliche Lektüre

- GraphQL entwerfen APIs- Dies ist ein umfassender Leitfaden für die Erstellung Ihres GraphQL mit einem leeren Schema ohne Datenquellen oder Resolver.

GraphQL APIs entwerfen mit AWS AppSync

AWS AppSync ermöglicht es Ihnen, GraphQL APIs mithilfe der Konsolenerfahrung zu erstellen. Sie haben im Abschnitt Starten eines Beispielschemas einen Blick darauf geworfen. In diesem Handbuch wurde jedoch nicht der gesamte Katalog der Optionen und Konfigurationen aufgeführt, die Sie nutzen könnten. AWS AppSync

Wenn Sie sich dafür entscheiden, eine GraphQL-API in der Konsole zu erstellen, stehen Ihnen mehrere Optionen zur Verfügung. Wenn Sie unseren Leitfaden zur Einführung eines Beispielschemas befolgt haben, haben wir Ihnen gezeigt, wie Sie eine API aus einem vordefinierten Modell erstellen. In den folgenden Abschnitten führen wir Sie durch die restlichen Optionen und Konfigurationen für die Erstellung von GraphQL APIs in AWS AppSync.

In diesem Abschnitt werden Sie sich mit den folgenden Konzepten befassen:

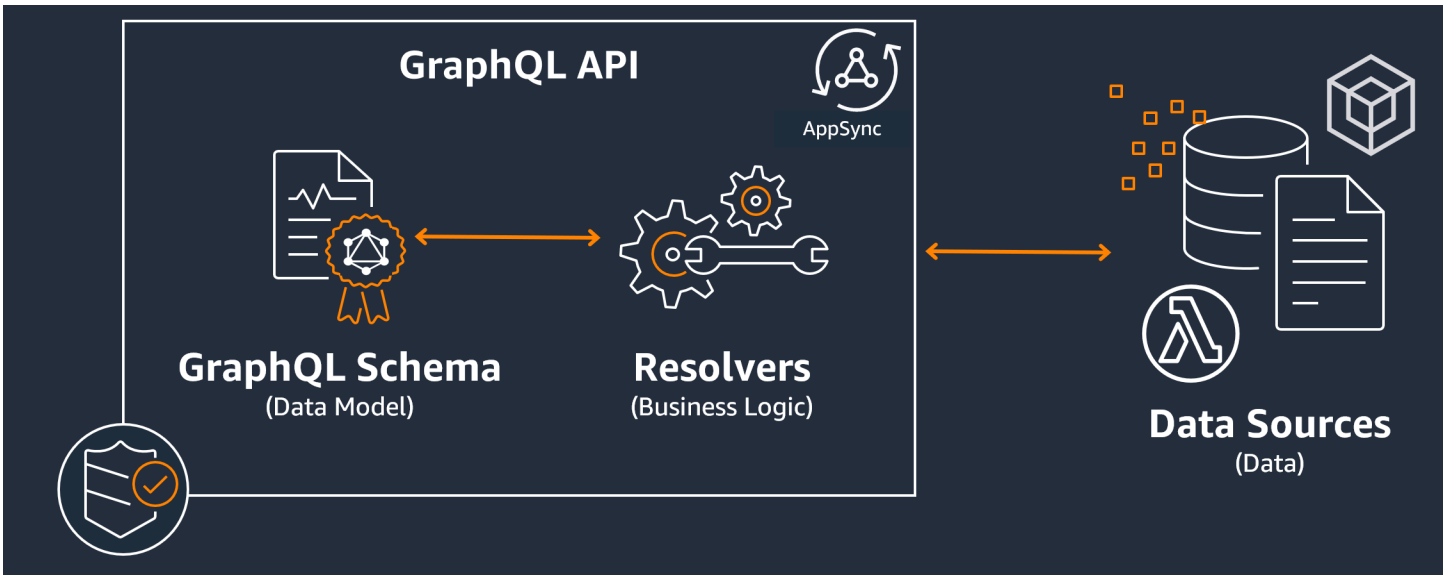
1. Blank APIs or imports: In diesem Handbuch wird der gesamte Erstellungsprozess für die Erstellung einer GraphQL-API beschrieben. Sie erfahren, wie Sie ein GraphQL aus einer leeren Vorlage ohne Modell erstellen, Datenquellen für Ihr Schema konfigurieren und Ihren ersten Resolver zu einem Feld hinzufügen.
2. Real-time data: Diese Anleitung zeigt Ihnen die möglichen Optionen für die Erstellung einer API mithilfe AWS AppSync der WebSocket Engine.
3. Merged APIs: Diese Anleitung zeigt Ihnen, wie Sie neues GraphQL erstellen, APIs indem Sie Daten aus mehreren vorhandenen GraphQL verknüpfen und zusammenführen. APIs
4. the section called “GraphQL APIs mit RDS-Introspektion erstellen”: In diesem Handbuch erfahren Sie, wie Sie Ihre Amazon RDS-Tabellen mithilfe einer Daten-API integrieren.

Strukturierung einer GraphQL-API (leer oder importiert APIs)

Bevor Sie Ihre GraphQL-API aus einer leeren Vorlage erstellen, wäre es hilfreich, die Konzepte rund um GraphQL zu überprüfen. Es gibt drei grundlegende Komponenten einer GraphQL-API:

1. Das Schema ist die Datei, die die Form und Definition Ihrer Daten enthält. Wenn ein Client eine Anfrage an Ihren GraphQL-Dienst stellt, folgen die zurückgegebenen Daten der Spezifikation des Schemas. Weitere Informationen finden Sie unter GraphQL-Schemas.
2. Die Datenquelle ist an Ihr Schema angehängt. Wenn eine Anfrage gestellt wird, werden die Daten hier abgerufen und geändert. Weitere Informationen finden Sie unter Data sources.

3. Der Resolver befindet sich zwischen dem Schema und der Datenquelle. Wenn eine Anfrage gestellt wird, führt der Resolver den Vorgang mit den Daten aus der Quelle aus und gibt dann das Ergebnis als Antwort zurück. Weitere Informationen finden Sie unter Resolvers.



AWS AppSync verwaltet Ihre, APIs indem es Ihnen ermöglicht, den Code für Ihre Schemas und Resolver zu erstellen, zu bearbeiten und zu speichern. Ihre Datenquellen stammen aus externen Repositorys wie Datenbanken, DynamoDB-Tabellen und Lambda-Funktionen. Wenn Sie einen AWS Dienst zum Speichern Ihrer Daten verwenden oder dies planen, AWS AppSync bietet es eine nahezu nahtlose Erfahrung beim Zuordnen von Daten aus Ihren AWS Konten zu Ihrem GraphQL. APIs

Im nächsten Abschnitt erfahren Sie, wie Sie jede dieser Komponenten mithilfe des Dienstes erstellen. AWS AppSync

Themen

- Entwerfen Ihres GraphQL-Schemas
- Eine Datenquelle anhängen in AWS AppSync
- Resolver konfigurieren in AWS AppSync
- Eine AWS AppSync API mit dem AWS CDK verwenden

Entwerfen Ihres GraphQL-Schemas

Das GraphQL-Schema ist die Grundlage jeder GraphQL-Serverimplementierung. Jede GraphQL-API wird durch ein einzelnes Schema definiert, das Typen und Felder enthält, die beschreiben, wie

die Daten aus Anfragen gefüllt werden. Die Daten, die durch Ihre API fließen, und die ausgeführten Operationen müssen anhand des Schemas validiert werden.

Im Allgemeinen beschreibt das GraphQL-Typsystem die Funktionen eines GraphQL-Servers und wird verwendet, um festzustellen, ob eine Abfrage gültig ist. Das Typsystem eines Servers wird oft als das Schema dieses Servers bezeichnet und kann aus verschiedenen Objekttypen, Skalartypen, Eingabetypen und mehr bestehen. GraphQL ist sowohl deklarativ als auch stark typisiert, was bedeutet, dass die Typen zur Laufzeit gut definiert sind und nur das zurückgeben, was angegeben wurde.

AWS AppSync ermöglicht es Ihnen, GraphQL-Schemas zu definieren und zu konfigurieren. Der folgende Abschnitt beschreibt, wie Sie GraphQL-Schemas mithilfe AWS AppSync der Dienste von Grund auf neu erstellen.

Strukturierung eines GraphQL-Schemas

> **Tip**
>
> Wir empfehlen, den Abschnitt Schemas zu lesen, bevor Sie fortfahren.

GraphQL ist ein leistungsstarkes Tool zur Implementierung von API-Diensten. Laut der Website von GraphQL ist GraphQL wie folgt:

„GraphQL ist eine Abfragesprache APIs und eine Laufzeit für die Ausführung dieser Abfragen mit Ihren vorhandenen Daten. GraphQL bietet eine vollständige und verständliche Beschreibung der Daten in Ihrer API, gibt Kunden die Möglichkeit, genau nach dem zu fragen, was sie benötigen, und nicht mehr, erleichtert die Weiterentwicklung APIs im Laufe der Zeit und ermöglicht leistungsstarke Entwicklertools. “

Dieser Abschnitt behandelt den allerersten Teil Ihrer GraphQL-Implementierung, das Schema. Unter Verwendung des obigen Zitats spielt ein Schema die Rolle, „eine vollständige und verständliche Beschreibung der Daten in Ihrer API bereitzustellen“. Mit anderen Worten, ein GraphQL-Schema ist eine textuelle Darstellung der Daten, Operationen und der Beziehungen zwischen ihnen Ihres Dienstes. Das Schema gilt als Haupteinstiegspunkt für Ihre GraphQL-Serviceimplementierung. Es überrascht nicht, dass es oft eines der ersten Dinge ist, die Sie in Ihrem Projekt machen. Wir empfehlen, den Abschnitt Schemas zu lesen, bevor Sie fortfahren.

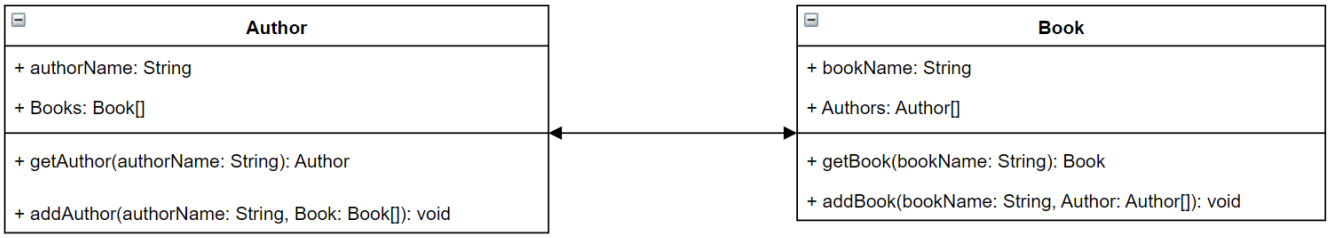
Um den Abschnitt Schemas zu zitieren: GraphQL-Schemas sind in der Schema Definition Language (SDL) geschrieben. SDL besteht aus Typen und Feldern mit einer festgelegten Struktur:

- Typen: Mit Typen definiert GraphQL die Form und das Verhalten der Daten. GraphQL unterstützt eine Vielzahl von Typen, die später in diesem Abschnitt erklärt werden. Jeder Typ, der in Ihrem Schema definiert ist, enthält seinen eigenen Bereich. Innerhalb des Bereichs befinden sich ein oder mehrere Felder, die einen Wert oder eine Logik enthalten können, die in Ihrem GraphQL-Dienst verwendet werden. Typen erfüllen viele verschiedene Rollen, am häufigsten sind Objekte oder Skalare (primitive Werttypen).
- Felder: Felder existieren innerhalb des Gültigkeitsbereichs eines Typs und enthalten den Wert, der vom GraphQL-Dienst angefordert wird. Diese sind Variablen in anderen Programmiersprachen sehr ähnlich. Die Form der Daten, die Sie in Ihren Feldern definieren, bestimmt, wie die Daten in einem Anforderungs-/Antwortvorgang strukturiert werden. Auf diese Weise können Entwickler vorhersagen, was zurückgegeben wird, ohne zu wissen, wie das Backend des Dienstes implementiert ist.

Die einfachsten Schemas werden drei verschiedene Datenkategorien enthalten:

1. Schemastämme: Stammverzeichnisse definieren die Einstiegspunkte Ihres Schemas. Es verweist auf die Felder, die bestimmte Operationen an den Daten ausführen, z. B. etwas hinzufügen, löschen oder ändern.
2. Typen: Dies sind Basistypen, die verwendet werden, um die Form der Daten darzustellen. Man kann sich diese fast als Objekte oder abstrakte Repräsentationen von etwas mit definierten Eigenschaften vorstellen. Sie könnten beispielsweise ein `Person` Objekt erstellen, das eine Person in einer Datenbank darstellt. Die Eigenschaften jeder Person werden in den `Person` AS-Feldern definiert. Sie können alles Mögliche wie Name, Alter, Beruf, Adresse usw. der Person sein.
3. Spezielle Objekttypen: Dies sind die Typen, die das Verhalten der Operationen in Ihrem Schema definieren. Jeder spezielle Objekttyp wird einmal pro Schema definiert. Sie werden zuerst im Schemastamm platziert und dann im Hauptteil des Schemas definiert. Jedes Feld in einem speziellen Objekttyp definiert eine einzelne Operation, die von Ihrem Resolver implementiert werden soll.

Um das ins rechte Licht zu rücken, stellen Sie sich vor, Sie erstellen einen Dienst, der Autoren und die Bücher, die sie geschrieben haben, speichert. Jeder Autor hat einen Namen und eine Reihe von Büchern, die er verfasst hat. Jedes Buch hat einen Namen und eine Liste der assoziierten Autoren. Wir möchten auch die Möglichkeit haben, Bücher und Autoren hinzuzufügen oder abzurufen. Eine einfache UML-Darstellung dieser Beziehung könnte wie folgt aussehen:



In GraphQL stellen die Entitäten `Author` und zwei verschiedene Objekttypen in Ihrem Schema `Book` dar:

```
type Author {
}

type Book {
}
```

`Author`enthält `authorName` und`Books`, während es `bookName` und `Book` `Authors` enthält. Diese können als Felder im Rahmen Ihrer Typen dargestellt werden:

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

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

Wie Sie sehen können, sind die Typdarstellungen dem Diagramm sehr ähnlich. Bei den Methoden wird es jedoch etwas schwieriger. Diese werden als Feld in einem von wenigen speziellen Objekttypen platziert. Ihre spezielle Objektkategorisierung hängt von ihrem Verhalten ab. GraphQL enthält drei grundlegende spezielle Objekttypen: Abfragen, Mutationen und Abonnements. Weitere Informationen finden Sie unter Spezielle Objekte.

Da `getAuthor` beide Daten anfordern, werden sie einem `Query` speziellen Objekttyp zugeordnet: `getBook`

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

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

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

Die Operationen sind mit der Abfrage verknüpft, die wiederum mit dem Schema verknüpft ist. Wenn Sie einen Schemastamm hinzufügen, wird der spezielle Objekttyp (`Query`in diesem Fall) als einer Ihrer Einstiegspunkte definiert. Dies kann mit dem `schema` Schlüsselwort geschehen:

```
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
}
```

Schauen Sie sich die letzten beiden Methoden an `addAuthor` und `addBook` fügen Ihrer Datenbank Daten hinzu, sodass sie in einem `Mutation` speziellen Objekttyp definiert werden. Aus der Seite „Typen“ wissen wir jedoch auch, dass Eingaben, die direkt auf Objekte verweisen, nicht zulässig

sind, da es sich ausschließlich um Ausgabetypen handelt. In diesem Fall können wir `Author` oder nicht verwendenBook, also müssen wir einen Eingabetyp mit denselben Feldern erstellen. In diesem Beispiel haben wir `AuthorInput` und hinzugefügt`BookInput`, die beide dieselben Felder ihres jeweiligen Typs akzeptieren. Dann erstellen wir unsere Mutation, indem wir die Eingaben als unsere Parameter verwenden:

```
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
}
```

Lassen Sie uns überprüfen, was wir gerade getan haben:

1. Wir haben ein Schema mit den `Author` Typen `Book` und zur Darstellung unserer Entitäten erstellt.
2. Wir haben die Felder hinzugefügt, die die Eigenschaften unserer Entitäten enthalten.
3. Wir haben eine Abfrage hinzugefügt, um diese Informationen aus der Datenbank abzurufen.
4. Wir haben eine Mutation hinzugefügt, um Daten in der Datenbank zu manipulieren.
5. Wir haben Eingabetypen hinzugefügt, um unsere Objektparameter in der Mutation zu ersetzen und den Regeln von GraphQL zu entsprechen.
6. Wir haben die Abfrage und die Mutation zu unserem Stammschema hinzugefügt, damit die GraphQL-Implementierung die Position des Root-Typs versteht.

Wie Sie sehen können, stützt sich der Prozess der Erstellung eines Schemas auf viele Konzepte aus der Datenmodellierung (insbesondere der Datenbankmodellierung) im Allgemeinen. Sie können sich das Schema so vorstellen, dass es der Form der Daten aus der Quelle entspricht. Es dient auch als Modell, das der Resolver implementieren wird. In den folgenden Abschnitten erfahren Sie, wie Sie mithilfe verschiedener AWS unterstützter Tools und Dienste ein Schema erstellen.

> **Note**
>
> Die Beispiele in den folgenden Abschnitten sind nicht für die Ausführung in einer echten Anwendung vorgesehen. Sie dienen nur dazu, die Befehle zu veranschaulichen, damit Sie Ihre eigenen Anwendungen erstellen können.

Erstellen von Schemata

Ihr Schema wird sich in einer Datei mit dem Namen befinden`schema.graphql`. AWS AppSync ermöglicht es Benutzern, mit verschiedenen Methoden neue Schemas für ihr GraphQL APIs zu erstellen. In diesem Beispiel erstellen wir eine leere API zusammen mit einem leeren Schema.

Console

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.
   a. Wählen Sie im Dashboard Create API (API erstellen) aus.
   b. Wählen Sie unter API-Optionen GraphQL APIs, Von Grund auf neu entwerfen und dann Weiter aus.

i. Ändern Sie für den API-Namen den vorausgefüllten Namen in den Namen, den Ihre Anwendung benötigt.

ii. Für Kontaktinformationen können Sie eine Kontaktperson angeben, um einen Manager für die API zu identifizieren. Dies ist ein optionales Feld.

iii. Unter Private API-Konfiguration können Sie private API-Funktionen aktivieren. Auf eine private API kann nur von einem konfigurierten VPC-Endpunkt (VPCE) aus zugegriffen werden. Weitere Informationen finden Sie unter Privat. APIs

Wir empfehlen, diese Funktion für dieses Beispiel nicht zu aktivieren. Wählen Sie „Weiter“, nachdem Sie Ihre Eingaben überprüft haben.

c. Unter GraphQL-Typ erstellen können Sie wählen, ob Sie eine DynamoDB-Tabelle erstellen möchten, die Sie als Datenquelle verwenden möchten, oder dies überspringen und später tun möchten.

Wählen Sie für dieses Beispiel Create GraphQL resources later aus. Wir werden eine Ressource in einem separaten Abschnitt erstellen.

d. Überprüfe deine Eingaben und wähle dann Create API.

2. Sie befinden sich im Dashboard Ihrer spezifischen API. Sie können es daran erkennen, dass der Name der API oben im Dashboard angezeigt wird. Wenn dies nicht der Fall ist, können Sie APIsin der Seitenleiste auswählen und dann Ihre API im APIs Dashboard auswählen.

   - Wählen Sie in der Seitenleiste unter dem Namen Ihrer API Schema aus.

3. Im Schema-Editor können Sie Ihre `schema.graphql` Datei konfigurieren. Sie kann leer sein oder mit Typen gefüllt sein, die aus einem Modell generiert wurden. Auf der rechten Seite befindet sich der Abschnitt Resolver, in dem Sie Resolver an Ihre Schemafelder anhängen können. In diesem Abschnitt werden wir uns nicht mit Resolvern befassen.

CLI

> **Note**
>
> Stellen Sie bei der Verwendung der CLI sicher, dass Sie über die richtigen Berechtigungen für den Zugriff und die Erstellung von Ressourcen im Service verfügen. Möglicherweise möchten Sie Richtlinien mit den geringsten Rechten für Benutzer ohne Administratorrechte festlegen, die auf den Service zugreifen müssen.

Weitere Informationen zu AWS AppSync Richtlinien finden Sie unter Identitäts- und Zugriffsverwaltung für. AWS AppSync
Darüber hinaus empfehlen wir, zuerst die Konsolenversion zu lesen, falls Sie dies noch nicht getan haben.

1. Falls Sie dies noch nicht getan haben, installieren Sie die AWS CLI und fügen Sie dann Ihre Konfiguration hinzu.
2. Erstellen Sie ein GraphQL-API-Objekt, indem Sie den `create-graphql-api`Befehl ausführen.

   Sie müssen zwei Parameter für diesen speziellen Befehl eingeben:

   1. Die `name` Ihrer API.
   2. Die oder die Art der Anmeldeinformationen`authentication-type`, die für den Zugriff auf die API verwendet werden (IAM, OIDC usw.).

   > **Note**
   >
   > Andere Parameter wie `Region` müssen konfiguriert werden, verwenden aber normalerweise standardmäßig Ihre CLI-Konfigurationswerte.

   Ein Beispielbefehl könnte wie folgt aussehen:

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

   Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

   ```
   {
       "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**
>
> Dies ist ein optionaler Befehl, der ein vorhandenes Schema verwendet und es mithilfe eines Base-64-Blobs in den AWS AppSync Dienst hochlädt. Wir werden diesen Befehl nicht für dieses Beispiel verwenden.

Führen Sie den Befehl `start-schema-creation` aus.

Für diesen speziellen Befehl müssen Sie zwei Parameter eingeben:

1. Ihr `api-id` aus dem vorherigen Schritt.
2. Das Schema `definition` ist ein Base-64-codierter binärer Blob.

Ein Beispielbefehl könnte wie folgt aussehen:

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

Es wird eine Ausgabe zurückgegeben:

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

Dieser Befehl gibt nach der Verarbeitung nicht die endgültige Ausgabe zurück. Sie müssen einen separaten Befehl verwenden `get-schema-creation-status`, um das Ergebnis zu sehen. Beachten Sie, dass diese beiden Befehle asynchron sind, sodass Sie den Ausgabestatus überprüfen können, auch wenn das Schema noch erstellt wird.

CDK

> **Tip**
>
> Bevor Sie das CDK verwenden, empfehlen wir, die offizielle Dokumentation des CDK zusammen mit AWS AppSync der CDK-Referenz zu lesen.
> Die unten aufgeführten Schritte zeigen nur ein allgemeines Beispiel für das Snippet, das zum Hinzufügen einer bestimmten Ressource verwendet wurde. Dies soll keine funktionierende Lösung in Ihrem Produktionscode sein. Wir gehen auch davon aus, dass Sie bereits eine funktionierende App haben.

1. Der Ausgangspunkt für das CDK ist etwas anders. Idealerweise sollte Ihre `schema.graphql` Datei bereits erstellt sein. Sie müssen nur eine neue Datei mit der `.graphql` Dateierweiterung erstellen. Dies kann eine leere Datei sein.
2. Im Allgemeinen müssen Sie möglicherweise die Import-Direktive zu dem Dienst hinzufügen, den Sie verwenden. Zum Beispiel kann es den folgenden Formen folgen:

   ```
   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'
   ```

   Um eine GraphQL-API hinzuzufügen, muss Ihre Stack-Datei den AWS AppSync Dienst importieren:

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

   > **Note**
   >
   > Das bedeutet, dass wir den gesamten Service unter dem `appsync` Schlüsselwort importieren. Um dies in Ihrer App zu verwenden, verwenden Ihre AWS AppSync Konstrukte das Format`appsync.construct_name`. Wenn wir zum Beispiel eine GraphQL-API erstellen wollten, würden wir sagen`new appsync.GraphqlApi(args_go_here)`. Der folgende Schritt zeigt dies.

3. Die grundlegendste GraphQL-API wird eine `name` für die API und den `schema` Pfad enthalten.

```
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**
>
> Sehen wir uns an, was dieser Ausschnitt bewirkt. Im Rahmen von `api` erstellen wir eine neue GraphQL-API, indem wir aufrufen`appsync.GraphqlApi(scope: Construct, id: string, props: GraphqlApiProps)`. Der Gültigkeitsbereich ist`this`, was sich auf das aktuelle Objekt bezieht. Die ID lautet*API_ID*, was der Ressourcenname Ihrer GraphQL-API sein wird, AWS CloudFormation wenn sie erstellt wird. Das `GraphqlApiProps` enthält die `name` Ihrer GraphQL-API und die`schema`. Das `schema` generiert ein Schema (`SchemaFile.fromAsset`), indem es den absoluten Pfad (`__dirname`) nach der `.graphql` Datei (*schema_name.graphql*) durchsucht. In einem realen Szenario befindet sich Ihre Schemadatei wahrscheinlich in der CDK-App.
> Um die an Ihrer GraphQL-API vorgenommenen Änderungen verwenden zu können, müssen Sie die App erneut bereitstellen.

Hinzufügen von Typen zu Schemas

Nachdem Sie Ihr Schema hinzugefügt haben, können Sie damit beginnen, sowohl Ihre Eingabe- als auch Ihre Ausgabetypen hinzuzufügen. Beachten Sie, dass die hier aufgeführten Typen nicht in echtem Code verwendet werden sollten. Sie sind lediglich Beispiele, die Ihnen helfen sollen, den Prozess zu verstehen.

Zuerst erstellen wir einen Objekttyp. In echtem Code müssen Sie nicht mit diesen Typen beginnen. Sie können jederzeit jeden beliebigen Typ erstellen, solange Sie die Regeln und die Syntax von GraphQL befolgen.

> **Note**
>
> In den nächsten Abschnitten wird der Schema-Editor verwendet, lassen Sie ihn also offen.

Console

- Sie können einen Objekttyp erstellen, indem Sie das `type` Schlüsselwort zusammen mit dem Namen des Typs verwenden:

```
type Type_Name_Goes_Here {}
```

Innerhalb des Gültigkeitsbereichs des Typs können Sie Felder hinzufügen, die die Eigenschaften des Objekts repräsentieren:

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

Ein Beispiel:

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

> **Note**
>
> In diesem Schritt haben wir einen generischen Objekttyp hinzugefügt, bei dem ein erforderliches `id` Feld gespeichert als`ID`, ein `title` Feld gespeichert als und ein `date` Feld`String`, das als gespeichert ist`AWSDateTime`. Eine Liste der Typen und Felder und ihrer Funktionsweise finden Sie unter Schemas. Eine Liste der Skalare und ihrer Funktionsweise finden Sie in der Typenreferenz.

CLI

> **Note**
>
> Wir empfehlen, zuerst die Konsolenversion zu lesen, falls Sie dies noch nicht getan haben.

- Sie können einen Objekttyp erstellen, indem Sie den `create-type`Befehl ausführen.

  Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

  1. Die `api-id` Ihrer API.
  2. Der `definition` oder der Inhalt Ihres Typs. Im Konsolenbeispiel war das:

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

  3. Die `format` deiner Eingabe. In diesem Beispiel verwenden wirSDL.

  Ein Beispielbefehl könnte so aussehen:

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

  Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

  ```
  {
      "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**
  >
  > In diesem Schritt haben wir einen generischen Objekttyp mit einem erforderlichen `id` Feld hinzugefügt, das als gespeichert istID, einem `title` Feld, das als gespeichert istString, und einem `date` Feld, das als gespeichert istAWSDateTime. Eine Liste

der Typen und Felder und ihrer Funktionsweise finden Sie unter Schemas. Eine Liste der Skalare und ihrer Funktionsweise finden Sie unter Typenreferenz.
Außerdem haben Sie vielleicht bemerkt, dass die direkte Eingabe der Definition für kleinere Typen funktioniert, für das Hinzufügen größerer oder mehrerer Typen jedoch nicht möglich ist. Sie können sich dafür entscheiden, alles in einer `.graphql` Datei hinzuzufügen und es dann als Eingabe zu übergeben.

CDK

> **Tip**
>
> Bevor Sie das CDK verwenden, empfehlen wir Ihnen, die offizielle Dokumentation des CDK zusammen mit der CDK-Referenz zu AWS AppSync lesen.
> Die unten aufgeführten Schritte zeigen nur ein allgemeines Beispiel für das Snippet, das zum Hinzufügen einer bestimmten Ressource verwendet wurde. Dies soll keine funktionierende Lösung in Ihrem Produktionscode sein. Wir gehen auch davon aus, dass Sie bereits eine funktionierende App haben.

Um einen Typ hinzuzufügen, müssen Sie ihn zu Ihrer `.graphql` Datei hinzufügen. Das Konsolenbeispiel lautete zum Beispiel:

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

Sie können Ihre Typen wie jede andere Datei direkt zum Schema hinzufügen.

> **Note**
>
> Um die an Ihrer GraphQL-API vorgenommenen Änderungen verwenden zu können, müssen Sie die App erneut bereitstellen.

Der Objekttyp hat Felder, bei denen es sich um skalare Typen wie Zeichenketten und Ganzzahlen handelt. AWS AppSync ermöglicht es Ihnen auch, erweiterte Skalartypen wie `AWSDateTime` zusätzlich zu den Basis-GraphQL-Skalaren zu verwenden. Außerdem ist jedes Feld, das mit einem Ausrufezeichen endet, erforderlich.

Insbesondere der `ID` skalare Typ ist ein eindeutiger Bezeichner, der entweder oder sein kann. `String` `Int` Sie können diese in Ihrem Resolver-Code für die automatische Zuweisung steuern.

Es gibt Ähnlichkeiten zwischen speziellen Objekttypen wie `Query` und „normalen“ Objekttypen wie dem obigen Beispiel, da sie beide das `type` Schlüsselwort verwenden und als Objekte betrachtet werden. Bei den speziellen Objekttypen (`QueryMutation`, und`Subscription`) unterscheidet sich ihr Verhalten jedoch erheblich, da sie als Einstiegspunkte für Ihre API bereitgestellt werden. Außerdem geht es bei ihnen eher um die Gestaltung von Vorgängen als um Daten. Weitere Informationen finden Sie unter Die Abfrage- und Mutationstypen.

Was spezielle Objekttypen angeht, könnte der nächste Schritt darin bestehen, einen oder mehrere von ihnen hinzuzufügen, um Operationen an den geformten Daten durchzuführen. In einem realen Szenario muss jedes GraphQL-Schema mindestens einen Root-Abfragetyp zum Anfordern von Daten haben. Sie können sich die Abfrage als einen der Einstiegspunkte (oder Endpunkte) für Ihren GraphQL-Server vorstellen. Fügen wir als Beispiel eine Abfrage hinzu.

Console

- Um eine Abfrage zu erstellen, können Sie sie einfach wie jeden anderen Typ zur Schemadatei hinzufügen. Eine Abfrage würde einen `Query` Typ und einen Eintrag im Stammverzeichnis wie folgt erfordern:

  ```
  schema {
    query: Name_of_Query
  }

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

  Beachten Sie, dass *`Name_of_Query`* in einer Produktionsumgebung `Query` in den meisten Fällen einfach aufgerufen wird. Wir empfehlen, diesen Wert beizubehalten. Innerhalb des Abfragetyps können Sie Felder hinzufügen. Jedes Feld führt eine Operation in der Anfrage aus. Infolgedessen werden die meisten, wenn nicht alle dieser Felder an einen Resolver

angehängt. In diesem Abschnitt befassen wir uns jedoch nicht damit. In Bezug auf das Format der Feldoperation könnte es so aussehen:

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

Ein Beispiel:

```
schema {
  query: Query
}

type Query {
  getObj: [Obj_Type_1]
}

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

> **Note**
>
> In diesem Schritt haben wir einen `Query` Typ hinzugefügt und ihn in unserem `schema` Stammverzeichnis definiert. Unser `Query` Typ hat ein `getObj` Feld definiert, das eine Liste von `Obj_Type_1` Objekten zurückgibt. Beachten Sie, `Obj_Type_1` dass dies das Objekt des vorherigen Schritts ist. Im Produktionscode arbeiten Ihre Außendienstmitarbeiter normalerweise mit Daten, die von Objekten wie geformt sind`Obj_Type_1`. Darüber hinaus verfügen Felder wie `getObj` normalerweise über einen Resolver, der die Geschäftslogik ausführt. Das wird in einem anderen Abschnitt behandelt.
>
> Als zusätzlicher Hinweis: Fügt bei Exporten AWS AppSync automatisch ein Schema-Root hinzu, sodass Sie es technisch gesehen nicht direkt zum Schema hinzufügen müssen. Unser Service verarbeitet automatisch doppelte Schemas. Wir fügen es hier als bewährte Methode hinzu.

CLI

> **Note**
>
> Wir empfehlen, zuerst die Konsolenversion zu lesen, falls Sie dies noch nicht getan haben.

1. Erstellen Sie ein `schema` Stammverzeichnis mit einer `query` Definition, indem `create-type`Sie den Befehl ausführen.

   Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

   1. Die `api-id` Ihrer API.
   2. Der `definition` oder der Inhalt Ihres Typs. Im Konsolenbeispiel war das:

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

   3. Die `format` deiner Eingabe. In diesem Beispiel verwenden wirSDL.

   Ein Beispielbefehl könnte so aussehen:

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

   Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

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

Note

Beachten Sie, dass Sie Ihren Schemastamm (oder einen beliebigen Typ im Schema) aktualisieren können, indem Sie den `create-type` Befehl ausführen, wenn Sie etwas nicht korrekt in den `update-type`Befehl eingegeben haben. In diesem Beispiel ändern wir vorübergehend den Schemastamm, sodass er eine `subscription` Definition enthält.
Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

1. Die `api-id` Ihrer API.
2. Die `type-name` deines Typs. Im Konsolenbeispiel war das`schema`.
3. Der `definition` oder der Inhalt Ihres Typs. Im Konsolenbeispiel war das:

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

Das Schema nach dem Hinzufügen von `subscription` sieht so aus:

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

4. Die `format` deiner Eingabe. In diesem Beispiel verwenden wirSDL.

Ein Beispielbefehl könnte so aussehen:

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

Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

```
{
    "type": {
        "definition": "schema {query: Query subscription: Subscription}",
```

```
        "arn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/schema",
        "format": "SDL"
    }
}
```

Das Hinzufügen vorformatierter Dateien funktioniert in diesem Beispiel weiterhin.

2. Erstellen Sie einen `Query` Typ, indem Sie den `create-type`Befehl ausführen.

   Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

   1. Die `api-id` Ihrer API.
   2. Der `definition` oder der Inhalt Ihres Typs. Im Konsolenbeispiel war das:

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

   3. Die `format` deiner Eingabe. In diesem Beispiel verwenden wir`SDL`.

   Ein Beispielbefehl könnte so aussehen:

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

   Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

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

> **Note**
>
> In diesem Schritt haben wir einen `Query` Typ hinzugefügt und ihn in Ihrem `schema` Stammverzeichnis definiert. Unser `Query` Typ hat ein `getObj` Feld definiert, das eine Liste von `Obj_Type_1` Objekten zurückgab.
> Im `schema` Stammcode gibt der `query:` Teil an`query: Query`, dass eine Abfrage in Ihrem Schema definiert wurde, während der `Query` Teil den tatsächlichen Namen des speziellen Objekts angibt.

CDK

> **Tip**
>
> Bevor Sie das CDK verwenden, empfehlen wir, die offizielle Dokumentation des CDK zusammen mit der CDK-Referenz zu AWS AppSync lesen.
> Die unten aufgeführten Schritte zeigen nur ein allgemeines Beispiel für das Snippet, das zum Hinzufügen einer bestimmten Ressource verwendet wurde. Dies soll keine funktionierende Lösung in Ihrem Produktionscode sein. Wir gehen auch davon aus, dass Sie bereits eine funktionierende App haben.

Sie müssen Ihre Abfrage und den Schemastamm zur `.graphql` Datei hinzufügen. Unser Beispiel sah wie das folgende Beispiel aus, aber Sie sollten es durch Ihren tatsächlichen Schemacode ersetzen:

```
schema {
  query: Query
}

type Query {
  getObj: [Obj_Type_1]
}

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

```
}
```

Sie können Ihre Typen wie jede andere Datei direkt zum Schema hinzufügen.

> **Note**
>
> Die Aktualisierung des Schemastammes ist optional. Wir haben es als bewährte Methode zu diesem Beispiel hinzugefügt.
> Um die an Ihrer GraphQL-API vorgenommenen Änderungen verwenden zu können, müssen Sie die App erneut bereitstellen.

Sie haben jetzt ein Beispiel für die Erstellung von Objekten und speziellen Objekten (Abfragen) gesehen. Sie haben auch gesehen, wie diese miteinander verbunden werden können, um Daten und Operationen zu beschreiben. Sie können Schemas verwenden, die nur die Datenbeschreibung und eine oder mehrere Abfragen enthalten. Wir möchten jedoch eine weitere Operation hinzufügen, um der Datenquelle Daten hinzuzufügen. Wir werden einen weiteren speziellen Objekttyp hinzufügen`Mutation`, der Daten modifiziert.

Console

- Eine Mutation wird aufgerufen`Mutation`. Die darin `Mutation` enthaltenen Feldoperationen beschreiben zum Beispiel `Query` eine Operation und werden an einen Resolver angehängt. Beachten Sie auch, dass wir es im `schema` Stammverzeichnis definieren müssen, da es sich um einen speziellen Objekttyp handelt. Hier ist ein Beispiel für eine Mutation:

  ```
  schema {
    mutation: Name_of_Mutation
  }

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

  Eine typische Mutation wird wie eine Abfrage im Stammverzeichnis aufgeführt. Die Mutation wird mit dem `type` Schlüsselwort zusammen mit dem Namen definiert. *`Name_of_Mutation`*wird normalerweise aufgerufen`Mutation`, daher empfehlen wir, es so zu belassen. Jedes Feld führt auch eine Operation aus. In Bezug auf das Format der Feldoperation könnte es so aussehen:

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

Ein Beispiel:

```
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

In diesem Schritt haben wir einen `Mutation` Typ mit einem `addObj` Feld hinzugefügt. Lassen Sie uns zusammenfassen, was dieses Feld tut:

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

`addObj`verwendet das `Obj_Type_1` Objekt, um eine Operation auszuführen. Das liegt an den Feldern, aber die Syntax beweist dies im `: Obj_Type_1` Rückgabetyp. `addObj`Im Inneren akzeptiert es die `date` Felder `id``title`, und aus dem `Obj_Type_1` Objekt als Parameter. Wie Sie vielleicht sehen, sieht es einer Methodendeklaration sehr ähnlich. Wir haben das Verhalten unserer Methode jedoch noch nicht beschrieben. Wie bereits erwähnt, dient das Schema nur dazu, zu definieren, wie die Daten und Operationen aussehen werden, und nicht, wie sie

funktionieren. Die Implementierung der eigentlichen Geschäftslogik erfolgt später, wenn wir unsere ersten Resolver erstellen.
Sobald Sie mit Ihrem Schema fertig sind, besteht die Möglichkeit, es als `schema.graphql` Datei zu exportieren. Im Schema-Editor können Sie Schema exportieren wählen, um die Datei in einem unterstützten Format herunterzuladen.
Ein zusätzlicher Hinweis: Fügt bei Exporten AWS AppSync automatisch ein Schema-Root hinzu, sodass Sie es technisch gesehen nicht direkt zum Schema hinzufügen müssen. Unser Service verarbeitet automatisch doppelte Schemas. Wir fügen es hier als bewährte Methode hinzu.

CLI

> **Note**
>
> Wir empfehlen, zuerst die Konsolenversion zu lesen, falls Sie dies noch nicht getan haben.

1. Aktualisieren Sie Ihr Stammschema, indem Sie den `update-type`Befehl ausführen.

   Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

   1. Die `api-id` Ihrer API.
   2. Die `type-name` deines Typs. Im Konsolenbeispiel war das`schema`.
   3. Der `definition` oder der Inhalt Ihres Typs. Im Konsolenbeispiel war das:

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

   4. Die `format` deiner Eingabe. In diesem Beispiel verwenden wir`SDL`.

   Ein Beispielbefehl könnte so aussehen:

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

Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

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

2. Erstellen Sie einen `Mutation` Typ, indem Sie den `create-type`Befehl ausführen.

   Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

   1. Die `api-id` Ihrer API.
   2. Der `definition` oder der Inhalt Ihres Typs. Im Konsolenbeispiel war das

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

   3. Die `format` deiner Eingabe. In diesem Beispiel verwenden wir`SDL`.

   Ein Beispielbefehl könnte so aussehen:

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

   Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

   ```
   {
       "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**
>
> Bevor Sie das CDK verwenden, empfehlen wir, die offizielle Dokumentation des CDK zusammen mit der CDK-Referenz zu AWS AppSync lesen.
> Die unten aufgeführten Schritte zeigen nur ein allgemeines Beispiel für das Snippet, das zum Hinzufügen einer bestimmten Ressource verwendet wurde. Dies soll keine funktionierende Lösung in Ihrem Produktionscode sein. Wir gehen auch davon aus, dass Sie bereits eine funktionierende App haben.

Sie müssen Ihre Abfrage und den Schemastamm zur `.graphql` Datei hinzufügen. Unser Beispiel sah wie das folgende Beispiel aus, aber Sie sollten es durch Ihren tatsächlichen Schemacode ersetzen:

```
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**
>
> Das Aktualisieren des Schema-Stammverzeichnisses ist optional. Wir haben es als bewährte Methode zu diesem Beispiel hinzugefügt.
> Um die an Ihrer GraphQL-API vorgenommenen Änderungen verwenden zu können, müssen Sie die App erneut bereitstellen.

Optionale Überlegungen — Verwendung von Enums als Status

Zu diesem Zeitpunkt wissen Sie, wie man ein grundlegendes Schema erstellt. Es gibt jedoch viele Dinge, die Sie hinzufügen könnten, um die Funktionalität des Schemas zu erhöhen. Eine häufig vorkommende Sache in Anwendungen ist die Verwendung von Enums als Status. Sie können eine Aufzählung verwenden, um zu erzwingen, dass beim Aufruf ein bestimmter Wert aus einer Menge von Werten ausgewählt wird. Das ist gut für Dinge, von denen Sie wissen, dass sie sich über lange Zeiträume nicht drastisch ändern werden. Hypothetisch gesehen könnten wir eine Aufzählung hinzufügen, die den Statuscode oder die Zeichenfolge in der Antwort zurückgibt.

Nehmen wir als Beispiel an, wir erstellen eine Social-Media-App, die die Beitragsdaten eines Benutzers im Backend speichert. Unser Schema enthält einen `Post` Typ, der die Daten eines einzelnen Beitrags darstellt:

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

Unser `Post` wird einen eindeutigen `id` Beitrag und eine `title` `date` Aufzählung mit dem Namen enthalten, `PostStatus` die den Status des Beitrags angibt, während er von der App verarbeitet wird. Für unseren Betrieb werden wir eine Abfrage haben, die alle Beitragsdaten zurückgibt:

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

Wir werden auch eine Mutation haben, die Beiträge zur Datenquelle hinzufügt:

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

Wenn wir uns unser Schema ansehen, könnte die `PostStatus` Aufzählung mehrere Status haben. Möglicherweise möchten wir, dass die drei grundlegenden Status `success` (Post erfolgreich verarbeitet), `pending` (Post wird verarbeitet) und `error` (Beitrag kann nicht verarbeitet werden) heißen. Um die Aufzählung hinzuzufügen, könnten wir Folgendes tun:

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

Das vollständige Schema könnte so aussehen:

```
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
```

```
}
```

Wenn ein Benutzer der Anwendung eine `Post` hinzufügt, wird die `addPost` Operation aufgerufen, um diese Daten zu verarbeiten. Während der damit verbundene Resolver die Daten `addPost` verarbeitet, aktualisiert er sie kontinuierlich `poststatus` mit dem Status des Vorgangs. Wenn sie abgefragt `Post` werden, enthält sie den endgültigen Status der Daten. Denken Sie daran, dass wir nur beschreiben, wie die Daten im Schema funktionieren sollen. Wir gehen von der Implementierung unserer Resolver (s) aus, die die eigentliche Geschäftslogik für den Umgang mit den Daten zur Erfüllung der Anfrage implementieren.

Optionale Überlegungen — Abonnements

Abonnements in AWS AppSync werden als Reaktion auf eine Mutation aufgerufen. Diese werden mit einem `Subscription`-Typ und einer `@aws_subscribe()`-Anweisung im Schema konfiguriert, um anzugeben, welche Mutationen ein oder mehrere Abonnements aufrufen. Weitere Informationen zur Konfiguration von Abonnements finden Sie unter Echtzeitdaten.

Optionale Überlegungen — Beziehungen und Seitennummerierung

Angenommen, Sie haben eine Million in einer DynamoDB-Tabelle `Posts` gespeichert und möchten einige dieser Daten zurückgeben. Die oben angegebene Beispielabfrage gibt jedoch nur alle Beiträge zurück. Sie möchten nicht jedes Mal, wenn Sie eine Anfrage stellen, alle abrufen. Stattdessen würden Sie sie paginieren wollen. Nehmen Sie dazu die folgenden Änderungen an Ihrem Schema vor:

- Fügen Sie im `getPosts` Feld zwei Eingabeargumente hinzu: `nextToken` (Iterator) und `limit` (Iterationslimit).
- Fügen Sie einen neuen `PostIterator` Typ hinzu, der `Posts` Felder (ruft die `Post` Objektliste ab) und `nextToken` (Iterator) enthält.
- Ändern Sie es `getPosts` so, dass es zurückgegeben wird `PostIterator` und keine Liste von `Post` Objekten.

```
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
}
```

Der `PostIterator` Typ ermöglicht es Ihnen, einen Teil der `Post` Objektliste zurückzugeben und einen, `nextToken` um den nächsten Teil abzurufen. Darin befindet `PostIterator` sich eine Liste von `Post` Elementen (`[Post]`), die mit einem Paginierungstoken (`nextToken`) zurückgegeben wird. AWS AppSyncIn würde dies über einen Resolver mit Amazon DynamoDB verbunden und automatisch als verschlüsseltes Token generiert. Dadurch wird der Wert des Arguments `limit` in den Parameter `maxResults` und des Arguments `nextToken` in den Parameter `exclusiveStartKey` konvertiert. Beispiele und die integrierten Vorlagenbeispiele in der AWS AppSync Konsole finden Sie unter Resolver-Referenz (). JavaScript

Eine Datenquelle anhängen in AWS AppSync

Datenquellen sind Ressourcen in Ihrem AWS Konto, mit denen GraphQL interagieren APIs kann. AWS AppSync unterstützt eine Vielzahl von Datenquellen wie AWS Lambda Amazon DynamoDB, relationale Datenbanken (Amazon Aurora Serverless), Amazon OpenSearch Service und HTTP-Endpunkte. Eine AWS AppSync API kann so konfiguriert werden, dass sie mit mehreren Datenquellen interagiert, sodass Sie Daten an einem einzigen Ort aggregieren können. AWS AppSync kann vorhandene AWS Ressourcen aus Ihrem Konto verwenden oder DynamoDB-Tabellen in Ihrem Namen anhand einer Schemadefinition bereitstellen.

Der folgende Abschnitt zeigt Ihnen, wie Sie eine Datenquelle an Ihre GraphQL-API anhängen.

Arten von Datenquellen

Nachdem Sie in der AWS AppSync Konsole ein Schema erstellt haben, können Sie eine Datenquelle daran anhängen. Wenn Sie zum ersten Mal eine API erstellen, besteht die Möglichkeit, während der Erstellung des vordefinierten Schemas eine Amazon DynamoDB-Tabelle bereitzustellen. Wir werden diese Option in diesem Abschnitt jedoch nicht behandeln. Ein Beispiel dafür finden Sie im Abschnitt Ein Schema starten.

Stattdessen werden wir uns alle AWS AppSync unterstützten Datenquellen ansehen. Es gibt viele Faktoren, die bei der Auswahl der richtigen Lösung für Ihre Anwendung eine Rolle spielen. Die folgenden Abschnitte bieten zusätzlichen Kontext für jede Datenquelle. Allgemeine Informationen zu Datenquellen finden Sie unter Datenquellen.

Amazon-DynamoDB

Amazon DynamoDB ist eine der wichtigsten AWS Speicherlösungen für skalierbare Anwendungen. Die Kernkomponente von DynamoDB ist die Tabelle, bei der es sich einfach um eine Sammlung von Daten handelt. In der Regel erstellen Sie Tabellen, die auf Entitäten wie `Book` oder basieren. `Author` Informationen zu Tabelleneinträgen werden als Elemente gespeichert. Dabei handelt es sich um Gruppen von Feldern, die für jeden Eintrag einzigartig sind. Ein vollständiges Element steht für ein Element row/record in der Datenbank. Ein Element für einen `Book` Eintrag kann beispielsweise `title` und `author` zusammen mit ihren Werten enthalten. Die einzelnen Felder wie das `title` und `author` werden als Attribute bezeichnet, was den Spaltenwerten in relationalen Datenbanken ähnelt.

Wie Sie sich vorstellen können, werden Tabellen zum Speichern von Daten aus Ihrer Anwendung verwendet. AWS AppSync ermöglicht es Ihnen, Ihre DynamoDB-Tabellen mit Ihrer GraphQL-API zu verbinden, um Daten zu manipulieren. Nehmen Sie diesen Anwendungsfall aus dem Frontend-Blog für Web und Mobilgeräte. Mit dieser Anwendung können sich Benutzer für eine Social-Media-App anmelden. Benutzer können Gruppen beitreten und Beiträge hochladen, die an andere Benutzer gesendet werden, die die Gruppe abonniert haben. Ihre Anwendung speichert Benutzer-, Post- und Benutzergruppeninformationen in DynamoDB. Die GraphQL-API (verwaltet von AWS AppSync) ist mit der DynamoDB-Tabelle verbunden. Wenn ein Benutzer eine Änderung am System vornimmt, die sich im Frontend widerspiegelt, ruft die GraphQL-API diese Änderungen ab und überträgt sie in Echtzeit an andere Benutzer.

AWS Lambda

Lambda ist ein ereignisgesteuerter Dienst, der automatisch die erforderlichen Ressourcen aufbaut, um Code als Reaktion auf ein Ereignis auszuführen. Lambda verwendet Funktionen, bei denen es sich um Gruppenanweisungen handelt, die den Code, die Abhängigkeiten und die Konfigurationen für die Ausführung einer Ressource enthalten. Funktionen werden automatisch ausgeführt, wenn sie einen Trigger erkennen, eine Gruppe von Aktivitäten, die Ihre Funktion aufrufen. Ein Trigger kann so etwas wie eine Anwendung sein, die einen API-Aufruf durchführt, ein AWS Dienst in Ihrem Konto, der eine Ressource hochlädt usw. Wenn sie ausgelöst werden, verarbeiten Funktionen Ereignisse, bei denen es sich um JSON-Dokumente handelt, die die zu ändernden Daten enthalten.

Lambda eignet sich gut zum Ausführen von Code, ohne die Ressourcen für die Ausführung bereitstellen zu müssen. Nehmen Sie diesen Anwendungsfall aus dem Frontend-Blog für Web und Mobilgeräte. Dieser Anwendungsfall ist dem im Abschnitt DynamoDB vorgestellten ein wenig ähnlich. In dieser Anwendung ist die GraphQL-API dafür verantwortlich, die Operationen für Dinge wie das Hinzufügen von Beiträgen (Mutationen) und das Abrufen dieser Daten (Abfragen) zu definieren. Um die Funktionalität ihrer Operationen zu implementieren (z. B.,`getPostsByAuthor ( author: String ! ) : [ Post ]`)`getPost ( id: String ! ) : Post`, verwenden sie Lambda-Funktionen, um eingehende Anfragen zu verarbeiten. Option 2: AWS AppSync Mit Lambda-Resolver verwenden sie den AWS AppSync Dienst, um ihr Schema zu verwalten und eine Lambda-Datenquelle mit einer der Operationen zu verknüpfen. Wenn der Vorgang aufgerufen wird, verbindet sich Lambda mit dem Amazon RDS-Proxy, um die Geschäftslogik in der Datenbank auszuführen.

Amazon RDS

Mit Amazon RDS können Sie schnell relationale Datenbanken erstellen und konfigurieren. In Amazon RDS erstellen Sie eine generische Datenbank-Instance, die als isolierte Datenbankumgebung in der Cloud dient. In diesem Fall verwenden Sie eine DB-Engine, bei der es sich um die eigentliche RDBMS-Software (PostgreSQL, MySQL usw.) handelt. Der Service entlastet einen Großteil der Backend-Arbeit, indem er Skalierbarkeit mithilfe AWS von Infrastruktur und Sicherheitsdiensten wie Patching und Verschlüsselung bietet und die Verwaltungskosten für Implementierungen senkt.

Nehmen Sie denselben Anwendungsfall aus dem Abschnitt Lambda. Unter Option 3: AWS AppSync mit Amazon RDS Resolver wird eine weitere Option vorgestellt, die GraphQL-API direkt mit Amazon RDS AWS AppSync zu verknüpfen. Mithilfe einer Daten-API verknüpfen sie die Datenbank mit der GraphQL-API. Ein Resolver ist an ein Feld angehängt (normalerweise eine Abfrage, Mutation oder ein Abonnement) und implementiert die SQL-Anweisungen, die für den Zugriff auf die Datenbank erforderlich sind. Wenn der Client eine Anfrage stellt, die das Feld aufruft, führt der Resolver die Anweisungen aus und gibt die Antwort zurück.

Amazon EventBridge

In EventBridge erstellen Sie Event-Busse. Dabei handelt es sich um Pipelines, die Ereignisse von Diensten oder Anwendungen empfangen, die Sie anhängen (die Ereignisquelle), und diese auf der Grundlage einer Reihe von Regeln verarbeiten. Ein Ereignis ist eine Zustandsänderung in einer Ausführungsumgebung, während eine Regel eine Reihe von Filtern für Ereignisse ist. Eine Regel folgt einem Ereignismuster oder den Metadaten der Statusänderung eines Ereignisses (ID, Region, Kontonummer, ARN (s) usw.). Wenn ein Ereignis mit dem Ereignismuster übereinstimmt, EventBridge wird das Ereignis über die Pipeline an den Zieldienst (Ziel) gesendet und die in der Regel angegebene Aktion ausgelöst.

EventBridge eignet sich gut für die Weiterleitung von Zustandsänderungsvorgängen an einen anderen Dienst. Nehmen Sie diesen Anwendungsfall aus dem Frontend-Blog für Web und Mobilgeräte. Das Beispiel zeigt eine E-Commerce-Lösung, bei der mehrere Teams unterschiedliche Dienste verwalten. Einer dieser Dienste bietet dem Kunden bei jedem Schritt der Lieferung (Bestellung aufgegeben, in Bearbeitung, versendet, geliefert usw.) über das Frontend aktuelle Informationen zur Bestellung. Das Frontend-Team, das diesen Service verwaltet, hat jedoch keinen direkten Zugriff auf die Daten des Bestellsystems, da diese von einem separaten Backend-Team verwaltet werden. Das Bestellsystem des Backend-Teams wird auch als Blackbox beschrieben, sodass es schwierig ist, Informationen darüber zu erhalten, wie sie ihre Daten strukturieren.
Das Backend-Team hat jedoch ein System eingerichtet, das Bestelldaten über einen Event-Bus veröffentlichte, der von verwaltet wird. EventBridge Um auf die vom Event-Bus kommenden Daten zuzugreifen und sie an das Front-End weiterzuleiten, erstellte das Front-End-Team ein neues Ziel, das auf die darin befindliche GraphQL-API verweist. AWS AppSync Außerdem wurde eine Regel erstellt, nach der nur Daten gesendet werden, die für die Aktualisierung der Bestellung relevant sind. Wenn ein Update durchgeführt wird, werden die Daten vom Event-Bus an die GraphQL-API gesendet. Das Schema in der API verarbeitet die Daten und leitet sie dann an das Frontend weiter.

Keine Datenquellen

Wenn Sie nicht vorhaben, eine Datenquelle zu verwenden, können Sie sie auf einstellen`none`.
Eine `none` Datenquelle ist zwar immer noch explizit als Datenquelle kategorisiert, aber kein Speichermedium. In der Regel ruft ein Resolver irgendwann eine oder mehrere Datenquellen auf, um die Anfrage zu verarbeiten. Es gibt jedoch Situationen, in denen Sie eine Datenquelle möglicherweise nicht manipulieren müssen. Wenn Sie die Datenquelle auf einstellen, `none` wird die Anfrage ausgeführt, der Datenaufruf-Schritt übersprungen und dann die Antwort ausgeführt.

Nehmen Sie denselben Anwendungsfall aus dem EventBridge Abschnitt. Im Schema verarbeitet die Mutation die Statusmeldung und sendet sie dann an die Abonnenten. Erinnert man sich daran,

wie Resolver funktionieren, gibt es in der Regel mindestens einen Datenquellenaufruf. Die Daten in diesem Szenario wurden jedoch bereits automatisch vom Event-Bus gesendet. Das bedeutet, dass die Mutation nicht erforderlich ist, um einen Datenquellenaufruf durchzuführen. Der Auftragsstatus kann einfach lokal bearbeitet werden. Die Mutation ist auf gesetztnone, was als Pass-Through-Wert ohne Datenquellenaufruf fungiert. Das Schema wird dann mit den Daten gefüllt, die an Abonnenten gesendet werden.

OpenSearch

Amazon OpenSearch Service ist eine Suite von Tools zur Implementierung von Volltextsuche, Datenvisualisierung und Protokollierung. Sie können diesen Service verwenden, um die von Ihnen hochgeladenen strukturierten Daten abzufragen.

In diesem Service erstellen Sie Instanzen von OpenSearch. Diese werden Knoten genannt. In einem Knoten fügen Sie mindestens einen Index hinzu. Indizes sind konzeptionell ein bisschen wie Tabellen in relationalen Datenbanken. (OpenSearch Ist jedoch nicht ACID-konform und sollte daher nicht auf diese Weise verwendet werden). Sie füllen Ihren Index mit Daten, die Sie in den OpenSearch Service hochladen. Wenn Ihre Daten hochgeladen werden, werden sie in einem oder mehreren Shards indexiert, die im Index vorhanden sind. Ein Shard ist wie eine Partition Ihres Indexes, die einige Ihrer Daten enthält und getrennt von anderen Shards abgefragt werden kann. Nach dem Hochladen werden Ihre Daten als JSON-Dateien, sogenannte Dokumente, strukturiert. Anschließend können Sie den Knoten nach Daten im Dokument abfragen.

HTTP-Endpunkte

Sie können HTTP-Endpunkte als Datenquellen verwenden. AWS AppSync kann Anfragen mit den relevanten Informationen wie Parametern und Nutzdaten an die Endpunkte senden. Die HTTP-Antwort wird dem Resolver zugänglich gemacht, der die endgültige Antwort zurückgibt, nachdem er seine Operation (en) abgeschlossen hat.

Eine Datenquelle hinzufügen

Wenn Sie eine Datenquelle erstellt haben, können Sie sie mit dem AWS AppSync Dienst und insbesondere mit der API verknüpfen.

Console

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.

   a. Wählen Sie Ihre API im Dashboard aus.

b. Wählen Sie in der Seitenleiste Datenquellen aus.

2. Klicken Sie auf Create data source.

   a. Geben Sie Ihrer Datenquelle einen Namen. Sie können ihr auch eine Beschreibung geben, aber das ist optional.

   b. Wählen Sie Ihren Datenquellentyp.

   c. Für DynamoDB müssen Sie Ihre Region und dann die Tabelle in der Region auswählen. Sie können Interaktionsregeln für Ihre Tabelle vorschreiben, indem Sie eine neue generische Tabellenrolle erstellen oder eine bestehende Rolle für die Tabelle importieren. Sie können die Versionsverwaltung aktivieren, sodass automatisch Datenversionen für jede Anfrage erstellt werden, wenn mehrere Clients versuchen, Daten gleichzeitig zu aktualisieren. Die Versionierung wird verwendet, um mehrere Datenvarianten zur Konflikterkennung und Konfliktlösung aufzubewahren und zu pflegen. Sie können auch die automatische Schemagenerierung aktivieren, bei der anhand Ihrer Datenquelle einige der CRUD- und `Query` Operationen generiert werden`List`, die für den Zugriff auf diese Daten in Ihrem Schema erforderlich sind.

      Dafür OpenSearch müssen Sie Ihre Region und dann die Domain (Cluster) in der Region auswählen. Sie können Interaktionsregeln mit Ihrer Domain diktieren, indem Sie eine neue generische Tabellenrolle erstellen oder eine bestehende Rolle für die Tabelle importieren.

      Für Lambda müssen Sie Ihre Region und dann den ARN der Lambda-Funktion in der Region auswählen. Sie können Interaktionsregeln mit Ihrer Lambda-Funktion diktieren, indem Sie eine neue generische Tabellenrolle erstellen oder eine vorhandene Rolle für die Tabelle importieren.

      Für HTTP müssen Sie Ihren HTTP-Endpunkt eingeben.

      Dafür EventBridge müssen Sie Ihre Region und dann den Event-Bus in der Region auswählen. Sie können Interaktionsregeln mit Ihrem Event-Bus vorschreiben, indem Sie eine neue generische Tabellenrolle erstellen oder eine bestehende Rolle für die Tabelle importieren.

      Für RDS müssen Sie Ihre Region, dann den geheimen Speicher (Benutzername und Passwort), den Datenbanknamen und das Schema auswählen.

Für keine Daten fügen Sie eine Datenquelle ohne tatsächliche Datenquelle hinzu. Dies dient dazu, Resolver lokal und nicht über eine tatsächliche Datenquelle zu verarbeiten.

> **Note**
>
> Wenn Sie vorhandene Rollen importieren, benötigen sie eine Vertrauensrichtlinie. Weitere Informationen finden Sie in der IAM-Vertrauensrichtlinie.

3. Wählen Sie Erstellen aus.

> **Note**
>
> Wenn Sie eine DynamoDB-Datenquelle erstellen, können Sie alternativ zur Schemaseite in der Konsole gehen, oben auf der Seite Ressourcen erstellen auswählen und dann ein vordefiniertes Modell ausfüllen, das in eine Tabelle umgewandelt werden soll. Bei dieser Option füllen Sie den Basistyp aus oder importieren ihn, konfigurieren die grundlegenden Tabellendaten einschließlich des Partitionsschlüssels und überprüfen die Schemaänderungen.

CLI

- Erstellen Sie Ihre Datenquelle, indem Sie den `create-data-source`Befehl ausführen.

  Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

  1. Die `api-id` Ihrer API.
  2. Die `name` deines Tisches.
  3. Die `type` der Datenquelle. Je nach ausgewähltem Datenquellentyp müssen Sie möglicherweise ein Tag `service-role-arn` und ein `-config` Tag eingeben.

  Ein Beispielbefehl könnte wie folgt aussehen:

```
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**
>
> Bevor Sie das CDK verwenden, empfehlen wir, die offizielle Dokumentation des CDK zusammen mit der CDK-Referenz zu AWS AppSync lesen.
> Die unten aufgeführten Schritte zeigen nur ein allgemeines Beispiel für das Snippet, das zum Hinzufügen einer bestimmten Ressource verwendet wurde. Dies soll keine funktionierende Lösung in Ihrem Produktionscode sein. Wir gehen auch davon aus, dass Sie bereits eine funktionierende App haben.

Um Ihre spezielle Datenquelle hinzuzufügen, müssen Sie das Konstrukt zu Ihrer Stack-Datei hinzufügen. Eine Liste der Datenquellentypen finden Sie hier:

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

1. Im Allgemeinen müssen Sie möglicherweise die Import-Direktive zu dem Dienst hinzufügen, den Sie verwenden. Zum Beispiel kann es den folgenden Formen folgen:

   ```
   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'
   ```

   So könnten Sie beispielsweise die Dienste AWS AppSync und DynamoDB importieren:

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

2. Einige Dienste wie RDS erfordern einige zusätzliche Einstellungen in der Stack-Datei, bevor die Datenquelle erstellt wird (z. B. VPC-Erstellung, Rollen und Zugangsdaten). Weitere Informationen finden Sie in den Beispielen auf den entsprechenden CDK-Seiten.

3. Für die meisten Datenquellen, insbesondere AWS Dienste, erstellen Sie eine neue Instanz der Datenquelle in Ihrer Stack-Datei. In der Regel sieht das wie folgt aus:

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

Hier ist zum Beispiel ein Beispiel für eine Amazon DynamoDB-Tabelle:

```
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**
>
> Für die meisten Datenquellen ist mindestens eine Requisite erforderlich (wird ohne Symbol gekennzeichnet). ? Schlagen Sie in der CDK-Dokumentation nach, welche Requisiten benötigt werden.

4. Als Nächstes müssen Sie die Datenquelle mit der GraphQL-API verknüpfen. Die empfohlene Methode besteht darin, sie hinzuzufügen, wenn Sie eine Funktion für Ihren Pipeline-Resolver erstellen. Das folgende Snippet ist beispielsweise eine Funktion, die alle Elemente in einer DynamoDB-Tabelle scannt:

```
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,
});
```

In den `dataSource` Requisiten können Sie die GraphQL-API (`add_api`) aufrufen und eine der integrierten Methoden (`addDynamoDbDataSource`) verwenden, um die Verknüpfung zwischen der Tabelle und der GraphQL-API herzustellen. Die Argumente sind der Name dieses Links, der in der AWS AppSync Konsole existieren wird (`data_source_name_in_console`in diesem Beispiel), und die Tabellenmethode (). `add_ddb_table` Mehr zu diesem Thema erfahren Sie im nächsten Abschnitt, wenn Sie mit der Erstellung von Resolvern beginnen.

Es gibt alternative Methoden zum Verknüpfen einer Datenquelle. Technisch gesehen könnten Sie `api` die Requisitenliste in der Tabellenfunktion erweitern. Hier ist zum Beispiel der Ausschnitt aus Schritt 3, aber mit `api` Requisiten, die eine GraphQL-API enthalten:

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

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

 ...

  api: add_api
});
```

Alternativ können Sie das Konstrukt separat aufrufen: `GraphqlApi`

```
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);
```

Wir empfehlen, die Assoziation nur in den Requisiten der Funktion zu erstellen. Andernfalls müssen Sie entweder Ihre Resolverfunktion manuell in der Konsole mit der Datenquelle verknüpfen (wenn Sie den AWS AppSync Konsolenwert weiterhin verwenden möchten`data_source_name_in_console`) oder eine separate Zuordnung in der Funktion unter einem anderen Namen wie erstellen. `data_source_name_in_console_2` Dies ist auf Einschränkungen bei der Verarbeitung von Informationen durch die Requisiten zurückzuführen.

> **Note**
>
> Sie müssen die App erneut bereitstellen, um Ihre Änderungen zu sehen.

IAM-Vertrauensrichtlinie

Wenn Sie eine bestehende IAM-Rolle für Ihre Datenquelle verwenden, müssen Sie dieser Rolle die entsprechenden Berechtigungen gewähren, um Operationen an Ihrer AWS Ressource, z. B. `PutItem` an einer Amazon DynamoDB-Tabelle, durchzuführen. Sie müssen auch die Vertrauensrichtlinie für diese Rolle ändern, damit sie für den AWS AppSync Ressourcenzugriff verwendet werden kann, wie in der folgenden Beispielrichtlinie dargestellt:

JSON

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

Sie können Ihrer Vertrauensrichtlinie auch Bedingungen hinzufügen, um den Zugriff auf die Datenquelle nach Bedarf einzuschränken. Derzeit `SourceArn` können unter diesen Bedingungen auch `SourceAccount` Schlüssel verwendet werden. Beispielsweise beschränkt die folgende Richtlinie den Zugriff auf Ihre Datenquelle auf das Konto`123456789012`:

JSON

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

Alternativ können Sie den Zugriff auf eine Datenquelle auf eine bestimmte API beschränken, z. B. `abcdefghijklmnopq` mithilfe der folgenden Richtlinie:

JSON

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

Sie können den Zugriff auf alle AWS AppSync APIs Benutzer aus einer bestimmten Region einschränken`us-east-1`, z. B. mithilfe der folgenden Richtlinie:

JSON

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

Im nächsten Abschnitt (Konfiguration von Resolvern) fügen wir unsere Resolver-Geschäftslogik hinzu und hängen sie an die Felder in unserem Schema an, um die Daten in unserer Datenquelle zu verarbeiten.

Weitere Informationen zur Konfiguration von Rollenrichtlinien finden Sie unter Ändern einer Rolle im IAM-Benutzerhandbuch.

Weitere Informationen zum kontoübergreifenden Zugriff auf AWS Lambda Resolver für finden Sie unter Kontenübergreifende AWS Lambda Resolver für AWS AppSync erstellen. AWS AppSync

Resolver konfigurieren in AWS AppSync

In den vorherigen Abschnitten haben Sie gelernt, wie Sie Ihr GraphQL-Schema und Ihre Datenquelle erstellen und diese dann im AWS AppSync Service miteinander verknüpft haben. In Ihrem Schema haben Sie möglicherweise ein oder mehrere Felder (Operationen) in Ihrer Abfrage und Mutation eingerichtet. Das Schema beschrieb zwar die Arten von Daten, die die Operationen von der Datenquelle anfordern würden, es wurde jedoch nie implementiert, wie sich diese Operationen in Bezug auf die Daten verhalten würden.

Das Verhalten einer Operation wird immer im Resolver implementiert, der mit dem Feld verknüpft wird, das die Operation ausführt. Weitere Informationen zur Funktionsweise von Resolvern im Allgemeinen finden Sie auf der Resolver-Seite.

In AWS AppSync ist Ihr Resolver an eine Runtime gebunden, das ist die Umgebung, in der Ihr Resolver ausgeführt wird. Laufzeiten bestimmen die Sprache, in der Ihr Resolver geschrieben wird. Derzeit werden zwei Laufzeiten unterstützt: APPSYNC_JS (JavaScript) und Apache Velocity Template Language (VTL).

Bei der Implementierung von Resolvern gibt es eine allgemeine Struktur, der sie folgen:

- Schritt vor dem: Wenn eine Anfrage vom Client gestellt wird, werden den Resolvern für die verwendeten Schemafelder (normalerweise Ihre Abfragen, Mutationen, Abonnements) die Anforderungsdaten übergeben. Der Resolver beginnt mit der Verarbeitung der Anforderungsdaten mit einem Before-Step-Handler, der es ermöglicht, einige Vorverarbeitungsvorgänge durchzuführen, bevor die Daten den Resolver passieren.
- Funktion (en): Nachdem der vorherige Schritt ausgeführt wurde, wird die Anforderung an die Funktionsliste übergeben. Die erste Funktion in der Liste wird für die Datenquelle ausgeführt. Eine Funktion ist eine Teilmenge des Codes Ihres Resolvers, die einen eigenen Anfrage- und Antworthandler enthält. Ein Anforderungshandler nimmt die Anforderungsdaten und führt Operationen an der Datenquelle durch. Der Antworthandler verarbeitet die Antwort der Datenquelle, bevor er sie an die Liste zurückgibt. Wenn es mehr als eine Funktion gibt, werden die Anforderungsdaten zur Ausführung an die nächste Funktion in der Liste gesendet. Die Funktionen in der Liste werden seriell in der vom Entwickler festgelegten Reihenfolge ausgeführt. Sobald alle Funktionen ausgeführt wurden, wird das Endergebnis an den nächsten Schritt übergeben.

- Nach dem Schritt: Der Nachschritt ist eine Handler-Funktion, mit der Sie einige letzte Operationen an der Antwort der endgültigen Funktion ausführen können, bevor Sie sie an die GraphQL-Antwort übergeben.

Dieser Flow ist ein Beispiel für einen Pipeline-Resolver. Pipeline-Resolver werden in beiden Laufzeiten unterstützt. Dies ist jedoch eine vereinfachte Erklärung dessen, was Pipeline-Resolver leisten können. Außerdem beschreiben wir nur eine mögliche Resolver-Konfiguration. Weitere Informationen zu unterstützten Resolver-Konfigurationen finden Sie in der Resolver-Übersicht für APPSYNC_JS oder in der Übersicht über JavaScript Resolver-Mapping-Vorlagen für VTL.

Wie Sie sehen können, sind Resolver modular aufgebaut. Damit die Komponenten des Resolvers ordnungsgemäß funktionieren, müssen sie in der Lage sein, von anderen Komponenten aus auf den Status der Ausführung zuzugreifen. Aus dem Abschnitt Resolver wissen Sie, dass jeder Komponente im Resolver wichtige Informationen über den Status der Ausführung als Satz von Argumenten (`argscontext`, usw.) übergeben werden können. Darin wird AWS AppSync dies ausschließlich von der abgewickelt. `context` Es ist ein Container für die Informationen über das Feld, das aufgelöst wird. Dies kann alles beinhalten, von übergebenen Argumenten über Ergebnisse bis hin zu Autorisierungsdaten, Header-Daten usw. Weitere Informationen zum Kontext finden Sie in der Resolver-Kontext-Objektreferenz für APPSYNC_JS oder in der Kontextreferenz für Resolver-Mapping-Vorlagen für VTL.

Der Kontext ist nicht das einzige Tool, mit dem Sie Ihren Resolver implementieren können. AWS AppSync unterstützt eine Vielzahl von Hilfsprogrammen für die Wertgenerierung, Fehlerbehandlung, Analyse, Konvertierung usw. Eine Liste der Dienstprogramme finden Sie hier für APPSYNC_JS oder hier für VTL.

In den folgenden Abschnitten erfahren Sie, wie Sie Resolver in Ihrer GraphQL-API konfigurieren.

Themen

- Einfache Abfragen erstellen (JavaScript)
- Basisabfragen erstellen (VTL)

Einfache Abfragen erstellen (JavaScript)

GraphQL-Resolver verbinden die Felder in einem Schema mit einer Datenquelle. Resolver sind der Mechanismus, mit dem Anfragen erfüllt werden.

Resolver, die AWS AppSync verwendet werden JavaScript , um einen GraphQL-Ausdruck in ein Format zu konvertieren, das die Datenquelle verwenden kann. Alternativ können Mapping-Vorlagen in Apache Velocity Template Language (VTL) geschrieben werden, um einen GraphQL-Ausdruck in ein Format zu konvertieren, das die Datenquelle verwenden kann.

In diesem Abschnitt wird beschrieben, wie Resolver mithilfe von konfiguriert werden. JavaScript Der Abschnitt Resolver-Tutorials (JavaScript) enthält ausführliche Anleitungen zur Implementierung von Resolvern mithilfe von. JavaScript Der Abschnitt Resolver-Referenz (JavaScript) enthält eine Erläuterung der Hilfsoperationen, die mit Resolvern verwendet werden können. JavaScript

Wir empfehlen, diese Anleitung zu befolgen, bevor Sie versuchen, eines der oben genannten Tutorials zu verwenden.

In diesem Abschnitt werden wir uns mit der Erstellung und Konfiguration von Resolvern für Abfragen und Mutationen befassen.

> **Note**
>
> In diesem Handbuch wird davon ausgegangen, dass Sie Ihr Schema erstellt haben und mindestens eine Abfrage oder Mutation haben. Wenn Sie nach Abonnements (Echtzeitdaten) suchen, finden Sie in dieser Anleitung weitere Informationen.

In diesem Abschnitt finden Sie einige allgemeine Schritte zur Konfiguration von Resolvern sowie ein Beispiel, das das folgende Schema verwendet:

```
// 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]
}
```

Einfache Abfragerolver erstellen

In diesem Abschnitt erfahren Sie, wie Sie einen einfachen Abfrageauflöser erstellen.

Console

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.
   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.
   b. Wählen Sie in der Seitenleiste Schema aus.
2. Geben Sie die Details Ihres Schemas und Ihrer Datenquelle ein. Weitere Informationen finden Sie in den Abschnitten Entwerfen Ihres Schemas und Anhängen einer Datenquelle.
3. Neben dem Schema-Editor befindet sich ein Fenster mit dem Namen Resolvers. Dieses Feld enthält eine Liste der Typen und Felder, wie sie in Ihrem Schemafenster definiert sind. Sie können Resolver an Felder anhängen. Höchstwahrscheinlich werden Sie Resolver an Ihre Feldoperationen anhängen. In diesem Abschnitt werden wir uns einfache Abfragekonfigurationen ansehen. Wählen Sie unter dem Abfragetyp neben dem Feld Ihrer Abfrage die Option Anhängen aus.
4. Auf der Seite Resolver anhängen können Sie unter Resolvertyp zwischen Pipeline- und Unit-Resolvern wählen. Weitere Informationen zu diesen Typen finden Sie unter Resolver. In diesem Handbuch werden verwendet. `pipeline resolvers`

> **Tip**
>
> Bei der Erstellung von Pipeline-Resolvern werden Ihre Datenquellen an die Pipeline-Funktion (en) angehängt. Funktionen werden erstellt, nachdem Sie den Pipeline-Resolver selbst erstellt haben. Aus diesem Grund gibt es auf dieser Seite keine Option, sie festzulegen. Wenn Sie einen Unit-Resolver verwenden, ist die Datenquelle direkt mit dem Resolver verknüpft, sodass Sie sie auf dieser Seite einstellen würden.

Wählen Sie für die Resolver-Runtime, ob Sie `APPSYNC_JS` die Laufzeit aktivieren möchten. JavaScript

5. Sie können das Caching für diese API aktivieren. Wir empfehlen, diese Funktion vorerst auszuschalten. Wählen Sie Erstellen aus.
6. Auf der Seite Resolver bearbeiten gibt es einen Code-Editor namens Resolver-Code, mit dem Sie die Logik für den Resolver-Handler und die Antwort (vor und nach den Schritten) implementieren können. Weitere Informationen finden Sie in der Übersicht über die JavaScript Resolver.

> **Note**
>
> In unserem Beispiel lassen wir die Anfrage einfach leer und die Antwort wird so eingestellt, dass sie das letzte Datenquellenergebnis aus dem Kontext zurückgibt:

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

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

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

Unterhalb dieses Abschnitts befindet sich eine Tabelle mit dem Namen Functions. Mit Funktionen können Sie Code implementieren, der für mehrere Resolver wiederverwendet werden kann. Anstatt Code ständig neu zu schreiben oder zu kopieren, können Sie den Quellcode als Funktion speichern, die Sie einem Resolver hinzufügen können, wann immer Sie ihn benötigen.

Funktionen machen den Großteil der Operationsliste einer Pipeline aus. Wenn Sie mehrere Funktionen in einem Resolver verwenden, legen Sie die Reihenfolge der Funktionen fest, und sie werden nacheinander in dieser Reihenfolge ausgeführt. Sie werden ausgeführt, nachdem die Anforderungsfunktion ausgeführt wurde und bevor die Antwortfunktion beginnt.

Um eine neue Funktion hinzuzufügen, wählen Sie unter Funktionen die Option Funktion hinzufügen und dann Neue Funktion erstellen aus. Alternativ wird Ihnen möglicherweise die Schaltfläche Funktion erstellen angezeigt, aus der Sie stattdessen auswählen können.

a. Wählen Sie eine Datenquelle aus. Dies ist die Datenquelle, auf die der Resolver reagiert.

> **Note**
>
> In unserem Beispiel hängen wir einen Resolver für an`getPost`, der ein Objekt von abruft. `Post` `id` Nehmen wir an, wir haben bereits eine DynamoDB-Tabelle für dieses Schema eingerichtet. Ihr Partitionsschlüssel ist auf „gesetzt“ `id` und leer.

b. Geben Sie ein`Function name`.

c. Unter Funktionscode müssen Sie das Verhalten der Funktion implementieren. Das mag verwirrend sein, aber jede Funktion wird ihren eigenen lokalen Anfrage- und Antworthandler haben. Die Anforderung wird ausgeführt, dann erfolgt der Datenquellenaufruf, um die Anfrage zu bearbeiten, und dann wird die Datenquellenantwort vom Antworthandler verarbeitet. Das Ergebnis wird im Kontextobjekt gespeichert. Danach wird die nächste Funktion in der Liste ausgeführt oder an den Antworthandler nach dem Schritt übergeben, falls es sich um die letzte handelt.

> **Note**
>
> In unserem Beispiel hängen wir einen Resolver an`getPost`, der eine Liste von `Post` Objekten aus der Datenquelle erhält. Unsere Anforderungsfunktion fordert die Daten aus unserer Tabelle an, die Tabelle leitet ihre Antwort an den Kontext weiter (ctx), dann gibt die Antwort das Ergebnis im Kontext zurück. AWS AppSync Die Stärke liegt in der Vernetzung mit anderen Diensten. AWS Da wir DynamoDB verwenden, verfügen wir über eine Reihe von Operationen, um solche Dinge zu vereinfachen. Wir haben auch einige Standardbeispiele für andere Datenquellentypen.
> Unser Code wird so aussehen:
>
> ```
> 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;
}
```

In diesem Schritt haben wir zwei Funktionen hinzugefügt:

- `request`: Der Anforderungshandler führt den Abrufvorgang für die Datenquelle aus. Das Argument enthält das Kontextobjekt (`ctx`) oder einige Daten, die allen Resolvern zur Verfügung stehen, die eine bestimmte Operation ausführen. Es kann beispielsweise Autorisierungsdaten, die aufgelösten Feldnamen usw. enthalten. Die Return-Anweisung führt eine `Scan`Operation aus (Beispiele finden Sie hier). Da wir mit DynamoDB arbeiten, dürfen wir einige der Operationen dieses Dienstes verwenden. Der Scan führt einen einfachen Abruf aller Elemente in unserer Tabelle durch. Das Ergebnis dieser Operation wird im Kontextobjekt als `result` Container gespeichert, bevor es an den Antworthandler übergeben wird. Das `request` wird vor der Antwort in der Pipeline ausgeführt.
- `response`: Der Antworthandler, der die Ausgabe von zurückgibt`request`. Das Argument ist das aktualisierte Kontextobjekt und die Rückgabeanweisung ist`ctx.prev.result`. Zu diesem Zeitpunkt in der Anleitung sind Sie mit diesem Wert möglicherweise noch nicht vertraut. `ctx`bezieht sich auf das Kontextobjekt. `prev`bezieht sich auf die vorherige Operation in der Pipeline, die unsere war`request`. Der `result` enthält die Ergebnisse des Resolvers, während er sich durch die Pipeline bewegt. Wenn Sie alles zusammenfügen, `ctx.prev.result` wird das Ergebnis der letzten ausgeführten Operation zurückgegeben, bei der es sich um den Request-Handler handelte.

d. Wenn Sie fertig sind, wählen Sie Create.

7. Zurück auf dem Resolver-Bildschirm wählen Sie unter Funktionen das Drop-down-Menü Funktion hinzufügen und fügen Sie Ihre Funktion zu Ihrer Funktionsliste hinzu.
8. Wählen Sie Speichern, um den Resolver zu aktualisieren.

CLI

Um Ihre Funktion hinzuzufügen

- Erstellen Sie mit dem `create-function` Befehl eine Funktion für Ihren Pipeline-Resolver.

  Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

  1. Die `api-id` Ihrer API.
  2. Die `name` der Funktion in der AWS AppSync Konsole.
  3. Der oder der Name der Datenquelle`data-source-name`, die die Funktion verwenden wird. Es muss bereits erstellt und mit Ihrer GraphQL-API im AWS AppSync Service verknüpft sein.
  4. Die `runtime` oder Umgebung und die Sprache der Funktion. Für JavaScript muss `APPSYNC_JS` der Name und die Laufzeit, lauten`1.0.0`.
  5. Die `code` oder Request- und Response-Handler Ihrer Funktion. Sie können es zwar manuell eingeben, es ist jedoch viel einfacher, es einer TXT-Datei (oder einem ähnlichen Format) hinzuzufügen und es dann als Argument zu übergeben.

  > **Note**
  >
  > Unser Abfragecode wird sich in einer Datei befinden, die als Argument übergeben wird:

```
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;
}
```

Ein Beispielbefehl könnte so aussehen:

```
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}
```

Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

```
{
    "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**
>
> Stellen Sie sicher, dass Sie `functionId` irgendwo aufzeichnen, da dies verwendet wird, um die Funktion an den Resolver anzuhängen.

Um deinen Resolver zu erstellen

- Erstellen Sie eine Pipeline-Funktion für, `Query` indem Sie den `create-resolver` Befehl ausführen.

  Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

  1. Die `api-id` Ihrer API.
  2. Der `type-name` oder der spezielle Objekttyp in Ihrem Schema (Query, Mutation, Subscription).
  3. Die `field-name` oder die Feldoperation innerhalb des speziellen Objekttyps, an den Sie den Resolver anhängen möchten.
  4. Der`kind`, der einen Units- oder Pipeline-Resolver angibt. Stellen Sie diesen Wert auf ein, `PIPELINE` um Pipeline-Funktionen zu aktivieren.
  5. Die oder die Funktion (en)`pipeline-config`, die an den Resolver angehängt werden sollen. Stellen Sie sicher, dass Sie die `functionId` Werte Ihrer Funktionen kennen. Die Reihenfolge der Auflistung ist wichtig.
  6. Das`runtime`, was war `APPSYNC_JS` (JavaScript). Das ist `runtimeVersion` derzeit`1.0.0`.
  7. Der`code`, der die Vorher- und Nachher-Schritthandler enthält.

  > **Note**
  >
  > Unser Abfragecode wird in einer Datei gespeichert, die als Argument übergeben wird:

```
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;
}
```

Ein Beispielbefehl könnte so aussehen:

```
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}
```

Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

```
{
    "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**
>
> Bevor Sie das CDK verwenden, empfehlen wir, die offizielle Dokumentation des CDK zusammen mit der CDK-Referenz zu AWS AppSync lesen.
> Die unten aufgeführten Schritte zeigen nur ein allgemeines Beispiel für das Snippet, das zum Hinzufügen einer bestimmten Ressource verwendet wurde. Dies soll keine funktionierende Lösung in Ihrem Produktionscode sein. Wir gehen auch davon aus, dass Sie bereits eine funktionierende App haben.

Eine Basis-App benötigt die folgenden Dinge:

1. Richtlinien für den Import von Diensten
2. Schemacode
3. Datenquellengenerator
4. Funktionscode
5. Resolver-Code

Aus den Abschnitten Entwerfen Ihres Schemas und Anhängen einer Datenquelle wissen wir, dass die Stack-Datei die Importanweisungen des Formulars enthalten wird:

```
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**
>
> In den vorherigen Abschnitten haben wir nur erklärt, wie AWS AppSync Konstrukte importiert werden. In echtem Code müssen Sie mehr Dienste importieren, nur um die App auszuführen. Wenn wir in unserem Beispiel eine sehr einfache CDK-App erstellen würden, würden wir zumindest den AWS AppSync Service zusammen mit unserer Datenquelle importieren, bei der es sich um eine DynamoDB-Tabelle handelt. Wir müssten auch einige zusätzliche Konstrukte importieren, um die App bereitzustellen:
>
> ```
> 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';
```

Um jedes dieser Dinge zusammenzufassen:

- `import * as cdk from 'aws-cdk-lib';`: Auf diese Weise können Sie Ihre CDK-App und Konstrukte wie den Stack definieren. Es enthält auch einige nützliche Hilfsfunktionen für unsere Anwendung, wie z. B. die Manipulation von Metadaten. Wenn Sie mit dieser Importdirektive vertraut sind, sich aber fragen, warum die CDK-Kernbibliothek hier nicht verwendet wird, lesen Sie die Migrationsseite.
- `import * as appsync from 'aws-cdk-lib/aws-appsync';`: Dadurch wird der AWS AppSync Dienst importiert.
- `import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';`: Dadurch wird der DynamoDB-Dienst importiert.
- `import { Construct } from 'constructs';`: Wir benötigen das, um das Root-Konstrukt zu definieren.

Die Art des Imports hängt von den Diensten ab, die Sie aufrufen. Wir empfehlen, in der CDK-Dokumentation nach Beispielen zu suchen. Das Schema oben auf der Seite wird eine separate Datei in Ihrer CDK-App als `.graphql` Datei sein. In der Stack-Datei können wir sie mit einem neuen GraphQL verknüpfen, indem wir das folgende Formular verwenden:

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

Note

In diesem Bereich `add_api` fügen wir eine neue GraphQL-API hinzu, die das `new` Schlüsselwort gefolgt von `appsync.GraphqlApi(scope: Construct, id: string , props: GraphqlApiProps)` verwendet. Unser Geltungsbereich lautet`this`, die CFN-ID lautet`graphQL-example`, und unsere Requisiten sind `my-first-api` (Name der API in der Konsole) und `schema.graphql` (der absolute Pfad zur Schemadatei).

Um eine Datenquelle hinzuzufügen, müssen Sie zuerst Ihre Datenquelle zum Stack hinzufügen. Anschließend müssen Sie es mithilfe der quellenspezifischen Methode mit der GraphQL-API verknüpfen. Die Zuordnung erfolgt, wenn Sie Ihren Resolver zum Laufen bringen. Lassen Sie uns in der Zwischenzeit ein Beispiel verwenden, indem wir die DynamoDB-Tabelle wie folgt erstellen: `dynamodb.Table`

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

> **Note**
>
> Wenn wir dies in unserem Beispiel verwenden würden, würden wir eine neue DynamoDB-Tabelle mit der CFN-ID von `posts-table` und dem Partitionsschlüssel von hinzufügen. `id (S)`

Als Nächstes müssen wir unseren Resolver in der Stack-Datei implementieren. Hier ist ein Beispiel für eine einfache Abfrage, die nach allen Elementen in einer DynamoDB-Tabelle sucht:

```
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

Zuerst haben wir eine Funktion namens erstellt. `add_func` Diese Reihenfolge der Erstellung mag etwas kontraintuitiv erscheinen, aber Sie müssen die Funktionen in Ihrem Pipeline-Resolver erstellen, bevor Sie den Resolver selbst erstellen. Eine Funktion folgt der Form:

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

Unser Anwendungsbereich war`this`, unsere CFN-ID war`func-get-posts`, und unsere Requisiten enthielten die tatsächlichen Funktionsdetails. Zu den Requisiten gehörten:

- Die Funktion, die in `name` der AWS AppSync Konsole vorhanden sein wird (`get_posts_func_1`).
- Die GraphQL-API, die wir zuvor erstellt haben (`add_api`).
- Die Datenquelle; dies ist der Punkt, an dem wir die Datenquelle mit dem GraphQL-API-Wert verknüpfen und sie dann an die Funktion anhängen. Wir nehmen die Tabelle, die wir erstellt haben (`add_ddb_table`) und hängen sie mit einer der `GraphqlApi` Methoden (`add_api`) an die GraphQL-API (`addDynamoDbDataSource`) an. Der ID-Wert (`table-for-posts`) ist der Name der Datenquelle in der AWS AppSync Konsole. Eine Liste quellenspezifischer Methoden finden Sie auf den folgenden Seiten:
  - DynamoDbDataSource
  - EventBridgeDataSource

  - HttpDataSource
  - LambdaDataSource
  - NoneDataSource
  - OpenSearchDataSource
  - RdsDataSource
- Der Code enthält die Request- und Response-Handler unserer Funktion, was ein einfaches Scannen und Zurückgeben ist.
- Die Laufzeit gibt an, dass wir die APPSYNC_JS-Laufzeitversion 1.0.0 verwenden möchten. Beachten Sie, dass dies derzeit die einzige Version ist, die für APPSYNC_JS verfügbar ist.

Als Nächstes müssen wir die Funktion an den Pipeline-Resolver anhängen. Wir haben unseren Resolver mit der folgenden Form erstellt:

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

Unser Umfang war`this`, unsere CFN-ID war`pipeline-resolver-get-posts`, und unsere Requisiten enthielten die tatsächlichen Funktionsdetails. Zu den Requisiten gehörten:

- Die GraphQL-API, die wir zuvor erstellt haben (`add_api`).
- Der Name des speziellen Objekttyps; dies ist eine Abfrageoperation, also haben wir einfach den Wert `Query` hinzugefügt.
- Der Feldname (`getPost`) ist der Name des Felds im Schema unter dem `Query` Typ.
- Der Code enthält Ihre Vorher- und Nachher-Handler. Unser Beispiel gibt nur die Ergebnisse zurück, die sich im Kontext befanden, nachdem die Funktion ihre Operation ausgeführt hat.
- Die Laufzeit gibt an, dass wir die APPSYNC_JS-Laufzeitversion 1.0.0 verwenden möchten. Beachten Sie, dass dies derzeit die einzige Version ist, die für APPSYNC_JS verfügbar ist.
- Die Pipeline-Konfiguration enthält den Verweis auf die von uns erstellte Funktion (). `add_func`

Um zusammenzufassen, was in diesem Beispiel passiert ist: Sie haben eine AWS AppSync Funktion gesehen, die einen Anfrage- und Antworthandler implementiert hat. Die Funktion war für die Interaktion mit Ihrer Datenquelle verantwortlich. Der Anforderungshandler hat eine `Scan` Operation an gesendet und ihm mitgeteilt AWS AppSync, welche Operation mit Ihrer DynamoDB-Datenquelle ausgeführt werden soll. Der Antworthandler hat die Liste der Elemente () zurückgegeben. `ctx.result.items` Die Liste der Elemente wurde dann automatisch dem `Post` GraphQL-Typ zugeordnet.

Erstellung grundlegender Mutationslöser

In diesem Abschnitt erfahren Sie, wie Sie einen einfachen Mutationslöser erstellen.

Console

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.

   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.

   b. Wählen Sie in der Seitenleiste Schema aus.

2. Wählen Sie unter dem Abschnitt Resolver und dem Mutationstyp neben Ihrem Feld die Option Anhängen aus.

   > **Note**
   >
   > In unserem Beispiel hängen wir einen Resolver für an`createPost`, der unserer Tabelle ein `Post` Objekt hinzufügt. Nehmen wir an, wir verwenden dieselbe DynamoDB-Tabelle aus dem letzten Abschnitt. Ihr Partitionsschlüssel ist auf „gesetzt“ `id` und leer.

3. Wählen Sie auf der Seite Attach Resolver unter Resolver-Typ die Option. `pipeline resolvers` Zur Erinnerung: Weitere Informationen zu Resolvern finden Sie hier. Wählen Sie für Resolver Runtime aus, ob Sie `APPSYNC_JS` die Laufzeit aktivieren möchten. JavaScript
4. Sie können das Caching für diese API aktivieren. Wir empfehlen, diese Funktion vorerst auszuschalten. Wählen Sie Erstellen aus.
5. Wähle „Funktion hinzufügen“ und dann „Neue Funktion erstellen“. Alternativ wird Ihnen möglicherweise die Schaltfläche Funktion erstellen angezeigt, aus der Sie stattdessen auswählen können.

a. Wählen Sie Ihre -Datenquelle aus. Dies sollte die Quelle sein, deren Daten Sie mit der Mutation manipulieren werden.

b. Geben Sie ein`Function name`.

c. Unter Funktionscode müssen Sie das Verhalten der Funktion implementieren. Da es sich um eine Mutation handelt, führt die Anfrage idealerweise einen Vorgang zur Änderung des Zustands an der aufgerufenen Datenquelle durch. Das Ergebnis wird von der Antwortfunktion verarbeitet.

> **Note**
>
> `createPost`fügt der Tabelle ein neues `Post` hinzu oder „fügt“ es mit unseren Parametern als Daten ein. Wir könnten so etwas hinzufügen:
>
> ```
> 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;
> }
> ```
>
> In diesem Schritt haben wir `request` auch `response` Funktionen hinzugefügt:
>
> - `request`: Der Request-Handler akzeptiert den Kontext als Argument. Die Return-Anweisung des Request-Handlers führt einen `PutItem`Befehl aus, bei dem es sich um einen integrierten DynamoDB-Vorgang handelt (Beispiele finden Sie hier oder hier). Der `PutItem` Befehl fügt unserer

DynamoDB-Tabelle ein `Post` Objekt hinzu, indem er den `key` Partitionswert (automatisch generiert von`util.autoid()`) und `attributes` die Eingabe des Kontextarguments verwendet (dies sind die Werte, die wir in unserer Anfrage übergeben werden). Die `key` Feldargumente `attributes` sind `id` und sind die `date` `title` Feldargumente. Sie sind beide über den `util.dynamodb.toMapValues`Helper vorformatiert, um mit der DynamoDB-Tabelle zu arbeiten.

- `response`: Die Antwort akzeptiert den aktualisierten Kontext und gibt das Ergebnis des Request-Handlers zurück.

d. Wählen Sie Create, wenn Sie fertig sind.

6. Zurück auf dem Resolver-Bildschirm wählen Sie unter Funktionen das Drop-down-Menü Funktion hinzufügen und fügen Sie Ihre Funktion zu Ihrer Funktionsliste hinzu.
7. Wählen Sie Speichern, um den Resolver zu aktualisieren.

CLI

Um Ihre Funktion hinzuzufügen

- Erstellen Sie mit dem `create-function` Befehl eine Funktion für Ihren Pipeline-Resolver.

  Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

  1. Die `api-id` Ihrer API.
  2. Die `name` der Funktion in der AWS AppSync Konsole.
  3. Der oder der Name der Datenquelle`data-source-name`, die die Funktion verwenden wird. Es muss bereits erstellt und mit Ihrer GraphQL-API im AWS AppSync Service verknüpft sein.
  4. Die `runtime` oder Umgebung und die Sprache der Funktion. Für JavaScript muss `APPSYNC_JS` der Name und die Laufzeit, lauten`1.0.0`.
  5. Die `code` oder Request- und Response-Handler Ihrer Funktion. Sie können es zwar manuell eingeben, aber es ist viel einfacher, es einer TXT-Datei (oder einem ähnlichen Format) hinzuzufügen und es dann als Argument zu übergeben.

> **Note**
>
> Unser Abfragecode wird sich in einer Datei befinden, die als Argument übergeben wird:

```
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;
}
```

Ein Beispielbefehl könnte so aussehen:

```
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}
```

Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

```
{
    "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**
>
> Stellen Sie sicher, dass Sie `functionId` irgendwo aufzeichnen, da dies verwendet wird, um die Funktion an den Resolver anzuhängen.

Um deinen Resolver zu erstellen

- Erstellen Sie eine Pipeline-Funktion für, `Mutation` indem Sie den `create-resolver` Befehl ausführen.

  Für diesen speziellen Befehl müssen Sie einige Parameter eingeben:

  1. Die `api-id` Ihrer API.
  2. Der `type-name` oder der spezielle Objekttyp in Ihrem Schema (Query, Mutation, Subscription).
  3. Die `field-name` oder die Feldoperation innerhalb des speziellen Objekttyps, an den Sie den Resolver anhängen möchten.
  4. Der`kind`, der einen Units- oder Pipeline-Resolver angibt. Stellen Sie diesen Wert auf ein, `PIPELINE` um Pipeline-Funktionen zu aktivieren.
  5. Die oder die Funktion (en)`pipeline-config`, die an den Resolver angehängt werden sollen. Stellen Sie sicher, dass Sie die `functionId` Werte Ihrer Funktionen kennen. Die Reihenfolge der Auflistung ist wichtig.
  6. Das`runtime`, was war `APPSYNC_JS` (JavaScript). Das ist `runtimeVersion` derzeit`1.0.0`.

7. Dercode, der den Vorher-Nachher-Schritt enthält.

> **Note**
>
> Unser Abfragecode wird in einer Datei enthalten sein, die als Argument übergeben wird:
>
> ```
> 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;
> }
> ```

Ein Beispielbefehl könnte so aussehen:

```
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}
```

Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

```
{
    "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**
>
> Bevor Sie das CDK verwenden, empfehlen wir, die offizielle Dokumentation des CDK zusammen mit der CDK-Referenz zu AWS AppSync lesen.
> Die unten aufgeführten Schritte zeigen nur ein allgemeines Beispiel für das Snippet, das zum Hinzufügen einer bestimmten Ressource verwendet wurde. Dies soll keine funktionierende Lösung in Ihrem Produktionscode sein. Wir gehen auch davon aus, dass Sie bereits eine funktionierende App haben.

- Um eine Mutation vorzunehmen, können Sie, vorausgesetzt, Sie befinden sich im selben Projekt, diese wie die Abfrage zur Stack-Datei hinzufügen. Hier sind eine modifizierte Funktion und ein modifizierter Resolver für eine Mutation, die der Tabelle eine neue `Post` hinzufügen:

```
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**
>
> Da diese Mutation und die Abfrage ähnlich strukturiert sind, erläutern wir nur die Änderungen, die wir vorgenommen haben, um die Mutation vorzunehmen.
> In der Funktion haben wir die CFN-ID in `func-add-post` und den Namen in geändert, `add_posts_func_1` um der Tatsache Rechnung zu tragen, dass wir der

Tabelle `Posts` etwas hinzufügen. In der Datenquelle haben wir eine neue Zuordnung zu unserer Tabelle (`add_ddb_table`) in der AWS AppSync Konsole vorgenommen, `table-for-posts-2` weil die `addDynamoDbDataSource` Methode dies erfordert. Denken Sie daran, dass diese neue Assoziation immer noch dieselbe Tabelle verwendet, die wir zuvor erstellt haben, aber wir haben jetzt zwei Verbindungen zu ihr in der AWS AppSync Konsole: eine für die Abfrage als `table-for-posts` und eine für die Mutation als`table-for-posts-2`. Der Code wurde dahingehend geändert, dass a hinzugefügt wurde, `Post` indem sein `id` Wert automatisch generiert wurde und die Eingabe eines Kunden für die restlichen Felder akzeptiert wurde.
Im Resolver haben wir den ID-Wert in geändert, `pipeline-resolver-create-posts` um der Tatsache Rechnung zu tragen, dass wir der Tabelle `Posts` etwas hinzufügen. Um die Mutation im Schema widerzuspiegeln, wurde der Typname in`Mutation`, und der Name, `createPost` geändert. Die Pipeline-Konfiguration wurde auf unsere neue Mutationsfunktion eingestellt`add_func_2`.

Um zusammenzufassen, was in diesem Beispiel passiert: Konvertiert die in dem `createPost` Feld definierten Argumente AWS AppSync automatisch aus Ihrem GraphQL-Schema in DynamoDB-Operationen. Das Beispiel speichert Datensätze in DynamoDB mit dem Schlüssel von`id`, der automatisch mit unserem `util.autoId()` Helper erstellt wird. Alle anderen Felder, die Sie aus Anfragen in der AWS AppSync Konsole oder auf andere Weise an die Kontextargumente (`ctx.args.input`) übergeben, werden als Attribute der Tabelle gespeichert. Sowohl der Schlüssel als auch die Attribute werden mithilfe des Helpers automatisch einem kompatiblen DynamoDB-Format zugeordnet. `util.dynamodb.toMapValues(values)`

AWS AppSync unterstützt auch Test- und Debug-Workflows für die Bearbeitung von Resolvern. Sie können ein `context` Scheinobjekt verwenden, um den transformierten Wert der Vorlage zu sehen, bevor Sie sie aufrufen. Optional können Sie die vollständige Anfrage an eine Datenquelle interaktiv anzeigen, wenn Sie eine Abfrage ausführen. Weitere Informationen finden Sie unter Resolver testen und debuggen (JavaScript) und Überwachen und Protokollieren.

Fortgeschrittene Resolver

Wenn Sie dem optionalen Abschnitt zur Paginierung unter Ihr Schema entwerfen folgen, müssen Sie dennoch Ihren Resolver zu Ihrer Anfrage hinzufügen, um die Paginierung nutzen zu können. In unserem Beispiel wurde eine Abfrage-Paginierung verwendet`getPosts`, die aufgerufen wurde, um nur einen Teil der gleichzeitig angeforderten Dinge zurückzugeben. Der Code unseres Resolvers in diesem Feld könnte so aussehen:

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

In der Anfrage geben wir den Kontext der Anfrage weiter. Unser `limit` ist*20*, was bedeutet, dass wir `Posts` bei der ersten Abfrage bis zu 20 zurückgeben. Unser `nextToken` Cursor ist auf den ersten `Post` Eintrag in der Datenquelle fixiert. Diese werden an die Argumente übergeben. Die Anfrage führt dann einen Scan von der ersten `Post` bis zur Anzahl der Scanlimits durch. Die Datenquelle speichert das Ergebnis im Kontext, der an die Antwort übergeben wird. Die Antwort gibt den `Posts` abgerufenen Wert zurück und setzt dann den `nextToken` Wert auf den `Post` Eintrag, der direkt hinter dem Grenzwert liegt. Die nächste Anfrage wird gesendet, um genau dasselbe zu tun, aber ab dem Offset direkt nach der ersten Abfrage. Denken Sie daran, dass diese Art von Anfragen sequentiell und nicht parallel ausgeführt werden.

Resolver in () testen und debuggen AWS AppSync JavaScript

AWS AppSync führt Resolver in einem GraphQL-Feld gegen eine Datenquelle aus. Bei der Arbeit mit Pipeline-Resolvern interagieren Funktionen mit Ihren Datenquellen. Wie in der Übersicht über JavaScript Resolver beschrieben, kommunizieren Funktionen mit Datenquellen, indem sie Anforderungs- und Antworthandler verwenden, die in der Laufzeit geschrieben sind JavaScript und dort ausgeführt werden. `APPSYNC_JS` Auf diese Weise können Sie vor und nach der Kommunikation mit der Datenquelle benutzerdefinierte Logik und Bedingungen bereitstellen.

Um Entwicklern beim Schreiben, Testen und Debuggen dieser Resolver zu helfen, bietet die AWS AppSync Konsole auch Tools zum Erstellen einer GraphQL-Anfrage und -Antwort mit Scheindaten bis hin zum einzelnen Field-Resolver. Darüber hinaus können Sie Abfragen, Mutationen und Abonnements in der AWS AppSync Konsole durchführen und einen detaillierten Protokollstream der

gesamten Anfrage von Amazon einsehen CloudWatch. Dies schließt Ergebnisse aus der Datenquelle ein.

Testen mit Scheindaten

Wenn ein GraphQL-Resolver aufgerufen wird, enthält er ein `context` Objekt, das relevante Informationen über die Anfrage enthält. Dazu gehören Argumente von einem Client, Identitätsinformationen sowie Daten aus dem übergeordneten GraphQL-Feld. Es speichert auch die Ergebnisse aus der Datenquelle, die im Antworthandler verwendet werden können. Weitere Informationen zu dieser Struktur und den verfügbaren Hilfsprogrammen, die beim Programmieren verwendet werden können, finden Sie in der Resolver-Kontextobjektreferenz.

Beim Schreiben oder Bearbeiten einer Resolverfunktion können Sie ein Modell- oder Testkontextobjekt an den Konsoleneditor übergeben. Auf diese Weise können Sie sehen, wie sowohl der Anforderungs- als auch der Antworthandler ausgewertet werden, ohne dass es tatsächlich mit einer Datenquelle zu tun hat. Beispiel: Sie können ein `firstname: Shaggy`-Testargument übergeben und sehen, wie es bei der Verwendung von `ctx.args.firstname` in Ihrem Vorlagencode ausgewertet wird. Sie können auch die Auswertung eines Dienstprogramm-Helferobjekts wie `util.autoId()` oder `util.time.nowISO8601()` testen.

Resolver werden getestet

In diesem Beispiel werden Resolver mithilfe der AWS AppSync Konsole getestet.

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.
   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.
   b. Wählen Sie in der Seitenleiste Funktionen.
2. Wählen Sie eine bestehende Funktion aus.
3. Wählen Sie oben auf der Seite „Funktion aktualisieren“ die Option Testkontext auswählen und anschließend Neuen Kontext erstellen aus.
4. Wählen Sie ein Beispielkontextobjekt aus oder füllen Sie die JSON-Datei manuell im Fenster Testkontext konfigurieren unten aus.
5. Geben Sie einen Namen für den Textkontext ein.
6. Wählen Sie die Schaltfläche Save (Speichern) aus.
7. Wählen Sie Run Test (Test ausführen), um den Resolver unter Verwendung dieses Mock-Kontexts auszuwerten.

Ein praktischeres Beispiel: Angenommen, Sie haben eine App, die einen GraphQL-Typ speichertDog, die automatische ID-Generierung für Objekte verwendet und diese in Amazon DynamoDB speichert. Sie möchten auch einige Werte aus den Argumenten einer GraphQL-Mutation schreiben und nur bestimmten Benutzern ermöglichen, eine Antwort zu sehen. Der folgende Ausschnitt zeigt, wie das Schema aussehen könnte:

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

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

Sie können eine AWS AppSync Funktion schreiben und sie Ihrem addDog Resolver hinzufügen, um die Mutation zu behandeln. Um Ihre AWS AppSync Funktion zu testen, können Sie ein Kontextobjekt wie im folgenden Beispiel auffüllen. Nachstehend werden die Argumente `name` und `age` des Clients verwendet und ein `username` im `identity`-Objekt mit Daten gefüllt:

```
{
    "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"
    }
}
```

Sie können Ihre AWS AppSync Funktion mit dem folgenden Code testen:

```
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();
}
```

Der ausgewertete Anforderungs- und Antworthandler enthält die Daten aus Ihrem Testkontextobjekt und den generierten Wert von`util.autoId()`. Wenn Sie `username` zu einem anderen Wert als `Nadia` ändern, werden die Ergebnisse nicht zurückgegeben, da die Autorisierungsprüfung fehlschlägt. Weitere Informationen zur detaillierten Zugriffskontrolle finden Sie unter Anwendungsfälle für die Autorisierung.

Testen von Anforderungs- und Antworthandlern mit AWS AppSync APIs

Sie können den `EvaluateCode` API-Befehl verwenden, um Ihren Code aus der Ferne mit simulierten Daten zu testen. Um mit dem Befehl zu beginnen, stellen Sie sicher, dass Sie die `appsync:evaluateMappingCode` entsprechende Berechtigung zu Ihrer Richtlinie hinzugefügt haben. Zum Beispiel:

JSON

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

```
    ]
}
```

Sie können den Befehl nutzen, indem Sie das AWS CLIoder verwenden AWS SDKs. Nehmen wir zum Beispiel das Dog Schema und seine AWS AppSync Funktionsanforderungs- und Antworthandler aus dem vorherigen Abschnitt. Speichern Sie den Code mithilfe der CLI auf Ihrer lokalen Station in einer Datei mit dem Namen `code.js` und speichern Sie das `context` Objekt anschließend in einer Datei mit dem Namen`context.json`. Führen Sie in Ihrer Shell den folgenden Befehl aus:

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

Die Antwort `evaluationResult` enthält eine, die die von Ihrem Handler zurückgegebene Nutzlast enthält. Es enthält auch ein `logs` Objekt, das die Liste der Protokolle enthält, die von Ihrem Handler während der Auswertung generiert wurden. Auf diese Weise können Sie Ihre Codeausführung einfach debuggen und Informationen zu Ihrer Evaluierung abrufen, um bei der Fehlerbehebung zu helfen. Zum Beispiel:

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

Das `evaluationResult` kann als JSON geparst werden, was Folgendes ergibt:

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

Mit dem SDK können Sie ganz einfach Tests aus Ihrer bevorzugten Testsuite integrieren, um das Verhalten Ihrer Handler zu überprüfen. Wir empfehlen, Tests mit dem Jest Testing Framework zu erstellen, aber jede Testsuite funktioniert. Das folgende Snippet zeigt einen hypothetischen

Validierungslauf. Beachten Sie, dass wir davon ausgehen, dass es sich bei der Bewertungsantwort um ein gültiges JSON handelt. Daher verwenden wir diese Methode, `JSON.parse` um JSON aus der Zeichenkettenantwort abzurufen:

```
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)
})
```

Dies führt zu dem folgenden Ergebnis:

```
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
```

Debuggen einer Live-Abfrage

Es gibt keinen Ersatz für einen end-to-end Test und eine Protokollierung, um eine Produktionsanwendung zu debuggen. AWS AppSync ermöglicht es Ihnen, Fehler und vollständige Anfragedetails über Amazon zu protokollieren CloudWatch. Darüber hinaus können Sie die AWS AppSync Konsole verwenden, um GraphQL-Abfragen, -Mutationen und -Abonnements zu testen und Protokolldaten für jede Anfrage live zurück in den Abfrage-Editor zu streamen, um sie in Echtzeit zu debuggen. Für Abonnements zeigen die Protokolle Verbindungszeitinformationen an.

Um dies durchzuführen, müssen Sie Amazon CloudWatch Logs im Voraus aktiviert haben, wie unter Überwachung und Protokollierung beschrieben. Wählen Sie als Nächstes in der AWS AppSync Konsole die Registerkarte Abfragen und geben Sie dann eine gültige GraphQL-Abfrage ein. Klicken Sie im unteren rechten Bereich auf das Protokollfenster und ziehen Sie es, um die Protokollansicht zu öffnen. Wählen Sie oben auf der Seite das Wiedergabe-Pfeilsymbol, um die GraphQL-Abfrage auszuführen. In wenigen Augenblicken werden Ihre vollständigen Anfrage- und Antwortprotokolle für den Vorgang in diesen Bereich gestreamt und Sie können sie in der Konsole einsehen.

Konfiguration und Verwendung von Pipeline-Resolvern in AWS AppSync () JavaScript

AWS AppSync führt Resolver auf einem GraphQL-Feld aus. In einigen Fällen müssen Anwendungen mehrere Operationen ausführen, um ein einziges GraphQL-Feld aufzulösen. Mit Pipeline-Resolvern können Entwickler jetzt Operationen, sogenannte Funktionen, zusammenstellen und sie nacheinander ausführen. Pipeline-Resolver sind nützlich für Anwendungen, die z. B. eine Autorisierungsprüfung ausführen müssen, bevor sie Daten für ein Feld abrufen.

Weitere Informationen zur Architektur eines JavaScript Pipeline-Resolvers finden Sie in der Übersicht über die JavaScriptResolver.

Schritt 1: Einen Pipeline-Resolver erstellen

Gehen Sie in der AWS AppSync Konsole zur Schemaseite.

Speichern Sie das folgende Schema:

```
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
}
```

Wir werden einen Pipeline-Resolver an das signUp-Feld mit dem Typ der Mutation anhängen. Wählen Sie im Bereich Mutationstyp auf der rechten Seite neben dem `signUp` Mutationsfeld die Option Anhängen aus. Stellen Sie den Resolver auf `pipeline resolver` und die `APPSYNC_JS` Laufzeit ein und erstellen Sie dann den Resolver.

Unser Pipeline-Resolver meldet einen Benutzer an, indem er zuerst die eingegebene E-Mail-Adresse validiert und den Benutzer dann im System speichert. Wir werden die E-Mail-Validierung in einer ValidateEmail-Funktion und das Speichern des Benutzers in einer SaveUser-Funktion kapseln. Die validateEmail-Funktion wird zuerst ausgeführt, und, sofern die E-Mail gültig ist, anschließend die saveUser-Funktion.

Der Ausführungsablauf wird wie folgt aussehen:

1. Anforderungshandler für Mutation.SignUp Resolver
2. Funktion validateEmail
3. Funktion saveUser
4. Antworthandler für den Mutation.SignUp-Resolver

Da wir die Funktion validateEmail wahrscheinlich in anderen Resolvern auf unserer API wiederverwenden werden, möchten wir den Zugriff vermeiden, `ctx.args` da sich diese von einem GraphQL-Feld zum anderen ändern. Wir können stattdessen `ctx.stash` verwenden, um das E-Mail-Attribut aus dem Eingabefeldargument `signUp(input: Signup)` zu speichern.

Aktualisieren Sie Ihren Resolver-Code, indem Sie Ihre Anfrage- und Antwortfunktionen ersetzen:

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

export function response(ctx) {
```

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

Wählen Sie Erstellen oder Speichern, um den Resolver zu aktualisieren.

Schritt 2: Eine Funktion erstellen

Klicken Sie auf der Pipeline-Resolver-Seite im Abschnitt Funktionen auf Funktion hinzufügen und dann auf Neue Funktion erstellen. Es ist auch möglich, Funktionen zu erstellen, ohne die Resolver-Seite aufrufen zu müssen. Rufen Sie dazu in der AWS AppSync Konsole die Seite Funktionen auf. Wählen Sie die Schaltfläche Create function (Funktion erstellen). Wir erstellen eine Funktion, die prüft, ob eine E-Mail gültig ist und von einer bestimmten Domäne stammt. Sofern die E-Mail nicht gültig ist, löst die Funktion einen Fehler aus. Andernfalls leitet sie sämtliche Eingaben weiter.

Stellen Sie sicher, dass Sie eine Datenquelle vom Typ NONE erstellt haben. Wählen Sie diese Datenquelle in der Liste Datenquellenname aus. Geben Sie für den Funktionsnamen in ein`validateEmail`. Überschreiben Sie im Funktionscode-Bereich alles mit diesem Snippet:

```
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;
}
```

Überprüfe deine Eingaben und wähle dann Erstellen. Wir haben unsere validateEmail-Funktion erstellt. Wiederholen Sie diese Schritte, um die SaveUser-Funktion mit dem folgenden Code zu erstellen (Der Einfachheit halber verwenden wir eine NONE-Datenquelle und tun so, als ob der Benutzer nach der Ausführung der Funktion im System gespeichert wurde.):

```
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;
}
```

Wir haben gerade unsere SaveUser-Funktion erstellt.

Schritt 3: Hinzufügen einer Funktion zu einem Pipeline-Resolver

Unsere Funktionen hätten automatisch zu dem gerade erstellten Pipeline-Resolver hinzugefügt werden sollen. Wenn dies nicht der Fall war oder Sie die Funktionen über die Funktionsseite erstellt haben, können Sie auf der **`signUp`** Resolver-Seite wieder auf Funktion hinzufügen klicken, um sie anzuhängen. Fügen Sie dem Resolver sowohl die Funktionen validateEmail als auch saveUser hinzu. Die Funktion validateEmail muss vor der Funktion saveUser platziert werden. Wenn Sie weitere Funktionen hinzufügen, können Sie die Optionen Nach oben und Nach unten verwenden, um die Reihenfolge der Ausführung Ihrer Funktionen neu zu organisieren. Überprüfen Sie Ihre Änderungen und wählen Sie dann Speichern.

Schritt 4: Eine Abfrage ausführen

Rufen Sie in der AWS AppSync Konsole die Seite Abfragen auf. Stellen Sie im Explorer sicher, dass Sie Ihre Mutation verwenden. Wenn nicht, wählen Sie `Mutation` in der Drop-down-Liste und dann+. Geben Sie die folgende Abfrage ein:

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

Dies sollte etwa Folgendes zurückgeben:

```
{
```

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

Wir haben mit einem Pipeline-Resolver unseren Benutzer erfolgreich angemeldet und die eingegebene E-Mail validiert.

Basisabfragen erstellen (VTL)

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

GraphQL-Resolver verbinden die Felder in einem Schema mit einer Datenquelle. Resolver sind der Mechanismus, mit dem Anfragen erfüllt werden. AWS AppSync kann Resolver automatisch aus einem Schema erstellen und verbinden oder ein Schema erstellen und Resolver aus einer vorhandenen Tabelle verbinden, ohne dass Sie Code schreiben müssen.

Resolver, die AWS AppSync verwendet werden JavaScript , um einen GraphQL-Ausdruck in ein Format zu konvertieren, das die Datenquelle verwenden kann. Alternativ können Mapping-Vorlagen in Apache Velocity Template Language (VTL) geschrieben werden, um einen GraphQL-Ausdruck in ein Format zu konvertieren, das die Datenquelle verwenden kann.

In diesem Abschnitt erfahren Sie, wie Sie Resolver mithilfe von VTL konfigurieren. Eine einführende Programmieranleitung im Stil eines Tutorials zum Schreiben von Resolvern finden Sie im Programmierhandbuch für Resolver-Mapping-Vorlagen. Hilfsprogramme, die Sie beim Programmieren verwenden können, finden Sie in der Kontextreferenz für Resolver-Mapping-Vorlagen. AWS AppSync verfügt außerdem über integrierte Test- und Debug-Workflows, die Sie verwenden können, wenn Sie von Grund auf neu bearbeiten oder Inhalte erstellen. Weitere Informationen finden Sie unter Resolver testen und debuggen.

Wir empfehlen, diese Anleitung zu befolgen, bevor Sie versuchen, eines der oben genannten Tutorials zu verwenden.

In diesem Abschnitt erfahren Sie, wie Sie einen Resolver erstellen, einen Resolver für Mutationen hinzufügen und erweiterte Konfigurationen verwenden.

Erstellen Sie Ihren ersten Resolver

In Anlehnung an die Beispiele aus den vorherigen Abschnitten besteht der erste Schritt darin, einen Resolver für Ihren `Query` Typ zu erstellen.

Console

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.
   b. Wählen Sie in der Seitenleiste Schema aus.
2. Auf der rechten Seite der Seite befindet sich ein Fenster namens Resolvers. Dieses Feld enthält eine Liste der Typen und Felder, wie sie in Ihrem Schemafenster auf der linken Seite der Seite definiert sind. Sie können Resolver an Felder anhängen. Wählen Sie beispielsweise unter dem Abfragetyp neben dem `getTodos` Feld die Option Anhängen aus.
3. Wählen Sie auf der Seite „Resolver erstellen“ die Datenquelle aus, die Sie im Handbuch Datenquellen anhängen erstellt haben. Im Fenster Zuordnungsvorlagen konfigurieren können Sie mithilfe der Dropdownliste auf der rechten Seite sowohl die generischen Vorlagen für die Anfrage als auch die Vorlage für die Antwortzuordnung auswählen oder Ihre eigenen erstellen.

> **Note**
>
> Die Kopplung einer Vorlage für die Zuordnung von Anfragen mit einer Vorlage für die Zuordnung von Antworten wird als Unit-Resolver bezeichnet. Unit-Resolver sind in der Regel für Routineoperationen vorgesehen. Wir empfehlen, sie nur für einzelne Operationen mit einer kleinen Anzahl von Datenquellen zu verwenden. Für komplexere Operationen empfehlen wir die Verwendung von Pipeline-Resolvern, die mehrere Operationen mit mehreren Datenquellen nacheinander ausführen können. Weitere Informationen zum Unterschied zwischen Vorlagen für die Zuordnung von Anfragen und Antworten finden Sie unter Unit-Resolver.
> Weitere Informationen zur Verwendung von Pipeline-Resolvern finden Sie unter Pipeline-Resolver.

4. Für allgemeine Anwendungsfälle verfügt die AWS AppSync Konsole über integrierte Vorlagen, mit denen Sie Elemente aus Datenquellen abrufen können (z. B. Abfragen aller Elemente, einzelne Suchvorgänge usw.). In der einfachen Version des Schemas aus Designing your schema, in der `getTodos` es keine Paginierung gab, sieht die Vorlage für die Anforderungszuweisung zum Auflisten von Elementen beispielsweise wie folgt aus:

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

5. Sie benötigen immer eine Vorlage für die Antwortzuweisung, die der Anfrage beiliegt. Die Konsole stellt eine Standardvorlage mit dem folgenden Pass-Through-Wert für Listen bereit:

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

   In diesem Beispiel hat das `context`-Objekt (mit dem Alias `$ctx`) für Listen mit Elementen die Form `$context.result.items`. Wenn die GraphQL-Operation ein einzelnes Element zurückgibt, würde es sich um `$context.result` handeln. AWS AppSync stellt Hilfsfunktionen für allgemeine Operationen bereit, z. B. die oben aufgelistete Funktion `$util.toJson`, mit denen die Antworten richtig formatiert werden können. Eine vollständige Liste der Funktionen finden Sie unter Referenz zum Resolver Mapping Template Utility.

6. Wählen Sie „Resolver speichern“.

API

1. Erstellen Sie ein Resolver-Objekt, indem Sie die `CreateResolver`API aufrufen.
2. Sie können die Felder Ihres Resolvers ändern, indem Sie die `UpdateResolver`API aufrufen.

CLI

1. Erstellen Sie einen Resolver, indem Sie den `create-resolver`Befehl ausführen.

   Für diesen speziellen Befehl müssen Sie 6 Parameter eingeben:

   1. Die `api-id` Ihrer API.

2. Der `type-name` Typ, den Sie in Ihrem Schema ändern möchten. Im Konsolenbeispiel war das`Query`.
3. Das `field-name` Feld, das Sie in Ihrem Typ ändern möchten. Im Konsolenbeispiel war das`getTodos`.
4. Die `data-source-name` der Datenquelle, die Sie im Handbuch Datenquellen anhängen erstellt haben.
5. Der`request-mapping-template`, welcher der Hauptteil der Anfrage ist. Im Konsolenbeispiel lautete das:

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

6. Das`response-mapping-template`, was der Hauptteil der Antwort ist. Im Konsolenbeispiel lautete das:

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

Ein Beispielbefehl könnte so aussehen:

```
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)"
```

Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

```
{
    "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. Führen Sie den `update-resolver`Befehl aus, um die Vorlagen für die and/or Feldzuweisung eines Resolvers zu ändern.

   Mit Ausnahme des `api-id` Parameters werden die im `create-resolver` Befehl verwendeten Parameter durch die neuen Werte aus dem `update-resolver` Befehl überschrieben.

Einen Resolver für Mutationen hinzufügen

Der nächste Schritt besteht darin, einen Resolver für Ihren `Mutation` Typ zu erstellen.

Console

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.

   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.

   b. Wählen Sie in der Seitenleiste Schema aus.

2. Wählen Sie unter dem Mutationstyp neben dem `addTodo` Feld die Option Anhängen aus.
3. Wählen Sie auf der Seite „Resolver erstellen“ die Datenquelle aus, die Sie im Handbuch Datenquellen anhängen erstellt haben.
4. Im Fenster Zuordnungsvorlagen konfigurieren müssen Sie die Anforderungsvorlage ändern, da es sich um eine Mutation handelt, bei der Sie DynamoDB ein neues Element hinzufügen. Verwenden Sie die folgende Abfrage-Zuweisungsvorlage:

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

5. AWS AppSync konvertiert automatisch die im `addTodo` Feld definierten Argumente aus Ihrem GraphQL-Schema in DynamoDB-Operationen. Im vorherigen Beispiel werden Datensätze in DynamoDB mit dem Schlüssel von gespeichert`id`, der vom

Mutationsargument as übergeben wird. `$ctx.args.id` Alle anderen Felder, die Sie durchlaufen, werden automatisch DynamoDB-Attributen mit zugeordnet. `$util.dynamodb.toMapValuesJson($ctx.args)`

Verwenden Sie für diesen Resolver die folgende Antwortzuweisungsvorlage:

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

AWS AppSync unterstützt auch Test- und Debug-Workflows für die Bearbeitung von Resolvern. Sie können ein `context`-Mock-Objekt verwenden, um den transformierten Wert der Vorlage vor dem Aufrufen anzuzeigen. Optional können Sie die vollständige Anforderungsausführung für eine Datenquelle interaktiv anzeigen, während Sie eine Abfrage ausführen. Weitere Informationen finden Sie unter Resolver testen und debuggen und Überwachen und Protokollieren.

6. Wählen Sie „Resolver speichern“.

API

Sie können dies auch tun, APIs indem Sie die Befehle im Abschnitt Erstellen Sie Ihren ersten Resolver und die Parameterdetails aus diesem Abschnitt verwenden.

CLI

Sie können dies auch in der CLI tun, indem Sie die Befehle im Abschnitt Create your first resolver und die Parameterdetails aus diesem Abschnitt verwenden.

Wenn Sie die erweiterten Resolver nicht verwenden, können Sie zu diesem Zeitpunkt mit der Verwendung Ihrer GraphQL-API beginnen, wie unter Verwenden Ihrer API beschrieben.

Fortgeschrittene Resolver

Wenn Sie dem Abschnitt „Erweitert“ folgen und unter Entwerfen Ihres Schemas für einen paginierten Scan ein Beispielschema erstellen, verwenden Sie stattdessen die folgende Anforderungsvorlage für das `getTodos` Feld:

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

```
    "nextToken": $util.toJson($util.defaultIfNullOrBlank($ctx.args.nextToken, null))
}
```

In diesem Paginierungsanwendungsfall ist die Antwortzuweisung mehr als nur ein Pass-Through, da sie sowohl den Cursor (damit der Kunde weiß, mit welcher Seite das nächste Mal begonnen wird) als auch den Ergebnissatz enthält. Die Zuweisungsvorlage sieht folgendermaßen aus:

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

Die Felder in der vorherigen Antwortzuweisungsvorlage sollten mit den in Ihrem `TodoConnection`-Typ definierten Feldern übereinstimmen.

Für Beziehungen, bei denen Sie eine `Comments` Tabelle haben und das Kommentarfeld für den Typ auflösen (was den `Todo` Typ von zurückgibt`[Comment]`), können Sie eine Zuordnungsvorlage verwenden, die eine Abfrage für die zweite Tabelle ausführt. Dazu müssen Sie bereits eine Datenquelle für die `Comments` Tabelle erstellt haben, wie unter Hinzufügen einer Datenquelle beschrieben.

> **Note**
>
> Wir verwenden einen Abfragevorgang für eine zweite Tabelle nur zur Veranschaulichung. Sie könnten stattdessen eine andere Operation gegen DynamoDB verwenden. Darüber hinaus könnten Sie die Daten aus einer anderen Datenquelle abrufen, z. B. AWS Lambda aus Amazon OpenSearch Service, da die Beziehung durch Ihr GraphQL-Schema gesteuert wird.

Console

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.
   b. Wählen Sie in der Seitenleiste Schema aus.
2. Wählen Sie unter dem Typ Todo neben dem Feld die `comments` Option Anhängen aus.
3. Wählen Sie auf der Seite „Resolver erstellen“ die Datenquelle für die Tabelle „Kommentare“ aus. Der Standardname für die Tabelle Kommentare aus den Schnellstartanleitungen

lautetAppSyncCommentTable, er kann jedoch variieren, je nachdem, welchen Namen Sie der Tabelle gegeben haben.

4. Fügen Sie Ihrer Vorlage für die Anforderungszuweisung den folgenden Ausschnitt hinzu:

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

5. `context.source` verweist auf das übergeordnete Objekt des aktuellen Felds, das aufgelöst wird. In diesem Beispiel verweist `source.id` auf das Todo-Objekt, das dann für den Abfrageausdruck verwendet wird.

   Sie können die Pass-Through-Antwortzuweisungsvorlage folgendermaßen verwenden:

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

6. Wählen Sie „Resolver speichern“.

7. Gehen Sie abschließend zurück zur Schemaseite in der Konsole, fügen Sie dem `addComment` Feld einen Resolver hinzu und geben Sie die Datenquelle für die `Comments` Tabelle an. Die Anforderungszuweisungsvorlage ist in diesem Fall eine einfache `PutItem`-Anweisung mit der spezifischen `todoid`, die als Argument kommentiert ist, aber Sie verwenden das Dienstprogramm `$utils.autoId()` zum Erstellen eines eindeutigen Sortierschlüssels für den Kommentar:

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

```
    },
    "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
}
```

Verwenden Sie folgendermaßen eine Pass-Through-Antwortvorlage:

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

API

Sie können dies auch tun, APIs indem Sie die Befehle im Abschnitt Erstellen Sie Ihren ersten Resolver und die Parameterdetails in diesem Abschnitt verwenden.

CLI

Sie können dies auch in der CLI tun, indem Sie die Befehle im Abschnitt Create your first resolver und die Parameterdetails aus diesem Abschnitt verwenden.

Deaktivieren von VTL-Mapping-Vorlagen mit direkten Lambda-Resolvern (VTL)

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Mit direkten Lambda-Resolvern können Sie die Verwendung von VTL-Mapping-Vorlagen bei der Verwendung von Datenquellen umgehen. AWS Lambda AWS AppSync kann eine Standardnutzlast für Ihre Lambda-Funktion sowie eine Standardübersetzung aus der Antwort einer Lambda-Funktion auf einen GraphQL-Typ bereitstellen. Sie können wählen, ob Sie eine Anforderungsvorlage, eine Antwortvorlage oder beides bereitstellen möchten, und AWS AppSync werden diese entsprechend behandeln.

Weitere Informationen über die standardmäßige Anforderungsnutzlast und die damit AWS AppSync verbundene Antwortübersetzung finden Sie in der Direct Lambda Resolver-Referenz. Weitere Informationen zum Einrichten einer AWS Lambda Datenquelle und zum Einrichten einer IAM-Vertrauensrichtlinie finden Sie unter Eine Datenquelle anhängen.

Direkte Lambda-Resolver konfigurieren

In den folgenden Abschnitten erfahren Sie, wie Sie Lambda-Datenquellen anhängen und Lambda-Resolver zu Ihren Feldern hinzufügen.

Fügen Sie eine Lambda-Datenquelle hinzu

Bevor Sie direkte Lambda-Resolver aktivieren können, müssen Sie eine Lambda-Datenquelle hinzufügen.

Console

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Konsole. AppSync
   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.
   b. Wählen Sie in der Seitenleiste Datenquellen aus.
2. Klicken Sie auf Create data source.
   a. Geben Sie unter Datenquellenname einen Namen für Ihre Datenquelle ein, z. B. **`myFunction`**
   b. Wählen Sie als Datenquellentyp die Option AWS Lambda Funktion aus.
   c. Wählen Sie für Region die entsprechende Region aus.
   d. Wählen Sie für Function ARN die Lambda-Funktion aus der Dropdownliste aus. Sie können nach dem Funktionsnamen suchen oder den ARN der Funktion, die Sie verwenden möchten, manuell eingeben.
   e. Erstellen Sie eine neue IAM-Rolle (empfohlen) oder wählen Sie eine vorhandene Rolle aus, die über die `lambda:invokeFunction` IAM-Berechtigung verfügt. Für bestehende Rollen ist eine Vertrauensrichtlinie erforderlich, wie im Abschnitt Eine Datenquelle anhängen beschrieben.

      Im Folgenden finden Sie ein Beispiel für eine IAM-Richtlinie, die über die erforderlichen Berechtigungen für die Ausführung von Vorgängen auf der Ressource verfügt:

      JSON

```
{
     "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. Wählen Sie die Schaltfläche Erstellen aus.

CLI

1. Erstellen Sie ein Datenquellenobjekt, indem Sie den `create-data-source`Befehl ausführen.

   Für diesen speziellen Befehl müssen Sie 4 Parameter eingeben:

   1. Die `api-id` Ihrer API.
   2. Die `name` Ihrer Datenquelle. Im Konsolenbeispiel ist dies der Name der Datenquelle.
   3. Der `type` Wert der Datenquelle. Im Konsolenbeispiel ist dies eine AWS Lambda Funktion.
   4. Das`lambda-config`, was die Funktion ARN im Konsolenbeispiel ist.

   > **Note**
   >
   > Es gibt andere Parameter wie diesen`Region`, die konfiguriert werden müssen, aber normalerweise werden sie standardmäßig auf Ihre CLI-Konfigurationswerte gesetzt.

   Ein Beispielbefehl könnte wie folgt aussehen:

```
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
```

Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

```
{
    "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. Führen Sie den `update-data-source`Befehl aus, um die Attribute einer Datenquelle zu ändern.

   Mit Ausnahme des `api-id` Parameters werden die im `create-data-source` Befehl verwendeten Parameter durch die neuen Werte aus dem `update-data-source` Befehl überschrieben.

Direkte Lambda-Resolver aktivieren

Nachdem Sie eine Lambda-Datenquelle erstellt und die entsprechende IAM-Rolle eingerichtet haben, um den Aufruf der Funktion AWS AppSync zu ermöglichen, können Sie sie mit einer Resolver- oder Pipeline-Funktion verknüpfen.

Console

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Konsole. AppSync

   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.

   b. Wählen Sie in der Seitenleiste Schema aus.

2. Wählen Sie im Resolver-Fenster ein Feld oder eine Operation aus und klicken Sie dann auf die Schaltfläche „Anhängen“.

3. Wählen Sie auf der Seite Neuen Resolver erstellen die Lambda-Funktion aus der Dropdownliste aus.

4. Um direkte Lambda-Resolver zu nutzen, stellen Sie sicher, dass Vorlagen für die Anfrage- und Antwortzuordnung im Abschnitt Zuordnungsvorlagen konfigurieren deaktiviert sind.
5. Wählen Sie die Schaltfläche „Resolver speichern“.

CLI

- Erstellen Sie einen Resolver, indem Sie den `create-resolver`Befehl ausführen.

  Für diesen speziellen Befehl müssen Sie 6 Parameter eingeben:

  1. Die `api-id` Ihrer API.
  2. Der `type-name` Typ in Ihrem Schema.
  3. Der `field-name` des Felds in Ihrem Schema.
  4. Der `data-source-name` oder der Name Ihrer Lambda-Funktion.
  5. Der`request-mapping-template`, welcher der Hauptteil der Anfrage ist. Im Konsolenbeispiel wurde dies deaktiviert:

     ```
     " "
     ```

  6. Das`response-mapping-template`, was der Hauptteil der Antwort ist. Im Konsolenbeispiel wurde dies ebenfalls deaktiviert:

     ```
     " "
     ```

  Ein Beispielbefehl könnte so aussehen:

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

  Eine Ausgabe wird in der CLI zurückgegeben. Ein Beispiel:

  ```
  {
      "resolver": {
          "resolverArn": "arn:aws:appsync:us-west-2:102847592837:apis/
  abcdefghijklmnopqrstuvwxyz/types/Subscription/resolvers/onCreateTodo",
          "typeName": "Subscription",
  ```

```
        "kind": "UNIT",
        "fieldName": "onCreateTodo",
        "dataSourceName": "LambdaTest"
    }
}
```

Wenn Sie Ihre Mapping-Vorlagen deaktivieren, gibt es mehrere zusätzliche Verhaltensweisen, die auftreten werden in AWS AppSync:

- Durch das Deaktivieren einer Zuordnungsvorlage signalisieren Sie, AWS AppSync dass Sie die in der Direct Lambda-Resolver-Referenz angegebenen Standarddatenübersetzungen akzeptieren.
- Wenn Sie die Vorlage für die Anforderungszuweisung deaktivieren, erhält Ihre Lambda-Datenquelle eine Nutzlast, die aus dem gesamten Context-Objekt besteht.
- Wenn Sie die Antwortzuordnungsvorlage deaktivieren, wird das Ergebnis Ihres Lambda-Aufrufs je nach Version der Anforderungszuordnungsvorlage übersetzt oder ob die Anforderungszuordnungsvorlage ebenfalls deaktiviert ist.

Resolver in (VTL) testen und debuggen AWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync führt Resolver in einem GraphQL-Feld gegen eine Datenquelle aus. Wie in der Übersicht über Resolver-Mapping-Vorlagen beschrieben, kommunizieren Resolver mithilfe einer Vorlagensprache mit Datenquellen. Auf diese Weise können Sie das Verhalten anpassen und Logik und Bedingungen vor und nach der Kommunikation mit der Datenquelle anwenden. Eine einführende Programmieranleitung im Stil eines Tutorials zum Schreiben von Resolvern finden Sie im Programmierleitfaden für Resolver-Mapping-Vorlagen.

Um Entwicklern beim Schreiben, Testen und Debuggen dieser Resolver zu helfen, bietet die AWS AppSync Konsole auch Tools zum Erstellen einer GraphQL-Anfrage und -Antwort mit Scheindaten bis hin zum einzelnen Field-Resolver. Darüber hinaus können Sie Abfragen, Mutationen und Abonnements in der AWS AppSync Konsole durchführen und einen detaillierten Protokollstream der

gesamten Anfrage CloudWatch von Amazon einsehen. Dieser umfasst die Ergebnisse aus einer Datenquelle.

Testen mit Scheindaten

Wenn ein GraphQL-Resolver aufgerufen wird, enthält er ein `context` Objekt, das Informationen über die Anfrage enthält. Dazu gehören Argumente von einem Client, Identitätsinformationen sowie Daten aus dem übergeordneten GraphQL-Feld. Es enthält auch die Ergebnisse aus der Datenquelle, die in der Antwortvorlage verwendet werden können. Weitere Informationen über diese Struktur und die verfügbaren Helferobjekt-Dienstprogramme finden Sie in der Referenz zur Resolver-Zuweisungsvorlage "Context".

Beim Schreiben oder Bearbeiten eines Resolvers können Sie ein Modell- oder Testkontextobjekt an den Konsoleneditor übergeben. Auf diese Weise können Sie sehen, wie sowohl die Anforderungs- als auch die Antwortvorlage ausgewertet werden, ohne tatsächlich auf eine Datenquelle zuzugreifen. Beispiel: Sie können ein `firstname: Shaggy`-Testargument übergeben und sehen, wie es bei der Verwendung von `$ctx.args.firstname` in Ihrem Vorlagencode ausgewertet wird. Sie können auch die Auswertung eines Dienstprogramm-Helferobjekts wie `$util.autoId()` oder `util.time.nowISO8601()` testen.

Resolver testen

In diesem Beispiel werden Resolver mithilfe der AWS AppSync Konsole getestet.

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.
   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.
   b. Wählen Sie in der Seitenleiste Schema aus.
2. Falls Sie dies noch nicht getan haben, wählen Sie unter dem Typ und neben dem Feld die Option Anhängen aus, um Ihren Resolver hinzuzufügen.

   Weitere Informationen zum Erstellen eines vollständigen Resolvers finden Sie unter Resolver konfigurieren.

   Wählen Sie andernfalls den Resolver aus, der sich bereits im Feld befindet.
3. Wählen Sie oben auf der Seite Resolver bearbeiten die Option Testkontext auswählen und dann Neuen Kontext erstellen aus.
4. Wählen Sie ein Beispielkontextobjekt aus oder füllen Sie die JSON-Datei manuell im Fenster mit dem Ausführungskontext unten aus.

5. Geben Sie einen Namen für den Textkontext ein.
6. Wählen Sie die Schaltfläche Save (Speichern) aus.
7. Wählen Sie oben auf der Seite „Resolver bearbeiten“ die Option Test ausführen aus.

Ein praktischeres Beispiel: Angenommen, Sie haben eine App, die einen GraphQL-Typ speichert`Dog`, die automatische ID-Generierung für Objekte verwendet und diese in Amazon DynamoDB speichert. Sie können außerdem einige Werte aus den Argumenten einer GraphQL-Mutation schreiben und nur bestimmten Benutzern das Anzeigen einer Antwort erlauben. Das Schema kann beispielsweise wie folgt aussehen:

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

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

Wenn Sie einen Resolver für die `addDog` Mutation hinzufügen, können Sie ein Kontextobjekt wie im folgenden Beispiel auffüllen. Nachstehend werden die Argumente `name` und `age` des Clients verwendet und ein `username` im `identity`-Objekt mit Daten gefüllt:

```
{
    "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"
```

```
    }
}
```

Sie können dies mithilfe der folgenden Anforderungs- und Antwortzuweisungsvorlagen testen:

Request Template

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

Antwortvorlage

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

Die ausgewertete Vorlage enthält die Daten aus Ihrem Testkontextobjekt und den generierten Wert aus `$util.autoId()`. Wenn Sie `username` zu einem anderen Wert als `Nadia` ändern, werden die Ergebnisse nicht zurückgegeben, da die Autorisierungsprüfung fehlschlägt. Weitere Informationen zur detaillierten Zugriffskontrolle finden Sie unter Anwendungsfälle für die Autorisierung.

Testen von Mapping-Vorlagen mit AWS AppSync APIs

Sie können den `EvaluateMappingTemplate` API-Befehl verwenden, um Ihre Mapping-Vorlagen mit simulierten Daten aus der Ferne zu testen. Um mit dem Befehl zu beginnen, stellen Sie sicher, dass Sie die `appsync:evaluateMappingTemplate` entsprechende Berechtigung zu Ihrer Richtlinie hinzugefügt haben. Zum Beispiel:

JSON

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

```
        {
            "Effect": "Allow",
            "Action": "appsync:evaluateMappingTemplate",
            "Resource": "arn:aws:appsync:<region>:<account>:*"
        }
    ]
}
```

Sie können den Befehl nutzen, indem Sie das AWS CLIoder verwenden AWS SDKs. Nehmen Sie zum Beispiel das Dog Schema und die zugehörigen request/response Mapping-Vorlagen aus dem vorherigen Abschnitt. Speichern Sie die Anforderungsvorlage mithilfe der CLI auf Ihrer lokalen Station in einer Datei mit dem Namen `request.vtl` und speichern Sie das `context` Objekt anschließend in einer Datei mit dem Namen`context.json`. Führen Sie in Ihrer Shell den folgenden Befehl aus:

```
aws appsync evaluate-mapping-template --template file://request.vtl --context file://
context.json
```

Der Befehl gibt die folgende Antwort zurück:

```
{
  "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"
}
```

Das `evaluationResult` enthält die Ergebnisse des Testens Ihrer bereitgestellten Vorlage mit der bereitgestelltencontext. Sie können Ihre Vorlagen auch mit dem testen AWS SDKs. Hier ist ein Beispiel für die Verwendung des AWS SDK für 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))
```

Mit dem SDK können Sie ganz einfach Tests aus Ihrer bevorzugten Testsuite integrieren, um das Verhalten Ihrer Vorlage zu überprüfen. Wir empfehlen, Tests mit dem Jest Testing Framework zu erstellen, aber jede Testsuite funktioniert. Der folgende Ausschnitt zeigt einen hypothetischen Validierungslauf. Beachten Sie, dass wir davon ausgehen, dass es sich bei der Bewertungsantwort um ein gültiges JSON handelt. Daher verwenden wir diese Methode, `JSON.parse` um JSON aus der Zeichenkettenantwort abzurufen:

```
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)
})
```

Dies führt zu dem folgenden Ergebnis:

```
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
```

Debuggen einer Live-Abfrage

Es gibt keinen Ersatz für einen end-to-end Test und eine Protokollierung, um eine Produktionsanwendung zu debuggen. AWS AppSync ermöglicht es Ihnen, Fehler und vollständige

Anfragedetails über Amazon zu protokollieren CloudWatch. Darüber hinaus können Sie die AWS AppSync Konsole verwenden, um GraphQL-Abfragen, -Mutationen und -Abonnements zu testen und Protokolldaten für jede Anfrage live zurück in den Abfrage-Editor zu streamen, um sie in Echtzeit zu debuggen. Für Abonnements zeigen die Protokolle Verbindungszeitinformationen an.

Um dies durchzuführen, müssen Sie Amazon CloudWatch Logs im Voraus aktiviert haben, wie unter Überwachung und Protokollierung beschrieben. Wählen Sie als Nächstes in der AWS AppSync Konsole die Registerkarte Abfragen und geben Sie dann eine gültige GraphQL-Abfrage ein. Klicken Sie im unteren rechten Bereich auf das Protokollfenster und ziehen Sie es, um die Protokollansicht zu öffnen. Wählen Sie oben auf der Seite das Wiedergabe-Pfeilsymbol, um die GraphQL-Abfrage auszuführen. In wenigen Augenblicken werden die vollständigen Anforderungs- und Antwortprotokolle für die Operation an diesen Abschnitt der Konsole gestreamt und können angezeigt werden.

Konfiguration und Verwendung von Pipeline-Resolvern in AWS AppSync (VTL)

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync führt Resolver auf einem GraphQL-Feld aus. In einigen Fällen müssen Anwendungen mehrere Operationen ausführen, um ein einziges GraphQL-Feld aufzulösen. Mit Pipeline-Resolvern können Entwickler jetzt Operationen, sogenannte Funktionen, zusammenstellen und sie nacheinander ausführen. Pipeline-Resolver sind nützlich für Anwendungen, die z. B. eine Autorisierungsprüfung ausführen müssen, bevor sie Daten für ein Feld abrufen.

Ein Pipeline-Resolver besteht aus einer Zuweisungsvorlage für Vorher, einer Zuweisungsvorlage für Nachher und einer Liste von Funktionen. Jede Funktion verfügt über eine Vorlage für die Zuordnung von Anfragen und Antworten, die sie anhand einer Datenquelle ausführt. Da ein Pipeline-Resolver die Ausführung an eine Liste von Funktionen delegiert, ist er mit keiner Datenquelle verknüpft. Unit-Resolver und Funktionen sind Primitive, die Operationen an Datenquellen ausführen. Weitere Informationen finden Sie in der Übersicht über die Resolver-Mapping-Vorlagen.

Schritt 1: Einen Pipeline-Resolver erstellen

Rufen Sie in der AWS AppSync Konsole die Schemaseite auf.

Speichern Sie das folgende Schema:

```
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
}
```

Wir werden einen Pipeline-Resolver an das signUp-Feld mit dem Typ der Mutation anhängen. Wählen Sie im Bereich Mutationstyp auf der rechten Seite neben dem `signUp` Mutationsfeld die Option Anhängen aus. Klicken Sie auf der Seite „Resolver erstellen“ auf Aktionen und dann auf Laufzeit aktualisieren. Wählen Sie`Pipeline Resolver`, wählen Sie dann `VTL` und wählen Sie dann Aktualisieren. Die Seite sollte nun drei Abschnitte enthalten: einen Textbereich Vor dem Zuordnen der Vorlage, einen Abschnitt mit Funktionen und einen Textbereich Nach dem Zuordnen der Vorlage.

Unser Pipeline-Resolver meldet einen Benutzer an, indem er zuerst die eingegebene E-Mail-Adresse validiert und den Benutzer dann im System speichert. Wir kapseln die E-Mail-Validierung in einer validateEmail-Funktion und das Speichern des Benutzers in einer saveUser-Funktion. Die validateEmail-Funktion wird zuerst ausgeführt, und, sofern die E-Mail gültig ist, anschließend die saveUser-Funktion.

Der Ausführungsablauf sieht wie folgt aus:

1. Resolver-Zuweisungsvorlage für Anforderungen Mutation.signUp
2. Funktion validateEmail

3. Funktion saveUser
4. Resolver-Zuweisungsvorlage für Antworten Mutation.signUp

Da wir die Funktion validateEmail wahrscheinlich in anderen Resolvern auf unserer API wiederverwenden werden, möchten wir den Zugriff vermeiden, `$ctx.args` da sich diese von einem GraphQL-Feld zum anderen ändern. Wir können stattdessen `$ctx.stash` verwenden, um das E-Mail-Attribut aus dem Eingabefeldargument `signUp(input: Signup)` zu speichern.

VOR der Zuordnungsvorlage:

```
## store email input field into a generic email key
$util.qr($ctx.stash.put("email", $ctx.args.input.email))
{}
```

Die Konsole stellt eine standardmäßige Passthrough-AFTER-Zuordnungsvorlage bereit, die wir verwenden werden:

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

Wählen Sie Erstellen oder Speichern, um den Resolver zu aktualisieren.

Schritt 2: Eine Funktion erstellen

Klicken Sie auf der Pipeline-Resolver-Seite im Abschnitt Funktionen auf Funktion hinzufügen und dann auf Neue Funktion erstellen. Es ist auch möglich, Funktionen zu erstellen, ohne die Resolver-Seite aufrufen zu müssen. Rufen Sie dazu in der AWS AppSync Konsole die Seite Funktionen auf. Wählen Sie die Schaltfläche Create function (Funktion erstellen). Wir erstellen eine Funktion, die prüft, ob eine E-Mail gültig ist und von einer bestimmten Domäne stammt. Sofern die E-Mail nicht gültig ist, löst die Funktion einen Fehler aus. Andernfalls leitet sie sämtliche Eingaben weiter.

Wählen Sie auf der Seite „Neue Funktionen“ die Option „Aktionen“ und dann „Laufzeit aktualisieren“. Wählen Sie „VTLAktualisieren“. Stellen Sie sicher, dass Sie eine Datenquelle vom Typ NONE erstellt haben. Wählen Sie diese Datenquelle in der Liste Datenquellenname aus. Geben Sie für den Funktionsnamen in ein`validateEmail`. Überschreiben Sie im Funktionscode-Bereich alles mit diesem Snippet:

```
#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}) }
}
```

Fügen Sie dies in die Antwortzuordnungsvorlage ein:

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

Überprüfen Sie Ihre Änderungen und wählen Sie dann Erstellen aus. Wir haben unsere validateEmail-Funktion erstellt. Wiederholen Sie diese Schritte, um die SaveUser-Funktion mit den folgenden Anforderungs- und Antwortzuordnungsvorlagen zu erstellen (Der Einfachheit halber verwenden wir eine NONE-Datenquelle und tun so, als ob der Benutzer nach der Ausführung der Funktion im System gespeichert wurde.):

Zuweisungsvorlage für Anforderungen:

```
## $ctx.prev.result contains the signup input values. We could have also
## used $ctx.args.input.
{
    "payload": $util.toJson($ctx.prev.result)
}
```

Zuweisungsvorlage für Antworten:

```
## 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)
```

Wir haben gerade unsere SaveUser-Funktion erstellt.

Schritt 3: Hinzufügen einer Funktion zu einem Pipeline-Resolver

Unsere Funktionen hätten automatisch zu dem Pipeline-Resolver hinzugefügt werden sollen, den wir gerade erstellt haben. Wenn dies nicht der Fall war oder Sie die Funktionen über die Seite Funktionen erstellt haben, können Sie auf der Resolver-Seite auf Funktion hinzufügen klicken, um sie anzuhängen. Fügen Sie dem Resolver sowohl die Funktionen validateEmail als auch saveUser hinzu. Die Funktion validateEmail muss vor der Funktion saveUser platziert werden. Wenn Sie weitere

Funktionen hinzufügen, können Sie die Optionen „Nach oben“ und „Nach unten“ verwenden, um die Reihenfolge der Ausführung Ihrer Funktionen neu zu organisieren. Überprüfen Sie Ihre Änderungen und wählen Sie dann Speichern.

Schritt 4: Eine Abfrage ausführen

Rufen Sie in der AWS AppSync Konsole die Seite Abfragen auf. Stellen Sie im Explorer sicher, dass Sie Ihre Mutation verwenden. Wenn nicht, wählen Sie `Mutation` in der Drop-down-Liste und dann+. Geben Sie die folgende Abfrage ein:

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

Dies sollte etwa Folgendes zurückgeben:

```
{
  "data": {
    "signUp": {
      "id": "256b6cc2-4694-46f4-a55e-8cb14cc5d7fc",
      "email": "nadia@myvaliddomain.com"
    }
  }
}
```

Wir haben mit einem Pipeline-Resolver unseren Benutzer erfolgreich angemeldet und die eingegebene E-Mail validiert. Ein detailliertes Tutorial zu Pipeline-Resolvern finden Sie unter Tutorial: Pipeline-Resolver

Eine AWS AppSync API mit dem AWS CDK verwenden

> **Tip**
>
> Bevor Sie das CDK verwenden, empfehlen wir, die offizielle Dokumentation des CDK zusammen mit der CDK-Referenz zu AWS AppSync lesen.

Wir empfehlen außerdem sicherzustellen, dass Ihre AWS CLI - und NPM-Installationen auf Ihrem System funktionieren.

In diesem Abschnitt werden wir eine einfache CDK-Anwendung erstellen, die Elemente aus einer DynamoDB-Tabelle hinzufügen und aus ihr abrufen kann. Dies ist als Schnellstartbeispiel gedacht, bei dem ein Teil des Codes aus den Abschnitten Schema entwerfen, Datenquelle anhängen und Resolvers konfigurieren () verwendet wird. JavaScript

Ein CDK-Projekt einrichten

Warning

Diese Schritte sind je nach Umgebung möglicherweise nicht ganz korrekt. Wir gehen davon aus, dass auf Ihrem System die erforderlichen Dienstprogramme installiert sind, dass eine Verbindung zu AWS Diensten hergestellt werden kann und dass die richtigen Konfigurationen vorhanden sind.

Der erste Schritt ist die Installation des AWS CDK. In Ihrer CLI können Sie den folgenden Befehl eingeben:

```
npm install -g aws-cdk
```

Als Nächstes müssen Sie ein Projektverzeichnis erstellen und dann dorthin navigieren. Ein Beispiel für einen Befehlssatz zum Erstellen und Navigieren zu einem Verzeichnis ist:

```
mkdir example-cdk-app
cd example-cdk-app
```

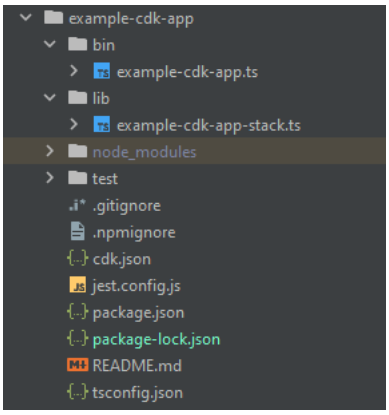
Als Nächstes müssen Sie eine App erstellen. Unser Service verwendet hauptsächlich TypeScript. Geben Sie in Ihrem Projektverzeichnis den folgenden Befehl ein:

```
cdk init app --language typescript
```

Wenn Sie dies tun, wird eine CDK-App zusammen mit ihren Initialisierungsdateien installiert:

```
Initializing a new git repository...
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Executing npm install...
✅All done!
```

Ihre Projektstruktur könnte so aussehen:



Sie werden feststellen, dass wir mehrere wichtige Verzeichnisse haben:

- `bin`: Die erste BIN-Datei erstellt die App. Wir werden das in diesem Handbuch nicht behandeln.
- `lib`: Das Verzeichnis lib enthält Ihre Stack-Dateien. Sie können sich Stack-Dateien als einzelne Ausführungseinheiten vorstellen. Konstrukte werden in unseren Stack-Dateien enthalten sein. Im Grunde sind dies Ressourcen für einen Dienst, der AWS CloudFormation bei der Bereitstellung der App aktiviert wird. Hier wird der Großteil unserer Codierung stattfinden.
- `node_modules`: Dieses Verzeichnis wurde von NPM erstellt und enthält alle Paketabhängigkeiten, die Sie mit dem `npm` Befehl installiert haben.

Unsere anfängliche Stack-Datei könnte etwa Folgendes enthalten:

```
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)
    // });
  }
}
```

Dies ist der Boilerplate-Code zum Erstellen eines Stacks in unserer App. Der größte Teil unseres Codes in diesem Beispiel wird in den Geltungsbereich dieser Klasse fallen.

Um zu überprüfen, ob sich Ihre Stack-Datei in der App im Verzeichnis Ihrer App befindet, führen Sie den folgenden Befehl im Terminal aus:

```
cdk ls
```

Eine Liste deiner Stacks sollte erscheinen. Ist dies nicht der Fall, musst du die Schritte möglicherweise erneut ausführen oder in der offiziellen Dokumentation nach Hilfe suchen.

Wenn Sie Ihre Codeänderungen vor der Bereitstellung erstellen möchten, können Sie jederzeit den folgenden Befehl im Terminal ausführen:

```
npm run build
```

Und um die Änderungen vor der Bereitstellung zu sehen:

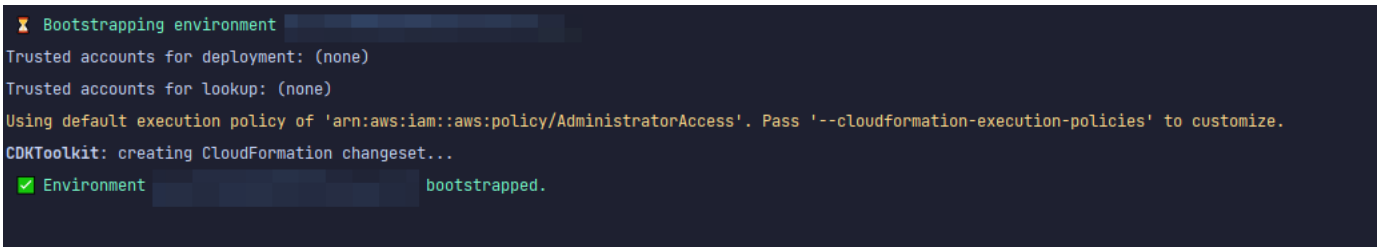
```
cdk diff
```

Bevor wir unseren Code zur Stack-Datei hinzufügen, führen wir einen Bootstrap durch. Bootstrapping ermöglicht es uns, Ressourcen für das CDK bereitzustellen, bevor die App bereitgestellt wird. Weitere Informationen zu diesem Prozess finden Sie hier. Um einen Bootstrap zu erstellen, lautet der Befehl:

```
cdk bootstrap aws://ACCOUNT-NUMBER/REGION
```

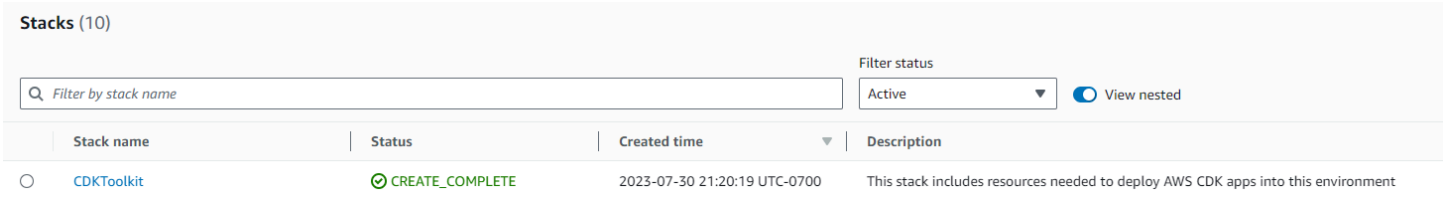
> **Tip**
>
> Für diesen Schritt sind mehrere IAM-Berechtigungen in Ihrem Konto erforderlich. Ihr Bootstrap wird verweigert, wenn Sie sie nicht haben. In diesem Fall müssen Sie möglicherweise unvollständige Ressourcen löschen, die durch den Bootstrap verursacht wurden, z. B. den von ihm generierten S3-Bucket.

Bootstrap wird mehrere Ressourcen bereitstellen. Die endgültige Nachricht wird so aussehen:

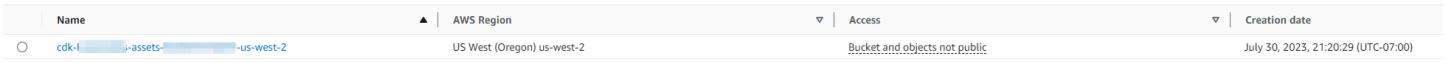


Dies erfolgt einmal pro Konto pro Region, sodass Sie dies nicht oft tun müssen. Die Hauptressourcen des Bootstrap sind der AWS CloudFormation Stack und der Amazon S3 S3-Bucket.

Der Amazon S3 S3-Bucket wird zum Speichern von Dateien und IAM-Rollen verwendet, die die für die Durchführung von Bereitstellungen erforderlichen Berechtigungen gewähren. Die erforderlichen Ressourcen werden in einem AWS CloudFormation Stack definiert, dem sogenannten Bootstrap-Stack, der normalerweise benannt ist. `CDKToolkit` Wie jeder AWS CloudFormation Stack erscheint er in der AWS CloudFormation Konsole, sobald er bereitgestellt wurde:



Das Gleiche gilt für den Bucket:



Um die Dienste, die wir benötigen, in unsere Stack-Datei zu importieren, können wir den folgenden Befehl verwenden:

```
npm install aws-cdk-lib # V2 command
```

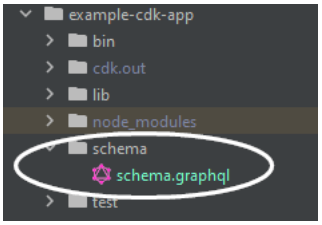
> **Tip**
>
> Wenn Sie Probleme mit V2 haben, können Sie die einzelnen Bibliotheken mit V1-Befehlen installieren:
>
> ```
> npm install @aws-cdk/aws-appsync @aws-cdk/aws-dynamodb
> ```
>
> Wir empfehlen dies nicht, da V1 veraltet ist.

Implementierung eines CDK-Projekts — Schema

Wir können jetzt mit der Implementierung unseres Codes beginnen. Zuerst müssen wir unser Schema erstellen. Sie können einfach eine `.graphql` Datei in Ihrer App erstellen:

```
mkdir schema
touch schema.graphql
```

In unserem Beispiel haben wir ein Verzeichnis auf oberster Ebene mit dem Namen hinzugefügt, das `schema` Folgendes enthält: `schema.graphql`



Lassen Sie uns in unser Schema ein einfaches Beispiel aufnehmen:

```
input CreatePostInput {
    title: String
    content: String
}

type Post {
    id: ID!
    title: String
    content: String
}
```

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

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

Zurück in unserer Stack-Datei müssen wir sicherstellen, dass die folgenden Importdirektiven definiert sind:

```
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';
```

Innerhalb der Klasse fügen wir Code hinzu, um unsere GraphQL-API zu erstellen und sie mit unserer `schema.graphql` Datei zu verbinden:

```
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'),
    });
  }
}
```

Wir werden auch etwas Code hinzufügen, um die GraphQL-URL, den API-Schlüssel und die Region auszudrucken:

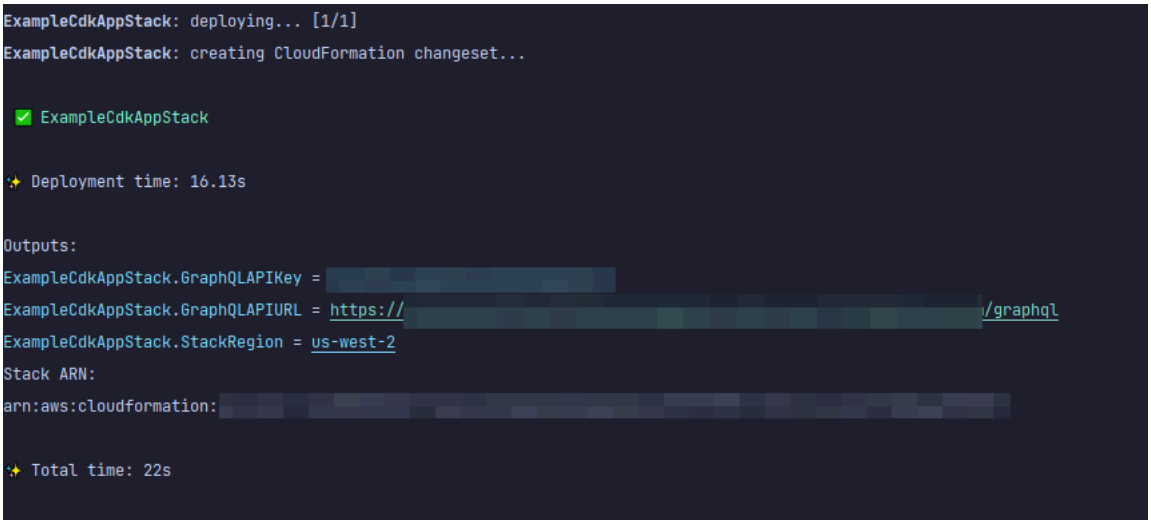
```
export class ExampleCdkAppStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // 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
    });
  }
}
```

An dieser Stelle werden wir unsere App erneut bereitstellen verwenden:

```
cdk deploy
```

Das ist das Ergebnis:



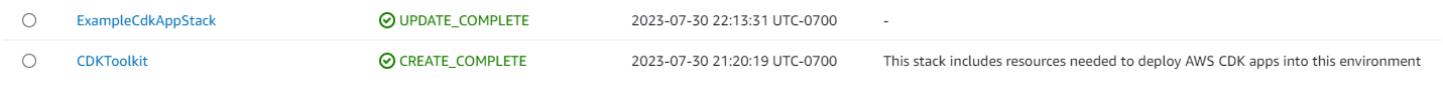
Es scheint, dass unser Beispiel erfolgreich war, aber schauen wir uns die AWS AppSync Konsole an, um zu bestätigen:

Es scheint, dass unsere API erstellt wurde. Jetzt überprüfen wir das an die API angehängte Schema:

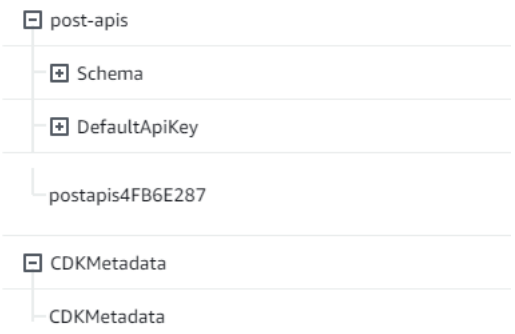
Schema

```
input CreatePostInput {
  title: String
  date: AWSDateTime
}

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

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

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

Dies scheint mit unserem Schemacode übereinzustimmen, es war also erfolgreich. Eine andere Möglichkeit, dies aus Sicht der Metadaten zu bestätigen, besteht darin, sich den AWS CloudFormation Stack anzusehen:



Wenn wir unsere CDK-App bereitstellen, werden Ressourcen AWS CloudFormation wie der Bootstrap aktiviert. Jeder Stapel in unserer App wird 1:1 einem AWS CloudFormation Stapel zugeordnet. Wenn Sie zum Stack-Code zurückkehren, wurde der Stackname aus dem Klassennamen `ExampleCdkAppStack` übernommen. Sie können die erstellten Ressourcen, die auch unseren Namenskonventionen entsprechen, in unserem GraphQL-API-Konstrukt sehen:



Implementierung eines CDK-Projekts — Datenquelle

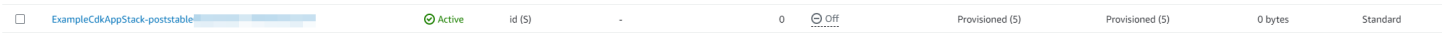
Als Nächstes müssen wir unsere Datenquelle hinzufügen. In unserem Beispiel wird eine DynamoDB-Tabelle verwendet. Innerhalb der Stack-Klasse fügen wir etwas Code hinzu, um eine neue Tabelle zu erstellen:

```
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
    });
  }
}
```

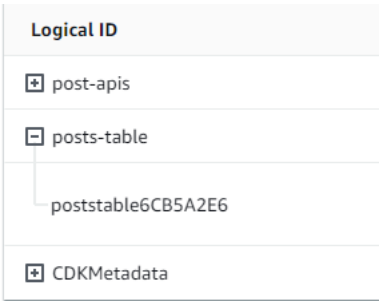
Lassen Sie uns an dieser Stelle erneut bereitstellen:

```
cdk deploy
```

Wir sollten in der DynamoDB-Konsole nach unserer neuen Tabelle suchen:



Unser Stack-Name ist korrekt und der Tabellenname entspricht unserem Code. Wenn wir unseren AWS CloudFormation Stack erneut überprüfen, sehen wir jetzt die neue Tabelle:



Implementierung eines CDK-Projekts - Resolver

In diesem Beispiel werden zwei Resolver verwendet: einer zum Abfragen der Tabelle und einer zum Hinzufügen. Da wir Pipeline-Resolver verwenden, müssen wir zwei Pipeline-Resolver mit jeweils einer Funktion deklarieren. In der Abfrage fügen wir den folgenden Code hinzu:

```
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
  });
}
```

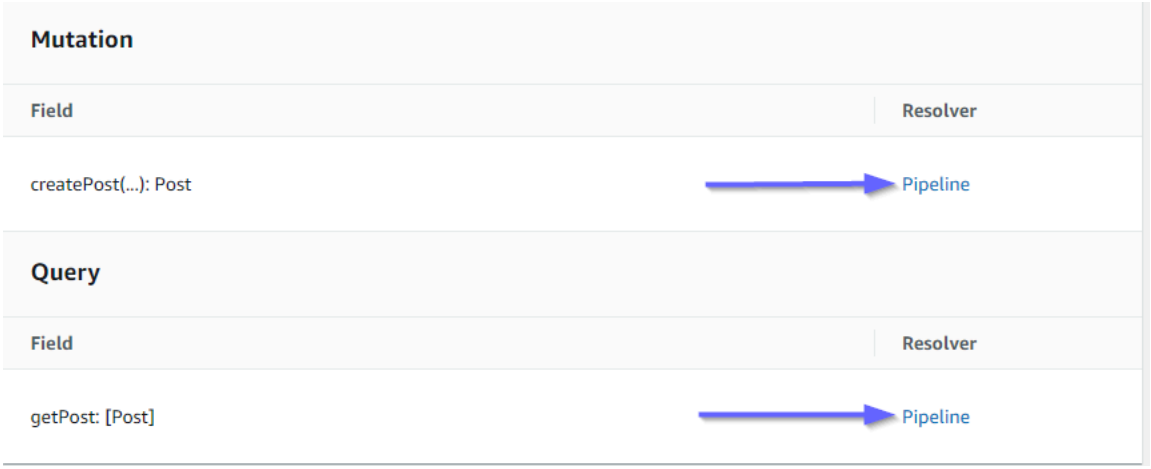
```
}
```

In diesem Snippet haben wir einen Pipeline-Resolver hinzugefügt, an den eine Funktion namens `func-add-post` angehängt ist. `pipeline-resolver-create-posts` Dies ist der Code, der der Tabelle `Posts` hinzugefügt wird. Der andere Pipeline-Resolver wurde `pipeline-resolver-get-posts` mit einer Funktion aufgerufen`func-get-post`, die zur Tabelle `Posts` hinzugefügte Daten abruft.

Wir werden das bereitstellen, um es dem AWS AppSync Service hinzuzufügen:

```
cdk deploy
```

Schauen wir in der AWS AppSync Konsole nach, ob sie an unsere GraphQL-API angehängt waren:



Es scheint richtig zu sein. Im Code wurden diese beiden Resolver an die von uns erstellte GraphQL-API angehängt (gekennzeichnet durch den `api` Props-Wert, der sowohl in den Resolvern als auch in den Funktionen vorhanden ist). In der GraphQL-API wurden die Felder, an die wir unsere Resolver angehängt haben, auch in den Requisiten angegeben (definiert durch die `typename` und `fieldname` -Requisiten in jedem Resolver).

Schauen wir uns an, ob der Inhalt der Resolver korrekt ist, beginnend mit: `pipeline-resolver-get-posts`

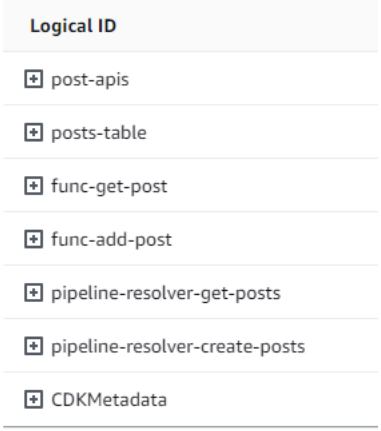
Die Vorher- und Nachher-Handler entsprechen unserem `code` Requisitenwert. Wir können auch sehen, dass eine Funktion aufgerufen wurde`add_posts_func_1`, die dem Namen der Funktion entspricht, die wir im Resolver angehängt haben.

Schauen wir uns den Codeinhalt dieser Funktion an:

Dies entspricht den `code` Requisiten der `add_posts_func_1` Funktion. Unsere Anfrage wurde erfolgreich hochgeladen, also schauen wir uns die Anfrage an:

Diese stimmen auch mit dem Code überein. Wenn wir uns ansehen`get_posts_func_1`:

Alles scheint an seinem Platz zu sein. Um dies aus Sicht der Metadaten zu bestätigen, können wir unseren Stack AWS CloudFormation erneut einchecken:



Jetzt müssen wir diesen Code testen, indem wir einige Anfragen ausführen.

Implementierung eines CDK-Projekts — Anfragen

Um unsere App in der AWS AppSync Konsole zu testen, haben wir eine Abfrage und eine Mutation durchgeführt:

```
query MyQuery {
  getPost {
    id
    date
    title
  }
}

mutation MyMutation {
  createPost(input: {date: "1970-01-01T12:30:00.000Z", title: "first post"}) {
    date
    id
    title
  }
}

```

`MyMutation`enthält eine `createPost` Operation mit den Argumenten `1970-01-01T12:30:00.000Z` und`first post`. Sie gibt den Wert `date` und zurück`title`, den wir übergeben haben, sowie den automatisch generierten `id` Wert. Das Ausführen der Mutation ergibt das Ergebnis:

```
{
  "data": {
    "createPost": {
      "date": "1970-01-01T12:30:00.000Z",
      "id": "4dc1c2dd-0aa3-4055-9eca-7c140062ada2",
      "title": "first post"
    }
  }
}
```

Wenn wir die DynamoDB-Tabelle schnell überprüfen, können wir unseren Eintrag in der Tabelle sehen, wenn wir sie scannen:

id *(String)*	date	title
9f62c4dd-49d5-48d5-b835-143284c72fe0	1970-01-01T12:30:00.000Z	first post

Wenn wir in der AWS AppSync Konsole die Abfrage ausführen, um dies abzurufen`Post`, erhalten wir das folgende Ergebnis:

```
{
  "data": {
    "getPost": [
      {
```

```
        "id": "9f62c4dd-49d5-48d5-b835-143284c72fe0",
        "date": "1970-01-01T12:30:00.000Z",
        "title": "first post"
      }
    ]
  }
}
```

Verwendung von Abonnements für Echtzeit-Datenanwendungen in AWS AppSync

> **Important**
>
> Ab dem 13. März 2025 können Sie WebSockets mithilfe von AWS AppSync Events eine PubSub Echtzeit-API erstellen. Weitere Informationen finden Sie unter Ereignisse veröffentlichen über WebSocket im AWS AppSync Events Developer Guide.

AWS AppSyncermöglicht es Ihnen, Abonnements zu nutzen, um Live-Anwendungsupdates, Push-Benachrichtigungen usw. zu implementieren. Wenn Clients die GraphQL-Abonnementvorgänge aufrufen, wird automatisch eine sichere WebSocket Verbindung hergestellt und aufrechterhalten von. AWS AppSync Anwendungen können dann Daten in Echtzeit von einer Datenquelle an Abonnenten verteilen und gleichzeitig die Verbindungs- und Skalierungsanforderungen der Anwendung AWS AppSync kontinuierlich verwalten. In den folgenden Abschnitten erfahren Sie, wie Abonnements AWS AppSync funktionieren.

GraphQL-Schemaabonnementanweisungen

Abonnements in AWS AppSync werden als Reaktion auf eine Mutation aufgerufen. Das bedeutet, dass Sie jede Datenquelle in AWS AppSync Echtzeit erstellen können, indem Sie eine GraphQL-Schemadirektive für eine Mutation angeben.

Die AWS Amplify Clientbibliotheken kümmern sich automatisch um die Verwaltung der Abonnementverbindungen. Die Bibliotheken verwenden Pure WebSockets als Netzwerkprotokoll zwischen dem Client und dem Dienst.

> **Note**
>
> Um die Autorisierung bei der Verbindung mit einem Abonnement zu steuern, können Sie AWS Identity and Access Management (IAM), AWS Lambda Amazon Cognito Cognito-Identitätspools oder Amazon Cognito Cognito-Benutzerpools für die Autorisierung auf Feldebene verwenden. Für detaillierte Zugriffskontrollen für Abonnements können Sie Resolver an Ihre Abonnementfelder anhängen und anhand der Identität des Anrufers und der Datenquellen eine Logik ausführen. AWS AppSync Weitere Informationen finden Sie unter Konfiguration von Autorisierung und Authentifizierung zur Sicherung Ihres GraphQL APIs.

Abonnements werden durch Mutationen ausgelöst und die Mutationsauswahlmenge wird an Abonnenten gesendet.

Das folgende Beispiel zeigt, wie Sie mit GraphQL-Abonnements arbeiten. Es gibt keine Datenquelle an, da es sich bei der Datenquelle um Lambda, Amazon DynamoDB oder Amazon Service handeln könnte. OpenSearch

Um mit Abonnements zu beginnen, müssen Sie Ihrem Schema wie folgt einen Abonnement-Einstiegspunkt hinzufügen:

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

Angenommen, Sie haben eine Blog-Website und möchten neue Blogs sowie Änderungen an vorhandenen Blogs abonnieren. Fügen Sie zu diesem Zweck folgende `Subscription`-Definition dem Schema hinzu:

```
type Subscription {
    addedPost: Post
    updatedPost: Post
    deletedPost: Post
}
```

Nehmen wir außerdem an, Sie haben die folgenden Mutationen:

```
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!
}
```

Sie können diese Felder in Echtzeitfelder umwandeln, indem Sie folgendermaßen eine `@aws_subscribe(mutations: ["mutation_field_1", "mutation_field_2"])`-Richtlinie für jedes Abonnement hinzufügen, für das Sie Benachrichtigungen erhalten möchten:

```
type Subscription {
    addedPost: Post
    @aws_subscribe(mutations: ["addPost"])
    updatedPost: Post
    @aws_subscribe(mutations: ["updatePost"])
    deletedPost: Post
    @aws_subscribe(mutations: ["deletePost"])
}
```

Da das eine Reihe von Mutationseingaben `@aws_subscribe(mutations: ["",..,""])` benötigt, können Sie mehrere Mutationen angeben, wodurch ein Abonnement initiiert wird. Wenn Sie das Abonnement von einem Client aus initiieren, sieht Ihre GraphQL-Abfrage möglicherweise wie folgt aus:

```
subscription NewPostSub {
    addedPost {
        __typename
        version
        title
        content
        author
        url
    }
}
```

Diese Abonnementabfrage wird für Client-Verbindungen und Tools benötigt.

Beim reinen WebSockets Client erfolgt die Filterung der Auswahlsätze pro Client, da jeder Client seinen eigenen Auswahlsatz definieren kann. In diesem Fall muss der Abonnementauswahlsatz eine Teilmenge des Mutationsauswahlsatzes sein. Beispiel: Abonnement `addPost(...){id author title url version}`, das mit Mutation `addedPost{author title}` verknüpft ist, erhält

nur den Autor und den Titel des Beitrags. Die anderen Felder werden nicht empfangen. Wenn der Mutation jedoch der Autor in ihrem Auswahlsatz fehlte, erhält der Abonnent einen `null`-Wert für das Autorenfeld (oder einen Fehler, falls das Autorenfeld im Schema als erforderlich/nicht NULL definiert ist).

Der Abonnement-Auswahlsatz ist bei der Verwendung von Pure unerlässlich WebSockets. Wenn ein Feld im Abonnement nicht explizit definiert ist, wird AWS AppSync das Feld nicht zurückgegeben.

Im vorherigen Beispiel hatten die Abonnements keine Argumente. Angenommen, Ihr Schema sieht wie folgt aus:

```
type Subscription {
    updatedPost(id:ID! author:String): Post
    @aws_subscribe(mutations: ["updatePost"])
}
```

In diesem Fall definiert der Client ein Abonnement wie folgt:

```
subscription UpdatedPostSub {
    updatedPost(id:"XYZ", author:"ABC") {
        title
        content
    }
}
```

Der Rückgabetyp eines `subscription`-Felds in Ihrem Schema muss dem Rückgabetyp des zugehörigen Mutationsfelds entsprechen. Im vorherigen Beispiel war dies als `addPost` und `addedPost` dargestellt, zurückgegeben als `Post`-Typ.

Informationen zum Einrichten von Abonnements auf dem Client finden Sie unterErstellen einer Client-Anwendung mit dem Amplify-Client.

Abonnementargumente verwenden

Ein wichtiger Teil der Verwendung von GraphQL-Abonnements besteht darin, zu verstehen, wann und wie Argumente verwendet werden. Sie können subtile Änderungen vornehmen, um zu ändern, wie und wann Clients über aufgetretene Mutationen informiert werden sollen. Sehen Sie sich dazu das Beispielschema aus dem Schnellstart-Kapitel an, in dem „Todos“ erstellt wird. Für dieses Beispielschema sind die folgenden Mutationen definiert:

```
type Mutation {
    createTodo(input: CreateTodoInput!): Todo
    updateTodo(input: UpdateTodoInput!): Todo
    deleteTodo(input: DeleteTodoInput!): Todo
}
```

Im Standardbeispiel können Clients Updates für alle abonnieren, `Todo` indem sie das `onUpdateTodo` `subscription` ohne Argumente verwenden:

```
subscription OnUpdateTodo {
  onUpdateTodo {
    description
    id
    name
    when
  }
}
```

Sie können Ihre `subscription` anhand ihrer Argumente filtern. Um beispielsweise nur eine auszulösen, `subscription` wenn eine `todo` mit einem bestimmten aktualisiert `ID` wird, geben Sie den folgenden `ID` Wert an:

```
subscription OnUpdateTodo {
  onUpdateTodo(id: "a-todo-id") {
    description
    id
    name
    when
  }
}
```

Sie können auch mehrere Argumente übergeben. Im Folgenden wird beispielsweise gezeigt, `subscription` wie Sie über `Todo` Aktualisierungen an einem bestimmten Ort und zu einer bestimmten Zeit benachrichtigt werden können:

```
subscription todosAtHome {
  onUpdateTodo(when: "tomorrow", where: "at home") {
    description
    id
    name
```

```
    when
    where
  }
}
```

Beachten Sie, dass alle Argumente optional sind. Wenn Sie in Ihrem keine Argumente angeben`subscription`, abonnieren Sie alle `Todo` Updates, die in Ihrer Anwendung auftreten. Sie könnten jedoch Ihre Felddefinition aktualisieren, sodass das `ID` Argument erforderlich `subscription` ist. Dies würde die Antwort eines bestimmten `todo` statt aller `todo` s erzwingen:

```
onUpdateTodo(
  id: ID!,
  name: String,
  when: String,
  where: String,
  description: String
): Todo
```

Argument-Null-Wert hat Bedeutung

Bei einer Abonnementabfrage filtert ein `null` Argumentwert die Ergebnisse anders AWS AppSync, als wenn das Argument vollständig weggelassen wird.

Kehren wir zum Todos-API-Beispiel zurück, in dem wir Todos erstellen könnten. Sehen Sie sich das Beispielschema aus dem Schnellstart-Kapitel an.

Lassen Sie uns unser Schema so ändern, dass es ein neues `owner` Feld für den `Todo` Typ enthält, das beschreibt, wer der Eigentümer ist. Das `owner` Feld ist nicht erforderlich und kann nur aktiviert werden`UpdateTodoInput`. Sehen Sie sich die folgende vereinfachte Version des Schemas an:

```
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"])
}
```

Das folgende Abonnement gibt alle Todo Updates zurück:

```
subscription MySubscription {
  onUpdateTodo {
    description
    id
    name
    when
    where
  }
}
```

Wenn Sie das vorherige Abonnement ändern, um das Feldargument hinzuzufügen`owner: null`, stellen Sie jetzt eine andere Frage. Dieses Abonnement registriert jetzt den Client, um über alle Todo Updates informiert zu werden, für die kein Eigentümer angegeben wurde.

```
subscription MySubscription {
  onUpdateTodo(owner: null) {
```

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

> **Note**
>
> Seit dem 1. Januar 2022 WebSockets ist MQTT over nicht mehr als Protokoll für GraphQL-Abonnements in verfügbar. AWS AppSync APIs Pure WebSockets ist das einzige Protokoll, das in unterstützt wird. AWS AppSync
> Clients, die auf dem AWS AppSync SDK oder den Amplify-Bibliotheken basieren und nach November 2019 veröffentlicht wurden, verwenden WebSockets standardmäßig automatisch Pure. Durch das Upgrade der Clients auf die neueste Version können sie die Pure WebSockets Engine verwenden AWS AppSync.
> Pure WebSockets bietet eine größere Nutzlast (240 KB), eine größere Auswahl an Client-Optionen und verbesserte CloudWatch Metriken. Weitere Informationen zur Verwendung von Pure WebSocket Clients finden Sie unterAufbau eines Echtzeit-Clients in WebSocket AWS AppSync.

Generisches pub/sub APIs Powered by Serverless WebSockets in AWS AppSync

> **Important**
>
> Ab dem 13. März 2025 können Sie WebSockets mithilfe von AWS AppSync Events eine PubSub Echtzeit-API erstellen. Weitere Informationen finden Sie unter Ereignisse veröffentlichen über WebSocket im AWS AppSync Events Developer Guide.

Bei einigen Anwendungen ist lediglich Simple erforderlich WebSocket APIs , bei dem sich die Kunden einen bestimmten Kanal oder ein bestimmtes Thema anhören. Generische JSON-Daten ohne spezifische Form oder stark typisierte Anforderungen können nach einem einfachen Publish-Subscribe-Muster (Pub/Sub) an Kunden weitergegeben werden, die sich einen dieser Kanäle anhören.

Wird verwendet AWS AppSync , um einfache Implementierungen pub/sub WebSocket APIs mit wenig bis gar keinen GraphQL-Kenntnissen innerhalb von Minuten durchzuführen, indem GraphQL-Code sowohl auf dem API-Backend als auch auf der Clientseite automatisch generiert wird.

Pub-Sub erstellen und konfigurieren APIs

Gehen Sie zunächst wie folgt vor:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
   - Wählen Sie im Dashboard Create API (API erstellen) aus.
2. Wählen Sie auf dem nächsten Bildschirm Create a real-time API und dann Next aus.
3. Geben Sie einen benutzerfreundlichen Namen für Ihre pub/sub API ein.
4. Sie können private API-Funktionen aktivieren, wir empfehlen jedoch, dies vorerst auszuschalten. Wählen Sie Weiter aus.
5. Sie können wählen, ob Sie mit automatisch eine funktionierende pub/sub API generieren möchten WebSockets. Wir empfehlen, diese Funktion vorerst ebenfalls auszuschalten. Wählen Sie Weiter aus.
6. Wählen Sie Create API und warten Sie dann einige Minuten. Eine neue vorkonfigurierte AWS AppSync Pub/Sub-API wird in Ihrem Konto erstellt. AWS

Die API verwendet AWS AppSync die integrierten lokalen Resolver (weitere Informationen zur Verwendung lokaler Resolver finden Sie unter Tutorial: Lokale Resolver im AWS AppSync Entwicklerhandbuch), um mehrere temporäre pub/sub Kanäle und WebSocket Verbindungen zu verwalten. Dabei werden Daten automatisch nur auf der Grundlage des Kanalnamens an abonnierte Clients übermittelt und gefiltert. API-Aufrufe werden mit einem API-Schlüssel autorisiert.

Nach der Bereitstellung der API stehen Ihnen einige zusätzliche Schritte zur Generierung von Client-Code und dessen Integration in Ihre Client-Anwendung zur Verfügung. Als Beispiel für die schnelle Integration eines Clients wird in diesem Handbuch eine einfache React-Webanwendung verwendet.

1. Erstellen Sie zunächst eine Boilerplate-React-App mit NPM auf Ihrem lokalen Computer:

```
$ npx create-react-app mypubsub-app
$ cd mypubsub-app
```

> **Note**
>
> In diesem Beispiel werden die Amplify-Bibliotheken verwendet, um Clients mit der Backend-API zu verbinden. Es ist jedoch nicht erforderlich, ein Amplify CLI-Projekt lokal zu erstellen. Während React in diesem Beispiel der Client der Wahl ist, unterstützen Amplify-Bibliotheken auch iOS-, Android- und Flutter-Clients und bieten dieselben Funktionen in diesen unterschiedlichen Laufzeiten. Die unterstützten Amplify-Clients bieten einfache Abstraktionen für die Interaktion mit AWS AppSync GraphQL-API-Backends mit wenigen Codezeilen, einschließlich integrierter WebSocket Funktionen, die vollständig mit dem Echtzeitprotokoll kompatibel sind:AWS AppSync WebSocket
>
> ```
> $ npm install @aws-amplify/api
> ```

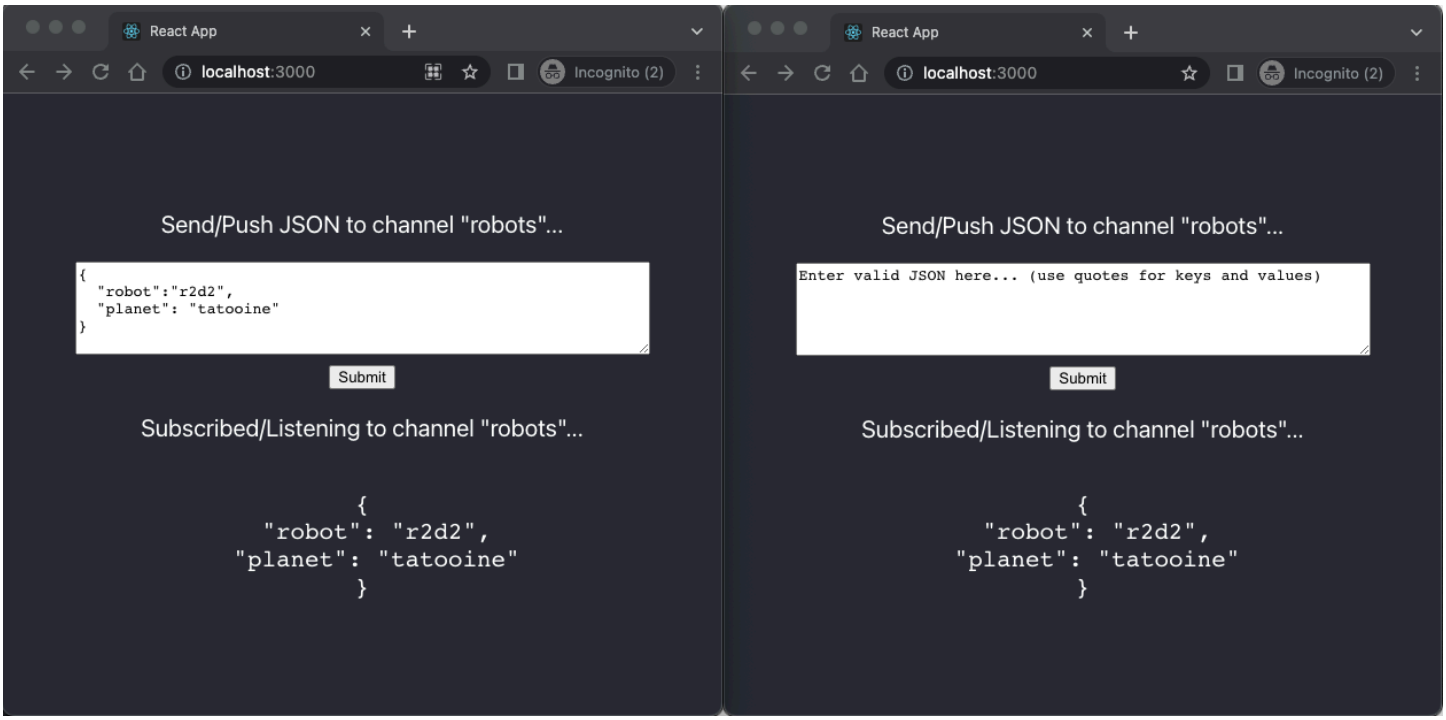
2. Wählen Sie in der AWS AppSync Konsole dann Herunterladen aus JavaScript, um eine einzelne Datei mit den API-Konfigurationsdetails und dem generierten GraphQL-Operationscode herunterzuladen.
3. Kopieren Sie die heruntergeladene Datei in den `/src` Ordner in Ihrem React-Projekt.
4. Ersetzen Sie als Nächstes den Inhalt der vorhandenen `src/App.js` Boilerplate-Datei durch den in der Konsole verfügbaren Beispiel-Client-Code.
5. Verwenden Sie den folgenden Befehl, um die Anwendung lokal zu starten:

```
$ npm start
```

6. Um das Senden und Empfangen von Echtzeitdaten zu testen, öffnen Sie zwei Browserfenster und greifen Sie darauf zu*`localhost:3000`*. Die Beispielanwendung ist so konfiguriert, dass sie generische JSON-Daten an einen fest codierten Kanal mit dem Namen *`robots`* sendet.
7. Geben Sie in einem der Browserfenster den folgenden JSON-Blob in das Textfeld ein und klicken Sie dann auf Senden:

```
{
  "robot":"r2d2",
  "planet": "tatooine"
}
```

Beide Browserinstanzen haben den *robots* Kanal abonniert und empfangen die veröffentlichten Daten in Echtzeit, die unten in der Webanwendung angezeigt werden:



Der gesamte erforderliche GraphQL-API-Code, einschließlich des Schemas, der Resolver und der Operationen, wird automatisch generiert, um einen generischen pub/sub Anwendungsfall zu ermöglichen. Im Backend werden Daten mit einer GraphQL-Mutation wie AWS AppSync der folgenden auf dem Echtzeit-Endpunkt veröffentlicht:

```
mutation PublishData {
    publish(data: "{\"msg\": \"hello world!\"}", name: "channel") {
        data
        name
    }
}
```

Abonnenten greifen mit einem entsprechenden GraphQL-Abonnement auf die veröffentlichten Daten zu, die an den jeweiligen temporären Kanal gesendet wurden:

```
subscription SubscribeToData {
    subscribe(name:"channel") {
        name
        data
    }
```

```
}
```

Implementierung von Pub-Sub in bestehende Anwendungen APIs

Falls Sie nur eine Echtzeitfunktion in einer vorhandenen Anwendung implementieren müssen, kann diese generische pub/sub API-Konfiguration problemlos in jede Anwendung oder API-Technologie integriert werden. Die Verwendung eines einzigen API-Endpunkts für den sicheren Zugriff, die Bearbeitung und Kombination von Daten aus einer oder mehreren Datenquellen in einem einzigen Netzwerkaufruf mit GraphQL bietet zwar Vorteile, es ist jedoch nicht erforderlich, eine bestehende REST-basierte Anwendung von Grund auf neu zu konvertieren oder neu zu erstellen, um die Echtzeitfunktionen nutzen zu können. AWS AppSync Sie könnten beispielsweise einen vorhandenen CRUD-Workload in einem separaten API-Endpunkt haben, wobei Clients Nachrichten oder Ereignisse von der vorhandenen Anwendung an die generische pub/sub API nur in Echtzeit und zu bestimmten pub/sub Zwecken senden und empfangen.

Definition erweiterter Abonnementfilter in AWS AppSync

> **Important**
>
> Ab dem 13. März 2025 können Sie WebSockets mithilfe von AWS AppSync Events eine PubSub Echtzeit-API erstellen. Weitere Informationen finden Sie unter Ereignisse veröffentlichen über WebSocket im AWS AppSync Events Developer Guide.

In AWS AppSync können Sie die Geschäftslogik für die Datenfilterung im Backend direkt in den GraphQL-API-Abonnement-Resolvern definieren und aktivieren, indem Sie Filter verwenden, die zusätzliche logische Operatoren unterstützen. Im Gegensatz zu den Abonnementargumenten, die in der Abonnementabfrage im Client definiert sind, können Sie diese Filter konfigurieren. Weitere Hinweise zur Verwendung von Abonnementargumenten finden Sie unterAbonnementargumente verwenden. Eine Liste der Operatoren finden Sie unterAWS AppSync Referenz zum Hilfsprogramm für Resolver-Mapping-Vorlagen.

Für die Zwecke dieses Dokuments unterteilen wir die Echtzeit-Datenfilterung in die folgenden Kategorien:

- Grundlegendes Filtern — Filterung auf der Grundlage von vom Kunden definierten Argumenten in der Abonnementabfrage.

- Verbesserte Filterung — Filterung auf der Grundlage einer Logik, die zentral im AWS AppSync Service-Backend definiert ist.

In den folgenden Abschnitten wird erklärt, wie erweiterte Abonnementfilter konfiguriert werden und wie sie in der Praxis angewendet werden.

Definieren von Abonnements in Ihrem GraphQL-Schema

Um erweiterte Abonnementfilter zu verwenden, definieren Sie das Abonnement im GraphQL-Schema und definieren dann den erweiterten Filter mithilfe einer Filtererweiterung. Um zu veranschaulichen, wie die erweiterte Abonnementfilterung funktioniert AWS AppSync, verwenden Sie als Beispiel das folgende GraphQL-Schema, das eine Ticket-Management-System-API definiert:

```
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
```

Angenommen, Sie erstellen eine `NONE` Datenquelle für Ihre API und fügen dann mithilfe dieser Datenquelle einen Resolver an die `createTicket` Mutation an. Ihre Handler könnten so aussehen:

```
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**
>
> Verbesserte Filter werden im Handler des GraphQL-Resolvers in einem bestimmten Abonnement aktiviert. Weitere Informationen finden Sie in der Resolver-Referenz.

Um das Verhalten des erweiterten Filters zu implementieren, müssen Sie die `extensions.setSubscriptionFilter()` Funktion verwenden, um einen Filterausdruck zu definieren, der anhand veröffentlichter Daten aus einer GraphQL-Mutation ausgewertet wird, an der die abonnierten Clients interessiert sein könnten. Weitere Informationen zu den Filtererweiterungen finden Sie unter Erweiterungen.

Im folgenden Abschnitt wird erklärt, wie Sie Filtererweiterungen verwenden, um erweiterte Filter zu implementieren.

Erstellen erweiterter Abonnementfilter mithilfe von Filtererweiterungen

Verbesserte Filter werden in JSON in den Antworthandler der Resolver des Abonnements geschrieben. Filter können in einer Liste zusammengefasst werden, die als a `filterGroup` bezeichnet wird. Filter werden mithilfe von mindestens einer Regel definiert, die jeweils Felder, Operatoren und Werte enthält. Definieren wir einen neuen Resolveron`SpecialTicketCreated`, der einen erweiterten Filter einrichtet. Sie können mehrere Regeln in einem Filter konfigurieren, die mithilfe der UND-Logik ausgewertet werden, während mehrere Filter in einer Filtergruppe mithilfe der OR-Logik ausgewertet werden:

```
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;
}
```

Basierend auf den im vorherigen Beispiel definierten Filtern werden wichtige Tickets automatisch an abonnierte API-Clients weitergeleitet, wenn ein Ticket erstellt wird mit:

- `priority`Stufe oder `high` `medium`

  AND

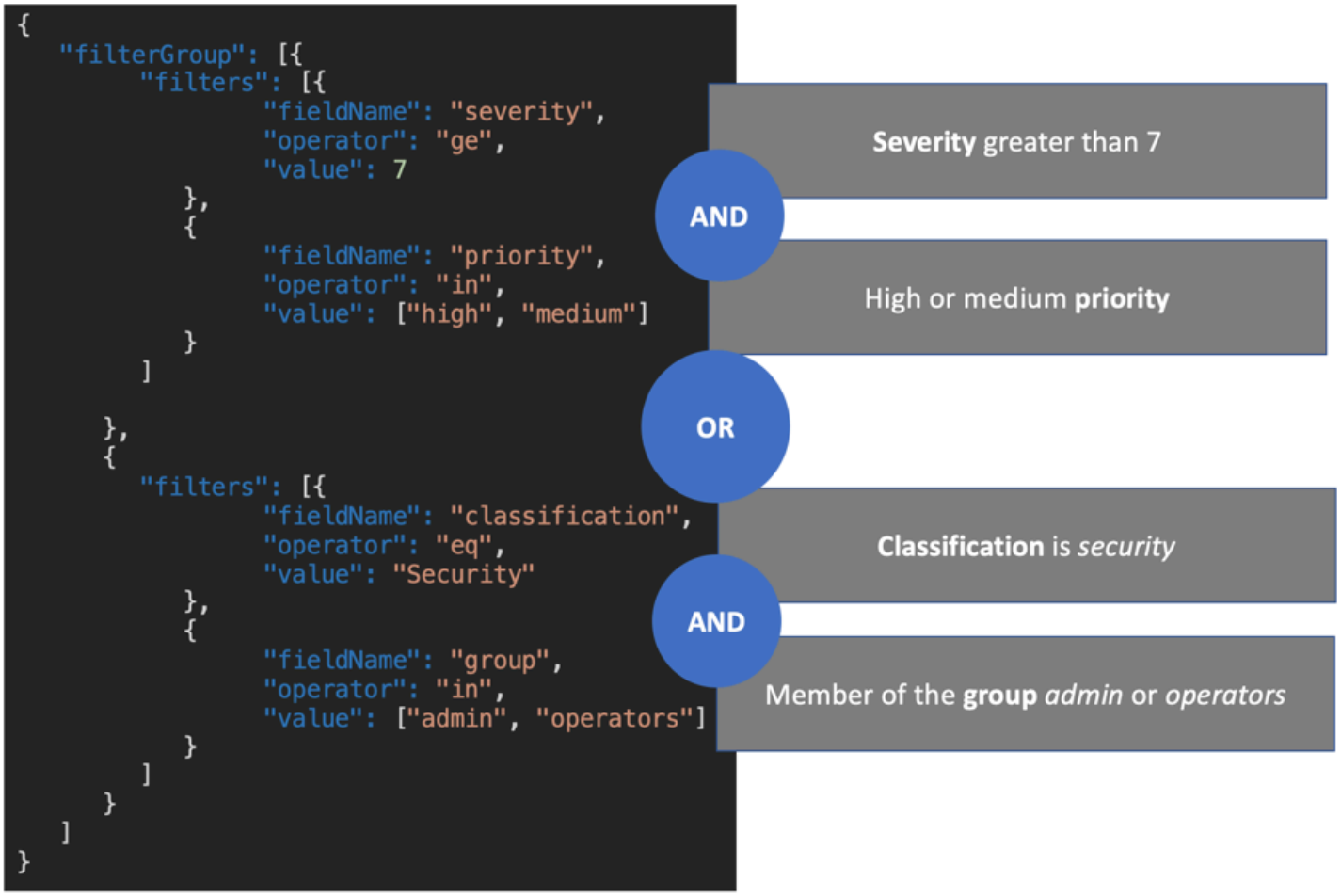
- `severity`Stufe größer oder gleich 7 (`ge`)

ODER

- `classification`Ticket gesetzt auf `Security`

  AND

- `group`Die Zuweisung ist auf `admin` oder gesetzt `operators`



Im Abonnement-Resolver definierte Filter (erweiterte Filterung) haben Vorrang vor Filtern, die nur auf Abonnementargumenten basieren (grundlegende Filterung). Weitere Informationen zur Verwendung von Abonnementargumenten finden Sie unter Abonnementargumente verwenden).

Wenn ein Argument im GraphQL-Schema des Abonnements definiert und erforderlich ist, erfolgt die Filterung auf der Grundlage des angegebenen Arguments nur, wenn das Argument in der Regel in der Methode des Resolvers definiert ist`extensions.setSubscriptionFilter()`. Wenn der Abonnement-Resolver jedoch keine `extensions` Filtermethoden enthält, werden die im Client definierten Argumente nur für die grundlegende Filterung verwendet. Sie können die Standardfilterung und die erweiterte Filterung nicht gleichzeitig verwenden.

Sie können die `contextVariable` in der Filtererweiterungslogik des Abonnements verwenden, um auf Kontextinformationen zur Anfrage zuzugreifen. Wenn Sie beispielsweise Amazon Cognito User Pools, OIDC oder benutzerdefinierte Lambda-Autorisierer für die Autorisierung verwenden, können Sie Informationen über Ihre Benutzer abrufen, sobald das Abonnement eingerichtet wird. `context.identity` Sie können diese Informationen verwenden, um Filter auf der Grundlage der Identität Ihrer Benutzer einzurichten.

Gehen Sie nun davon aus, dass Sie das erweiterte Filterverhalten für implementieren möchtenon`GroupTicketCreated`. Für das on`GroupTicketCreated` Abonnement ist ein obligatorischer `group` Name als Argument erforderlich. Bei der Erstellung wird den Tickets automatisch ein `pending` Status zugewiesen. Sie können einen Abonnementfilter einrichten, um nur neu erstellte Tickets zu erhalten, die zu der angegebenen Gruppe gehören:

```
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;
}
```

Wenn Daten mithilfe einer Mutation wie im folgenden Beispiel veröffentlicht werden:

```
mutation CreateTicket {
  createTicket(input: {priority: medium, severity: 2, group: "aws"}) {
    id
    priority
    severity
```

```
    status
    group
    createdAt
  }
}
```

Abonnierte Kunden warten darauf, dass die Daten automatisch weitergeleitet werden, WebSockets sobald ein Ticket mit der `createTicket` Mutation erstellt wird:

```
subscription OnGroup {
  onGroupTicketCreated(group: "aws") {
    category
    status
    severity
    priority
    id
    group
    createdAt
    content
  }
}
```

Clients können ohne Argumente abonniert werden, da die Filterlogik im AWS AppSync Service mit erweiterter Filterung implementiert ist, wodurch der Client-Code vereinfacht wird. Clients erhalten Daten nur, wenn die definierten Filterkriterien erfüllt sind.

Definition erweiterter Filter für verschachtelte Schemafelder

Sie können die erweiterte Abonnementfilterung verwenden, um verschachtelte Schemafelder zu filtern. Angenommen, wir haben das Schema aus dem vorherigen Abschnitt geändert, um Standort- und Adresstypen einzubeziehen:

```
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
```

Mit diesem Schema können Sie ein `.` Trennzeichen verwenden, um die Verschachtelung darzustellen. Im folgenden Beispiel wird eine Filterregel für ein verschachteltes Schemafeld unter

hinzugefügt. `location.address.country` Das Abonnement wird ausgelöst, wenn die Adresse des Tickets wie folgt gesetzt USA ist:

```
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;
}
```

Steht im obigen Beispiel `location` für Verschachtelungsebene eins, `address` für Verschachtelungsebene zwei und `country` für Verschachtelungsebene drei, die alle durch das Trennzeichen getrennt sind. `.`

Sie können dieses Abonnement testen, indem Sie die folgende Mutation verwenden: `createTicket`

```
mutation CreateTicketInUSA {
  createTicket(input: {location: "{\"address\":{\"country\":\"USA\"}}"}) {
    category
    content
    createdAt
    group
    id
    location {
      address {
        country
      }
    }
    priority
    severity
    status
  }
}
```

Definition erweiterter Filter vom Client aus

Sie können die grundlegende Filterung in GraphQL mit Abonnementargumenten verwenden. Der Client, der den Aufruf in der Abonnementabfrage durchführt, definiert die Werte der Argumente. Wenn erweiterte Filter in einem AWS AppSync Abonnement-Resolver mit der `extensions` Filterung aktiviert werden, haben die im Resolver definierten Backend-Filter Vorrang und Priorität.

Konfigurieren Sie dynamische, vom Client definierte erweiterte Filter mithilfe eines Arguments im Abonnement. `filter` Wenn Sie diese Filter konfigurieren, müssen Sie das GraphQL-Schema aktualisieren, um das neue Argument widerzuspiegeln:

```
...
type Subscription {
    onSpecialTicketCreated(filter: String): Ticket
        @aws_subscribe(mutations: ["createTicket"])
}
...
```

Der Client kann dann eine Abonnementabfrage wie im folgenden Beispiel senden:

```
subscription onSpecialTicketCreated($filter: String) {
     onSpecialTicketCreated(filter: $filter) {
        id
        group
        description
        priority
        severity
     }
 }
```

Sie können die Abfragevariable wie im folgenden Beispiel konfigurieren:

```
{"filter" : "{\"severity\":{\"le\":2}}"}
```

Das `util.transform.toSubscriptionFilter()` Resolver-Hilfsprogramm kann in der Vorlage für die Zuordnung von Abonnementantworten implementiert werden, um den im Abonnementargument definierten Filter für jeden Client anzuwenden:

```
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;
}
```

Mit dieser Strategie können Kunden ihre eigenen Filter definieren, die eine erweiterte Filterlogik und zusätzliche Operatoren verwenden. Filter werden zugewiesen, wenn ein bestimmter Client die Abonnementabfrage in einer sicheren WebSocket Verbindung aufruft. Weitere Informationen zum Transformationsprogramm für erweiterte Filterung, einschließlich des Formats der Payload der `filter` Abfragevariablen, finden Sie unter Übersicht über JavaScriptResolver.

Zusätzliche erweiterte Filtereinschränkungen

Im Folgenden sind mehrere Anwendungsfälle aufgeführt, in denen erweiterte Filter zusätzlichen Einschränkungen unterliegen:

- Verbesserte Filter unterstützen keine Filterung für Objektlisten der obersten Ebene. In diesem Anwendungsfall werden veröffentlichte Daten aus der Mutation bei erweiterten Abonnements ignoriert.
- AWS AppSync unterstützt bis zu fünf Verschachtelungsebenen. Filter für Schemafelder, die nach der Verschachtelungsebene fünf liegen, werden ignoriert. Nehmen Sie die unten stehende GraphQL-Antwort. Das `continent` Eingabefeld `venue.address.country.metadata.continent` ist zulässig, da es sich um ein Nest der Stufe 5 handelt. Da `venue.address.country.metadata.capital.financial` es `financial` sich jedoch um ein Nest der Stufe sechs handelt, funktioniert der Filter nicht:

```
{
    "data": {
        "onCreateFilterEvent": {
            "venue": {
                "address": {
                    "country": {
                        "metadata": {
```

```
                            "capital": {
                                "financial": "New York"
                            },
                            "continent" : "North America"
                        }
                    },
                    "state": "WA"
                },
                "builtYear": 2023
            },
            "private": false,
        }
    }
}
```

WebSocket Verbindungen mithilfe von Filtern abbestellen in AWS AppSync

> **Important**
>
> Ab dem 13. März 2025 können Sie WebSockets mithilfe von AWS AppSync Events eine PubSub Echtzeit-API erstellen. Weitere Informationen finden Sie unter Ereignisse veröffentlichen über WebSocket im AWS AppSync Events Developer Guide.

In AWS AppSync können Sie eine WebSocket Verbindung mit einem verbundenen Client auf der Grundlage einer bestimmten Filterlogik zwangsweise abbestellen und schließen (für ungültig erklären). Dies ist in autorisierungsbezogenen Szenarien nützlich, z. B. wenn Sie einen Benutzer aus einer Gruppe entfernen.

Die Invalidierung eines Abonnements erfolgt als Reaktion auf eine in einer Mutation definierte Nutzlast. Wir empfehlen, Mutationen, mit denen Abonnementverbindungen ungültig gemacht werden, als administrative Operationen in Ihrer API zu behandeln und die Berechtigungen entsprechend einzuschränken, indem Sie ihre Verwendung auf einen Admin-Benutzer, eine Gruppe oder einen Back-End-Dienst beschränken. Verwenden Sie beispielsweise Schemaautorisierungsrichtlinien wie `@aws_auth(cognito_groups: ["Administrators"])` oder. `@aws_iam` Weitere Informationen finden Sie unter Zusätzliche Autorisierungsmodi verwenden.

Invalidierungsfilter verwenden dieselbe Syntax und Logik wie erweiterte Abonnementfilter. Definieren Sie diese Filter mithilfe der folgenden Dienstprogramme:

- `extensions.invalidateSubscriptions()`— Definiert im Response-Handler des GraphQL-Resolvers für eine Mutation.
- `extensions.setSubscriptionInvalidationFilter()`— Definiert im Response-Handler des GraphQL-Resolvers für die mit der Mutation verknüpften Abonnements.

Weitere Informationen zu Erweiterungen der Invalidierungsfilterung finden Sie JavaScript in der Übersicht über Resolver.

Verwenden der Abonnement-Invalidierung

Verwenden Sie das folgende GraphQL-Schema, um zu sehen AWS AppSync, wie die Invalidierung von Abonnements funktioniert:

```
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
```

```
}
```

Definieren Sie einen Invalidierungsfilter im `removeUserFromGroup` Mutationsauflösungscode:

```
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 };
}
```

Wenn die Mutation aufgerufen wird, werden die im `payload` Objekt definierten Daten verwendet, um das in definierte Abonnement zu kündigen. `subscriptionField` In der Antwortzuordnungsvorlage des `onGroupMessageCreated` Abonnements ist auch ein Invalidierungsfilter definiert.

Wenn die `extensions.invalidateSubscriptions()` Payload eine ID enthält, die der ID des IDs abonnierten Clients entspricht, wie im Filter definiert, wird das entsprechende Abonnement abgemeldet. Außerdem wird die Verbindung geschlossen. WebSocket Definieren Sie den Abonnement-Resolver-Code für das `onGroupMessageCreated` Abonnement:

```
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;
}
```

Beachten Sie, dass im Antworthandler für Abonnements sowohl Abonnementfilter als auch Invalidierungsfilter gleichzeitig definiert werden können.

Nehmen wir beispielsweise an, dass Client A mithilfe der folgenden Abonnementanforderung einen neuen Benutzer mit der ID *user-1* für die Gruppe *group-1* mit der ID abonniert:

```
onGroupMessageCreated(userId : "user-1", groupId: :"group-1"){...}
```

AWS AppSync führt den Abonnement-Resolver aus, der Abonnement- und Invalidierungsfilter generiert, wie in der vorherigen `onGroupMessageCreated` Antwortzuordnungsvorlage definiert. Für Client A ermöglichen die Abonnementfilter das Senden von Daten nur an*group-1*, und die Invalidierungsfilter sind sowohl für als auch definiert. *user-1 group-1*

Gehen Sie nun davon aus, dass Client B mithilfe der folgenden Abonnementanforderung *user-2* einen Benutzer mit der ID für eine Gruppe *group-2* mit der ID abonniert:

```
onGroupMessageCreated(userId : "user-2", groupId: :"group-2"){...}
```

AWS AppSync führt den Abonnement-Resolver aus, der Abonnement- und Invalidierungsfilter generiert. Für Client B ermöglichen die Abonnementfilter das Senden von Daten nur an*group-2*, und die Invalidierungsfilter sind sowohl für als auch definiert. *user-2 group-2*

Gehen Sie als Nächstes davon aus, dass eine neue Gruppennachricht mit der ID mithilfe einer Mutationsanforderung wie im folgenden Beispiel erstellt *message-1* wird:

```
createGroupMessage(id: "message-1", groupId :
      "group-1", message: "test message"){...}
```

Abonnierte Clients, die den definierten Filtern entsprechen, erhalten automatisch die folgenden Datennutzdaten über: WebSockets

```
{
  "data": {
    "onGroupMessageCreated": {
      "id": "message-1",
      "groupId": "group-1",
```

```
      "message": "test message",
    }
  }
}
```

Client A erhält die Nachricht, weil die Filterkriterien mit dem definierten Abonnementfilter übereinstimmen. Client B empfängt die Nachricht jedoch nicht, da der Benutzer nicht Teil von ist*`group-1`*. Außerdem entspricht die Anfrage nicht dem Abonnementfilter, der im Abonnement-Resolver definiert ist.

Gehen Sie abschließend davon aus, dass *`user-1`* dies aus der *`group-1`* Verwendung der folgenden Mutationsanforderung entfernt wurde:

```
removeUserFromGroup(userId: "user-1", groupId : "group-1"){...}
```

Die Mutation leitet eine Abonnement-Invalidierung ein, wie sie in ihrem `extensions.invalidateSubscriptions()` Resolver-Antworthandlercode definiert ist. AWS AppSync meldet dann Client A ab und schließt seine Verbindung. WebSocket Client B ist davon nicht betroffen, da die in der Mutation definierte Payload für die Invalidierung nicht mit seinem Benutzer oder seiner Gruppe übereinstimmt.

Wenn eine Verbindung für AWS AppSync ungültig erklärt wird, erhält der Client eine Nachricht, in der bestätigt wird, dass er sich abgemeldet hat:

```
{
  "message": "Subscription complete."
}
```

Verwendung von Kontextvariablen in Filtern für die Invalidierung von Abonnements

Wie bei erweiterten Abonnementfiltern können Sie die `contextVariable` in der Erweiterung für den Abonnement-Invalidierungsfilter verwenden, um auf bestimmte Daten zuzugreifen.

Beispielsweise ist es möglich, eine E-Mail-Adresse als Payload für die Invalidierung in der Mutation zu konfigurieren und sie dann mit dem E-Mail-Attribut oder dem Anspruch eines abonnierten Benutzers abzugleichen, der mit Amazon Cognito Cognito-Benutzerpools oder OpenID Connect autorisiert ist. Der im `extensions.setSubscriptionInvalidationFilter()` Abonnement-Invalidator definierte Invalidierungsfilter prüft, ob die durch die `extensions.invalidateSubscriptions()` Payload der Mutation festgelegte E-Mail-Adresse

mit der E-Mail-Adresse übereinstimmt, die aus dem JWT-Token des Benutzers abgerufen wurde, wodurch die Ungültigerklärung eingeleitet wird. `context.identity.claims.email`

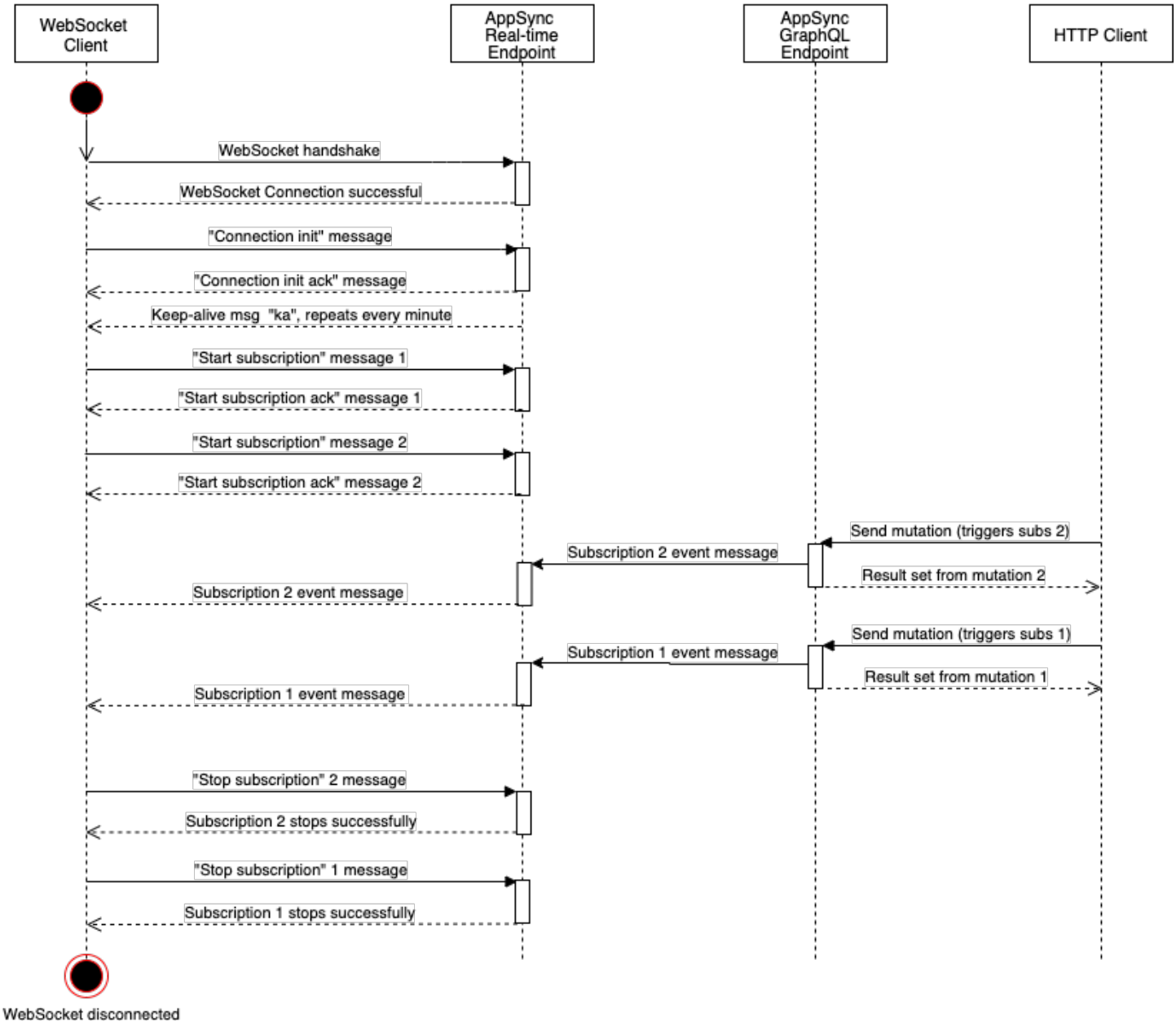
Aufbau eines Echtzeit-Clients in WebSocket AWS AppSync

> **Important**
>
> Ab dem 13. März 2025 können Sie WebSockets mithilfe von AWS AppSync Events eine PubSub Echtzeit-API erstellen. Weitere Informationen finden Sie unter Ereignisse veröffentlichen über WebSocket im AWS AppSync Events Developer Guide.

AWS AppSyncDer WebSocket Echtzeitclient ermöglicht GraphQL-Abonnements in einem mehrstufigen Prozess. Der Client stellt zunächst eine WebSocket Verbindung mit dem AWS AppSync Echtzeit-Endpunkt her, sendet eine Nachricht zur Verbindungsinitialisierung und wartet auf die Bestätigung. Nach erfolgreicher Verbindung registriert der Client Abonnements, indem er Startnachrichten mit eindeutigen IDs und GraphQL-Abfragen sendet. AWS AppSync bestätigt erfolgreiche Abonnements mit Bestätigungsnachrichten. Der Client wartet dann auf Abonnementereignisse, die durch entsprechende Mutationen ausgelöst werden. AWS AppSync Sendet regelmäßig Keep-Alive-Nachrichten, um die Verbindung aufrechtzuerhalten. Wenn der Vorgang abgeschlossen ist, hebt der Client die Registrierung der Abonnements auf, indem er Stoppnachrichten sendet. Dieses System unterstützt mehrere Abonnements über eine einzige WebSocket Verbindung und unterstützt verschiedene Autorisierungsmodi, darunter API-Schlüssel, Amazon Cognito Cognito-Benutzerpools, IAM und Lambda.

WebSocket Client-Implementierung in Echtzeit für GraphQL-Abonnements

Das folgende Sequenzdiagramm und die folgenden Schritte zeigen den Echtzeit-Abonnement-Workflow zwischen dem WebSocket Client, dem HTTP-Client und AWS AppSync.



1. Der Client stellt eine WebSocket Verbindung mit dem AWS AppSync Echtzeit-Endpunkt her. Bei einem Netzwerkfehler sollte der Client einen exponentiellen Backoff-Jitter ausführen. Weitere Informationen finden Sie unter Exponentieller Backoff und Jitter im Architektur-Blog. AWS
2. (Optional) Nach dem erfolgreichen WebSocket Verbindungsaufbau sendet der Client eine Nachricht. `connection_init`
3. Wenn `connection_init` gesendet, wartet der Client auf eine `connection_ack` Nachricht von AWS AppSync. Diese Nachricht enthält einen `connectionTimeoutMs` Parameter, der die maximale Wartezeit in Millisekunden für eine `"ka"` (Keep-Alive-) Nachricht angibt.

4. AWS AppSync sendet regelmäßig Nachrichten. `"ka"` Der Client verfolgt die Zeit, zu der er jede `"ka"` Nachricht empfangen hat. Wenn der Client innerhalb von `connectionTimeoutMs` Millisekunden keine `"ka"` Nachricht empfängt, sollte der Client die Verbindung schließen.
5. Der Client registriert das Abonnement, indem er eine `start`-Abonnementnachricht sendet. Eine einzelne WebSocket Verbindung unterstützt mehrere Abonnements, auch wenn sie sich in unterschiedlichen Autorisierungsmodi befinden.
6. Der Client wartet darauf AWS AppSync , dass `start_ack` Nachrichten gesendet werden, um erfolgreiche Abonnements zu bestätigen. Wenn ein Fehler auftritt, wird eine `"type": "error"` Meldung AWS AppSync zurückgegeben.
7. Der Client wartet auf Abonnementereignisse, die gesendet werden, nachdem eine entsprechende Mutation aufgerufen wurde. Abfragen und Mutationen werden normalerweise `https://` an den AWS AppSync GraphQL-Endpunkt gesendet. Abonnements werden mithilfe von secure WebSocket (`wss://`) über den AWS AppSync Echtzeit-Endpunkt abgewickelt.
8. Der Client hebt die Registrierung des Abonnements auf, indem er eine `stop`-Abonnementnachricht sendet.
9. Nachdem der Client die Registrierung aller Abonnements aufgehoben und überprüft hat, dass keine Nachrichten über das übertragen werden WebSocket, kann er die WebSocket Verbindung trennen.

Handshake-Details zum Herstellen der Verbindung WebSocket

Um eine Verbindung herzustellen und einen erfolgreichen Handshake mit zu initiieren AWS AppSync, benötigt ein WebSocket Client Folgendes:

- Der AWS AppSync Echtzeit-Endpunkt
- Header — Enthalten Informationen, die für den AWS AppSync Endpunkt und die Autorisierung relevant sind. AWS AppSync unterstützt die folgenden drei Methoden zur Bereitstellung von Headern:
  - Header per Abfragezeichenfolge
    - Die Header-Informationen sind als Base64-Zeichenfolge codiert, die von einem stringifizierten JSON-Objekt abgeleitet wird. Dieses JSON-Objekt enthält Details, die für den Endpunkt und die AWS AppSync Autorisierung relevant sind. Der Inhalt des JSON-Objekts variiert je nach Autorisierungsmodus.
  - Header über `Sec-WebSocket-Protocol`

    - Eine Base64URL-kodierte Zeichenfolge aus dem stringifizierten JSON-Objekt, die Informationen enthält, die für den AWS AppSync Endpunkt und die Autorisierung relevant sind, wird als Protokoll im Header übergeben. `Sec-WebSocket-Protocol` Der Inhalt des JSON-Objekts variiert je nach Autorisierungsmodus.
  - Header über Standard-HTTP-Header:
    - Header können als Standard-HTTP-Header in der Verbindungsanforderung übergeben werden, ähnlich wie Header für GraphQL-Abfragen und -Mutationen an übergeben werden. AWS AppSync Die Übergabe von Headern über Standard-HTTP-Header wird jedoch für private API-Verbindungsanfragen nicht unterstützt.
- `payload`— Base64-kodierte Zeichenfolge von. `payload` Payload wird nur benötigt, wenn Header mithilfe einer Abfragezeichenfolge bereitgestellt werden

Mit diesen Anforderungen kann ein WebSocket Client eine Verbindung zu der URL herstellen, die den Echtzeit-Endpunkt mit der Abfragezeichenfolge enthält, die `graphql-ws` WebSocket als Protokoll verwendet wird.

Erkennung des -Echtzeitendpunkts über den -GraphQL-Endpunkt

Der AWS AppSync GraphQL-Endpunkt und der AWS AppSync Echtzeit-Endpunkt unterscheiden sich geringfügig in Protokoll und Domäne. Sie können den GraphQL-Endpunkt mit dem Befehl AWS Command Line Interface `aws appsync get-graphql-api` (AWS CLI) abrufen.

AWS AppSync GraphQL-Endpunkt:

```
https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql
```

AWS AppSync Echtzeit-Endpunkt:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/
graphql
```

Anwendungen können über einen beliebigen HTTP-Client für Abfragen und Mutationen eine Verbindung zum AWS AppSync GraphQL-Endpunkt (`https://`) herstellen. Anwendungen können sich mit einem beliebigen WebSocket Client für Abonnements mit dem AWS AppSync Echtzeit-Endpunkt (`wss://`) verbinden.

Mit benutzerdefinierten Domainnamen können Sie über eine einzige Domain mit beiden Endpunkten interagieren. Wenn Sie beispielsweise `api.example.com` als Ihre benutzerdefinierte Domain

konfigurieren, können Sie mit Ihren GraphQL- und Echtzeit-Endpunkten über diese interagieren: URLs

AWS AppSync GraphQL-Endpunkt für benutzerdefinierte Domänen:

```
https://api.example.com/graphql
```

AWS AppSync Echtzeit-Endpunkt für benutzerdefinierte Domänen:

```
wss://api.example.com/graphql/realtime
```

Das Format der Header-Parameter basiert auf dem AWS AppSync API-Autorisierungsmodus

Das Format des in der Verbindungsabfragezeichenfolge verwendeten `header` Objekts hängt vom AWS AppSync API-Autorisierungsmodus ab. Das `host` Feld im Objekt bezieht sich auf den AWS AppSync GraphQL-Endpunkt, der verwendet wird, um die Verbindung zu validieren, auch wenn der `wss://` Aufruf gegen den Echtzeit-Endpunkt erfolgt. Um den Handshake zu initiieren und die autorisierte Verbindung herzustellen, sollte es sich bei `payload` um ein leeres JSON-Objekt handeln. Payload wird nur benötigt, wenn Header per Abfragestring übergeben werden.

In den folgenden Abschnitten werden die Header-Formate für jeden Autorisierungsmodus veranschaulicht.

API-Schlüssel

Kopfzeile des API-Schlüssels

Inhalt der Kopfzeile

- `"host": <string>`: Der Host für den AWS AppSync GraphQL-Endpunkt oder Ihren benutzerdefinierten Domainnamen.
- `"x-api-key": <string>`: Der für die API konfigurierte AWS AppSync API-Schlüssel.

Beispiel

```
{
    "host":"example1234567890000.appsync-api.us-east-1.amazonaws.com",
    "x-api-key":"da2-12345678901234567890123456"
```

```
}
```

Header per Abfragezeichenfolge

Zunächst wird ein JSON-Objekt, das das `host` und `x-api-key` das enthält, in eine Zeichenfolge umgewandelt. Als Nächstes wird diese Zeichenfolge mithilfe der Base64-Kodierung codiert. Die resultierende Base64-kodierte Zeichenfolge wird als Abfrageparameter mit dem Namen der WebSocket URL hinzugefügt`header`, um die Verbindung mit dem Echtzeit-Endpunkt herzustellen. AWS AppSync Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJob3N0IjoiZXhhbXBsZTEyMzQ1Njc4OTAwMDAuYXBwc3luYy1hcGkudXMtZWFzdC0xLmFtYXpvbmF3cy5jb20i
```

Es ist wichtig zu beachten, dass neben dem Base64-codierten Header-Objekt auch ein leeres JSON-Objekt {} Base64-codiert und als separater Abfrageparameter in der URL enthalten ist. `payload` WebSocket

Header über **`Sec-WebSocket-Protocol`**

Ein JSON-Objekt, das das `host` und das enthält, `x-api-key` wird in eine Zeichenfolge konvertiert und dann mit der Base64URL-Codierung codiert. Der resultierenden Base64URL-codierten Zeichenfolge wird ein Präfix vorangestellt. `header-` Diese Zeichenfolge mit Präfix wird dann zusätzlich zum `Sec-WebSocket-Protocol` Header als neues Unterprotokoll verwendet, wenn die Verbindung mit dem `graphql-ws` Echtzeit-Endpunkt hergestellt wird. WebSocket AWS AppSync

Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Der `Sec-WebSocket-Protocol` Header enthält den folgenden Wert:

```
"sec-websocket-protocol" : ["graphql-ws", "header-
ewogICAgImhvc3QiOiJleGFtcGxlMTIzNDU2Nzg5MDAwMC5hcHBzeW5jLWFwaS51cy1lYXN0LTEuYW1hem9uYXdzLmNvbSI
```

Header über Standard-HTTP-Header

Bei dieser Methode werden die Host- und API-Schlüsselinformationen mithilfe von Standard-HTTP-Headern übertragen, wenn die WebSocket Verbindung mit dem AWS AppSync Echtzeit-Endpunkt hergestellt wird. Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Die Anforderungsheader würden Folgendes beinhalten:

```
"sec-websocket-protocol" : ["graphql-ws"]
"host":"example1234567890000.appsync-api.us-east-1.amazonaws.com",
"x-api-key":"da2-12345678901234567890123456"
```

Amazon Cognito-Benutzerpools und OpenID Connect (OIDC)

Amazon Cognito und OIDCheader

Inhalt des Headers:

- `"Authorization": <string>`: Ein JWT-ID-Token. Der Header kann ein Bearer-Schema verwenden.
- `"host": <string>`: Der Host für den AWS AppSync GraphQL-Endpunkt oder Ihren benutzerdefinierten Domainnamen.

Beispiel:

```
{

 "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"
}
```

Header per Abfragezeichenfolge

Zunächst wird ein JSON-Objekt, das das `host` und `Authorization` das enthält, in eine Zeichenfolge umgewandelt. Als Nächstes wird diese Zeichenfolge mithilfe der Base64-Kodierung codiert. Die resultierende Base64-kodierte Zeichenfolge wird als Abfrageparameter mit dem Namen der WebSocket URL hinzugefügt`header`, um die Verbindung mit dem Echtzeit-Endpunkt herzustellen. AWS AppSync Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJBdXRob3JpemF0aW9uIjoiZXlKcmFXUWlPaUpqYkc1eGIzQTVlVzVNSzA5UVlYSXJNVEpIV0VGTFNYQmllVTVX
```

Es ist wichtig zu beachten, dass neben dem Base64-codierten Header-Objekt auch ein leeres JSON-Objekt {} Base64-codiert und als separater Abfrageparameter in der URL enthalten ist. `payload` WebSocket

Header über **`Sec-WebSocket-Protocol`**

Ein JSON-Objekt, das das `host` und das enthält, `Authorization` wird in eine Zeichenfolge konvertiert und dann mit der Base64URL-Codierung codiert. Der resultierenden Base64URL-codierten Zeichenfolge wird ein Präfix vorangestellt. `header-` Diese Zeichenfolge mit Präfix wird dann zusätzlich zum `Sec-WebSocket-Protocol` Header als neues Unterprotokoll verwendet, wenn die Verbindung mit dem `graphql-ws` Echtzeit-Endpunkt hergestellt wird. WebSocket AWS AppSync

Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Der `Sec-WebSocket-Protocol` Header enthält den folgenden Wert:

```
"sec-websocket-protocol" : ["graphql-ws", "header-
ewogICAgImhvc3QiOiJleGFtcGxlMTIzNDU2Nzg5MDAwMC5hcHBzeW5jLWFwaS51cy1lYXN0LTEuYW1hem9uYXdzLmNvbSI
```

Header über Standard-HTTP-Header

Bei dieser Methode werden die Host- und Autorisierungsinformationen beim WebSocket Verbindungsaufbau mit dem Echtzeit-Endpunkt mithilfe von Standard-HTTP-Headern übertragen. AWS AppSync Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Die Anforderungsheader würden Folgendes beinhalten:

```
"sec-websocket-protocol" : ["graphql-ws"]
"Authorization":"eyEXAMPLEiJjbG5xb3A5eW5MK09QYXIrMTJHWEFLSXBieU5WNHhsQjEXAMPLEnM2WldvPSIsImFsZy
zEE2DJH7sHOl2zxYi7f-SmEGoh2AD8emxQRYajByz-rE4Jh0QOymN2Ys-ZIkMpVBTPgu-
TMWDyOHhDUmUj2OP82yeZ3wlZAtr_gM4LzjXUXmI_K2yGjuXfXTaa1mvQEBG0mQfVd7SfwXB-
```

```
jcv4RYVi6j25qgow9Ew52ufurPqaK-3WAKG32KpV8J4-Wejq8t0c-
yA7sb8EnB551b7TU93uKRiVVK3E55Nk5ADPoam_WYE45i3s5qVAP_-InW75NUoOCGTsS8YWMfb6ecHYJ-1j-
bzA27zaT9VjctXn9byNFZmEXAMPLExw",
"host":"example1234567890000.appsync-api.us-east-1.amazonaws.com"
```

IAM

IAM-Header

Inhalt der Kopfzeile

- `"accept": "application/json, text/javascript"`: Ein konstanter `<string>`-Parameter.
- `"content-encoding": "amz-1.0"`: Ein konstanter `<string>`-Parameter.
- `"content-type": "application/json; charset=UTF-8"`: Ein konstanter `<string>`-Parameter.
- `"host": <string>`: Dies ist der Host für den AWS AppSync GraphQL-Endpunkt.
  - `"x-amz-date": <string>`: Der Zeitstempel muss in UTC und im folgenden ISO 8601-Format sein: YYYYMMDD'T'HHMMSS'Z'. Ein gültiger Zeitstempel ist beispielsweise 20150830T123600Z. Der Zeitstempel darf keine Millisekunden enthalten. Weitere Informationen finden Sie unter Umgang mit Daten in Signature Version 4 in der. Allgemeine AWS-Referenz
  - `"X-Amz-Security-Token": <string>`: Das AWS Sitzungstoken, das bei der Verwendung temporärer Sicherheitsanmeldedaten erforderlich ist. Weitere Informationen finden Sie unter Verwenden von temporären Anmeldeinformationen mit AWS -Ressourcen im IAM-Benutzerhandbuch.
  - `"Authorization": <string>`: Signaturinformationen der Version 4 (Sigv4) für den AWS AppSync Endpunkt. Weitere Informationen zum Signiervorgang finden Sie unter Aufgabe 4: Hinzufügen der Signatur zur HTTP-Anfrage in der Allgemeine AWS-Referenz.

Die Sigv4-Signatur-HTTP-Anforderung enthält eine kanonische URL. Dies ist der AWS AppSync -GraphQL-Endpunkt, dem `/connect` angefügt wurde. Die AWS Service-Endpunkt-Region ist dieselbe Region, in der Sie die AWS AppSync API verwenden, und der Dienstname ist „appsync“. Die HTTP-Anforderung für die Signatur ist wie folgt:

```
{
  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",
  }
}
```

Beispiel

```
{
  "accept": "application/json, text/javascript",
  "content-encoding": "amz-1.0",
  "content-type": "application/json; charset=UTF-8",
  "host": "example1234567890000.appsync-api.us-east-1.amazonaws.com",
  "x-amz-date": "20200401T001010Z",
  "X-Amz-Security-Token":
 "AgEXAMPLEZ2luX2VjEAoaDmFwLXNvdXRoZWFEXAMPLEcwRQIgAh97Cljq7wOPL8KsxP3YtDuyc/9hAj8PhJ7Fvf38SgoC
+
+pEagWCveZUjKEn0zyUhBEXAMPLEjj//////////8BEXAMPLExODk2NDgyNzg1NSIMo1mWnpESWUoYw4BkKqEFSrm3DXuL8
+ZbVc4JKjDP4vUCKNR6Le9C9pZp9PsW0NoFy3vLBUdAXEXAMPLEOVG8feXfiEEA+1khgFK/
wEtwR+9zF7NaMMMse07wN2gG2tH0eKMEXAMPLEQX+sMbytQo8iepP9PZOzlZsSFb/
dP5Q8hk6YEXAMPLEYcKZsTkDAq2uKFQ8mYUVA9EtQnNRiFLEY83aKvG/tqLWNnGlSNVx7SMcfovkFDqQamm
+88y1OwwAEYK7qcoceX6Z7GGcaYuIfGpaX2MCCELeQvZ+8WxEgOnIfz7GYvsYNjLZSaRnV4G
+ILY1F0QNW64S9Nvj
+BwDg3ht2CrNvpwjVYlj9U3nmxE0UG5ne83LL5hhqMpm25kmL7enVgw2kQzmU2id4IKu0C/
WaoDRuO2F5zE63vJbxN8AYs7338+4B4HBb6BZ6OUgg96Q15RA41/
gIqxaVPxyTpDfTU5GfSLxocdYeniqqpFMtZG2n9d0u7GsQNcFkNcG3qDZm4tDo8tZbuym0a2VcF2E5hFEgXBa
+XLJCfXi/77OqAEjP0x7Qdk3B43p8KG/BaioP5RsV8zBGvH1zAgyPha2rN70/
tT13yrmPd5QYEfwzexjKrV4mWIuRg8NTHYSZJUaeyCwTom80VFUJXG
+GYTUyv5W22aBcnoRGiCiKEYTLOkgXecdKFTHmcIAejQ9Welr0a196Kq87w5KNMCkcCGFnwBNFLmfnbpNqT6rUBxxs3X5nt
aox0FtHX21eF6qIGT8j1z+l2opU+ggwUgkhUUgCH2TfqBj+MLMVVvpgqJsPKt582caFKArIFIvO
+9QupxLnEH2hz04TMTfnU6bQC6z1buVe7h
+tOLnh1YPFsLQ88anib/7TTC8k9DsBTq0ASe8R2GbSEsmO9qbbMwgEaYUhOKtGeyQsSJdhSk6XxXThrWL9EnwBCXDkICMqd
+WgtPtKOOweDlCaRs3R2qXcbNgVhleMk4IWnF8D1695AenU1LwHjOJLkCjxgNFiWAFEPH9aEXAMPLExA==",
  "Authorization": "AWS4-HMAC-SHA256 Credential=XXXXXXXXXXXXXXXXXXX/20200401/
us-east-1/appsync/aws4_request, SignedHeaders=accept;content-
encoding;content-type;host;x-amz-date;x-amz-security-token,
 Signature=83EXAMPLEbcc1fe3ee69f75cd5ebbf4cb4f150e4f99cec869f149c5EXAMPLEdc"
}
```

Header per Abfragezeichenfolge

Zunächst wird ein JSON-Objekt, das den `host` (AWS AppSync GraphQL-Endpunkt) und die anderen Autorisierungsheader enthält, in eine Zeichenfolge umgewandelt. Als Nächstes wird diese Zeichenfolge mithilfe der Base64-Kodierung codiert. Die resultierende Base64-kodierte Zeichenfolge wird der WebSocket URL als Abfrageparameter mit dem Namen hinzugefügt. `header` Die resultierende Anforderungs-URL hat das folgende Format:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJBdXRob3JpemF0aW9uIjoiZXlKcmFXUWlPaUpqYkc1eGIzQTVlVzVNSzA5UVlYSXJNVEpIV0VGTFNYQmllVTVX
```

Es ist wichtig zu beachten, dass neben dem Base64-codierten Header-Objekt auch ein leeres JSON-Objekt {} Base64-codiert und als separater Abfrageparameter in der URL enthalten ist. `payload` WebSocket

Header über **`Sec-WebSocket-Protocol`**

Ein JSON-Objekt, das die `host` und die anderen Autorisierungsheader enthält, wird in eine Zeichenfolge konvertiert und dann mit der Base64URL-Kodierung codiert. Der resultierenden Base64URL-codierten Zeichenfolge wird ein Präfix vorangestellt. `header-` Diese Zeichenfolge mit Präfix wird dann zusätzlich zum `Sec-WebSocket-Protocol` Header als neues Unterprotokoll verwendet, wenn die Verbindung mit dem `graphql-ws` Echtzeit-Endpunkt hergestellt wird. WebSocket AWS AppSync

Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Der `Sec-WebSocket-Protocol` Header enthält den folgenden Wert:

```
"sec-websocket-protocol" : ["graphql-ws", "header-
ew0KICAiYWNjZXB0IjogImFwcGxpY2F0aW9uL2pzb24sIHRleHQvamF2YXNjcmlwdCIsDQogICJjb250ZW50LWVuY29kaW5
```

Header über Standard-HTTP-Header

Bei dieser Methode werden der Host und die anderen Autorisierungsinformationen beim WebSocket Verbindungsaufbau mit dem Echtzeit-Endpunkt mithilfe von Standard-HTTP-Headern übertragen. AWS AppSync Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Die Anforderungsheader würden Folgendes beinhalten:

```
"sec-websocket-protocol" : ["graphql-ws"]
"accept": "application/json, text/javascript",
"content-encoding": "amz-1.0",
"content-type": "application/json; charset=UTF-8",
"host": "example1234567890000.appsync-api.us-east-1.amazonaws.com",
"x-amz-date": "20200401T001010Z",
"X-Amz-Security-Token":
 "AgEXAMPLEZ2luX2VjEAoaDmFwLXNvdXRoZWFEXAMPLEcwRQIgAh97Cljq7wOPL8KsxP3YtDuyc/9hAj8PhJ7Fvf38SgoC
+
+pEagWCveZUjKEn0zyUhBEXAMPLEjj//////////8BEXAMPLExODk2NDgyNzg1NSIMo1mWnpESWUoYw4BkKqEFSrm3DXuL8
+ZbVc4JKjDP4vUCKNR6Le9C9pZp9PsW0NoFy3vLBUdAXEXAMPLEOVG8feXfiEEA+1khgFK/
wEtwR+9zF7NaMMMse07wN2gG2tH0eKMEXAMPLEQX+sMbytQo8iepP9PZOzlZsSFb/
dP5Q8hk6YEXAMPLEYcKZsTkDAq2uKFQ8mYUVA9EtQnNRiFLEY83aKvG/tqLWNnGlSNVx7SMcfovkFDqQamm
+88y1OwwAEYK7qcoceX6Z7GGcaYuIfGpaX2MCCELeQvZ+8WxEgOnIfz7GYvsYNjLZSaRnV4G
+ILY1F0QNW64S9Nvj
+BwDg3ht2CrNvpwjVYlj9U3nmxE0UG5ne83LL5hhqMpm25kmL7enVgw2kQzmU2id4IKu0C/
WaoDRuO2F5zE63vJbxN8AYs7338+4B4HBb6BZ6OUgg96Q15RA41/
gIqxaVPxyTpDfTU5GfSLxocdYeniqqpFMtZG2n9d0u7GsQNcFkNcG3qDZm4tDo8tZbuym0a2VcF2E5hFEgXBa
+XLJCfXi/77OqAEjP0x7Qdk3B43p8KG/BaioP5RsV8zBGvH1zAgyPha2rN70/
tT13yrmPd5QYEfwzexjKrV4mWIuRg8NTHYSZJUaeyCwTom80VFUJXG
+GYTUyv5W22aBcnoRGiCiKEYTLOkgXecdKFTHmcIAejQ9Welr0a196Kq87w5KNMCkcCGFnwBNFLmfnbpNqT6rUBxxs3X5nt
aox0FtHX21eF6qIGT8j1z+l2opU+ggwUgkhUUgCH2TfqBj+MLMVVvpgqJsPKt582caFKArIFIvO
+9QupxLnEH2hz04TMTfnU6bQC6z1buVe7h
+tOLnh1YPFsLQ88anib/7TTC8k9DsBTq0ASe8R2GbSEsmO9qbbMwgEaYUhOKtGeyQsSJdhSk6XxXThrWL9EnwBCXDkICMqd
+WgtPtKOOweDlCaRs3R2qXcbNgVhleMk4IWnF8D1695AenU1LwHjOJLkCjxgNFiWAFEPH9aEXAMPLExA==",
"Authorization": "AWS4-HMAC-SHA256 Credential=XXXXXXXXXXXXXXXXXXX/20200401/
us-east-1/appsync/aws4_request, SignedHeaders=accept;content-
encoding;content-type;host;x-amz-date;x-amz-security-token,
 Signature=83EXAMPLEbcc1fe3ee69f75cd5ebbf4cb4f150e4f99cec869f149c5EXAMPLEdc"
```

Um die Anfrage mit einer benutzerdefinierten Domain zu signieren:

```
{
  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",
  }
```

```
}
```

Beispiel

```
{
  "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"
}
```

URL mit Abfragezeichenfolge anfordern

```
wss://api.example.com/graphql?
header=eyEXAMPLEHQiOiJhcHBsaWNhdGlvbi9qc29uLCB0ZXh0L2phdmFEXAMPLEQiLCJjb250ZW50LWVuY29kaW5nIjoE
```

> **Note**
>
> Eine WebSocket Verbindung kann mehrere Abonnements haben (auch mit unterschiedlichen Authentifizierungsmodi). Eine Möglichkeit, dies zu implementieren, besteht darin, eine WebSocket Verbindung für das erste Abonnement herzustellen und sie dann zu schließen, wenn das letzte Abonnement nicht registriert ist. Sie können dies optimieren, indem Sie einige Sekunden warten, bevor Sie die WebSocket Verbindung schließen, falls die App unmittelbar nach der Aufhebung der Registrierung des letzten Abonnements abonniert wird. Beispiel: Wenn eine mobile App von einem Bildschirm auf einen anderen wechselt, beendet sie beim Aushängen ein Abonnement und beim Einhängen des Ereignisses startet sie ein anderes Abonnement.

Lambda-Autorisierung

Lambda-Autorisierungsheader

Inhalt der Kopfzeile

- `"Authorization": <string>`: Der Wert, der als übergeben wird`authorizationToken`.
- `"host": <string>`: Der Host für den AWS AppSync GraphQL-Endpunkt oder Ihren benutzerdefinierten Domainnamen.

Beispiel

```
{

 "Authorization":"M0UzQzM1MkQtMkI0Ni00OTZCLUI1NkQtMUM0MTQ0QjVBRTczCkI1REEzRTIxLTk5NzItNDJENi1BQ
    "host":"example1234567890000.appsync-api.us-east-1.amazonaws.com"
}
```

Header per Abfragezeichenfolge

Zunächst wird ein JSON-Objekt, das das `host` und `Authorization` das enthält, in eine Zeichenfolge umgewandelt. Als Nächstes wird diese Zeichenfolge mithilfe der Base64-Kodierung codiert. Die resultierende Base64-kodierte Zeichenfolge wird als Abfrageparameter mit dem Namen der WebSocket URL hinzugefügt`header`, um die Verbindung mit dem Echtzeit-Endpunkt herzustellen. AWS AppSync Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJBdXRob3JpemF0aW9uIjoiZXlKcmFXUWlPaUpqYkc1eGIzQTVlVzVNSzA5UVlYSXJNVEpIV0VGTFNYQmllVTVX
```

Es ist wichtig zu beachten, dass neben dem Base64-codierten Header-Objekt auch ein leeres JSON-Objekt {} Base64-codiert und als separater Abfrageparameter in der URL enthalten ist. `payload` WebSocket

Header über **`Sec-WebSocket-Protocol`**

Ein JSON-Objekt, das das `host` und das enthält, `Authorization` wird in eine Zeichenfolge konvertiert und dann mit der Base64URL-Codierung codiert. Der resultierenden Base64URL-codierten Zeichenfolge wird ein Präfix vorangestellt. `header-` Diese Zeichenfolge mit Präfix wird dann zusätzlich zum `Sec-WebSocket-Protocol` Header als neues Unterprotokoll verwendet, wenn die Verbindung mit dem `graphql-ws` Echtzeit-Endpunkt hergestellt wird. WebSocket AWS AppSync

Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Der `Sec-WebSocket-Protocol` Header enthält den folgenden Wert:

```
"sec-websocket-protocol" : ["graphql-ws", "header-
ewogICAgImhvc3QiOiJleGFtcGxlMTIzNDU2Nzg5MDAwMC5hcHBzeW5jLWFwaS51cy1lYXN0LTEuYW1hem9uYXdzLmNvbSI
```

Header über Standard-HTTP-Header

Bei dieser Methode werden die Host- und Autorisierungsinformationen beim WebSocket Verbindungsaufbau mit dem Echtzeit-Endpunkt mithilfe von Standard-HTTP-Headern übertragen. AWS AppSync Die resultierende Anforderungs-URL hat die folgende Form:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Die Anforderungsheader würden Folgendes beinhalten:

```
"sec-websocket-protocol" : ["graphql-ws"]
"Authorization":"eyEXAMPLEiJjbG5xb3A5eW5MK09QYXIrMTJHWEFLSXBieU5WNHhsQjEXAMPLEnM2WldvPSIsImFsZy
zEE2DJH7sHOl2zxYi7f-SmEGoh2AD8emxQRYajByz-rE4Jh0QOymN2Ys-ZIkMpVBTPgu-
TMWDyOHhDUmUj2OP82yeZ3wlZAtr_gM4LzjXUXmI_K2yGjuXfXTaa1mvQEBG0mQfVd7SfwXB-
jcv4RYVi6j25qgow9Ew52ufurPqaK-3WAKG32KpV8J4-Wejq8t0c-
```

```
yA7sb8EnB551b7TU93uKRiVVK3E55Nk5ADPoam_WYE45i3s5qVAP_-InW75NUoOCGTsS8YWMfb6ecHYJ-1j-
bzA27zaT9VjctXn9byNFZmEXAMPLExw",
"host":"example1234567890000.appsync-api.us-east-1.amazonaws.com"
```

Betrieb in Echtzeit WebSocket

Nach dem Initiieren eines erfolgreichen WebSocket Handshakes mit muss der Client eine nachfolgende Nachricht senden AWS AppSync, mit der er sich AWS AppSync für verschiedene Operationen verbinden kann. Diese Nachrichten benötigen die folgenden Daten:

- `type`: Typ der Operation.
- `id`: Eine eindeutige Kennung für das Abonnement. Für diesen Zweck sollte eine UUID verwendet werden.
- `payload`: Die zugehörige Nutzlast, abhängig vom Vorgangstyp.

Das `type` Feld ist das einzige Pflichtfeld; die `payload` Felder `id` und sind optional.

Reihenfolge der Ereignisse

Um die Abonnementanfrage erfolgreich zu initiieren, einzurichten, zu registrieren und zu verarbeiten, muss der Kunde die folgende Reihenfolge einhalten:

1. Initialisierung der Verbindung (`connection_init`)
2. Bestätigung der Verbindung (`connection_ack`)
3. Registrierung des Abonnements (`start`)
4. Bestätigung des Abonnements (`start_ack`)
5. Verarbeitung des Abonnements (`data`)
6. Aufhebung der Registrierung des Abonnements (`stop`)

Nachricht über die Initialisierung der Verbindung

(Optional) Nach einem erfolgreichen Handshake kann der Client die `connection_init` Nachricht senden, um mit der Kommunikation mit dem AWS AppSync Echtzeit-Endpunkt zu beginnen. Die Nachricht ist eine Zeichenfolge, die durch die Stringifizierung des JSON-Objekts wie folgt abgerufen wurde:

```
{ "type": "connection_init" }
```

Nachricht über die Bestätigung der Verbindung

Nach dem Senden der Nachricht `connection_init` muss der Client auf die Nachricht `connection_ack` warten. Alle Nachrichten, die vor dem Empfang gesendet wurden, `connection_ack` werden ignoriert. Die Nachricht sollte wie folgt lauten:

```
{
  "type": "connection_ack",
  "payload": {
    // Time in milliseconds waiting for ka message before the client should terminate
 the WebSocket connection
    "connectionTimeoutMs": 300000
  }
}
```

Keep-Alive-Nachricht

Zusätzlich zur Verbindungsbestätigungsnachricht empfängt der Client regelmäßig Keep-Alive-Nachrichten. Wenn der Client innerhalb des Verbindungstimeouts keine Keep-Alive-Nachricht erhält, sollte der Client die Verbindung schließen. AWS AppSync sendet weiterhin diese Nachrichten und verwaltet die registrierten Abonnements, bis die Verbindung automatisch beendet wird (nach 24 Stunden). Keep-Alive-Nachrichten sind Heartbeats und müssen vom Client nicht bestätigt werden.

```
{ "type": "ka" }
```

Nachricht über die Abonnementregistrierung

Nachdem der Client eine `connection_ack` Nachricht erhalten hat, kann er Abonnementregistrierungsnachrichten an senden. AWS AppSync Bei diesem Nachrichtentyp handelt es sich um ein stringifiziertes JSON-Objekt, das die folgenden Felder enthält:

- `"id": <string>`: Die ID des Abonnements. Diese ID muss für jedes Abonnement eindeutig sein, andernfalls gibt der Server einen Fehler zurück, der darauf hinweist, dass die Abonnement-ID doppelt vorhanden ist.
- `"type": "start"`: Ein konstanter `<string>`-Parameter.
- `"payload": <0bject>`: Ein Objekt, das die für das Abonnement relevanten Informationen enthält.
  - `"data": <string>`: Ein stringifiziertes JSON-Objekt, das eine GraphQL-Abfrage und Variablen enthält.

    - `"query": <string>`: Eine GraphQL-Operation.
    - `"variables": <Object>`: Ein Objekt, das die Variablen für die Abfrage enthält.
  - `"extensions": <Object>`: Ein Objekt, das ein Autorisierungsobjekt enthält.
- `"authorization": <Object>`: Ein Objekt, das die für die Autorisierung erforderlichen Felder enthält.

Autorisierungsobjekt für die Abonnementregistrierung

Für das Autorisierungsobjekt gelten dieselben Regeln wie im Das Format der Header-Parameter basiert auf dem AWS AppSync API-Autorisierungsmodus Abschnitt. Die einzige Ausnahme ist IAM, wo sich die SigV4-Signaturinformationen geringfügig unterscheiden. Weitere Details finden Sie im IAM-Beispiel.

Beispiel mit Verwendung von Amazon Cognito-Benutzerpools:

```
{
  "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"
}
```

Beispiel mit Verwendung von 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"
}
```

Beispiel mit einem benutzerdefinierten Domainnamen:

```
{
  "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"
}
```

Die SigV4-Signatur muss nicht `/connect` an die URL angehängt werden, und die stringifizierte JSON-GraphQL-Operation ersetzt sie. `data` Im Folgenden finden Sie ein Beispiel für eine SigV4-Signaturanforderung:

```
{
  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",
  }
}
```

Bestätigungsnachricht für das Abonnement

Nach dem Senden der Abonnementstartnachricht sollte der Client warten, AWS AppSync bis die Nachricht gesendet wird`start_ack`. Die `start_ack` Nachricht weist darauf hin, dass das Abonnement erfolgreich ist.

Beispiel für eine Abonnementbestätigung:

```
{
  "type": "start_ack",
  "id": "eEXAMPLE-cf23-1234-5678-152EXAMPLE69"
}
```

Fehlermeldung

Wenn der Verbindungsaufbau oder die Abonnementregistrierung fehlschlägt oder wenn ein Abonnement vom Server aus beendet wird, sendet der Server eine Fehlermeldung an den Client. Tritt der Fehler während der Verbindungsinitialisierung auf, wird die Verbindung vom Server geschlossen.

- `"type": "error"`: Ein konstanter `<string>`-Parameter.
- `"id": <string>`: Die ID des entsprechenden registrierten Abonnements, falls relevant.
- `"payload" <Object>`: Ein Objekt, das die entsprechenden Fehlerinformationen enthält.

Beispiel:

```
{
  "type": "error",
  "payload": {
    "errors": [
      {
        "errorType": "LimitExceededError",
        "message": "Rate limit exceeded"
      }
    ]
  }
}
```

Verarbeitung von Datennachrichten

Wenn ein Client eine Mutation einreicht, AWS AppSync identifiziert er alle Abonnenten, die daran interessiert sind, und sendet eine `"type":"data"` Nachricht an jeden, der das entsprechende Abonnement `id` aus dem `"start"` Abonnementvorgang verwendet. Es wird erwartet, `id` dass der Client den Überblick über das von ihm gesendete Abonnement behält, sodass der Client, wenn er eine Datennachricht empfängt, diese dem entsprechenden Abonnement zuordnen kann.

- `"type": "data"`: Ein konstanter `<string>`-Parameter.
- `"id": <string>`: Die ID des entsprechenden registrierten Abonnements.
- `"payload" <Object>`: Ein Objekt, das die Abonnementinformationen enthält.

Beispiel:

```
{
  "type": "data",
  "id": "ee849ef0-cf23-4cb8-9fcb-152ae4fd1e69",
  "payload": {
    "data": {
      "onCreateMessage": {
        "__typename": "Message",
        "message": "test"
      }
    }
  }
}
```

Nachricht über die Aufhebung der Abonnementregistrierung

Wenn die App die Abonnementereignisse nicht mehr abhören möchte, sollte der Client eine Nachricht mit dem folgenden stringifizierten JSON-Objekt senden:

- `"type": "stop"`: Ein konstanter `<string>`-Parameter.
- `"id": <string>`: Die ID des Abonnements, für das die Registrierung aufgehoben werden soll.

Beispiel:

```
{
  "type":"stop",
  "id":"ee849ef0-cf23-4cb8-9fcb-152ae4fd1e69"
}
```

AWS AppSync sendet eine Bestätigungsnachricht mit dem folgenden stringifizierten JSON-Objekt zurück:

- `"type": "complete"`: Ein konstanter `<string>`-Parameter.
- `"id": <string>`: Die ID des nicht registrierten Abonnements.

Nachdem der Client die Bestätigungsnachricht erhalten hat, erhält er keine weiteren Nachrichten für dieses spezielle Abonnement.

Beispiel:

```
{
  "type":"complete",
  "id":"eEXAMPLE-cf23-1234-5678-152EXAMPLE69"
}
```

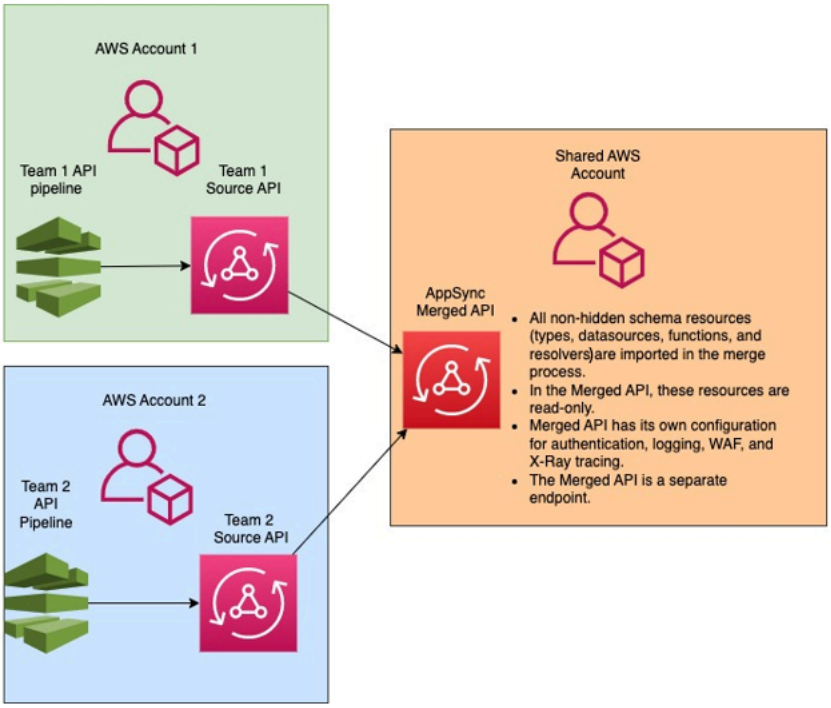
Trennen der Verbindung WebSocket

Um Datenverlust zu vermeiden, sollte der Client vor dem Trennen der Verbindung über die erforderliche Logik verfügen, um zu überprüfen, ob derzeit kein Vorgang über die WebSocket Verbindung ausgeführt wird. Vor dem Trennen der Verbindung mit dem sollten alle Abonnements abgemeldet werden. WebSocket

Zusammenführen APIs AWS AppSync

Wenn der Einsatz von GraphQL innerhalb eines Unternehmens zunimmt, können Kompromisse zwischen API ease-of-use - und API-Entwicklungsgeschwindigkeit entstehen. Einerseits setzen AWS AppSync Unternehmen GraphQL ein, um die Anwendungsentwicklung zu vereinfachen, indem sie Entwicklern eine flexible API zur Verfügung stellen, mit der sie mit einem einzigen Netzwerkaufruf sicher auf Daten aus einer oder mehreren Datendomänen zugreifen, diese bearbeiten und kombinieren können. Andererseits möchten Teams innerhalb einer Organisation, die für die verschiedenen Datendomänen verantwortlich sind, die zu einem einzigen GraphQL-API-Endpunkt zusammengefasst sind, möglicherweise die Möglichkeit haben, API-Updates unabhängig voneinander zu erstellen, zu verwalten und bereitzustellen, um ihre Entwicklungsgeschwindigkeit zu erhöhen.

Um dieses Problem zu lösen, ermöglicht die AWS AppSync APIs Merged-Funktion es Teams aus verschiedenen Datendomänen, unabhängig voneinander zu erstellen und bereitzustellen AWS AppSync APIs (z. B. GraphQL-Schemas, Resolver, Datenquellen und Funktionen), die dann zu einer einzigen, zusammengeführten API kombiniert werden können. Dies gibt Unternehmen die Möglichkeit, eine einfach zu verwendende, domänenübergreifende API zu verwalten, und bietet den verschiedenen Teams, die an dieser API beteiligt sind, die Möglichkeit, API-Updates schnell und unabhängig voneinander vorzunehmen.



Mithilfe von APIs Merged können Unternehmen die Ressourcen mehrerer, unabhängiger Quellen AWS AppSync APIs in einen einzigen AWS AppSync zusammengeführten API-Endpunkt importieren. Zu diesem Zweck AWS AppSync können Sie eine Liste mit Quellquellen erstellen und anschließend alle mit der AWS AppSync Quelle verknüpften Metadaten APIs, APIs einschließlich Schema, Typen, Datenquellen, Resolver und Funktionen, zu einer neuen AWS AppSync zusammengeführten API zusammenführen.

Bei Zusammenführungen besteht die Möglichkeit, dass aufgrund von Inkonsistenzen im Inhalt der Quell-API-Daten, wie z. B. Typbenennungskonflikte beim Kombinieren mehrerer Schemas, ein Zusammenführungskonflikt auftritt. Für einfache Anwendungsfälle, in denen keine Definitionen in der Quelle in APIs Konflikt stehen, müssen die Quell-API-Schemas nicht geändert werden. Die daraus resultierende Merged API importiert einfach alle Typen, Resolver, Datenquellen und Funktionen aus der Originalquelle. AWS AppSync APIs In komplexen Anwendungsfällen, in denen Konflikte auftreten, users/teams müssen sie die Konflikte auf verschiedene Weise lösen. AWS AppSync stellt Benutzern mehrere Tools und Beispiele zur Verfügung, mit denen Zusammenführungskonflikte reduziert werden können.

Nachfolgende Zusammenführungen, die in AWS AppSync konfiguriert sind, übertragen die in der Quelle vorgenommenen Änderungen APIs an die zugehörige zusammengeführte API.

Zusammengeführt APIs und Verbund

In der GraphQL-Community gibt es viele Lösungen und Muster, um GraphQL-Schemas zu kombinieren und die Teamzusammenarbeit über einen gemeinsamen Graphen zu ermöglichen. AWS AppSync Merged APIs verwendet bei der Schemakomposition einen Build-Time-Ansatz, bei dem die Quelldateien in einer separaten, zusammengeführten API zusammengefasst APIs werden. Ein alternativer Ansatz besteht darin, einen Runtime-Router über mehrere Quell APIs - oder Untergraphen zu verteilen. Bei diesem Ansatz empfängt der Router eine Anfrage, verweist auf ein kombiniertes Schema, das er als Metadaten verwaltet, erstellt einen Anforderungsplan und verteilt dann die Anforderungselemente auf die zugrunde liegenden Subgraphen/Server. In der folgenden Tabelle wird der Build-Time-Ansatz der AWS AppSync zusammengeführten API mit routerbasierten Laufzeitansätzen für die GraphQL-Schemakomposition verglichen:

Funktion	AppSync Zusammengeführte API	Router-basierte Lösungen
Unterdiagramme werden unabhängig voneinander verwaltet	Ja	Ja
Untergraphen können unabhängig voneinander adressiert werden	Ja	Ja
Automatisierte Schemazusammenstellung	Ja	Ja
Automatisierte Konflikterkennung	Ja	Ja
Konfliktlösung über Schemadirektiven	Ja	Ja
Unterstützte Subgraph-Server	AWS AppSync*	Variiert
Komplexität des Netzwerks	Eine einzige, zusammengeführte API bedeutet keine zusätzlichen Netzwerk-Hops.	Eine mehrschichtige Architektur erfordert die Planung und Delegierung von Abfragen, das Parsen und Serialisieren/

		Deserialisieren von Unterabfragen sowie Referenz-Resolver in Untergraphen, um Verknüpfungen durchzuführen.
Unterstützung der Beobachtbarkeit	Integrierte Überwachung, Protokollierung und Rückverfolgung. Ein einziger, zusammengeführter API-Server bedeutet vereinfachtes Debugging.	Build-your-own Beobachtbarkeit über den Router und alle zugehörigen Subgraph-Server hinweg. Komplexes Debugging auf verteilten Systemen.
Unterstützung bei der Autorisierung	Integrierte Unterstützung für mehrere Autorisierungsmodi.	Build-your-own Autorisierungsregeln.
Kontenübergreifende Sicherheit	Integrierte Unterstützung für AWS cloudübergreifende Kontozuordnungen.	Build-your-own Sicherheitsmodell.
Unterstützung für Abonnements	Ja	Nein

* AWS AppSync Zusammengeführt APIs kann nur mit der AWS AppSync Quelle verknüpft werden APIs. Wenn Sie Unterstützung für die Schemakomposition zwischen AWS AppSync und ohne AWS AppSync Unterdiagramme benötigen, können Sie ein oder mehrere AWS AppSync GraphQL Merged APIs zu einer and/or routerbasierten Lösung verbinden. Im Referenz-Blog finden Sie beispielsweise Informationen zum Hinzufügen AWS AppSync APIs als Untergraph mithilfe einer routerbasierten Architektur mit Apollo Federation v2: Apollo GraphQL Federation with. AWS AppSync

Themen

- Konfliktlösung zusammengeführter APIs
- Schemas konfigurieren
- Autorisierungsmodi konfigurieren
- Konfiguration von Ausführungsrollen
- Konfiguration von Cross-Account-Zusammengeführt mit APIs AWS RAM
- Zusammenführen

- Zusätzliche Unterstützung für Merged APIs
- Einschränkungen der zusammengeführten APIs
- Merged erstellen APIs

Konfliktlösung zusammengeführter APIs

AWS AppSync Stellt Benutzern im Falle eines Zusammenführungskonflikts verschiedene Tools und Beispiele zur Verfügung, um die Probleme zu beheben.

Richtlinien für zusammengeführte API-Schemas

AWS AppSync hat mehrere GraphQL-Direktiven eingeführt, die verwendet werden können, um Konflikte zwischen den Quellen APIs zu reduzieren oder zu lösen:

- @canonical: Diese Direktive legt den Vorrang von Typen/Feldern mit ähnlichen Namen und Daten fest. Wenn zwei oder mehr Quellen denselben GraphQL-Typ oder dasselbe GraphQL-Feld APIs haben, APIs kann eine von ihnen ihren Typ oder ihr Feld als kanonisch annotieren, was bei der Zusammenführung priorisiert wird. Konflikte types/fields , die in anderen Quellen nicht mit dieser Direktive annotiert sind, werden beim Zusammenführen ignoriert. APIs
- @hidden: Diese Direktive kapselt bestimmte, um sie aus types/fields dem Zusammenführungsprozess zu entfernen. Teams möchten möglicherweise bestimmte Typen oder Operationen in der Quell-API entfernen oder verbergen, sodass nur interne Kunden auf bestimmte typisierte Daten zugreifen können. Wenn diese Direktive angehängt ist, werden Typen oder Felder nicht in der zusammengeführten API zusammengeführt.
- @renamed: Diese Direktive ändert die Namen von types/fields , um Namenskonflikte zu reduzieren. Es gibt Situationen, in denen verschiedene denselben Typ oder denselben Feldnamen APIs haben. Sie müssen jedoch alle im zusammengeführten Schema verfügbar sein. Eine einfache Möglichkeit, sie alle in die zusammengeführte API aufzunehmen, besteht darin, das Feld in etwas Ähnliches, aber anderes umzubenennen.

Sehen Sie sich das folgende Beispiel an, um zu zeigen, was die Utility-Schema-Direktiven bieten:

Gehen wir in diesem Beispiel davon aus, dass wir zwei Quellen zusammenführen möchten APIs. Wir erhalten zwei Schemas, mit denen Beiträge erstellt und abgerufen werden können (z. B. Kommentarbereich oder Beiträge in sozialen Netzwerken). Unter der Annahme, dass sich die Typen und Felder sehr ähnlich sind, besteht bei einer Zusammenführung ein hohes Konfliktpotenzial. Die folgenden Ausschnitte zeigen die Typen und Felder der einzelnen Schemas.

Die erste Datei mit dem Namen Source1.graphQL ist ein GraphQL-Schema, das es einem Benutzer ermöglicht, mithilfe der Mutation zu erstellen`Posts`. `putPost` Jede `Post` enthält einen Titel und eine ID. Die ID wird verwendet`User`, um auf die Informationen des Posters (E-Mail und Adresse) und auf die `Message` Payload (Inhalt) zu verweisen. Der `User` Typ ist mit dem @canonical -Tag annotiert.

```
# 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
}
```

Die zweite Datei namens Source2.graphQL ist ein GraphQL-Schema, das sehr ähnliche Dinge tut wie Source1.graphql. Beachten Sie jedoch, dass die Felder der einzelnen Typen unterschiedlich sind. Beim Zusammenführen dieser beiden Schemas kommt es aufgrund dieser Unterschiede zu Zusammenführungskonflikten.

Beachten Sie auch, dass Source2.graphql auch mehrere Direktiven enthält, um diese Konflikte zu reduzieren. Der `Post` Typ ist mit einem @hidden -Tag versehen, um sich während des Zusammenführungsvorgangs selbst zu verschleiern. Der `Message` Typ ist mit dem @renamed

-Tag versehen, in den der Typname geändert werden kann, falls es zu `ChatMessage` einem Namenskonflikt mit einem anderen Typ kommt. `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")
}
```

Wenn die Zusammenführung erfolgt, ergibt das Ergebnis die folgende `MergedSchema.graphql` Datei:

```
# 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
}
```

Bei der Zusammenführung sind mehrere Dinge passiert:

- Der *`User`* Typ aus Source1.graphql hatte aufgrund der @canonical -Annotation Vorrang vor dem *`User`* von Source2.graphql.
- Der *`Message`* Typ aus Source1.graphql wurde in die Zusammenführung aufgenommen. Bei dem `Message` von Source2.graphql gab es jedoch einen Namenskonflikt. Aufgrund seiner @renamed -Annotation wurde es ebenfalls in die Zusammenführung aufgenommen, jedoch mit dem alternativen Namen. `ChatMessage`
- Der `Post` Typ aus Source1.graphql war enthalten, der `Post` Typ aus Source2.graphql jedoch nicht. Normalerweise würde es bei diesem Typ einen Konflikt geben, aber da der `Post` Typ aus Source2.graphql eine @hidden -Annotation hatte, wurden seine Daten verschleiert und nicht in die Zusammenführung aufgenommen. Dies führte zu keinen Konflikten.

- Der `Query` Typ wurde aktualisiert, sodass er den Inhalt beider Dateien enthält. Eine `GetMessage` Abfrage wurde jedoch `GetChatMessage` aufgrund der Direktive in umbenannt. Dadurch wurde der Namenskonflikt zwischen den beiden Abfragen mit demselben Namen behoben.

Es kommt auch vor, dass einem widersprüchlichen Typ keine Direktiven hinzugefügt wurden. Hier beinhaltet der zusammengeführte Typ die Vereinigung aller Felder aus allen Quelldefinitionen dieses Typs. Sehen Sie sich dazu das folgende Beispiel an:

Dieses Schema, Source1.graphQL genannt, ermöglicht das Erstellen und Abrufen. `Posts` Die Konfiguration ähnelt dem vorherigen Beispiel, enthält jedoch weniger Informationen.

```
# 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
}
```

Dieses Schema mit dem Namen Source2.graphql ermöglicht das Erstellen und Abrufen `Reviews` (z. B. Filmbewertungen oder Restaurantkritiken). `Reviews`sind mit demselben ID-Wert `Post` verknüpft. Zusammen enthalten sie den Titel, die Beitrags-ID und die Nutzdatennachricht des vollständigen Bewertungsbeitrags.

Beim Zusammenführen wird es zu einem Konflikt zwischen den beiden `Post` Typen kommen. Da es keine Anmerkungen zur Lösung dieses Problems gibt, besteht das Standardverhalten darin, eine Vereinigung der widersprüchlichen Typen durchzuführen.

```
# 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
}
```

Wenn die Zusammenführung erfolgt, ergibt das Ergebnis die folgende Datei: `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
}
```

Bei der Zusammenführung sind mehrere Dinge passiert:

- Der `Mutation` Typ hatte keine Konflikte und wurde zusammengeführt.
- Die `Post` Typfelder wurden im Rahmen einer Union-Operation kombiniert. Beachten Sie, wie die Vereinigung der beiden zu einem Single `id` `title`, einem und einem Single führte`reviews`.
- Der `Review` Typ hatte keine Konflikte und wurde zusammengeführt.
- Für den `Query` Typ gab es keine Konflikte und er wurde zusammengeführt.

Verwaltung von Resolvern auf gemeinsam genutzten Typen

Stellen Sie sich im obigen Beispiel den Fall vor, dass Source1.graphql einen Unit-Resolver konfiguriert hat`Query.getPost`, der eine DynamoDB-Datenquelle mit dem Namen verwendet. `PostDatasource` Dieser Resolver gibt das und eines Typs zurück. `id` `title` `Post` Stellen Sie sich nun vor, dass Source2.graphql einen Pipeline-Resolver konfiguriert hat`Post.reviews`, auf dem zwei Funktionen ausgeführt werden. `Function1`hat eine `None` Datenquelle angehängt, um benutzerdefinierte Autorisierungsprüfungen durchzuführen. `Function2`hat eine DynamoDB-Datenquelle angehängt, um die `reviews` Tabelle abzufragen.

```
query GetPostQuery {
    getPost(id: "1") {
        id,
        title,
        reviews
    }
}
```

Wenn die obige Abfrage von einem Client für den Merged API-Endpunkt ausgeführt wird, führt der AWS AppSync Dienst zunächst den Unit-Resolver für `Query.getPost` from aus`Source1`, der DynamoDB aufruft `PostDatasource` und die Daten von DynamoDB zurückgibt. Anschließend führt er den `Post.reviews` Pipeline-Resolver aus, der `Function1` eine benutzerdefinierte Autorisierungslogik ausführt und die Bewertungen `Function2` zurückgibt, die in gefunden wurden. `id` `$context.source` Der Dienst verarbeitet die Anfrage als einen einzigen GraphQL-Lauf, und für diese einfache Anfrage ist nur ein einziges Anforderungstoken erforderlich.

Verwaltung von Lösungskonflikten bei gemeinsam genutzten Typen

Stellen Sie sich den folgenden Fall vor, `Query.getPost` in dem wir auch einen Resolver auf implementieren, um mehrere Felder gleichzeitig bereitzustellen, die über den Feld-Resolver in hinausgehen. `Source2` Source1.graphql könnte so aussehen:

```
# This snippet represents a file called Source1.graphql

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

type Query {
    getPost(id: ID!): Post
}
```

Source2.graphql könnte so aussehen:

```
# This snippet represents a file called Source2.graphql

type Post  {
  id: ID!
  content: String!
  contentHash: String!
  author: String!
}

type Query {
    getPost(id: ID!): Post
}
```

Der Versuch, diese beiden Schemas zusammenzuführen, führt zu einem Zusammenführungsfehler, da AWS AppSync Merged APIs nicht zulässt, dass mehrere Quell-Resolver an dasselbe Feld angehängt werden. Um diesen Konflikt zu lösen, können Sie ein Feldauflösungsmuster implementieren, bei dem Source2.graphql einen separaten Typ hinzufügen müsste, der die Felder, die es besitzt, vom Typ definiert. `Post` Im folgenden Beispiel fügen wir einen Typ namens hinzu, der die Inhalts- und Autorenfelder enthält*`PostInfo`*, die von Source2.graphql aufgelöst werden. Source1.graphql implementiert den an angehängten Resolver`Query.getPost`, wohingegen Source2.graphql nun einen Resolver anhängt, um sicherzustellen, dass alle Daten erfolgreich abgerufen werden können: `Post.postInfo`

```
type Post  {
  id: ID!
  postInfo: PostInfo
}
```

```
type PostInfo {
   content: String!
   contentHash: String!
   author: String!
}

type Query {
    getPost(id: ID!): Post
}
```

Die Lösung eines solchen Konflikts erfordert zwar, dass die Quell-API-Schemas neu geschrieben werden und die Clients möglicherweise ihre Abfragen ändern müssen. Der Vorteil dieses Ansatzes besteht jedoch darin, dass die Eigentümer der zusammengeführten Resolver für alle Quellteams weiterhin klar sind.

Schemas konfigurieren

Zwei Parteien sind für die Konfiguration der Schemas verantwortlich, um eine zusammengeführte API zu erstellen:

- Besitzer zusammengeführter APIs — Besitzer zusammengeführter APIs müssen die Autorisierungslogik der zusammengeführten API und erweiterte Einstellungen wie Protokollierung, Tracing, Caching und WAF-Unterstützung konfigurieren.
- Eigentümer der assoziierten Quell-API — Besitzer assoziierter APIs müssen die Schemas, Resolver und Datenquellen konfigurieren, aus denen die zusammengeführte API besteht.

Da das Schema Ihrer zusammengeführten API aus den Schemas Ihrer verknüpften Quelle erstellt wird APIs, ist es schreibgeschützt. Das bedeutet, dass Änderungen am Schema in Ihrer Quelle APIs initiiert werden müssen. In der AWS AppSync Konsole können Sie mithilfe der Dropdownliste über dem Schemafenster zwischen Ihrem zusammengeführten Schema und den einzelnen Schemas der Quelle, die in Ihrer zusammengeführten API APIs enthalten ist, wechseln.

Autorisierungsmodi konfigurieren

Zum Schutz Ihrer zusammengeführten API stehen mehrere Autorisierungsmodi zur Verfügung. Weitere Informationen zu den Autorisierungsmodi finden Sie unter Autorisierung und Authentifizierung. AWS AppSync

Die folgenden Autorisierungsmodi stehen für Merged zur Verfügung APIs:

- API-Schlüssel: Die einfachste Autorisierungsstrategie. Alle Anfragen müssen einen API-Schlüssel unter dem `x-api-key` Anforderungsheader enthalten. Abgelaufene API-Schlüssel werden 60 Tage nach dem Ablaufdatum aufbewahrt.
- AWS Identity and Access Management (IAM): Die AWS IAM-Autorisierungsstrategie autorisiert alle Anfragen, die mit Sigv4 signiert sind.
- Amazon Cognito Cognito-Benutzerpools: Autorisieren Sie Ihre Benutzer über Amazon Cognito Cognito-Benutzerpools, um eine genauere Kontrolle zu erhalten.
- AWS Lambda Authorizers: Eine serverlose Funktion, mit der Sie den Zugriff auf Ihre API mithilfe benutzerdefinierter Logik authentifizieren und autorisieren können. AWS AppSync
- OpenID Connect: Dieser Autorisierungstyp erzwingt OpenID Connect-Token (OIDC), die von einem OIDC-kompatiblen Dienst bereitgestellt werden. Ihre Anwendung kann Benutzer und Berechtigungen nutzen, die von Ihrem OIDC-Anbieter zur Kontrolle des Zugriffs definiert wurden.

Die Autorisierungsmodi einer zusammengeführten API werden vom Eigentümer der zusammengeführten API konfiguriert. Zum Zeitpunkt eines Zusammenführungsvorgangs muss die zusammengeführte API den auf einer Quell-API konfigurierten primären Autorisierungsmodus entweder als eigenen primären Autorisierungsmodus oder als sekundären Autorisierungsmodus enthalten. Andernfalls ist sie inkompatibel und der Zusammenführungsvorgang schlägt mit einem Konflikt fehl. Wenn Multi-Auth-Direktiven in der Quelle verwendet werden APIs, kann der Zusammenführungsprozess diese Direktiven automatisch mit dem vereinheitlichten Endpunkt zusammenführen. Falls der primäre Autorisierungsmodus der Quell-API nicht mit dem primären Autorisierungsmodus der zusammengeführten API übereinstimmt, werden diese Authentifizierungsdirektiven automatisch hinzugefügt, um sicherzustellen, dass der Autorisierungsmodus für die Typen in der Quell-API konsistent ist.

Konfiguration von Ausführungsrollen

Wenn Sie eine zusammengeführte API erstellen, müssen Sie eine Servicerolle definieren. Eine AWS Servicerolle ist eine AWS Identity and Access Management (IAM) -Rolle, die von AWS Diensten verwendet wird, um Aufgaben in Ihrem Namen auszuführen.

In diesem Zusammenhang ist es erforderlich, dass Ihre zusammengeführte API Resolver ausführt, die auf Daten aus den in Ihrer Quelle konfigurierten Datenquellen zugreifen. APIs Die dafür erforderliche Servicerolle ist die`mergedApiExecutionRole`, und sie muss über die `appsync:SourceGraphQL` IAM-Berechtigung expliziten Zugriff haben, um Anfragen auf der Quelle auszuführen, die in Ihrer zusammengeführten API APIs enthalten ist. Während der Ausführung einer

GraphQL-Anfrage übernimmt der AWS AppSync Dienst diese Dienstrolle und autorisiert die Rolle, die Aktion auszuführen. `appsync:SourceGraphQL`

AWS AppSync unterstützt das Zulassen oder Verweigern dieser Berechtigung für bestimmte Felder der obersten Ebene innerhalb der Anfrage, z. B. wie der IAM-Autorisierungsmodus für IAM funktioniert. APIs Für non-top-level Felder AWS AppSync müssen Sie die Berechtigung für den Quell-API-ARN selbst definieren. Um den Zugriff auf bestimmte non-top-level Felder in der Merged-API einzuschränken, empfehlen wir, eine benutzerdefinierte Logik in Ihrem Lambda zu implementieren oder die Quell-API-Felder mithilfe der @hidden -Direktive vor der zusammengeführten API auszublenden. Wenn Sie der Rolle erlauben möchten, alle Datenoperationen innerhalb einer Quell-API auszuführen, können Sie die folgende Richtlinie hinzufügen. Beachten Sie, dass der erste Ressourceneintrag den Zugriff auf alle Felder der obersten Ebene ermöglicht und der zweite Eintrag untergeordnete Resolver behandelt, die für die Quell-API-Ressource selbst autorisieren:

JSON

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

Wenn Sie den Zugriff nur auf ein bestimmtes Feld der obersten Ebene beschränken möchten, können Sie eine Richtlinie wie die folgende verwenden:

JSON

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

Sie können auch den AWS AppSync Konsolen-API-Erstellungsassistenten verwenden, um eine Servicerolle zu generieren, damit Ihre zusammengeführte API auf Ressourcen zugreifen kann APIs , die in der Quelle konfiguriert sind und sich in demselben Konto wie Ihre zusammengeführte API befinden. Falls sich Ihre Quelle APIs nicht in demselben Konto wie Ihre zusammengeführte API befindet, müssen Sie zunächst Ihre Ressourcen mithilfe von AWS Resource Access Manager (AWS RAM) gemeinsam nutzen.

Konfiguration von Cross-Account-Zusammengeführt mit APIs AWS RAM

Wenn Sie eine zusammengeführte API erstellen, können Sie optional Quellen APIs von anderen Konten zuordnen, die über AWS Resource Access Manager (AWS RAM) gemeinsam genutzt wurden. AWS RAM hilft Ihnen dabei, Ihre Ressourcen sicher zwischen AWS Konten, innerhalb Ihrer Organisation oder Organisationseinheiten (OUs) und mit IAM-Rollen und -Benutzern gemeinsam zu nutzen.

AWS AppSync integriert sich AWS RAM in, um die Konfiguration und den Zugriff auf Quellen APIs über mehrere Konten hinweg über eine einzige zusammengeführte API zu unterstützen. AWS RAM ermöglicht es Ihnen, eine gemeinsame Ressourcennutzung oder einen Container mit Ressourcen und den Berechtigungssätzen zu erstellen, die für jede Ressource gemeinsam genutzt werden. Sie können AWS AppSync APIs zu einer Ressourcenfreigabe hinzufügen in AWS RAM. AWS AppSync Stellt innerhalb einer Ressourcenfreigabe drei verschiedene Berechtigungssätze bereit, die einer AWS AppSync API im RAM zugeordnet werden können:

1. `AWSRAMPermissionAppSyncSourceApiOperationAccess`: Der Standard-Berechtigungssatz, der hinzugefügt wird, wenn eine AWS AppSync API gemeinsam genutzt wird, AWS RAM sofern keine andere Berechtigung angegeben ist. Dieser Berechtigungssatz wird für die gemeinsame Nutzung einer AWS AppSync Quell-API mit einem Besitzer einer zusammengeführten API verwendet. Dieser Berechtigungssatz umfasst die Berechtigung für

`appsync:AssociateMergedGraphqlApi` die Quell-API sowie die für den Zugriff auf die Quell-API-Ressourcen zur Laufzeit erforderlichen `appsync:SourceGraphQL` Berechtigungen.

2. `AWSRAMPermissionAppSyncMergedApiOperationAccess`: Dieser Berechtigungssatz sollte konfiguriert werden, wenn eine zusammengeführte API mit einem Eigentümer der Quell-API geteilt wird. Dieser Berechtigungssatz gibt der Quell-API die Möglichkeit, die zusammengeführte API zu konfigurieren, einschließlich der Möglichkeit, jede Quelle, die dem Zielprinzipal APIs gehört, der zusammengeführten API zuzuordnen und die Quell-API-Verknüpfungen der zusammengeführten API zu lesen und zu aktualisieren.
3. `AWSRAMPermissionAppSyncAllowSourceGraphQLAccess`: Dieser Berechtigungssatz ermöglicht die Verwendung der `appsync:SourceGraphQL` Berechtigung mit einer AWS AppSync API. Es ist für die gemeinsame Nutzung einer Quell-API mit einem Eigentümer einer zusammengeführten API vorgesehen. Im Gegensatz zum Standardberechtigungssatz für den Zugriff auf Quell-API-Operationen umfasst dieser Berechtigungssatz nur die Laufzeitberechtigung`appsync:SourceGraphQL`. Wenn sich ein Benutzer dafür entscheidet, den Zugriff auf den Vorgang der zusammengeführten API mit einem Eigentümer der Quell-API zu teilen, muss er diese Berechtigung auch von der Quell-API an den Eigentümer der zusammengeführten API weitergeben, um über den Endpunkt der zusammengeführten API Laufzeitzugriff zu erhalten.

AWS AppSync unterstützt auch vom Kunden verwaltete Berechtigungen. Wenn eine der bereitgestellten AWS verwalteten Berechtigungen nicht funktioniert, können Sie Ihre eigene vom Kunden verwaltete Berechtigung erstellen. Kundenverwaltete Berechtigungen sind verwaltete Berechtigungen, die Sie erstellen und verwalten, indem Sie genau angeben, welche Aktionen unter welchen Bedingungen mit gemeinsam genutzten Ressourcen ausgeführt werden können. AWS RAM AWS AppSync ermöglicht es Ihnen, bei der Erstellung Ihrer eigenen Berechtigungen aus den folgenden Aktionen zu wählen:

1. `appsync:AssociateSourceGraphqlApi`
2. `appsync:AssociateMergedGraphqlApi`
3. `appsync:GetSourceApiAssociation`
4. `appsync:UpdateSourceApiAssociation`
5. `appsync:StartSchemaMerge`
6. `appsync:ListTypesByAssociation`
7. `appsync:SourceGraphQL`

Sobald Sie eine Quell-API oder eine zusammengeführte API ordnungsgemäß freigegeben haben AWS RAM und, falls erforderlich, die Einladung zur gemeinsamen Nutzung von Ressourcen akzeptiert wurde, wird sie in der AWS AppSync Konsole angezeigt, wenn Sie die Quell-API-Verknüpfungen auf Ihrer zusammengeführten API erstellen oder aktualisieren. Sie können auch unabhängig von der eingestellten Berechtigung alle AWS AppSync APIs Dateien auflisten, die AWS RAM mit Ihrem Konto geteilt wurden, indem Sie den von bereitgestellten `ListGraphqlApis` Vorgang aufrufen AWS AppSync und den `OTHER_ACCOUNTS` Besitzerfilter verwenden.

> **Note**
>
> Für das Teilen über AWS RAM muss der Anrufer AWS RAM die Erlaubnis haben, die `appsync:PutResourcePolicy` Aktion auf jeder API auszuführen, die gemeinsam genutzt wird.

Zusammenführen

Zusammenführungen verwalten

Zusammengeführt APIs sollen die Teamzusammenarbeit auf einem einheitlichen AWS AppSync Endpunkt unterstützen. Teams können unabhängig voneinander ihre eigene isolierte GraphQL-Quelle APIs im Backend weiterentwickeln, während der AWS AppSync Service die Integration der Ressourcen in den einzigen zusammengeführten API-Endpunkt verwaltet, um Reibungsverluste bei der Zusammenarbeit zu reduzieren und die Entwicklungsvorlaufzeiten zu verkürzen.

Automatische Zusammenführungen

Die mit Ihrer AWS AppSync zusammengeführten API APIs verknüpfte Quelle kann so konfiguriert werden, dass sie automatisch mit der zusammengeführten API zusammengeführt (Auto-Merge) wird, nachdem Änderungen an der Quell-API vorgenommen wurden. Dadurch wird sichergestellt, dass die Änderungen aus der Quell-API immer im Hintergrund an den Endpunkt der zusammengeführten API weitergegeben werden. Jede Änderung am Quell-API-Schema wird in der zusammengeführten API aktualisiert, sofern sie nicht zu einem Zusammenführungskonflikt mit einer vorhandenen Definition in der zusammengeführten API führt. Wenn das Update in der Quell-API einen Resolver, eine Datenquelle oder eine Funktion aktualisiert, wird auch die importierte Ressource aktualisiert. Wenn ein neuer Konflikt eingeführt wird, der nicht automatisch gelöst werden kann (automatisch gelöst), wird die Aktualisierung des Schemas der zusammengeführten API aufgrund eines nicht unterstützten Konflikts während des Zusammenführungsvorgangs abgelehnt. Die Fehlermeldung

ist in der Konsole für jede Quell-API-Zuordnung verfügbar, die den Status hat. `MERGE_FAILED` Sie können die Fehlermeldung auch überprüfen, indem Sie den `GetSourceApiAssociation` Vorgang für eine bestimmte Quell-API-Zuordnung mit dem AWS SDK oder der AWS CLI wie folgt aufrufen:

```
aws appsync get-source-api-association --merged-api-identifier <Merged API ARN> --
association-id <SourceApiAssociation id>
```

Dies führt zu einem Ergebnis im folgenden Format:

```
{
    "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."
    }
}
```

Manuelle Zusammenführungen

Die Standardeinstellung für eine Quell-API ist eine manuelle Zusammenführung. Um alle Änderungen zusammenzuführen, die APIs seit der letzten Aktualisierung der zusammengeführten API an der Quelle vorgenommen wurden, kann der Eigentümer der Quell-API eine manuelle Zusammenführung von der AWS AppSync Konsole aus oder über den im AWS SDK und der AWS CLI verfügbaren `StartSchemaMerge` Vorgang aufrufen.

Zusätzliche Unterstützung für Merged APIs

Konfiguration von Abonnements

Im Gegensatz zu routerbasierten Ansätzen zur GraphQL-Schemakomposition bietet Merged AWS AppSync integrierte APIs Unterstützung für GraphQL-Abonnements. Alle Abonnementvorgänge,

die in Ihrer zugehörigen Quelle definiert sind, APIs werden automatisch zusammengeführt und funktionieren in Ihrer zusammengeführten API ohne Änderung. Weitere Informationen darüber, wie Abonnements über serverlose WebSockets Verbindungen AWS AppSync unterstützt werden, finden Sie unter Echtzeitdaten.

Beobachtbarkeit konfigurieren

AWS AppSync APIs Merged bietet integrierte Protokollierung, Überwachung und Metriken über Amazon CloudWatch. AWS AppSync bietet auch integrierte Unterstützung für die Rückverfolgung über AWS X-Ray.

Konfiguration benutzerdefinierter Domänen

AWS AppSync APIs Merged bietet integrierte Unterstützung für die Verwendung benutzerdefinierter Domains mit den GraphQL- und Echtzeit-Endpunkten Ihrer zusammengeführten API.

Caching konfigurieren

AWS AppSync APIs Merged bietet integrierte Unterstützung für das optionale Zwischenspeichern von Antworten auf Anfrage- und/oder Resolverebene sowie für die Antwortkomprimierung. Weitere Informationen finden Sie unter Zwischenspeichern und Komprimierung.

Private Konfiguration APIs

AWS AppSync APIs Merged bietet integrierte Unterstützung für Private APIs , die den Zugriff auf die GraphQL- und Echtzeit-Endpunkte Ihrer zusammengeführten API auf Datenverkehr beschränkt, der von VPC-Endpunkten stammt, die Sie konfigurieren können.

Konfiguration von Firewall-Regeln

AWS AppSync APIs Merged bietet integrierte Unterstützung für AWS WAF, sodass Sie Ihre Daten schützen können, APIs indem Sie Firewall-Regeln für Webanwendungen definieren.

Konfiguration von Audit-Logs

AWS AppSync APIs Merged bietet integrierte Unterstützung für AWS CloudTrail, mit der Sie Auditprotokolle konfigurieren und verwalten können.

Einschränkungen der zusammengeführten APIs

Beachten Sie bei der APIs Entwicklung von Merged die folgenden Regeln:

1. Eine zusammengeführte API kann keine Quell-API für eine andere zusammengeführte API sein.
2. Eine Quell-API kann nicht mehr als einer zusammengeführten API zugeordnet werden.
3. Die Standardgrößenbeschränkung für ein zusammengeführtes API-Schemadokument beträgt 10 MB.
4. Die Standardanzahl von Quellen APIs , die einer zusammengeführten API zugeordnet werden können, ist 10. Sie können jedoch eine Erhöhung des Limits beantragen, wenn Sie mehr als 10 Quellen APIs in Ihrer zusammengeführten API benötigen.

Merged erstellen APIs

Um eine zusammengeführte API in der Konsole zu erstellen

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AWS AppSync Konsole.
   - Wählen Sie im Dashboard Create API aus.
2. Wählen Sie „Zusammengeführte API“ und anschließend „Weiter“.
3. Geben Sie auf der Seite „API-Details angeben“ die folgenden Informationen ein:
   a. Geben Sie unter API-Details die folgenden Informationen ein:
      i. Geben Sie den API-Namen Ihrer zusammengeführten API an. Dieses Feld ist eine Möglichkeit, Ihre GraphQL-API zu kennzeichnen, um sie bequem von anderen APIs GraphQL zu unterscheiden.
      ii. Geben Sie die Kontaktdaten an. Dieses Feld ist optional und fügt der GraphQL-API einen Namen oder eine Gruppe hinzu. Es ist nicht mit anderen Ressourcen verknüpft oder wird von diesen generiert und funktioniert ähnlich wie das API-Namensfeld.
   b. Unter Servicerolle müssen Sie Ihrer zusammengeführten API eine IAM-Ausführungsrolle zuordnen, damit Sie Ihre Ressourcen zur Laufzeit sicher importieren und verwenden AWS AppSync können. Sie können wählen, ob Sie eine neue Servicerolle erstellen und verwenden möchten, sodass Sie angeben können, welche Richtlinien und Ressourcen verwendet AWS AppSync werden sollen. Sie können auch eine vorhandene IAM-Rolle importieren, indem Sie „Bestehende Servicerolle verwenden“ und dann die Rolle aus der Dropdownliste auswählen.
   c. Unter Private API-Konfiguration können Sie festlegen, ob private API-Funktionen aktiviert werden sollen. Beachten Sie, dass diese Auswahl nach der Erstellung der

zusammengeführten API nicht geändert werden kann. Weitere Informationen zu Private APIs finden Sie unter AWS AppSync Private verwenden APIs.

Wählen Sie Weiter, wenn Sie fertig sind.

4. Als Nächstes müssen Sie das GraphQL hinzufügen APIs , das als Grundlage für Ihre zusammengeführte API verwendet wird. Geben Sie auf der APIs Seite Quelle auswählen die folgenden Informationen ein:

   a. Wählen Sie in der Tabelle APIs Aus Ihrem AWS Konto die Option Quelle hinzufügen aus APIs. In der Liste von GraphQL APIs enthält jeder Eintrag die folgenden Daten:

      i. Name: Das API-Namensfeld der GraphQL-API.

      ii. API-ID: Der eindeutige ID-Wert der GraphQL-API.

      iii. Primärer Authentifizierungsmodus: Der Standardautorisierungsmodus für die GraphQL-API. Weitere Informationen zu den Autorisierungsmodi finden Sie unter Autorisierung und Authentifizierung. AWS AppSync

      iv. Zusätzlicher Authentifizierungsmodus: Die sekundären Autorisierungsmodi, die in der GraphQL-API konfiguriert wurden.

      v. Wählen Sie den aus APIs , den Sie in der zusammengeführten API verwenden möchten, indem Sie das Kontrollkästchen neben dem Feld Name der API aktivieren. Wählen Sie anschließend Quelle hinzufügen APIs. Das ausgewählte GraphQL APIs wird in der Tabelle APIs von Ihren AWS Konten angezeigt.

   b. Wählen Sie in der Tabelle „APIs Von anderen AWS Konten“ die Option Quelle APIs hinzufügen aus. Die GraphQL APIs in dieser Liste stammen von anderen Konten, die ihre Ressourcen über AWS Resource Access Manager (AWS RAM) mit Ihren teilen. Das Verfahren zur Auswahl von GraphQL APIs in dieser Tabelle ist derselbe wie im vorherigen Abschnitt. Weitere Informationen zur gemeinsamen Nutzung von Ressourcen finden Sie AWS RAM unter Was ist AWS Resource Access Manager? .

      Wählen Sie Weiter, wenn Sie fertig sind.

   c. Fügen Sie Ihren primären Authentifizierungsmodus hinzu. Weitere Informationen finden Sie unter Autorisierung und Authentifizierung. Wählen Sie Weiter aus.

   d. Überprüfe deine Eingaben und wähle dann Create API.

GraphQL APIs mit RDS-Introspektion erstellen

AWS AppSync Das Introspection-Hilfsprogramm kann Modelle aus Datenbanktabellen erkennen und GraphQL-Typen vorschlagen. Der Create API Wizard der AWS AppSync Konsole kann sofort eine API aus einer Aurora MySQL- oder PostgreSQL-Datenbank generieren. Er erstellt automatisch Typen und JavaScript Resolver zum Lesen und Schreiben von Daten.

AWS AppSync bietet direkte Integration mit Amazon Aurora Aurora-Datenbanken über die Amazon RDS Data API. Anstatt eine persistente Datenbankverbindung zu benötigen, bietet die Amazon RDS Data API einen sicheren HTTP-Endpunkt, AWS AppSync mit dem eine Verbindung hergestellt wird, um ausgeführt zu werden SQL Aussagen. Sie können dies verwenden, um eine relationale Datenbank-API für Ihre MySQL- und PostgreSQL-Workloads auf Aurora zu erstellen.

Das Erstellen einer API für Ihre relationale Datenbank mit hat mehrere Vorteile: AWS AppSync

- Ihre Datenbank ist nicht direkt für Clients zugänglich, wodurch der Access Point von der Datenbank selbst entkoppelt wird.
- Sie können speziell APIs auf die Bedürfnisse verschiedener Anwendungen zugeschnittene Lösungen erstellen, sodass keine benutzerdefinierte Geschäftslogik in den Frontends erforderlich ist. Dies entspricht dem (BFF- Backend-For-Frontend) Muster.
- Autorisierung und Zugriffskontrolle können auf der AWS AppSync Ebene implementiert werden, wobei verschiedene Autorisierungsmodi zur Zugriffskontrolle verwendet werden. Für die Verbindung mit der Datenbank sind keine zusätzlichen Rechenressourcen erforderlich, z. B. das Hosten eines Webservers oder Proxyverbindungen.
- Echtzeitfunktionen können über Abonnements hinzugefügt werden, wobei Datenmutationen AppSync automatisch an die verbundenen Clients übertragen werden.
- Clients können über HTTPS eine Verbindung zur API herstellen, indem sie gemeinsame Ports wie 443 verwenden.

AWS AppSync macht das Erstellen APIs aus vorhandenen relationalen Datenbanken einfach. Das Introspektionsprogramm kann Modelle aus Datenbanktabellen erkennen und GraphQL-Typen vorschlagen. Der Create API Wizard der AWS AppSync Konsole kann sofort eine API aus einer Aurora MySQL- oder PostgreSQL-Datenbank generieren. Er erstellt automatisch Typen und JavaScript Resolver zum Lesen und Schreiben von Daten.

AWS AppSync bietet integrierte JavaScript Hilfsprogramme, um das Schreiben von SQL-Anweisungen in Resolvern zu vereinfachen. Sie können die AWS AppSync`sql` Tag-Vorlagen für

statische Anweisungen mit dynamischen Werten oder die `rds` Modul-Dienstprogramme verwenden, um Anweisungen programmgesteuert zu erstellen. Weitere Informationen finden Sie in der Resolver-Funktionsreferenz für RDS-Datenquellen und integrierte Module.

Verwenden der Introspektionsfunktion (Konsole)

Ein ausführliches Tutorial und eine Anleitung für die ersten Schritte finden Sie unter Tutorial: Aurora PostgreSQL Serverless with Data API.

Mit der AWS AppSync Konsole können Sie in nur wenigen Minuten eine AWS AppSync GraphQL-API aus Ihrer vorhandenen Aurora-Datenbank erstellen, die mit der Daten-API konfiguriert ist. Dadurch wird schnell ein Betriebsschema generiert, das auf Ihrer Datenbankkonfiguration basiert. Sie können die API unverändert verwenden oder darauf aufbauen, um Funktionen hinzuzufügen.

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.

   - Wählen Sie im Dashboard Create API (API erstellen) aus.

2. Wählen Sie unter API-Optionen GraphQL APIs, Mit einem Amazon Aurora Aurora-Cluster beginnen und dann Weiter aus.

   a. Geben Sie einen API-Namen ein. Dieser wird als ID für die API in der Konsole verwendet.

   b. Für Kontaktinformationen können Sie eine Kontaktstelle angeben, um einen Manager für die API zu identifizieren. Dies ist ein optionales Feld.

   c. Unter Private API-Konfiguration können Sie private API-Funktionen aktivieren. Auf eine private API kann nur von einem konfigurierten VPC-Endpunkt (VPCE) aus zugegriffen werden. Weitere Informationen finden Sie unter Privat. APIs

      Wir empfehlen, diese Funktion für dieses Beispiel nicht zu aktivieren. Wählen Sie „Weiter“, nachdem Sie Ihre Eingaben überprüft haben.

3. Wählen Sie auf der Datenbankseite die Option Datenbank auswählen aus.

   a. Sie müssen Ihre Datenbank aus Ihrem Cluster auswählen. Der erste Schritt besteht darin, die Region auszuwählen, in der Ihr Cluster existiert.

   b. Wählen Sie den Aurora-Cluster aus der Drop-down-Liste aus. Beachten Sie, dass Sie eine entsprechende Daten-API erstellt und aktiviert haben müssen, bevor Sie die Ressource verwenden können.

   c. Als Nächstes müssen Sie dem Dienst die Anmeldeinformationen für Ihre Datenbank hinzufügen. Dies erfolgt hauptsächlich mit AWS Secrets Manager. Wählen Sie die Region,

in der Ihr Geheimnis existiert. Weitere Informationen zum Abrufen geheimer Informationen finden Sie unter Geheimnisse finden oder Geheimnisse abrufen.

d. Fügen Sie Ihr Geheimnis aus der Drop-down-Liste hinzu. Beachten Sie, dass der Benutzer über Leseberechtigungen für Ihre Datenbank verfügen muss.

4. Wählen Sie Importieren aus.

AWS AppSync beginnt mit der Introspektion Ihrer Datenbank und entdeckt Tabellen, Spalten, Primärschlüssel und Indizes. Es prüft, ob die erkannten Tabellen in einer GraphQL-API unterstützt werden können. Beachten Sie, dass Tabellen zur Unterstützung der Erstellung neuer Zeilen einen Primärschlüssel benötigen, der mehrere Spalten verwenden kann. AWS AppSync ordnet Tabellenspalten wie folgt Typfeldern zu:

Datentyp	Feldtyp
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	Boolesch

BOOLEAN	Boolesch
BIT	Int
TINYINT	Int
SMALLINT	Int
MEDIUMINT	Int
INT	Int
INTEGER	Int
BIGINT	Int
JAHR	Int
FLOAT	Gleitkommazahl
DOUBLE	Gleitkommazahl
DECIMAL	Gleitkommazahl
DEC	Gleitkommazahl
NUMERIC	Gleitkommazahl
DATUM	AWSDate
TIMESTAMP (ZEITSTEMPEL)	String
DATETIME	String
TIME	AWSTime
JSON	AWSJson
ENUM	ENUM

5. Sobald die Tabellenerkennung abgeschlossen ist, wird der Datenbankbereich mit Ihren Informationen gefüllt. Im neuen Abschnitt Datenbanktabellen sind die Daten aus der Tabelle möglicherweise bereits gefüllt und in einen Typ für Ihr Schema konvertiert. Wenn Sie einige der

erforderlichen Daten nicht sehen, können Sie nach ihnen suchen, indem Sie Tabellen hinzufügen auswählen, im angezeigten Modal auf die Kontrollkästchen für diese Typen klicken und dann Hinzufügen auswählen.

Um einen Typ aus dem Abschnitt Datenbanktabellen zu entfernen, klicken Sie auf das Kontrollkästchen neben dem Typ, den Sie entfernen möchten, und wählen Sie dann Entfernen. Die entfernten Typen werden im Modal Tabellen hinzufügen platziert, falls Sie sie später wieder hinzufügen möchten.

Beachten Sie, dass die Tabellennamen als Typnamen AWS AppSync verwendet werden, aber Sie können sie umbenennen, z. B. indem Sie einen Plural-Tabellennamen wie *`movies`* den Typnamen *`Movie`* ändern. Um einen Typ im Abschnitt Datenbanktabellen umzubenennen, klicken Sie auf das Kontrollkästchen des Typs, den Sie umbenennen möchten, und klicken Sie dann auf das Stiftsymbol in der Spalte Typname.

Um eine Vorschau des Schemainhalts auf der Grundlage Ihrer Auswahl anzuzeigen, wählen Sie Schemavorschau. Beachten Sie, dass dieses Schema nicht leer sein darf. Sie müssen also mindestens eine Tabelle in einen Typ konvertieren lassen. Außerdem darf dieses Schema eine Größe von 1 MB nicht überschreiten.

- Wählen Sie unter Servicerolle aus, ob Sie eine neue Servicerolle speziell für diesen Import erstellen oder eine vorhandene Rolle verwenden möchten.

6. Wählen Sie Weiter.
7. Wählen Sie als Nächstes aus, ob Sie eine schreibgeschützte API (nur Abfragen) oder eine API zum Lesen und Schreiben von Daten (mit Abfragen und Mutationen) erstellen möchten. Letzteres unterstützt auch Echtzeitabonnements, die durch Mutationen ausgelöst werden.
8. Wählen Sie Weiter.
9. Überprüfen Sie Ihre Auswahl und wählen Sie dann Create API. AWS AppSync erstellt die API und hängt Resolver an Abfragen und Mutationen an. Die generierte API ist voll funktionsfähig und kann nach Bedarf erweitert werden.

Verwendung der Introspektionsfunktion (API)

Sie können die `StartDataSourceIntrospection` Introspektion-API verwenden, um Modelle in Ihrer Datenbank programmgesteuert zu erkennen. Weitere Informationen zu diesem Befehl finden Sie unter Verwenden der API. `StartDataSourceIntrospection`

Geben Sie zur Verwendung `StartDataSourceIntrospection` den Amazon-Ressourcennamen (ARN) Ihres Aurora-Clusters, den Datenbanknamen und den AWS Secrets Manager geheimen ARN an. Der Befehl startet den Introspektionsprozess. Sie können die Ergebnisse mit dem `GetDataSourceIntrospection` Befehl abrufen. Sie können angeben, ob der Befehl die SDL-Zeichenfolge (Storage Definition Language) für die erkannten Modelle zurückgeben soll. Dies ist nützlich, um eine SDL-Schemadefinition direkt aus den erkannten Modellen zu generieren.

Wenn Sie beispielsweise die folgende DDL-Anweisung (Data Definition Language) für eine einfache `Todos` Tabelle haben:

```
create table if not exists public.todos
(
id serial constraint todos_pk primary key,
description text,
due timestamp,
"createdAt" timestamp default now()
);
```

Sie beginnen die Introspektion mit dem Folgenden.

```
aws appsync start-data-source-introspection \
  --rds-data-api-config resourceArn=<cluster-arn>,secretArn=<secret-
arn>,databaseName=database
```

Verwenden Sie als Nächstes den `GetDataSourceIntrospection` Befehl, um das Ergebnis abzurufen.

```
aws appsync get-data-source-introspection \
  --introspection-id a1234567-8910-abcd-efgh-identifier \
  --include-models-sdl
```

Dies gibt das folgende Ergebnis zurück.

```
{
    "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
    }
}
```

Erstellen einer Client-Anwendung mit dem Amplify-Client

Sie können mit jedem AWS AppSync GraphQL-Client eine Verbindung zu Ihrer GraphQL-API herstellen, aber wir empfehlen dringend den Amplify v6-Client. Amplify generiert nicht nur automatisch stark typisierte Clients SDKs für Ihre GraphQL-API, sondern bietet auch Unterstützung für Echtzeitdaten und erweiterte GraphQL-Abfragefunktionen in Client-Anwendungen. Für Webanwendungen kann Amplify einen JavaScript Client erstellen. Für diejenigen, die auf plattformübergreifende oder mobile Umgebungen abzielen, bietet Amplify Android, iOS und React Native. Weitere Informationen zur Generierung von Client-Code für diese Plattformen finden Sie in der Amplify-Dokumentation. Hier ist eine Anleitung, mit der Sie Ihre Reise mit einer JavaScript React-Anwendung beginnen können:

> **Note**
>
> Sie müssen sowohl npm als auch die Amazon CLI installieren und konfigurieren, bevor Sie beginnen können. Wenn Sie den Amplify v6-Client verwenden, folgen Sie dieser Anleitung.

Erste Schritte:

1. Navigieren Sie auf Ihrem lokalen Computer zum Verzeichnis Ihres Projekts. Installieren Sie die Amplify-Bibliothek mit dem folgenden Befehl:

   ```
   npm install aws-amplify
   ```

2. Laden Sie Ihre Konfigurationsdatei herunter und platzieren Sie sie in Ihrem Projektordner. Ihre Konfigurationsdatei enthält normalerweise eine `config` Variable, in der einige Einstellungen (Endpunkt, Region, Autorisierungsmodus usw.) definiert sind. Sie könnte zum Beispiel so aussehen:

   ```
   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. Importieren Sie in Ihrem Code die Amplify-Bibliothek und Ihre Konfiguration, um Amplify einzurichten:

```
import { Amplify } from 'aws-amplify';
import config from './aws-exports.js';

Amplify.configure(config);
```

   Verwenden Sie alternativ das Snippet in Ihrer API-Konfiguration, um Amplify direkt einzurichten:

```
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. Mit der Amplify-Toolchain haben Sie die Möglichkeit, Operationen auf der Grundlage Ihres Schemas automatisch zu generieren, was Ihnen den Aufwand manueller Skripterstellung erspart. Verwenden Sie im Stammverzeichnis Ihrer Anwendung den folgenden CLI-Befehl:

```
npx @aws-amplify/cli codegen add --apiId <id goes here> --region <region goes here>
```

   Dadurch wird das Schema Ihrer API heruntergeladen und standardmäßig der Client-Hilfscode in den `src/graphql` Ordner generiert. Nach jeder API-Bereitstellung können Sie den folgenden Befehl erneut ausführen, um aktualisierte GraphQL-Anweisungen und -Typen zu generieren:

```
npx @aws-amplify/cli codegen
```

5. Sie können jetzt Modelle für Android, Swift, Flutter und generieren. JavaScript DataStore Verwenden Sie den folgenden Befehl, um Ihr Schema herunterzuladen:

```
aws appsync get-introspection-schema --api-id <id goes here> --region <region goes
 here> --format SDL schema.graphql
```

Führen Sie dann den folgenden Befehl im Stammverzeichnis Ihrer Anwendung aus:

```
npx @aws-amplify/cli codegen models \
--model-schema schema.graphql \
--target [android|ios|flutter|javascript|typescript] \
--output-dir ./
```

JavaScript Resolver-Tutorials für AWS AppSync

Datenquellen und Resolver werden verwendet, AWS AppSync um GraphQL-Anfragen zu übersetzen und Informationen aus Ihren Ressourcen abzurufen. AWS AWS AppSync unterstützt die automatische Bereitstellung und Verbindungen mit bestimmten Datenquellentypen. AWS AppSync unterstützt AWS Lambda auch Amazon DynamoDB, relationale Datenbanken (Amazon Aurora Serverless), Amazon OpenSearch Service und HTTP-Endpunkte als Datenquellen. Sie können eine GraphQL-API mit Ihren vorhandenen AWS Ressourcen verwenden oder Datenquellen und Resolver von Grund auf neu erstellen. Die folgenden Abschnitte sollen einige der häufigsten GraphQL-Anwendungsfälle in Form von Tutorials erläutern.

Themen

- Erstellen einer einfachen Post-Anwendung mit DynamoDB-Resolvern JavaScript
- Verwendung von Resolvern in AWS LambdaAWS AppSync
- Verwendung lokaler Resolver in AWS AppSync
- Kombinieren von GraphQL-Resolvern in AWS AppSync
- Verwendung von Amazon OpenSearch Service-Resolvern in AWS AppSync
- Durchführen von DynamoDB-Transaktionen in AWS AppSync
- Verwenden von DynamoDB-Batchoperationen in AWS AppSync
- Verwendung von HTTP-Resolvern in AWS AppSync
- Verwenden von Aurora PostgreSQL mit Daten-API in AWS AppSync

Erstellen einer einfachen Post-Anwendung mit DynamoDB-Resolvern JavaScript

In diesem Tutorial importieren Sie Ihre Amazon DynamoDB-Tabellen in AWS AppSync und verbinden sie, um mithilfe von JavaScript Pipeline-Resolvern, die Sie in Ihrer eigenen Anwendung nutzen können, eine voll funktionsfähige GraphQL-API zu erstellen.

Sie verwenden die AWS AppSync Konsole, um Ihre Amazon DynamoDB DynamoDB-Ressourcen bereitzustellen, Ihre Resolver zu erstellen und sie mit Ihren Datenquellen zu verbinden. Sie werden auch in der Lage sein, über GraphQL-Anweisungen in Ihre Amazon DynamoDB DynamoDB-Datenbank zu lesen und in sie zu schreiben und Echtzeitdaten zu abonnieren.

Es gibt bestimmte Schritte, die abgeschlossen werden müssen, damit GraphQL-Anweisungen in Amazon DynamoDB DynamoDB-Operationen und Antworten wieder in GraphQL übersetzt werden können. In diesem Tutorial wird der Konfigurationsprozess anhand mehrerer Szenarien und Datenzugriffsmuster aus der Praxis veranschaulicht.

Erstellen Sie Ihre GraphQL-API

Um eine GraphQL-API zu erstellen in AWS AppSync

1. Öffnen Sie die AppSync Konsole und wählen Sie Create API.
2. Wählen Sie Von Grund auf neu entwerfen und dann Weiter.
3. Geben Sie Ihrer API `PostTutorialAPI` einen Namen und wählen Sie dann Weiter. Gehen Sie zur Bewertungsseite, behalten Sie für die übrigen Optionen die Standardwerte bei und wählen Sie`Create`.

Die AWS AppSync Konsole erstellt eine neue GraphQL-API für Sie. Standardmäßig wird der API-Schlüsselauthentifizierungsmodus verwendet. Sie können in der Konsole den Rest der GraphQL-API einrichten und im weiteren Verlauf dieses Tutorials abfragen.

Definition einer grundlegenden Post-API

Jetzt, wo Sie Ihre GraphQL-API haben, können Sie ein grundlegendes Schema einrichten, das das grundlegende Erstellen, Abrufen und Löschen von Post-Daten ermöglicht.

Um Daten zu Ihrem Schema hinzuzufügen

1. Wählen Sie in Ihrer API die Registerkarte Schema.
2. Wir werden ein Schema erstellen, das einen `Post` Typ und eine Operation `addPost` zum Hinzufügen und Abrufen von `Post` Objekten definiert. Ersetzen Sie im Schemabereich den Inhalt durch den folgenden Code:

```
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. Wählen Sie Save Schema (Schema speichern).

Einrichtung Ihrer Amazon DynamoDB-Tabelle

Die AWS AppSync Konsole kann Ihnen helfen, die AWS Ressourcen bereitzustellen, die Sie zum Speichern Ihrer eigenen Ressourcen in einer Amazon DynamoDB-Tabelle benötigen. In diesem Schritt erstellen Sie eine Amazon DynamoDB-Tabelle zum Speichern Ihrer Beiträge. Sie richten auch einen sekundären Index ein, den wir später verwenden werden.

So erstellen Sie Ihre Amazon DynamoDB-Tabelle

1. Wählen Sie auf der Schemaseite Create Resources aus.
2. Wählen Sie Bestehenden Typ verwenden und wählen Sie dann den `Post` Typ aus.
3. Wählen Sie im Abschnitt Zusätzliche Indizes die Option Index hinzufügen aus.
4. Benennen Sie den Index`author-index`.
5. Stellen Sie `Primary key` das Auf `author` und den `Sort` Schlüssel auf ein`None`.
6. Deaktivieren Sie GraphQL automatisch generieren. In diesem Beispiel erstellen wir den Resolver selbst.

7. Wählen Sie Create (Erstellen) aus.

Sie haben jetzt eine neue Datenquelle namens`PostTable`, die Sie unter Datenquellen auf der seitlichen Registerkarte einsehen können. Sie verwenden diese Datenquelle, um Ihre Abfragen und Mutationen mit Ihrer Amazon DynamoDB-Tabelle zu verknüpfen.

Einen AddPost-Resolver einrichten (Amazon DynamoDB) PutItem

Da Sie AWS AppSync nun die Amazon DynamoDB-Tabelle kennen, können Sie sie mit einzelnen Abfragen und Mutationen verknüpfen, indem Sie Resolver definieren. Der erste Resolver, den Sie erstellen, ist der verwendete `addPost` Pipeline-Resolver JavaScript, mit dem Sie einen Beitrag in Ihrer Amazon DynamoDB-Tabelle erstellen können. Ein Pipeline-Resolver besteht aus den folgenden Komponenten:

- Die Stelle im GraphQL-Schema, an der der Resolver angefügt werden soll. In diesem Fall richten Sie einen Resolver im Feld `createPost` im Typ `Mutation` ein. Dieser Resolver wird aufgerufen, wenn der Aufrufer Mutation aufruft. `{ addPost(...){...} }`
- Die Datenquelle für diesen Resolver. In diesem Fall möchten Sie die zuvor definierte DynamoDB-Datenquelle verwenden, damit Sie der `post-table-for-tutorial` DynamoDB-Tabelle Einträge hinzufügen können.
- Der Request-Handler. Der Request-Handler ist eine Funktion, die die eingehende Anfrage des Aufrufers verarbeitet und sie in Anweisungen AWS AppSync zur Ausführung gegen DynamoDB übersetzt.
- Der Antworthandler. Die Aufgabe des Response-Handlers besteht darin, die Antwort von DynamoDB zu verarbeiten und sie wieder in etwas zu übersetzen, das GraphQL erwartet. Dies ist nützlich, wenn sich das Format der Daten in DynamoDB vom `Post`-Typ in GraphQL unterscheidet. In diesem Fall stimmt die Form jedoch überein, daher werden die Daten direkt übergeben.

Um Ihren Resolver einzurichten

1. Wählen Sie in Ihrer API den Tab Schema.
2. Suchen Sie im Bereich Resolver nach dem `addPost` Feld unter dem `Mutation` Typ und wählen Sie dann Attach aus.
3. Wählen Sie Ihre Datenquelle und dann Erstellen aus.
4. Ersetzen Sie den Code in Ihrem Code-Editor durch diesen Codeausschnitt:

```
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. Wählen Sie Save (Speichern) aus.

> **Note**
>
> In diesem Code verwenden Sie die DynamoDB-Modul-Utils, mit denen Sie auf einfache Weise DynamoDB-Anfragen erstellen können.

AWS AppSync enthält ein Hilfsprogramm zur automatischen ID-Generierung namens`util.autoId()`, das verwendet wird, um eine ID für Ihren neuen Beitrag zu generieren. Wenn Sie keine ID angeben, generiert das Tool sie automatisch für Sie.

```
const key = { id: ctx.args.id ?? util.autoId() }
```

Weitere Informationen zu den verfügbaren Dienstprogrammen für JavaScript finden Sie unter JavaScriptLaufzeitfunktionen für Resolver und Funktionen.

Rufen Sie die API auf, um einen Beitrag hinzuzufügen

Jetzt, da der Resolver konfiguriert wurde, AWS AppSync kann er eine eingehende `addPost` Mutation in einen Amazon DynamoDB DynamoDB-Vorgang `PutItem` übersetzen. Sie können jetzt eine Mutation ausführen, um Inhalte in der Tabelle hinzuzufügen.

Um den Vorgang auszuführen

1. Wählen Sie in Ihrer API die Registerkarte Abfragen aus.

2. Fügen Sie im Bereich Abfragen die folgende Mutation hinzu:

```
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. Wählen Sie „Ausführen“ (die orangefarbene Play-Schaltfläche) und anschließend`addPost`. Die Ergebnisse des neu erstellten Beitrags sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

Die folgende Erklärung zeigt, was passiert ist:

1. AWS AppSync hat eine `addPost` Mutationsanfrage erhalten.
2. AWS AppSync führt den Request-Handler des Resolvers aus. Die `ddb.put` Funktion erstellt eine `PutItem` Anfrage, die wie folgt aussieht:

```
{
  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 verwendet diesen Wert, um eine Amazon DynamoDB `PutItem` DynamoDB-Anfrage zu generieren und auszuführen.
4. AWS AppSync nahm die Ergebnisse der `PutItem` Anfrage und konvertierte sie wieder in GraphQL-Typen.

```
{
    "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. Der Antworthandler gibt das Ergebnis sofort zurück (`return ctx.result`).
6. Das Endergebnis ist in der GraphQL-Antwort sichtbar.

Einrichtung des GetPost-Resolvers (Amazon DynamoDB) GetItem

Jetzt, da Sie der Amazon DynamoDB-Tabelle Daten hinzufügen können, müssen Sie die `getPost` Abfrage so einrichten, dass sie diese Daten aus der Tabelle abrufen kann. Hierzu richten Sie einen weiteren Resolver ein.

Um Ihren Resolver hinzuzufügen

1. Wählen Sie in Ihrer API den Tab Schema.
2. Suchen Sie im Bereich Resolver auf der rechten Seite das `getPost` Feld für den `Query` Typ und wählen Sie dann Attach aus.
3. Wählen Sie Ihre Datenquelle und dann Erstellen aus.
4. Ersetzen Sie den Code im Code-Editor durch diesen Codeausschnitt:

```
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. Speichern Sie Ihren Resolver.

> **Note**
>
> In diesem Resolver verwenden wir einen Pfeilfunktionsausdruck für den Antworthandler.

Rufen Sie die API auf, um einen Beitrag zu erhalten

Jetzt, da der Resolver eingerichtet wurde, AWS AppSync weiß er, wie eine eingehende `getPost` Anfrage in einen Amazon DynamoDB `GetItem` DynamoDB-Vorgang übersetzt wird. Sie können jetzt eine Abfrage ausführen, um den zuvor erstellten Post abzurufen.

Um Ihre Abfrage auszuführen

1. Wählen Sie in Ihrer API den Tab Abfragen aus.

2. Fügen Sie im Bereich Abfragen den folgenden Code hinzu und verwenden Sie die ID, die Sie nach der Erstellung Ihres Beitrags kopiert haben:

```
query getPost {
  getPost(id: "123") {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Wähle „Ausführen“ (die orangefarbene Play-Schaltfläche) und dann`getPost`. Die Ergebnisse des neu erstellten Beitrags sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden.
4. Der von Amazon DynamoDB abgerufene Beitrag sollte im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

Nehmen Sie alternativ das folgende Beispiel:

```
query getPost {
  getPost(id: "123") {
```

```
    id
    author
    title
  }
}
```

Wenn Ihre `getPost` Abfrage nur das`id`, und benötigt`author`, können Sie Ihre Anforderungsfunktion so ändern`title`, dass Projektionsausdrücke verwendet werden, um nur die gewünschten Attribute aus Ihrer DynamoDB-Tabelle anzugeben, um unnötige Datenübertragungen von DynamoDB zu zu zu vermeiden. AWS AppSync Die Anforderungsfunktion könnte beispielsweise wie der folgende Ausschnitt aussehen:

```
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
```

Sie können auch ein selectionSetListmit verwenden`getPost`, um Folgendes darzustellen: `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
```

Eine UpdatePost-Mutation erstellen (Amazon DynamoDB) UpdateItem

Bisher können Sie `Post` Objekte in Amazon DynamoDB erstellen und abrufen. Als Nächstes richten Sie eine neue Mutation ein, um ein Objekt zu aktualisieren. Im Vergleich zu der `addPost` Mutation, bei der alle Felder angegeben werden müssen, können Sie mit dieser Mutation nur die Felder

angeben, die Sie ändern möchten. Außerdem wurde ein neues `expectedVersion` Argument eingeführt, mit dem Sie die Version angeben können, die Sie ändern möchten. Sie richten eine Bedingung ein, die sicherstellt, dass Sie die neueste Version des Objekts ändern. Dazu verwenden Sie die `UpdateItem` Amazon DynamoDB DynamoDB-Operation.sc

Um Ihren Resolver zu aktualisieren

1. Wählen Sie in Ihrer API den Tab Schema.
2. Ändern Sie im Schemabereich den `Mutation` Typ, um eine neue `updatePost` Mutation hinzuzufügen, wie folgt:

```
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. Wählen Sie Save Schema (Schema speichern).
4. Suchen Sie im Bereich Resolver auf der rechten Seite nach dem neu erstellten `updatePost` Feld für den `Mutation` Typ und wählen Sie dann Anhängen aus. Erstellen Sie Ihren neuen Resolver mithilfe des folgenden Snippets:

```
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. Speichern Sie alle Änderungen, die Sie vorgenommen haben.

Dieser Resolver wird verwendet`ddb.update`, um eine Amazon DynamoDB `UpdateItem` DynamoDB-Anfrage zu erstellen. Anstatt den gesamten Artikel zu schreiben, bitten Sie Amazon DynamoDB lediglich, bestimmte Attribute zu aktualisieren. Dies erfolgt mithilfe von Amazon DynamoDB DynamoDB-Aktualisierungsausdrücken.

Die `ddb.update` Funktion verwendet einen Schlüssel und ein Aktualisierungsobjekt als Argumente. Anschließend überprüfen Sie die Werte der eingehenden Argumente. Wenn ein Wert auf gesetzt ist`null`, verwenden Sie die `remove` DynamoDB-Operation, um zu signalisieren, dass der Wert aus dem DynamoDB-Element entfernt werden soll.

Es gibt auch einen neuen Abschnitt. `condition` Ein Bedingungsausdruck ermöglicht es Ihnen, Amazon DynamoDB mitzuteilen AWS AppSync , ob die Anfrage erfolgreich sein soll, basierend auf dem Zustand des Objekts, das sich bereits in Amazon DynamoDB befand, bevor der Vorgang ausgeführt wird. In diesem Fall möchten Sie, dass die `UpdateItem` Anfrage nur erfolgreich ist, wenn das `version` Feld des Elements, das sich derzeit in Amazon DynamoDB befindet, genau mit dem `expectedVersion` Argument übereinstimmt. Wenn der Artikel aktualisiert wird, möchten wir den Wert von erhöhen. `version` Dies ist mit der Bedienfunktion `increment` einfach zu bewerkstelligen.

Weitere Informationen zu Bedingungsausdrücken finden Sie in der Dokumentation zu Bedingungsausdrücken.

Weitere Informationen zu der `UpdateItem` Anfrage finden Sie in der UpdateItemDokumentation und in der Dokumentation zum DynamoDB-Modul.

Weitere Informationen zum Schreiben von Aktualisierungsausdrücken finden Sie in der DynamoDB-Dokumentation UpdateExpressions.

Rufen Sie die API auf, um einen Beitrag zu aktualisieren

Versuchen wir, das `Post` Objekt mit dem neuen Resolver zu aktualisieren.

Um dein Objekt zu aktualisieren

1. Wählen Sie in Ihrer API den Tab Abfragen aus.
2. Fügen Sie im Bereich Abfragen die folgende Mutation hinzu. Außerdem müssen Sie das `id` Argument auf den Wert aktualisieren, den Sie sich zuvor notiert haben:

```
mutation updatePost {
  updatePost(
    id:123
    title: "An empty story"
    content: null
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Wähle „Ausführen“ (die orangefarbene Play-Schaltfläche) und dann „Ausführen“`updatePost`.
4. Der aktualisierte Beitrag in Amazon DynamoDB sollte im Ergebnisbereich rechts neben dem Bereich Abfragen angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

In dieser Anfrage haben Sie Amazon DynamoDB gebeten AWS AppSync , nur die `content` Felder `title` und zu aktualisieren. Alle anderen Felder wurden unverändert gelassen (mit Ausnahme der Erhöhung des `version` Felds). Sie haben das `title` Attribut auf einen neuen Wert gesetzt und das `content` Attribut aus dem Beitrag entfernt. Die Felder `author`, `url`, `ups` und `downs` blieben unverändert. Versuchen Sie erneut, die Mutationsanforderung auszuführen, während Sie die Anfrage genau so lassen, wie sie ist. Es wird eine Antwort ähnlich der folgenden angezeigt:

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

```
}
```

Die Anforderung schlägt fehl, weil der Bedingungsausdruck wie folgt ausgewertet wird: `false`

1. Als Sie die Anfrage zum ersten Mal ausgeführt haben, war der Wert des `version` Felds des Beitrags in Amazon DynamoDB1, der `expectedVersion` dem Argument entsprach. Die Anfrage war erfolgreich, was bedeutete, dass das `version` Feld in Amazon DynamoDB auf erhöht wurde. 2
2. Als Sie die Anfrage zum zweiten Mal ausgeführt haben, war der Wert des `version` Felds des Beitrags in Amazon DynamoDB2, der nicht mit dem `expectedVersion` Argument übereinstimmte.

Dieses Muster wird normalerweise als optimistische Sperre bezeichnet.

Stimmenmutationen erstellen (Amazon DynamoDB UpdateItem)

Der `Post` Typ enthält `downs` Felder, um die Aufzeichnung von positiven `ups` und negativen Stimmen zu ermöglichen. Im Moment erlaubt uns die API jedoch nichts mit ihnen zu machen. Fügen wir eine Mutation hinzu, damit wir die Beiträge positiv und negativ bewerten können.

Um deine Mutation hinzuzufügen

1. Wählen Sie in Ihrer API den Tab Schema.
2. Ändern Sie im Schemabereich den `Mutation` Typ und fügen Sie die Aufzählung hinzu, `DIRECTION` um neue Stimmenmutationen hinzuzufügen:

```
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. Wählen Sie Save Schema (Schema speichern).
4. Suchen Sie im Bereich Resolver auf der rechten Seite nach dem neu erstellten **`vote`** Feld für den **`Mutation`** Typ und wählen Sie dann Anhängen aus. Erstellen Sie einen neuen Resolver, indem Sie den Code erstellen und durch den folgenden Codeausschnitt ersetzen:

```
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. Speichern Sie alle Änderungen, die Sie vorgenommen haben.

Rufen Sie die API auf, um einen Beitrag positiv oder negativ zu bewerten

Jetzt, da die neuen Resolver eingerichtet wurden, AWS AppSync weiß, wie man einen eingehenden Vorgang `upvotePost` oder eine `downvote` Mutation in einen Amazon DynamoDB DynamoDB-Vorgang `UpdateItem` übersetzt. Sie können jetzt Mutationen ausführen, um Upvotes oder Downvotes für den zuvor erstellten Post auszuführen.

Um Ihre Mutation auszuführen

1. Wählen Sie in Ihrer API den Tab Abfragen aus.

2. Fügen Sie im Bereich Abfragen die folgende Mutation hinzu. Außerdem müssen Sie das `id` Argument auf den Wert aktualisieren, den Sie sich zuvor notiert haben:

```
mutation votePost {
  vote(id:123, direction: UP) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Wähle „Ausführen“ (die orangefarbene Play-Schaltfläche) und dann „Ausführen“`votePost`.
4. Der aktualisierte Beitrag in Amazon DynamoDB sollte im Ergebnisbereich rechts neben dem Bereich Abfragen angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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. Wählen Sie noch ein paar Mal Ausführen. Sie sollten sehen, dass die `version` Felder `ups` und bei 1 jeder Ausführung der Abfrage inkrementiert werden.
6. Ändern Sie die Abfrage so, dass sie mit einer anderen `DIRECTION` aufgerufen wird.

```
mutation votePost {
  vote(id:123, direction: DOWN) {
    id
```

```
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

7. Wählen Sie Ausführen (die orangefarbene Play-Schaltfläche) und anschließend`votePost`.

   Diesmal sollten Sie sehen, dass die `version` Felder `downs` und bei `1` jeder Ausführung der Abfrage inkrementiert werden.

Einen DeletePost-Resolver einrichten (Amazon DynamoDB) DeleteItem

Als Nächstes möchten Sie eine Mutation erstellen, um einen Beitrag zu löschen. Dazu verwenden Sie den `DeleteItem` Amazon DynamoDB DynamoDB-Vorgang.

Um Ihre Mutation hinzuzufügen

1. Wählen Sie in Ihrem Schema die Registerkarte Schema.
2. Ändern Sie im Schemabereich den `Mutation` Typ, um eine neue `deletePost` Mutation hinzuzufügen:

```
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. Dieses Mal haben Sie das `expectedVersion` Feld optional gemacht. Wählen Sie als Nächstes Schema speichern aus.
4. Suchen Sie im Bereich Resolver auf der rechten Seite nach dem neu erstellten `delete` Feld im `Mutation` Typ und wählen Sie dann Anhängen aus. Erstellen Sie mit dem folgenden Code einen neuen Resolver:

```
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**
>
> Das `expectedVersion` Argument ist ein optionales Argument. Wenn der Aufrufer ein `expectedVersion` Argument in der Anfrage festlegt, fügt der Anforderungshandler eine Bedingung hinzu, die nur dann den Erfolg der `DeleteItem` Anfrage ermöglicht, wenn

das Element bereits gelöscht wurde oder wenn das `version` Attribut des Beitrags in Amazon DynamoDB genau dem entspricht. `expectedVersion` Wenn es ausgelassen wird, wird kein Bedingungsausdruck für die `DeleteItem`-Anforderung angegeben. Es ist erfolgreich, unabhängig vom Wert von `version` oder ob das Element in Amazon DynamoDB vorhanden ist oder nicht.
Auch wenn Sie einen Artikel löschen, können Sie den gelöschten Artikel zurücksenden, sofern er nicht bereits gelöscht wurde.

Weitere Informationen zu der `DeleteItem` Anfrage finden Sie in der DeleteItemDokumentation.

Rufen Sie die API auf, um einen Beitrag zu löschen

Jetzt, da der Resolver eingerichtet wurde, AWS AppSync weiß er, wie eine eingehende `delete` Mutation in eine Amazon DynamoDB `DeleteItem` DynamoDB-Operation übersetzt werden kann. Sie können jetzt eine Mutation ausführen, um Inhalte in der Tabelle zu löschen.

Um Ihre Mutation auszuführen

1. Wählen Sie in Ihrer API den Tab Abfragen aus.
2. Fügen Sie im Bereich Abfragen die folgende Mutation hinzu. Außerdem müssen Sie das `id` Argument auf den Wert aktualisieren, den Sie sich zuvor notiert haben:

```
mutation deletePost {
  deletePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Wähle „Ausführen“ (die orangefarbene Play-Schaltfläche) und dann „Ausführen“`deletePost`.
4. Der Beitrag wurde aus Amazon DynamoDB gelöscht. Beachten Sie, dass der Wert des Elements AWS AppSync zurückgegeben wird, das aus Amazon DynamoDB gelöscht wurde und der im

Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden sollte. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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. Der Wert wird nur zurückgegeben, wenn dieser Aufruf von derjenige `deletePost` ist, der ihn tatsächlich aus Amazon DynamoDB löscht. Wählen Sie erneut Ausführen.
6. Der Aufruf ist immer noch erfolgreich, aber es wird kein Wert zurückgegeben:

```
{
  "data": {
    "deletePost": null
  }
}
```

7. Versuchen wir nun, einen Beitrag zu löschen, aber diesmal geben wir einen an`expectedValue`. Zunächst müssen Sie einen neuen Beitrag erstellen, da Sie gerade den gelöscht haben, mit dem Sie bisher gearbeitet haben.
8. Fügen Sie im Bereich Abfragen die folgende Mutation hinzu:

```
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. Wählen Sie „Ausführen“ (die orangefarbene Play-Schaltfläche) und anschließend `addPost`.
10. Die Ergebnisse des neu erstellten Beitrags sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Notieren Sie den Wert `id` des neu erstellten Objekts, da Sie ihn gleich benötigen werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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. Versuchen wir nun, den Beitrag mit einem illegalen Wert für expectedVersion zu löschen. Fügen Sie im Bereich Abfragen die folgende Mutation hinzu. Außerdem müssen Sie das `id` Argument auf den Wert aktualisieren, den Sie sich zuvor notiert haben:

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 9999
  ) {
    id
    author
```

```
    title
    content
    url
    ups
    downs
    version
  }
}
```

12. Wähle „Ausführen“ (die orangefarbene Play-Schaltfläche) und dann „Ausführen“`deletePost`. Das folgende Ergebnis wird zurückgegeben:

```
{
  "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. Die Anforderung ist fehlgeschlagen, da der Bedingungsausdruck zu `false` ausgewertet wird. Der Wert für `version` des Beitrags in Amazon DynamoDB entspricht nicht dem in den Argumenten `expectedValue` angegebenen Wert. Der aktuelle Wert des Objekts wird im Feld `data` im Abschnitt `errors` der GraphQL-Antwort zurückgegeben. Wiederholen Sie die Anforderung und korrigieren Sie dabei `expectedVersion`:

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

14. Wählen Sie Ausführen (die orangefarbene Play-Schaltfläche) und dann. `deletePost`

    Diesmal ist die Anfrage erfolgreich und der Wert, der aus Amazon DynamoDB gelöscht wurde, wird zurückgegeben:

```
{
  "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. Wählen Sie erneut Ausführen. Der Aufruf ist immer noch erfolgreich, aber diesmal wird kein Wert zurückgegeben, da der Beitrag bereits in Amazon DynamoDB gelöscht wurde.

```
{ "data": { "deletePost": null } }
```

Einen AllPost-Resolver einrichten (Amazon DynamoDB Scan)

Bisher ist die API nur nützlich, wenn Sie die einzelnen Beiträge kennen, `id` die Sie sich ansehen möchten. Fügen wir jetzt einen neuen Resolver hinzu, der alle Posts in der Tabelle zurückgibt.

Um deine Mutation hinzuzufügen

1. Wählen Sie in Ihrer API den Tab Schema.
2. Ändern Sie im Bereich Schema den `Query` Typ, um eine neue `allPost` Abfrage hinzuzufügen, wie folgt:

```
type Query {
    allPost(limit: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
}
```

3. Fügen Sie einen neuen `PaginationPosts`-Typ hinzu:

```
type PaginatedPosts {
    posts: [Post!]!
    nextToken: String
}
```

4. Wählen Sie Save Schema (Schema speichern).
5. Suchen Sie im Bereich Resolver auf der rechten Seite nach dem neu erstellten `allPost` Feld für den `Query` Typ und wählen Sie dann Anhängen aus. Erstellen Sie einen neuen Resolver mit dem folgenden Code:

```
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 };
}
```

   Der Request-Handler dieses Resolvers erwartet zwei optionale Argumente:

- `limit`- Gibt die maximale Anzahl von Elementen an, die in einem einzigen Aufruf zurückgegeben werden sollen.
- `nextToken`- Wird verwendet, um die nächsten Ergebnisse abzurufen (wir werden später zeigen, woher der Wert für `nextToken` stammt).

6. Speichern Sie alle an Ihrem Resolver vorgenommenen Änderungen.

Weitere Informationen zur `Scan` Anfrage finden Sie in der Referenzdokumentation zum Scannen.

Rufen Sie die API auf, um alle Beiträge zu scannen

Jetzt, da der Resolver eingerichtet wurde, AWS AppSync weiß er, wie eine eingehende `allPost` Anfrage in einen Amazon DynamoDB `Scan` DynamoDB-Vorgang übersetzt wird. Sie können jetzt die Tabelle scannen, um alle Posts abzurufen. Bevor Sie dies ausprobieren können, muss allerdings die Tabelle mit Daten gefüllt werden, da Sie alle Daten gelöscht haben, mit denen Sie bislang gearbeitet haben.

Um Daten hinzuzufügen und abzufragen

1. Wählen Sie in Ihrer API den Tab Abfragen aus.
2. Fügen Sie im Bereich Abfragen die folgende Mutation hinzu:

```
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. Wählen Sie Ausführen (die orangefarbene Play-Schaltfläche).
4. Scannen wir jetzt die Tabelle so, dass jeweils fünf Ergebnisse zurückgegeben werden. Fügen Sie im Bereich Abfragen die folgende Abfrage hinzu:

```
query allPost {
  allPost(limit: 5) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

5. Wählen Sie Ausführen (die orangefarbene Play-Schaltfläche) und anschließend`allPost`.

   Die ersten fünf Beiträge sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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. Sie haben fünf Ergebnisse und ein Ergebnis erhaltennextToken, das Sie verwenden können, um die nächsten Ergebnisse zu erhalten. Aktualisieren Sie die allPost-Abfrage, um das nextToken aus dem vorherigen Ergebnissatz einzuschließen:

```
query allPost {
  allPost(
    limit: 5
    nextToken: "<token>"
  ) {
    posts {
      id
      author
    }
    nextToken
  }
}
```

7. Wähle „Ausführen“ (die orangefarbene Play-Schaltfläche) und dann „Ausführen“allPost.

   Die verbleibenden vier Beiträge sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. nextTokenIn dieser Ergebnisgruppe gibt es keine, da Sie alle neun Beiträge durchgeblättert haben und keiner mehr übrig ist. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

Einen allPostsBy Author-Resolver einrichten (Amazon DynamoDB Query)

Sie können Amazon DynamoDB nicht nur nach allen Beiträgen durchsuchen, sondern auch Amazon DynamoDB abfragen, um Beiträge abzurufen, die von einem bestimmten Autor erstellt wurden. Die Amazon DynamoDB-Tabelle, die Sie zuvor erstellt haben, hat bereits einen `GlobalSecondaryIndex` Aufrufauthor-index, den Sie mit einer Amazon DynamoDB `Query` DynamoDB-Operation verwenden können, um alle Beiträge abzurufen, die von einem bestimmten Autor erstellt wurden.

Um Ihre Abfrage hinzuzufügen

1. Wählen Sie in Ihrer API den Tab Schema.
2. Ändern Sie im Bereich Schema den `Query` Typ, um eine neue `allPostsByAuthor` Abfrage hinzuzufügen, wie folgt:

```
type Query {
    allPostsByAuthor(author: String!, limit: Int, nextToken: String): PaginatedPosts!
    allPost(limit: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
}
```

   Beachten Sie, dass dabei derselbe `PaginatedPosts` Typ verwendet wird, den Sie für die `allPost` Abfrage verwendet haben.

3. Wählen Sie Save Schema (Schema speichern).
4. Suchen Sie im Bereich Resolver auf der rechten Seite nach dem neu erstellten `allPostsByAuthor` Feld für den `Query` Typ, und wählen Sie dann Anhängen aus. Erstellen Sie mit dem folgenden Codeausschnitt einen Resolver:

```
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 };
}
```

   Wie der `allPost` Resolver hat auch dieser Resolver zwei optionale Argumente:

   - `limit`- Gibt die maximale Anzahl von Elementen an, die in einem einzigen Aufruf zurückgegeben werden sollen.
   - `nextToken`- Ruft den nächsten Satz von Ergebnissen ab (der Wert für `nextToken` kann aus einem vorherigen Aufruf abgerufen werden).
5. Speichert alle an Ihrem Resolver vorgenommenen Änderungen.

Weitere Informationen zu der `Query` Anfrage finden Sie in der Query-Referenzdokumentation.

Rufen Sie die API auf, um alle Beiträge nach Autor abzufragen

Jetzt, da der Resolver eingerichtet wurde, AWS AppSync weiß er, wie man eine eingehende `allPostsByAuthor` Mutation in eine `Query` DynamoDB-Operation anhand des Index übersetzt. `author-index` Sie können nun die Tabelle abfragen, um alle Posts von einem bestimmten Autor abzurufen.

Vorher sollten wir die Tabelle jedoch mit einigen weiteren Beiträgen füllen, da bisher alle Beiträge denselben Autor haben.

Um Daten hinzuzufügen und abzufragen

1. Wählen Sie in Ihrer API den Tab Abfragen aus.
2. Fügen Sie im Bereich Abfragen die folgende Mutation hinzu:

```
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. Wählen Sie „Ausführen“ (die orangefarbene Play-Schaltfläche) und anschließend`addPost`.
4. Wir fragen jetzt die Tabelle ab, so dass alle Posts von `Nadia` zurückgegeben werden. Fügen Sie im Bereich Abfragen die folgende Abfrage hinzu:

```
query allPostsByAuthor {
  allPostsByAuthor(author: "Nadia") {
    posts {
      id
      title
    }
    nextToken
  }
}
```

5. Wählen Sie Ausführen (die orangefarbene Play-Schaltfläche) und anschließend`allPostsByAuthor`. Alle Beiträge, die von verfasst wurden, `Nadia` sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
```

```
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you know...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

6. Die Paginierung funktioniert für `Query` genauso wie für `Scan`. Suchen wir z. B. nach allen Posts von `AUTHORNAME`, wobei jeweils fünf Posts abgerufen werden.
7. Fügen Sie im Bereich Abfragen die folgende Abfrage hinzu:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    limit: 5
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

8. Wählen Sie Ausführen (die orangefarbene Play-Schaltfläche) und anschließend`allPostsByAuthor`. Alle Beiträge, die von verfasst wurden, `AUTHORNAME` sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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. Aktualisieren Sie das `nextToken`-Argument mit dem Wert, der in der vorherigen Abfrage zurückgegeben wurde:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    limit: 5
    nextToken: "<token>"
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

10. Wählen Sie „Ausführen“ (die orangefarbene Play-Schaltfläche) und anschließend. `allPostsByAuthor` Die verbleibenden Beiträge, die von verfasst wurden, `AUTHORNAME` sollten

im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:


```
{
  "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
    }
  }
}
```


Verwenden von Sets

Bis zu diesem Zeitpunkt war der `Post` Typ ein flaches Schlüssel/Wert-Objekt. Sie können mit Ihrem Resolver auch komplexe Objekte wie Sets, Listen und Maps modellieren. Aktualisieren wir jetzt den `Post`-Typ, so dass er Tags enthält. Ein Beitrag kann null oder mehr Tags haben, die in DynamoDB als String-Set gespeichert werden. Außerdem richten Sie einige Mutationen ein, um Tags hinzuzufügen und zu entfernen, sowie eine neue Abfrage zum Scannen nach Posts mit einem bestimmten Tag.

Um Ihre Daten einzurichten

1. Wählen Sie in Ihrer API den Tab Schema.

2. Ändern Sie im Bereich Schema den `Post` Typ wie folgt, um ein neues `tags` Feld hinzuzufügen:

```
type Post {
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
  tags: [String!]
}
```

3. Ändern Sie im Schemabereich den `Query` Typ, um eine neue `allPostsByTag` Abfrage hinzuzufügen, wie folgt:

```
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. Ändern Sie im Schemabereich den `Mutation` Typ, um neue `addTag` und `removeTag` Mutationen hinzuzufügen, wie folgt:

```
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. Wählen Sie Save Schema (Schema speichern).
6. Suchen Sie im Bereich Resolver auf der rechten Seite nach dem neu erstellten `allPostsByTag` Feld für den `Query` Typ und wählen Sie dann Anhängen aus. Erstellen Sie Ihren Resolver mithilfe des folgenden Snippets:

```
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. Speichern Sie alle Änderungen, die Sie an Ihrem Resolver vorgenommen haben.
8. Gehen Sie nun `addTag` mit dem folgenden Codeausschnitt genauso für das `Mutation` Feld vor:

> **Note**
>
> Obwohl die DynamoDB-Utils derzeit keine Mengenoperationen unterstützen, können Sie trotzdem mit Sets interagieren, indem Sie die Anfrage selbst erstellen.

```
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. Speichern Sie alle an Ihrem Resolver vorgenommenen Änderungen.
10. Wiederholen Sie dies noch einmal für das `Mutation` Feld `removeTag` mit dem folgenden Codeausschnitt:

```
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. Speichern Sie alle an Ihrem Resolver vorgenommenen Änderungen.

Aufrufen der API zum Arbeiten mit Tags

Nachdem Sie die Resolver eingerichtet haben, AWS AppSync weiß er, wie eingehende `addTag` `allPostsByTag` Anfragen und Anfragen in DynamoDB `UpdateItem` und Operationen übersetzt werden. `removeTag` `Scan` Wählen Sie zum Ausprobieren einen der zuvor erstellten Posts aus. Verwenden wir als Beispiel einen Post, der von `Nadia` verfasst wurde.

Um Tags zu verwenden

1. Wählen Sie in Ihrer API den Tab Abfragen aus.
2. Fügen Sie im Bereich Abfragen die folgende Abfrage hinzu:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "Nadia"
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

3. Wählen Sie Ausführen (die orangefarbene Play-Schaltfläche) und anschließend`allPostsByAuthor`.
4. Alle Beiträge von Nadia sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you known...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

5. Verwenden wir den mit dem Titel Der süßeste Hund der Welt. Nimm es auf`id`, weil du es später benutzen wirst. Versuchen wir nun, ein dog Tag hinzuzufügen.

6. Fügen Sie im Bereich Abfragen die folgende Mutation hinzu. Außerdem müssen Sie das `id`-Argument mit dem zuvor notierten Wert aktualisieren.

```
mutation addTag {
  addTag(id:10 tag: "dog") {
    id
    title
    tags
  }
}
```

7. Wählen Sie „Ausführen“ (die orangefarbene Play-Schaltfläche) und anschließend`addTag`. Der Beitrag wurde mit dem neuen Tag aktualisiert:

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

8. Sie können weitere Tags hinzufügen. Aktualisieren Sie die Mutation, um das `tag` Argument wie folgt zu ändern`puppy`:

```
mutation addTag {
  addTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

9. Wählen Sie „Ausführen“ (die orangefarbene Play-Schaltfläche) und anschließend`addTag`. Der Beitrag wurde mit dem neuen Tag aktualisiert:

```
{
  "data": {
```

```
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog",
        "puppy"
      ]
    }
  }
}
```

10. Sie können Tags auch löschen. Fügen Sie im Bereich Abfragen die folgende Mutation hinzu. Außerdem müssen Sie das `id` Argument auf den Wert aktualisieren, den Sie sich zuvor notiert haben:

```
mutation removeTag {
  removeTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

11. Wähle „Ausführen“ (die orangefarbene Play-Schaltfläche) und dann „Ausführen“`removeTag`. Der Post wird aktualisiert und das Tag `puppy` wird gelöscht.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

12. Du kannst auch nach allen Beiträgen suchen, die ein Schlagwort haben. Fügen Sie im Bereich Abfragen die folgende Abfrage hinzu:

```
query allPostsByTag {
```

```
  allPostsByTag(tag: "dog") {
    posts {
      id
      title
      tags
    }
    nextToken
  }
}
```

13. Wählen Sie Ausführen (die orangefarbene Play-Schaltfläche) und anschließend `allPostsByTag`. Alle Posts mit dem Tag dog werden folgendermaßen zurückgegeben:

```
{
  "data": {
    "allPostsByTag": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world",
          "tags": [
            "dog",
            "puppy"
          ]
        }
      ],
      "nextToken": null
    }
  }
}
```

Schlussfolgerung

In diesem Tutorial haben Sie eine API erstellt, mit der Sie `Post` Objekte in DynamoDB mithilfe AWS AppSync von GraphQL bearbeiten können.

Zum Aufräumen können Sie die AWS AppSync GraphQL-API von der Konsole löschen.

Um die mit Ihrer DynamoDB-Tabelle verknüpfte Rolle zu löschen, wählen Sie Ihre Datenquelle in der Tabelle Datenquellen aus und klicken Sie auf Bearbeiten. Notieren Sie sich den Wert der Rolle

unter Eine bestehende Rolle erstellen oder verwenden. Gehen Sie zur IAM-Konsole, um die Rolle zu löschen.

Um Ihre DynamoDB-Tabelle zu löschen, klicken Sie in der Datenquellenliste auf den Namen der Tabelle. Dadurch gelangen Sie zur DynamoDB-Konsole, in der Sie die Tabelle löschen können.

Verwendung von Resolvern in AWS LambdaAWS AppSync

Sie können AWS Lambda with verwenden AWS AppSync , um jedes GraphQL-Feld aufzulösen. Beispielsweise könnte eine GraphQL-Abfrage einen Aufruf an eine Amazon Relational Database Service (Amazon RDS) -Instance senden, und eine GraphQL-Mutation könnte in einen Amazon Kinesis Kinesis-Stream schreiben. In diesem Abschnitt zeigen wir Ihnen, wie Sie eine Lambda-Funktion schreiben, die Geschäftslogik auf der Grundlage des Aufrufs einer GraphQL-Feldoperation ausführt.

Erstellen einer Lambda-Funktion

Das folgende Beispiel zeigt eine in `Node.js` (Runtime: Node.js 18.x) geschriebene Lambda-Funktion, die als Teil einer Blogpost-Anwendung verschiedene Operationen an Blogbeiträgen ausführt. Beachten Sie, dass der Code in einem Dateinamen mit der Erweiterung.mis gespeichert werden sollte.

```
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)
  }
}
```

Diese Lambda-Funktion ruft einen Beitrag nach ID ab, fügt einen Beitrag hinzu, ruft eine Liste von Beiträgen ab und ruft verwandte Beiträge für einen bestimmten Beitrag ab.

> **Note**
>
> Die Lambda-Funktion verwendet die `switch` Anweisung on`event.field`, um zu ermitteln, welches Feld gerade aufgelöst wird.

Erstellen Sie diese Lambda-Funktion mit der AWS Management Console.

Eine Datenquelle für Lambda konfigurieren

Nachdem Sie die Lambda-Funktion erstellt haben, navigieren Sie in der AWS AppSync Konsole zu Ihrer GraphQL-API und wählen Sie dann die Registerkarte Datenquellen.

Wählen Sie Datenquelle erstellen, geben Sie einen benutzerfreundlichen Datenquellennamen ein (z. B.**`Lambda`**), und wählen Sie AWS Lambda dann als Datenquellentyp die Option Funktion aus. Wählen Sie für Region dieselbe Region wie Ihre Funktion aus. Wählen Sie für Function ARN den Amazon Resource Name (ARN) Ihrer Lambda-Funktion.

Nachdem Sie Ihre Lambda-Funktion ausgewählt haben, können Sie entweder eine neue AWS Identity and Access Management (IAM-) Rolle erstellen (für die die entsprechenden Berechtigungen AWS AppSync zugewiesen werden) oder eine vorhandene Rolle mit der folgenden Inline-Richtlinie auswählen:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:REGION:ACCOUNTNUMBER:function/
LAMBDA_FUNCTION"
        }
    ]
}
```

Sie müssen außerdem wie folgt eine Vertrauensbeziehung mit AWS AppSync der IAM-Rolle einrichten:

JSON

```
{
```

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

Erstellen Sie ein GraphQL-Schema

Nachdem die Datenquelle mit Ihrer Lambda-Funktion verbunden ist, erstellen Sie ein GraphQL-Schema.

Stellen Sie im Schema-Editor in der AWS AppSync Konsole sicher, dass Ihr Schema dem folgenden Schema entspricht:

```
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]
```

```
}
```

Resolver konfigurieren

Nachdem Sie nun eine Lambda-Datenquelle und ein gültiges GraphQL-Schema registriert haben, können Sie Ihre GraphQL-Felder mithilfe von Resolvern mit Ihrer Lambda-Datenquelle verbinden.

Sie werden einen Resolver erstellen, der die Laufzeit AWS AppSync JavaScript (`APPSYNC_JS`) verwendet und mit Ihren Lambda-Funktionen interagiert. Weitere Informationen zum Schreiben von AWS AppSync Resolvern und Funktionen mit JavaScript finden Sie unter JavaScript Laufzeitfunktionen für Resolver und Funktionen.

Weitere Informationen zu Lambda-Mapping-Vorlagen finden Sie in der JavaScript Resolver-Funktionsreferenz für Lambda.

In diesem Schritt fügen Sie der Lambda-Funktion einen Resolver für die folgenden Felder hinzu:`getPost(id:ID!): Post`, `allPosts: [Post]addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!`, und. `Post.relatedPosts: [Post]` Wählen Sie im Schema-Editor in der AWS AppSync Konsole im Bereich Resolver neben dem Feld die Option Attach aus. `getPost(id:ID!): Post` Wählen Sie Ihre Lambda-Datenquelle. Geben Sie als Nächstes den folgenden Code ein:

```
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;
}
```

Dieser Resolver-Code übergibt den Feldnamen, die Liste der Argumente und den Kontext zum Quellobjekt an die Lambda-Funktion, wenn sie diese aufruft. Wählen Sie Speichern.

Sie haben nun den ersten Resolver angehängt. Wiederholen Sie diesen Vorgang für die übrigen Felder.

Testen Sie Ihre GraphQL-API

Jetzt ist Ihre Lambda-Funktion mit GraphQL-Resolvern verbunden und Sie können einige Mutationen und Abfragen mithilfe der Konsole oder einer Client-Anwendung ausführen.

Wählen Sie auf der linken Seite der AWS AppSync Konsole Abfragen aus und fügen Sie dann den folgenden Code ein:

addPost-Mutation

```
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
    }
}
```

getPost-Abfrage

```
query GetPost {
    getPost(id: "2") {
        id
        author
        title
        content
        url
        ups
        downs
    }
}
```

allPosts-Abfrage

```
query AllPosts {
    allPosts {
        id
        author
        title
        content
        url
        ups
        downs
        relatedPosts {
            id
            title
        }
    }
}
```

Zurückkehrende Fehler

Jede gegebene Feldauflösung kann zu einem Fehler führen. Mit AWS AppSync können Sie Fehler aus den folgenden Quellen melden:

- Resolver-Antworthandler
- Lambda-Funktion

Aus dem Resolver-Response-Handler

Um absichtliche Fehler auszulösen, können Sie die `util.error` Utility-Methode verwenden. Sie benötigt ein Argument an `errorMessageerrorType`, an und einen optionalen `data` Wert. `data` ist hilfreich, um zusätzliche Daten an den Client zurückzugeben, wenn ein Fehler auftritt. Das Objekt `data` wird `errors` der endgültigen GraphQL-Antwort hinzugefügt.

Das folgende Beispiel zeigt, wie es im `Post.relatedPosts: [Post]` Resolver-Response-Handler verwendet wird.

```
// the Post.relatedPosts response handler
export function response(ctx) {
    util.error("Failed to fetch relatedPosts", "LambdaFailure", ctx.result)
    return ctx.result;
```

```
}
```

Dies löst eine GraphQL-Antwort ähnlich der Folgenden aus:

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

Dabei ist `allPosts[0].relatedPosts` gleich null, weil ein Fehler aufgetreten ist und `errorMessage`, `errorType` und `data` im Objekt `data.errors[0]` vorhanden sind.

Von der Lambda-Funktion

AWS AppSync versteht auch Fehler, die die Lambda-Funktion auslöst. Mit dem Lambda-Programmiermodell können Sie behandelte Fehler melden. Wenn die Lambda-Funktion einen Fehler ausgibt, AWS AppSync kann das aktuelle Feld nicht aufgelöst werden. Nur die von Lambda zurückgegebene Fehlermeldung ist in der Antwort enthalten. Derzeit können Sie keine überflüssigen Daten an den Client zurückgeben, indem Sie einen Fehler in der Lambda-Funktion auslösen.

> **Note**
>
> Wenn Ihre Lambda-Funktion einen unbehandelten Fehler auslöst, AWS AppSync verwendet sie die von Lambda festgelegte Fehlermeldung.

Die folgenden Lambda-Funktion löst einen Fehler aus:

```
export const handler = async (event) => {
  console.log('Received event {}', JSON.stringify(event, 3))
  throw new Error('I always fail.')
}
```

Der Fehler wird in Ihrem Antworthandler empfangen. Sie können es in der GraphQL-Antwort zurücksenden, indem Sie den Fehler mit an die Antwort anhängen. `util.appendError` Ändern Sie dazu Ihren AWS AppSync Funktionsantwort-Handler wie folgt:

```
// the lambdaInvoke response handler
export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type, result);
  }
  return result;
}
```

Dadurch wird eine GraphQL-Antwort ähnlich der folgenden ausgegeben:

```
{
```

```
  "data": {
    "allPosts": null
  },
  "errors": [
    {
      "path": [
        "allPosts"
      ],
      "data": null,
      "errorType": "Lambda:Unhandled",
      "errorInfo": null,
      "locations": [
        {
          "line": 2,
          "column": 3,
          "sourceName": null
        }
      ],
      "message": "I fail. always"
    }
  ]
}
```

Anwendungsfall für Fortgeschrittene: Batching

Die Lambda-Funktion in diesem Beispiel hat ein `relatedPosts` Feld, das eine Liste verwandter Beiträge für einen bestimmten Beitrag zurückgibt. In den Beispielabfragen gibt der `allPosts` Feldaufruf der Lambda-Funktion fünf Beiträge zurück. Da wir angegeben haben, dass wir auch `relatedPosts` für jeden zurückgegebenen Beitrag eine Auflösung durchführen möchten, wird die `relatedPosts` Feldoperation fünfmal aufgerufen.

```
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
        }
    }
}
```

Das klingt in diesem speziellen Beispiel vielleicht nicht wesentlich, aber dieses verstärkte Überabrufen kann die Anwendung schnell untergraben.

Wenn wir z. B. in derselben Abfrage erneut `relatedPosts` für die zurückgegebenen verwandten `Posts` abrufen würden, würde die Anzahl der Abrufe dramatisch ansteigen.

```
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
            }
        }
    }
}
```

In dieser relativ einfachen Abfrage AWS AppSync würde die Lambda-Funktion 1 + 5 + 25 = 31 mal aufgerufen.

Dies ist eine häufig vorkommende Herausforderung, die oft als N+1-Problem (in diesem Fall N = 5) bezeichnet wird und zu höherer Latenz und höheren Kosten für die Anwendung führen kann.

Ein Ansatz zur Lösung dieses Problems besteht darin, ähnliche Feld-Resolver-Anforderungen zu bündeln. In diesem Beispiel könnte die Lambda-Funktion, anstatt eine Liste verwandter Beiträge für

einen bestimmten Beitrag auflösen zu lassen, stattdessen eine Liste verwandter Beiträge für einen bestimmten Stapel von Beiträgen auflösen.

Um dies zu demonstrieren, aktualisieren wir den Resolver für die `relatedPosts` Batchverarbeitung.

```
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;
}
```

Der Code ändert nun den Vorgang von `Invoke` zu dem `BatchInvoke` Zeitpunkt, an dem das Problem gelöst `fieldName` wird. `relatedPosts` Aktivieren Sie jetzt die Batchverarbeitung für die Funktion im Abschnitt Batching konfigurieren. Stellen Sie die maximale Batchgröße auf ein. 5 Wählen Sie Speichern.

Mit dieser Änderung erhält die Lambda-Funktion beim Auflösen `relatedPosts` Folgendes als Eingabe:

```
[
    {
        "field": "relatedPosts",
        "source": {
            "id": 1
        }
    },
    {
        "field": "relatedPosts",
        "source": {
            "id": 2
```

```
        }
    },
    ...
]
```

Wenn in der Anfrage angegeben `BatchInvoke` ist, empfängt die Lambda-Funktion eine Liste von Anfragen und gibt eine Ergebnisliste zurück.

Insbesondere muss die Ergebnisliste der Größe und Reihenfolge der Payload-Einträge der Anfrage entsprechen, damit die Ergebnisse entsprechend abgeglichen werden AWS AppSync können.

In diesem Batching-Beispiel gibt die Lambda-Funktion eine Reihe von Ergebnissen wie folgt zurück:

```
[
    [{"id":"2","title":"Second book"}, {"id":"3","title":"Third book"}],   //
 relatedPosts for id=1
    [{"id":"3","title":"Third book"}]                                      //
 relatedPosts for id=2
]
```

Sie können Ihren Lambda-Code aktualisieren, um Batching für Folgendes zu übernehmen: `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)
  }
}
```

Rückgabe einzelner Fehler

Die vorherigen Beispiele zeigen, dass es möglich ist, einen einzelnen Fehler von der Lambda-Funktion zurückzugeben oder einen Fehler von Ihrem Antworthandler auszulösen. Bei Batch-

Aufrufen wird durch das Auslösen eines Fehlers über die Lambda-Funktion ein ganzer Batch als fehlgeschlagen gekennzeichnet. Dies kann für bestimmte Szenarien akzeptabel sein, in denen ein nicht behebbarer Fehler auftritt, z. B. eine fehlgeschlagene Verbindung zu einem Datenspeicher. In Fällen, in denen einige Elemente im Stapel erfolgreich sind und andere fehlschlagen, ist es jedoch möglich, sowohl Fehler als auch gültige Daten zurückzugeben. Da die Batch-Antwort AWS AppSync erfordert, um Elemente aufzulisten, die der ursprünglichen Größe des Stapels entsprechen, müssen Sie eine Datenstruktur definieren, die gültige Daten von fehlerhaften Daten unterscheiden kann.

Wenn beispielsweise erwartet wird, dass die Lambda-Funktion einen Stapel verwandter Beiträge zurückgibt, könnten Sie sich dafür entscheiden, eine Liste von *Response* Objekten zurückzugeben, in der jedes Objekt optionale Daten -, errorMessage - und ErrorType-Felder hat. Wenn das Feld errorMessage angezeigt wird, bedeutet dies, dass ein Fehler aufgetreten ist.

Der folgende Code zeigt, wie Sie die Lambda-Funktion aktualisieren können:

```
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)
  }
}
```

Aktualisieren Sie den `relatedPosts` Resolver-Code:

```
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
  }
}
```

Der Response-Handler sucht nun nach Fehlern, die von der Lambda-Funktion bei `Invoke` Operationen zurückgegeben wurden, sucht nach Fehlern, die für einzelne Elemente für `BatchInvoke` Operationen zurückgegeben wurden, und überprüft schließlich die`fieldName`. Für`relatedPosts`, die Funktion kehrt zurück`result.data`. Für alle anderen Felder kehrt die Funktion einfach zurück`result`. Sehen Sie sich zum Beispiel die folgende Abfrage an:

```
query AllPosts {
  allPosts {
    id
    title
    content
    url
    ups
    downs
    relatedPosts {
      id
    }
    author
  }
}
```

Diese Abfrage gibt eine GraphQL-Antwort zurück, die der folgenden ähnelt:

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

Konfiguration der maximalen Batchgröße

Um die maximale Batchgröße auf einem Resolver zu konfigurieren, verwenden Sie den folgenden Befehl in der 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**
>
> Wenn Sie eine Vorlage für die Anforderungszuweisung bereitstellen, müssen Sie den `BatchInvoke` Vorgang verwenden, um Batching zu verwenden.

Verwendung lokaler Resolver in AWS AppSync

AWS AppSync ermöglicht es Ihnen AWS Lambda, unterstützte Datenquellen (Amazon DynamoDB oder Amazon OpenSearch Service) zu verwenden, um verschiedene Operationen durchzuführen. Doch in bestimmten Szenarien ist ein Aufruf an eine unterstützte Datenquelle möglicherweise nicht erforderlich.

Hier ist der lokale Resolver nützlich. Anstatt eine Remote-Datenquelle aufzurufen, leitet der lokale Resolver einfach das Ergebnis des Request-Handlers an den Response-Handler weiter. Das Feldauflösung verlässt AWS AppSync nicht.

Lokale Resolver sind in einer Vielzahl von Situationen nützlich. Der beliebtest Anwendungsfall ist das Veröffentlichen von Benachrichtigungen ohne Auslösen eines Datenquellenaufrufs. Um diesen Anwendungsfall zu demonstrieren, erstellen wir eine Pub/Sub-Anwendung, in der Benutzer Nachrichten veröffentlichen und abonnieren können. In diesem Beispiel werden Abonnements verwendet. Wenn Sie noch nicht mit Abonnements vertraut sind, sehen Sie sich das Tutorial Echtzeitdaten an.

Die Pub/Sub-App erstellen

Erstellen Sie zunächst eine leere GraphQL-API, indem Sie die Option Von Grund auf neu entwerfen wählen und die optionalen Details konfigurieren, wenn Sie Ihre GraphQL-API erstellen.

In unserer Pub/Sub-Anwendung können Kunden Nachrichten abonnieren und veröffentlichen. Jede veröffentlichte Nachricht enthält einen Namen und Daten. Fügen Sie dies dem Schema hinzu:

```
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"])
}
```

Als Nächstes fügen wir dem `Mutation.publish` Feld einen Resolver hinzu. Suchen Sie im Bereich Resolver neben dem Schemabereich den **`Mutation`** Typ und dann das **`publish(...): Channel`** Feld und klicken Sie dann auf Anhängen.

Erstellen Sie eine None-Datenquelle und geben Sie ihr PageDataSourceeinen Namen. Hängen Sie sie an Ihren Resolver an.

Fügen Sie Ihre Resolver-Implementierung mithilfe des folgenden Snippets hinzu:

```
export function request(ctx) {
  return { payload: ctx.args };
}

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

Stellen Sie sicher, dass Sie den Resolver erstellen und die von Ihnen vorgenommenen Änderungen speichern.

Nachrichten senden und abonnieren

Damit Kunden Nachrichten empfangen können, müssen sie zunächst einen Posteingang abonniert haben.

Führen Sie im Bereich Abfragen das `SubscribeToData` Abonnement aus:

```
subscription SubscribeToData {
    subscribe(name:"channel") {
        name
        data
    }
}
```

Der Abonnent erhält Nachrichten, wann immer die `publish` Mutation aufgerufen wird, aber nur, wenn die Nachricht an das `channel` Abonnement gesendet wird. Lassen Sie uns das im Bereich Abfragen versuchen. Während Ihr Abonnement noch in der Konsole läuft, öffnen Sie eine andere Konsole und führen Sie im Bereich Abfragen die folgende Anfrage aus:

> **Note**
>
> In diesem Beispiel verwenden wir gültige JSON-Zeichenketten.

```
mutation PublishData {
    publish(data: "{\"msg\": \"hello world!\"}", name: "channel") {
        data
        name
    }
}
```

Das Ergebnis sieht etwa wie folgt aus:

```
{
  "data": {
    "publish": {
      "data": "{\"msg\":\"hello world!\"}",
      "name": "channel"
    }
  }
}
```

Wir haben gerade die Verwendung lokaler Resolver demonstriert, indem wir eine Nachricht veröffentlicht und empfangen haben, ohne den AWS AppSync Dienst zu verlassen.

Kombinieren von GraphQL-Resolvern in AWS AppSync

Resolver und Felder in einem GraphQL-Schema zeichnen sich durch 1-zu-1-Beziehungen mit einem hohen Maß an Flexibilität aus. Da eine Datenquelle unabhängig von einem Schema auf einem Resolver konfiguriert wird, haben Sie die Möglichkeit, Ihre GraphQL-Typen über verschiedene Datenquellen aufzulösen oder zu manipulieren, sodass Sie ein Schema kombinieren und anpassen können, um Ihren Anforderungen am besten gerecht zu werden.

Die folgenden Szenarien zeigen, wie Sie Datenquellen in Ihrem Schema mischen und abgleichen können. Bevor Sie beginnen, sollten Sie mit der Konfiguration von Datenquellen und Resolvern für AWS Lambda Amazon DynamoDB und Amazon Service vertraut sein. OpenSearch

Beispielschema

Das folgende Schema hat den Typ `Post` mit jeweils drei `Query` `Mutation` Operationen:

```
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
}
```

In diesem Beispiel hätten Sie insgesamt sechs Resolver, von denen jeder eine Datenquelle benötigt. Eine Möglichkeit, dieses Problem zu lösen, besteht darin, diese mit einer einzigen Amazon DynamoDB-Tabelle namens, zu verbinden`Posts`, in der das `AllPost` Feld einen Scan und das `searchPosts` Feld eine Abfrage ausführt (siehe JavaScriptResolver-Funktionsreferenz für DynamoDB). Sie sind jedoch nicht auf Amazon DynamoDB beschränkt; es gibt verschiedene Datenquellen wie Lambda oder OpenSearch Service, um Ihre Geschäftsanforderungen zu erfüllen.

Änderung von Daten mithilfe von Resolvern

Möglicherweise müssen Sie Ergebnisse aus einer Drittanbieter-Datenbank zurückgeben, die nicht direkt von AWS AppSync Datenquellen unterstützt wird. Möglicherweise müssen Sie auch komplexe Änderungen an den Daten vornehmen, bevor sie an die API-Clients zurückgegeben werden. Dies kann durch eine falsche Formatierung der Datentypen verursacht werden, z. B. durch Zeitstempelunterschiede auf Clients oder durch den Umgang mit Abwärtskompatibilitätsproblemen. In diesem Fall ist es die geeignete Lösung, AWS Lambda Funktionen als Datenquelle mit Ihrer AWS AppSync API zu verbinden. Zur Veranschaulichung: Im folgenden Beispiel manipuliert eine AWS Lambda Funktion Daten, die aus einem Datenspeicher eines Drittanbieters abgerufen wurden:

```
export const handler = (event, context, callback) => {
    // fetch data
    const result = fetcher()

    // apply complex business logic
    const data = transform(result)

    // return to AppSync
    return data
};
```

Hierbei handelt es sich um eine absolut gültige Lambda-Funktion, die dem `AllPost`-Feld im GraphQL-Schema zugewiesen werden könnte. Auf diese Weise würde jeder Abfrage, die alle Ergebnisse zurückgibt, Zufallszahlen für die positiven und negativen Stimmen vergeben.

DynamoDB und Service OpenSearch

Bei einigen Anwendungen führen Sie möglicherweise Mutationen oder einfache Suchabfragen für DynamoDB durch und lassen einen Hintergrundprozess Dokumente an OpenSearch Service übertragen. Sie könnten den `searchPosts` Resolver einfach an die OpenSearch Service-Datenquelle anhängen und Suchergebnisse (aus Daten, die ihren Ursprung in DynamoDB haben)

mithilfe einer GraphQL-Abfrage zurückgeben. Dies kann äußerst leistungsfähig sein, wenn Sie Ihren Anwendungen erweiterte Suchoperationen wie Stichwörter, unscharfe Wortübereinstimmungen oder sogar Geodatensuchen hinzufügen. Die Übertragung von Daten aus DynamoDB könnte über einen ETL-Prozess erfolgen, oder Sie könnten alternativ mithilfe von Lambda aus DynamoDB streamen.

Informationen zu den ersten Schritten mit diesen speziellen Datenquellen finden Sie in unseren DynamoDB - und Lambda-Tutorials.

Wenn Sie beispielsweise das Schema aus unserem vorherigen Tutorial verwenden, fügt die folgende Mutation DynamoDB ein Element hinzu:

```
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
  }
}
```

Dadurch werden Daten in DynamoDB geschrieben, das dann Daten über Lambda an Amazon OpenSearch Service streamt, mit dem Sie dann anhand verschiedener Felder nach Beiträgen suchen. Da sich die Daten beispielsweise in Amazon OpenSearch Service befinden, können Sie entweder die Autoren- oder Inhaltsfelder mit Freiformtext, auch mit Leerzeichen, wie folgt durchsuchen:

```
query searchName{
    searchAuthor(name:"   Nadia   "){
        id
        title
        content
    }
```

```
}

---------- or ----------

query searchContent{
    searchContent(text:"test"){
        id
        title
        content
    }
}
```

Da die Daten direkt in DynamoDB geschrieben werden, können Sie mit den Abfragen `allPost{...}` und `getPost{...}` dennoch effiziente Listen- oder Element-Suchvorgänge für die Tabelle durchführen. Dieser Stack verwendet den folgenden Beispielcode für DynamoDB-Streams:

> **Note**
>
> Dieser Python-Code ist ein Beispiel und sollte nicht im Produktionscode verwendet werden.

```
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.'
```

Sie können dies dann mithilfe von DynamoDB-Streams an eine DynamoDB-Tabelle mit dem Primärschlüssel von anhängen`id`, und alle Änderungen an der DynamoDB-Quelle würden in Ihre Service-Domain gestreamt. OpenSearch Weitere Informationen zum Konfigurieren dieser Einstellung finden Sie in der Dokumentation zu DynamoDB Streams.

Verwendung von Amazon OpenSearch Service-Resolvern in AWS AppSync

AWS AppSync unterstützt die Nutzung von Amazon OpenSearch Service von Domains aus, die Sie in Ihrem eigenen AWS Konto bereitgestellt haben, sofern sie nicht in einer VPC existieren. Nachdem Ihre Domänen bereitgestellt wurden, können Sie sich über eine Datenquelle mit ihnen verbinden. Zu diesem Zeitpunkt können Sie auch einen Resolver im Schema konfigurieren, um GraphQL-Operationen, wie z. B. Abfragen, Mutationen und Abonnements, auszuführen. Diese Anleitung führt Sie durch einige häufig auftretende Beispiele.

Weitere Informationen finden Sie in unserer Referenz zu JavaScript Resolver-Funktionen für. OpenSearch

Erstellen Sie eine neue OpenSearch Dienstdomäne

Um mit diesem Tutorial beginnen zu können, benötigen Sie eine bestehende OpenSearch Service-Domain. Wenn Sie noch keine haben, können Sie das folgende Bespiel verwenden. Beachten Sie, dass es bis zu 15 Minuten dauern kann, bis eine OpenSearch Dienstdomäne erstellt ist, bevor Sie mit der Integration in eine AWS AppSync Datenquelle fortfahren können.

```
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
```

Sie können den folgenden AWS CloudFormation Stack in der Region US-West-2 (Oregon) in Ihrem AWS Konto starten:

Konfigurieren Sie eine Datenquelle für Service OpenSearch

Nachdem die OpenSearch Service-Domain erstellt wurde, navigieren Sie zu Ihrer AWS AppSync GraphQL-API und wählen Sie die Registerkarte Datenquellen. Wählen Sie Datenquelle erstellen und geben Sie einen benutzerfreundlichen Namen für die Datenquelle ein, z. B. „*oss*“. Wählen Sie dann OpenSearch Amazon-Domain als Datenquellentyp und wählen Sie die entsprechende Region aus, und Ihre OpenSearch Service-Domain sollte aufgeführt sein. Nachdem Sie sie ausgewählt haben, können Sie entweder eine neue Rolle erstellen und AWS AppSync ihnen die entsprechenden Berechtigungen zuweisen, oder Sie können eine bestehende Rolle auswählen, für die die folgende Inline-Richtlinie gilt:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1234234",
            "Effect": "Allow",
            "Action": [
                "es:ESHttpDelete",
                "es:ESHttpHead",
                "es:ESHttpGet",
                "es:ESHttpPost",
                "es:ESHttpPut"
            ],
            "Resource": [
                "arn:aws:es:us-east-1:ACCOUNTNUMBER:domain/democluster/*"
            ]
        }
    ]
```

```
}
```

Sie müssen außerdem eine Vertrauensbeziehung AWS AppSync für diese Rolle einrichten mit:

JSON

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

Darüber hinaus verfügt die OpenSearch Service-Domain über eine eigene Zugriffsrichtlinie, die Sie über die Amazon OpenSearch Service-Konsole ändern können. Sie müssen eine Richtlinie hinzufügen, die der folgenden ähnelt und die entsprechenden Aktionen und Ressourcen für die OpenSearch Service-Domain enthält. Beachten Sie, dass der Principal die AWS AppSync Datenquellenrolle sein wird. Sie finden diese Rolle in der IAM-Konsole, sofern Sie sie von dieser Konsole erstellen lassen.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/service-role/
APPSYNC_DATASOURCE_ROLE"
            },
            "Action": [
```

```
                "es:ESHttpDelete",
                "es:ESHttpHead",
                "es:ESHttpGet",
                "es:ESHttpPost",
                "es:ESHttpPut"
            ],
            "Resource": "arn:aws:es:us-east-1:ACCOUNTNUMBER:domain/DOMAIN_NAME/*"
        }
    ]
}
```

Einen Resolver anschließen

Nachdem die Datenquelle nun mit Ihrer OpenSearch Service-Domain verbunden ist, können Sie sie mit einem Resolver mit Ihrem GraphQL-Schema verbinden, wie im folgenden Beispiel gezeigt:

```
 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
}
```

Beachten Sie, dass es einen benutzerdefinierten `Post`-Typ mit einem `id`-Feld gibt. In den folgenden Beispielen gehen wir davon aus, dass es einen Prozess gibt (der automatisiert werden kann), um diesen Typ in Ihre OpenSearch Service-Domain einzufügen, der einer Pfadwurzel zugeordnet würde, `/post/_doc` wo sich der Index `post` befindet. Von diesem Stammpfad aus können Sie einzelne Dokumente, Platzhaltersuchen mit `/id/post*` oder Suchen in mehreren Dokumenten mit dem Pfad

von durchführen. `/post/_search` Wenn Sie beispielsweise einen anderen Typ haben`User`, können Sie Dokumente unter einem neuen Index namens `user` indexieren und dann Suchen mit dem Pfad von durchführen. `/user/_search`

Ändern Sie im Schema-Editor in der AWS AppSync Konsole das vorherige `Posts` Schema, sodass es eine `searchPosts` Abfrage enthält:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  searchPosts: [Post]
}
```

Speichern Sie das Schema. Suchen Sie im Bereich Resolver nach Attach **`searchPosts`** und wählen Sie es aus. Wählen Sie Ihre OpenSearch Service-Datenquelle und speichern Sie den Resolver. Aktualisieren Sie den Code Ihres Resolvers mithilfe des folgenden Snippets:

```
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)
```

```
}
```

Dabei wird davon ausgegangen, dass das vorherige Schema Dokumente enthält, die in OpenSearch Service indexiert wurden, unter dem Feld. `post` Wenn Sie Ihre Daten anders strukturieren, müssen Sie sie entsprechend aktualisieren.

Ihre Suchanfragen ändern

Der vorherige Resolver-Request-Handler führt eine einfache Abfrage für alle Datensätze durch. Angenommen, Sie möchten nach einem bestimmten Autor suchen. Nehmen wir außerdem an, Sie möchten, dass dieser Autor ein in Ihrer GraphQL-Abfrage definiertes Argument ist. Fügen Sie im Schema-Editor der AWS AppSync Konsole eine `allPostsByAuthor` Abfrage hinzu:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  allPostsByAuthor(author: String!): [Post]
  searchPosts: [Post]
}
```

Suchen Sie im Bereich Resolver nach Attach **`allPostsByAuthor`** und wählen Sie es aus. Wählen Sie die OpenSearch Service-Datenquelle aus und verwenden Sie den folgenden Code:

```
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)
}
```

Beachten Sie, dass der body mit einer Begriffsabfrage für das `author`-Feld ausgefüllt wurde und vom Client als Argument übergeben wurde. Optional können Sie vorab ausgefüllte Informationen verwenden, z. B. Standardtext.

Daten zum OpenSearch Service hinzufügen

Möglicherweise möchten Sie Ihrer OpenSearch Service-Domain aufgrund einer GraphQL-Mutation Daten hinzufügen. Hierbei handelt es sich um einen äußerst effektiven Mechanismus für Suchvorgänge und andere Zwecke. Da Sie GraphQL-Abonnements verwenden können, um Ihre Daten in Echtzeit zu erstellen, kann dies als Mechanismus dienen, um Clients über Aktualisierungen von Daten in Ihrer OpenSearch Service-Domain zu informieren.

Kehren Sie zur Schemaseite in der AWS AppSync Konsole zurück und wählen Sie Attach für die `addPost()` Mutation aus. Wählen Sie erneut die OpenSearch Service-Datenquelle aus und verwenden Sie den folgenden Code:

```
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
}
```

Wie zuvor ist dies ein Beispiel dafür, wie Ihre Daten strukturiert sein könnten. Wenn Sie unterschiedliche Feldnamen oder Indizes haben, müssen Sie `path` und aktualisierenbody. Dieses Beispiel zeigt auch, wie Sie das`context.arguments`, was auch als geschrieben werden kann`ctx.args`, in Ihrem Request-Handler verwenden.

Ein einzelnes Dokument wird abgerufen

Wenn Sie schließlich die `getPost(id:ID)` Abfrage in Ihrem Schema verwenden möchten, um ein einzelnes Dokument zurückzugeben, suchen Sie diese Abfrage im Schema-Editor der AWS AppSync Konsole und wählen Sie Anhängen aus. Wählen Sie erneut die OpenSearch Service-Datenquelle aus und verwenden Sie den folgenden Code:

```
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
}
```

Führen Sie Abfragen und Mutationen durch

Sie sollten jetzt in der Lage sein, GraphQL-Operationen für Ihre OpenSearch Service-Domain durchzuführen. Navigieren Sie zur Registerkarte Abfragen der AWS AppSync Konsole und fügen Sie einen neuen Datensatz hinzu:

```
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
       )
}
```

Sie werden das Ergebnis der Mutation auf der rechten Seite sehen. In ähnlicher Weise können Sie jetzt eine `searchPosts` Abfrage für Ihre OpenSearch Service-Domain ausführen:

```
query search {
    searchPosts {
        id
        title
        author
        content
    }
}
```

Bewährte Methoden

- OpenSearch Der Dienst sollte zum Abfragen von Daten dienen, nicht als primäre Datenbank. Möglicherweise möchten Sie OpenSearch Service in Verbindung mit Amazon DynamoDB verwenden, wie unter Kombinieren von GraphQL-Resolvern beschrieben.
- Gewähren Sie nur Zugriff auf Ihre Domain, indem Sie der AWS AppSync Servicerolle den Zugriff auf den Cluster gestatten.
- Sie können mit dieser Entwicklung klein anfangen, indem Sie zunächst nur den preisgünstigsten Cluster verwenden, und später während der Produktion zu einem größeren Cluster überwechseln, der über eine hohe Verfügbarkeit verfügt.

Durchführen von DynamoDB-Transaktionen in AWS AppSync

AWS AppSync unterstützt die Verwendung von Amazon DynamoDB-Transaktionsoperationen für eine oder mehrere Tabellen in einer einzigen Region. Zu den unterstützten Operationen gehören `TransactGetItems` und `TransactWriteItems`. Mithilfe dieser Funktionen können Sie Aufgaben wie die folgenden ausführen: AWS AppSync

- Übergeben einer Schlüsselliste in einer einzigen Abfrage und Rückgabe der Ergebnisse aus einer Tabelle
- Lesen von Datensätzen aus einer oder mehreren Tabellen in einer einzigen Abfrage
- In all-or-nothing gewisser Weise Datensätze in Transaktionen in eine oder mehrere Tabellen schreiben
- Transaktionen werden ausgeführt, wenn einige Bedingungen erfüllt sind

Berechtigungen

Wie bei anderen Resolvern müssen Sie eine Datenquelle in erstellen AWS AppSync und entweder eine Rolle erstellen oder eine vorhandene verwenden. Da Transaktionsvorgänge unterschiedliche Berechtigungen für DynamoDB-Tabellen erfordern, müssen Sie den konfigurierten Rollen Berechtigungen für Lese- oder Schreibaktionen gewähren:

JSON

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-east-1:accountId:table/TABLENAME",
                "arn:aws:dynamodb:region:accountId:table/TABLENAME/*"
            ]
        }
    ]
}
```

> **Note**
>
> Rollen sind an Datenquellen in einer Datenquelle gebunden AWS AppSync, und Resolver für Felder werden für eine Datenquelle aufgerufen. Für Datenquellen, die für den Abruf von DynamoDB konfiguriert sind, ist nur eine Tabelle angegeben, um die Konfiguration zu vereinfachen. Möchten Sie jedoch einen Transaktionsvorgang an mehreren Tabellen mit nur einem Resolver ausführen, stellt dies eine erweiterte Aufgabe dar, und Sie müssen der Rolle dieser Datenquelle Zugriff auf alle Tabellen gewähren, mit denen der Resolver interagieren wird. Dies kann im Feld Ressource (Resource) in der IAM-Richtlinie weiter oben eingerichtet werden. Die Konfiguration der Transaktionsaufrufe für die Tabellen erfolgt im Resolver-Code, den wir weiter unten beschreiben.

Datenquelle

Der Einfachheit halber verwenden wir die gleiche Datenquelle für alle in diesem Tutorial verwendeten Resolver.

Wir werden zwei Tabellen namens SavingAccounts und checkingAccounts haben, beide mit dem `accountNumber` als Partitionsschlüssel und eine TransactionHistory-Tabelle mit als

Partitionsschlüssel. `transactionId` Sie können die folgenden CLI-Befehle verwenden, um Ihre Tabellen zu erstellen. Stellen Sie sicher, dass Sie es `region` durch Ihre Region ersetzen.

Mit der 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
```

Erstellen Sie in der AWS AppSync Konsole unter Datenquellen eine neue DynamoDB-Datenquelle und geben Sie ihr einen Namen. TransactTutorial Wählen Sie SavingAccounts als Tabelle aus (obwohl die spezifische Tabelle bei der Verwendung von Transaktionen keine Rolle spielt). Wählen Sie, ob Sie eine neue Rolle und die Datenquelle erstellen möchten. Sie können die Datenquellenkonfiguration überprüfen, um den Namen der generierten Rolle zu sehen. In der IAM-Konsole können Sie eine Inline-Richtlinie hinzufügen, die es der Datenquelle ermöglicht, mit allen Tabellen zu interagieren.

Ersetzen Sie `region` und `accountID` durch Ihre Region und Konto-ID:

JSON

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

```
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-east-1:accountId:table/savingAccounts",
                "arn:aws:dynamodb:region:accountId:table/savingAccounts/*",
                "arn:aws:dynamodb:us-east-1:accountId:table/checkingAccounts",
                "arn:aws:dynamodb:region:accountId:table/checkingAccounts/*",
                "arn:aws:dynamodb:us-east-1:accountId:table/transactionHistory",
                "arn:aws:dynamodb:region:accountId:table/transactionHistory/*"
            ]
        }
    ]
}
```

Transaktionen

Für dieses Beispiel ist der Kontext eine klassische Banktransaktion, bei der wir `TransactWriteItems` für folgende Aktionen verwenden:

- Überweisen von Geld von Sparkonten auf Girokonten
- Generieren neuer Transaktionsdatensätze für jede Transaktion

Anschließend verwenden wir `TransactGetItems`, um Details der Sparkonten und Girokonten abzurufen.

> **Warning**
>
> `TransactWriteItems`wird nicht unterstützt, wenn es zusammen mit Konflikterkennung und Konfliktlösung verwendet wird. Diese Einstellungen müssen deaktiviert werden, um mögliche Fehler zu vermeiden.

Wir definieren unser GraphQL-Schema wie folgt:

```
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 - Konten auffüllen

Um Geld zwischen Konten zu übertragen, müssen wir die Tabelle mit den Details füllen. Wir verwenden dazu die GraphQL-Operation `Mutation.populateAccounts`.

Klicken Sie im Abschnitt Schema neben dem `Mutation.populateAccounts` Vorgang auf Anhängen. Wählen Sie die `TransactTutorial` Datenquelle aus und klicken Sie auf Erstellen.

Verwenden Sie nun den folgenden Code:

```
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 }
}
```

Speichern Sie den Resolver und navigieren Sie zum Abschnitt Abfragen der AWS AppSync Konsole, um die Konten zu füllen.

Führen Sie die folgende Mutation aus:

```
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
    }
  }
```

```
}
```

Wir haben drei Sparkonten und drei Girokonten in einer Mutation gefüllt.

Verwenden Sie die DynamoDB-Konsole, um zu überprüfen, ob Daten sowohl in den Tabellen SavingAccounts als auch in checkingAccounts angezeigt werden.

TransactWriteItems - Geld überweisen

Hängen Sie mit dem folgenden Code einen Resolver an die `transferMoney` Mutation an. Für jede Überweisung benötigen wir einen Erfolgsmodifikator sowohl für das Giro- als auch für das Sparkonto, und wir müssen die Übertragung in Transaktionen nachverfolgen.

```
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 }
}
```

Navigieren Sie nun zum Abschnitt Abfragen der AWS AppSync Konsole und führen Sie die TransferMoney-Mutation wie folgt aus:

```
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
    }
  }
}
```

Wir haben drei Banktransaktionen in einer Mutation gesendet. Verwenden Sie die DynamoDB-Konsole, um zu überprüfen, ob Daten in den Tabellen SavingAccounts, checkingAccounts und TransactionHistory angezeigt werden.

TransactGetItems - Konten abrufen

Um die Details von Spar- und Girokonten in einer einzigen Transaktionsanfrage abzurufen, fügen wir der `Query.getAccounts` GraphQL-Operation in unserem Schema einen Resolver hinzu. Wählen Sie Anhängen und wählen Sie dieselbe `TransactTutorial` Datenquelle aus, die Sie zu Beginn des Tutorials erstellt haben. Verwenden Sie folgenden Code:

```
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 }
}
```

Speichern Sie den Resolver, und navigieren Sie zu den Queries (Abfragen)-Abschnitten der AWS AppSync -Konsole. Führen Sie die folgende Abfrage aus, um die Spar- und Girokonten abzurufen:

```
query getAccounts {
  getAccounts(
    savingAccountNumbers: ["1", "2", "3"],
    checkingAccountNumbers: ["1", "2"]
  ) {
    savingAccounts {
      accountNumber
      username
      balance
    }
    checkingAccounts {
      accountNumber
      username
      balance
    }
  }
}
```

Wir haben erfolgreich die Verwendung von DynamoDB-Transaktionen unter Verwendung von demonstriert. AWS AppSync

Verwenden von DynamoDB-Batchoperationen in AWS AppSync

AWS AppSync unterstützt die Verwendung von Amazon DynamoDB-Batchoperationen für eine oder mehrere Tabellen in einer einzigen Region. Zu den unterstützten Vorgängen gehören `BatchGetItem`, `BatchPutItem` und `BatchDeleteItem`. Mithilfe dieser Funktionen können Sie Aufgaben wie die folgenden ausführen: AWS AppSync

- Übergeben einer Schlüsselliste in einer einzigen Abfrage und Rückgabe der Ergebnisse aus einer Tabelle
- Lesen von Datensätzen aus einer oder mehreren Tabellen in einer einzigen Abfrage
- Speichern von Datensätzen in großen Mengen in eine oder mehrere Tabellen
- Bedingtes Schreiben oder Löschen von Datensätzen in mehreren Tabellen, die möglicherweise eine Beziehung haben

Batch-Operationen AWS AppSync weisen zwei wesentliche Unterschiede zu Vorgängen ohne Batch auf:

- Die Datenquellenrolle muss über Berechtigungen für alle Tabellen verfügen, auf die der Resolver zugreift.
- Die Tabellenspezifikation für einen Resolver ist Teil des Anforderungsobjekts.

Batches aus einzelnen Tabellen

> **Warning**
>
> `BatchPutItem`und `BatchDeleteItem` werden nicht unterstützt, wenn sie zusammen mit der Konflikterkennung und -lösung verwendet werden. Diese Einstellungen müssen deaktiviert werden, um mögliche Fehler zu vermeiden.

Lassen Sie uns zunächst eine neue GraphQL-API erstellen. Wählen Sie in der AWS AppSync Konsole Create API, GraphQL APIs und Design from scratch aus. Geben Sie Ihrer API einen Namen**`BatchTutorial API`**, wählen Sie Weiter und wählen Sie im Schritt GraphQL-Ressourcen angeben die Option GraphQL-Ressourcen später erstellen aus und klicken Sie auf Weiter. Überprüfen Sie Ihre Daten und erstellen Sie die API. Gehen Sie zur Schema-Seite und fügen Sie das folgende Schema ein. Beachten Sie dabei, dass wir für die Abfrage eine Liste mit folgenden Elementen übergeben IDs:

```
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]
}
```

Speichern Sie Ihr Schema und wählen Sie oben auf der Seite Ressourcen erstellen aus. Wählen Sie Bestehenden Typ verwenden und wählen Sie den `Post` Typ aus. Benennen Sie Ihre Tabelle`Posts`. Stellen Sie sicher, dass der Primärschlüssel auf eingestellt ist**`id`**, deaktivieren Sie die Option GraphQL automatisch generieren (Sie geben Ihren eigenen Code an) und wählen Sie Erstellen aus. AWS AppSync Erstellt zunächst eine neue DynamoDB-Tabelle und eine mit der Tabelle verbundene Datenquelle mit den entsprechenden Rollen. Es gibt jedoch noch einige Berechtigungen, die Sie der Rolle hinzufügen müssen. Gehen Sie zur Seite Datenquellen und wählen Sie die neue Datenquelle aus. Unter Wählen Sie eine vorhandene Rolle aus werden Sie feststellen, dass automatisch eine Rolle für die Tabelle erstellt wurde. Notieren Sie sich die Rolle (sollte ungefähr so aussehen`appsync-ds-ddb-aaabbbcccddd-Posts`) und wechseln Sie dann zur IAM-Konsole (https://console.aws.amazon.com/iam/). Wählen Sie in der IAM-Konsole Rollen und dann Ihre Rolle aus der Tabelle aus. Klicken Sie in Ihrer Rolle unter Berechtigungsrichtlinien auf das "+" neben der Richtlinie (sollte einen ähnlichen Namen wie der Rollenname haben). Wenn die Richtlinie angezeigt wird, wählen Sie oben in der zusammenklappbaren Ansicht die Option Bearbeiten aus. Sie müssen Ihrer Richtlinie Batch-Berechtigungen hinzufügen, insbesondere `dynamodb:BatchGetItem` und`dynamodb:BatchWriteItem`. Es wird ungefähr so aussehen:

JSON

```
{
    "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:…"
            ]
        }
    ]
}
```

Wählen Sie Weiter und dann Änderungen speichern. Ihre Richtlinie sollte jetzt die Stapelverarbeitung zulassen.

Gehen Sie zurück in der AWS AppSync Konsole zur Schemaseite und wählen Sie neben dem `Mutation.batchAdd` Feld die Option Anhängen aus. Erstellen Sie Ihren Resolver mit der `Posts` Tabelle als Datenquelle. Ersetzen Sie im Code-Editor die Handler durch das folgende Snippet. Dieses Snippet nimmt automatisch jedes Element `input PostInput` vom Typ GraphQL und erstellt eine Map, die für den Vorgang benötigt wird: `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;
}
```

Navigieren Sie zur Seite „Abfragen“ der AWS AppSync Konsole und führen Sie die folgende Mutation aus: `batchAdd`

```
mutation add {
    batchAdd(posts:[{
            id: 1 title: "Running in the Park"},{
            id: 2 title: "Playing fetch"
        }]){
            id
            title
    }
}
```

Sie sollten die Ergebnisse auf dem Bildschirm sehen. Sie können dies überprüfen, indem Sie die DynamoDB-Konsole überprüfen, um nach den in die Tabelle geschriebenen Werten zu suchen. `Posts`

Als Nächstes wiederholen Sie den Vorgang des Anfügens eines Resolvers, jedoch für das `Query.batchGet` Feld, wobei die `Posts` Tabelle als Datenquelle verwendet wird. Ersetzen Sie die Handler durch den folgenden Code. Hierdurch wird automatisch jedes Element im GraphQL `ids:[]`-Typ berücksichtigt und eine Zuordnung erstellt, wie sie für die `BatchGetItem`-Operation erforderlich ist:

```
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;
```

```
}
```

Gehen Sie jetzt zurück zur Abfrageseite der AWS AppSync Konsole und führen Sie die folgende **`batchGet`** Abfrage aus:

```
query get {
    batchGet(ids:[1,2,3]){
        id
        title
    }
}
```

Dies sollte die Ergebnisse für die beiden `id`-Werte, die Sie zuvor hinzugefügt haben, zurückgeben. Beachten Sie, dass für den ein `null` Wert `id` mit dem Wert von zurückgegeben wurde`3`. Das liegt daran, dass es in Ihrer `Posts` Tabelle noch keinen Datensatz mit diesem Wert gab. Beachten Sie außerdem, dass die Ergebnisse in derselben Reihenfolge AWS AppSync zurückgegeben werden wie die an die Abfrage übergebenen Schlüssel. Dies ist eine zusätzliche Funktion, die AWS AppSync in Ihrem Namen ausgeführt wird. Wenn Sie also zu wechselnbatchGet(ids:[1,3,2]), werden Sie feststellen, dass sich die Reihenfolge geändert hat. Sie können darüber hinaus erkennen, welche `id` einen `null`-Wert zurückgegeben hat.

Fügen Sie abschließend einen weiteren Resolver an das `Mutation.batchDelete` Feld an, indem Sie die `Posts` Tabelle als Datenquelle verwenden. Ersetzen Sie die Handler durch den folgenden Code. Hierdurch wird automatisch jedes Element im GraphQL `ids:[]`-Typ berücksichtigt und eine Zuordnung erstellt, wie sie für die `BatchGetItem`-Operation erforderlich ist:

```
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;
}
```

Gehen Sie jetzt zurück zur Abfrageseite der AWS AppSync Konsole und führen Sie die folgende `batchDelete` Mutation aus:

```
mutation delete {
    batchDelete(ids:[1,2]){ id }
}
```

Die Datensätze mit der `id` 1 und 2 sollten jetzt gelöscht sein. Wenn Sie die frühere `batchGet()`-Abfrage ausführen, sollten diese `null` zurückgeben.

Batch mit mehreren Tabellen

> **Warning**
>
> `BatchPutItem`und `BatchDeleteItem` werden nicht unterstützt, wenn sie zusammen mit der Konflikterkennung und -lösung verwendet werden. Diese Einstellungen müssen deaktiviert werden, um mögliche Fehler zu vermeiden.

AWS AppSync ermöglicht es Ihnen auch, tabellenübergreifende Batch-Operationen durchzuführen. Lassen Sie uns hierzu eine komplexere Anwendung erstellen. Stellen Sie sich vor, wir entwickeln eine App zur Tiergesundheit, bei der Sensoren den Standort und die Körpertemperatur des Tieres melden. Diese Sensoren sind batteriebetrieben und versuchen, alle paar Minuten eine Verbindung mit dem Netzwerk aufzubauen. Wenn ein Sensor eine Verbindung herstellt, sendet er seine Messwerte an unsere AWS AppSync API. Die Daten werden von Auslösern analysiert und dem Besitzer des Haustiers in einem Dashboard angezeigt. Lassen Sie uns jetzt die Interaktionen zwischen dem Sensor und dem Backend-Datenspeicher näher untersuchen.

Wählen Sie in der AWS AppSync Konsole Create API, GraphQL APIs und Design from scratch aus. Geben Sie Ihrer API einen Namen**`MultiBatchTutorial API`**, wählen Sie Weiter und wählen Sie im Schritt GraphQL-Ressourcen angeben die Option GraphQL-Ressourcen später erstellen aus und klicken Sie auf Weiter. Überprüfen Sie Ihre Daten und erstellen Sie die API. Gehen Sie zur Schema-Seite und fügen Sie das folgende Schema ein und speichern Sie es:

```
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
}
```

Wir müssen zwei DynamoDB-Tabellen erstellen:

- `locationReadings`speichert die Messwerte der Sensorposition.
- `temperatureReadings`speichert die Sensortemperaturwerte.

Beide Tabellen werden dieselbe Primärschlüsselstruktur haben: `sensorId (String)` als Partitionsschlüssel und `timestamp (String)` als Sortierschlüssel.

Wählen Sie oben auf der Seite Create Resources aus. Wählen Sie Bestehenden Typ verwenden und wählen Sie den `locationReadings` Typ aus. Benennen Sie Ihre Tabelle`locationReadings`. Stellen Sie sicher, dass der Primärschlüssel auf `sensorId` und der Sortierschlüssel auf eingestellt sind`timestamp`. Deaktivieren Sie die Option GraphQL automatisch generieren (Sie stellen Ihren eigenen Code bereit) und wählen Sie Erstellen aus. Wiederholen Sie diesen Vorgang, um den `temperatureReadings` als Typ und Tabellennamen zu `temperatureReadings` verwenden. Verwenden Sie dieselben Tasten wie oben.

Ihre neuen Tabellen werden automatisch generierte Rollen enthalten. Es gibt noch einige Berechtigungen, die Sie diesen Rollen hinzufügen müssen. Gehen Sie zur Seite Datenquellen und wählen Sie`locationReadings`. Unter Wählen Sie eine bestehende Rolle aus können Sie die Rolle sehen. Notieren Sie sich die Rolle (sollte ungefähr so aussehen`appsync-ds-ddb-aaabbbcccddd-locationReadings`) und wechseln Sie dann zur IAM-Konsole (https://console.aws.amazon.com/iam/). Wählen Sie in der IAM-Konsole Rollen und dann Ihre Rolle aus der Tabelle aus. Klicken Sie in Ihrer Rolle unter Berechtigungsrichtlinien auf das "+" neben der Richtlinie (sollte einen ähnlichen Namen wie der Rollenname haben). Wenn die Richtlinie angezeigt wird, wählen Sie oben in der zusammenklappbaren Ansicht die Option Bearbeiten aus. Sie müssen dieser Richtlinie Berechtigungen hinzufügen. Es wird ungefähr so aussehen:

JSON

```
{
    "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:us-east-1:account:table/locationReadings",
                "arn:aws:dynamodb:region:account:table/locationReadings/*",
                "arn:aws:dynamodb:us-east-1:account:table/temperatureReadings",
                "arn:aws:dynamodb:region:account:table/temperatureReadings/*"
            ]
        }
    ]
}
```

Wählen Sie Weiter und dann Änderungen speichern. Wiederholen Sie diesen Vorgang für die `temperatureReadings` Datenquelle mit demselben Richtlinienausschnitt oben.

BatchPutItem - Aufzeichnen von Sensormesswerten

Nachdem eine Verbindung zum Internet hergestellt wurde, müssen unsere Sensoren in der Lage sein, ihre Messwerte zu senden. Sie verwenden dazu die API GraphQL-Feld `Mutation.recordReadings`. Wir müssen diesem Feld einen Resolver hinzufügen.

Wählen Sie auf der Schemaseite der AWS AppSync Konsole neben dem `Mutation.recordReadings` Feld die Option Anhängen aus. Erstellen Sie auf dem nächsten Bildschirm Ihren Resolver mit der `locationReadings` Tabelle als Datenquelle.

Nachdem Sie Ihren Resolver erstellt haben, ersetzen Sie die Handler im Editor durch den folgenden Code. Diese `BatchPutItem` Operation ermöglicht es uns, mehrere Tabellen anzugeben:

```
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
}
```

Bei Stapelvorgängen können durch den Aufruf sowohl Fehler als auch Ergebnisse zurückgegeben werden. In diesem Fall können wir eine zusätzliche Fehlerbehandlung vornehmen.

> **Note**
>
> Die Verwendung von `utils.appendError()` ähnelt der`util.error()`, mit dem wesentlichen Unterschied, dass sie die Auswertung des Anfrage- oder Antworthandlers nicht unterbricht. Stattdessen signalisiert es, dass ein Fehler mit dem Feld aufgetreten ist, ermöglicht aber die Auswertung des Handlers und damit die Rückgabe von Daten an den Aufrufer. Wir empfehlen die Verwendung, `utils.appendError()` wenn Ihre Anwendung Teilergebnisse zurückgeben muss.

Speichern Sie den Resolver und navigieren Sie in der AWS AppSync Konsole zur Seite Abfragen. Wir können jetzt einige Sensorwerte senden.

Führen Sie die folgende Mutation aus:

```
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
    }
  }
}
```

Wir haben zehn Sensorwerte in einer Mutation gesendet, wobei die Messwerte auf zwei Tabellen aufgeteilt wurden. Verwenden Sie die DynamoDB-Konsole, um zu überprüfen, ob die Daten sowohl in der als auch in der `locationReadings` Tabelle angezeigt werden. `temperatureReadings`

BatchDeleteItem - Löschen von Sensormesswerten

In ähnlicher Weise müssten wir auch in der Lage sein, Chargen von Sensormesswerten zu löschen. Dazu verwenden wir das GraphQL-Feld `Mutation.deleteReadings`. Wählen Sie auf der Schemaseite der AWS AppSync Konsole neben dem `Mutation.deleteReadings` Feld die Option Anhängen aus. Erstellen Sie auf dem nächsten Bildschirm Ihren Resolver mit der `locationReadings` Tabelle als Datenquelle.

Nachdem Sie Ihren Resolver erstellt haben, ersetzen Sie die Handler im Code-Editor durch das folgende Snippet. In diesem Resolver verwenden wir einen Hilfsfunktions-Mapper, der die `sensorId` und aus den bereitgestellten Eingaben extrahiert. `timestamp`

```
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
}
```

Speichern Sie den Resolver und navigieren Sie zur Seite „Abfragen“ in der Konsole. AWS AppSync Lassen Sie uns nun ein paar Sensormesswerte löschen.

Führen Sie die folgende Mutation aus:

```
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**
>
> Im Gegensatz zum `DeleteItem`-Vorgang wird nicht das vollständig gelöschte Element in der Antwort zurückgegeben. Nur der übergebene Schlüssel wird zurückgegeben. Weitere Informationen finden Sie BatchDeleteItem in der JavaScript Resolver-Funktionsreferenz für DynamoDB.

Überprüfen Sie über die DynamoDB-Konsole, ob diese beiden Messwerte aus den Tabellen `locationReadings` und `temperatureReadings` gelöscht wurden.

BatchGetItem - Messwerte abrufen

Eine weitere übliche Operation für unsere App wäre das Abrufen der Messwerte für einen Sensor zu einem bestimmten Zeitpunkt. Fügen wir dazu einen Resolver zum GraphQL-Feld `Query.getReadings` unseres Schemas hinzu. Wählen Sie auf der Schemaseite der AWS AppSync Konsole neben dem `Query.getReadings` Feld die Option Anhängen aus. Erstellen Sie auf dem nächsten Bildschirm Ihren Resolver mit der `locationReadings` Tabelle als Datenquelle.

Verwenden wir den folgenden Code:

```
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' })),
 ]
}
```

Speichern Sie den Resolver und navigieren Sie zur Seite „Abfragen“ in der AWS AppSync Konsole. Lassen Sie uns nun unsere Sensorwerte abrufen.

Führen Sie die folgende Abfrage aus:

```
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
    }
  }
}
```

Wir haben die Verwendung von DynamoDB-Batchoperationen mit erfolgreich demonstriert. AWS AppSync

Fehlerbehandlung

In AWS AppSync Datenquellenoperationen können manchmal Teilergebnisse zurückgegeben werden. Wir verwenden den Begriff Teilergebnis, wenn bei der Ausgabe einer Operation

einige Daten fehlen und darin ein Fehler enthalten ist. Da die Fehlerbehandlung von Natur aus anwendungsspezifisch ist, AWS AppSync haben Sie die Möglichkeit, Fehler im Antworthandler zu behandeln. Sollte beim Resolver ein Aufruffehler vorliegen, lässt sich dieser im Kontext an `ctx.error` erkennen. Aufruffehler umfassen immer eine Nachricht und eine Art, auf die über die Eigenschaften `ctx.error.message` und `ctx.error.type`zugegriffen werden kann. Im Antworthandler können Sie Teilergebnisse auf drei Arten verarbeiten:

1. Überwinden Sie den Aufruffehler, indem Sie einfach Daten zurückgeben.
2. Einen Fehler (using`util.error(...)`) auslösen, indem Sie die Auswertung des Handlers beenden, wodurch keine Daten zurückgegeben werden.
3. Fügt einen Fehler an (using`util.appendError(...)`) und gibt auch Daten zurück.

Lassen Sie uns jeden der drei oben genannten Punkte anhand von DynamoDB-Batchoperationen demonstrieren.

DynamoDB-Stapelvorgänge

Mit DynamoDB-Stapelvorgängen ist es möglich, dass ein Stapel teilweise abgeschlossen wird. Das bedeutet, dass einige der angeforderten Elemente oder Schlüssel nicht verarbeitet werden. Wenn AWS AppSync ein Batch nicht abgeschlossen werden kann, werden unverarbeitete Elemente und ein Aufruffehler für den Kontext angezeigt.

Wir werden die Fehlerbehandlung mithilfe der `Query.getReadings`-Feldkonfiguration der `BatchGetItem`-Operation aus dem vorherigen Abschnitt dieser Anleitung implementieren. Dieses Mal nehmen wir jedoch an, dass bei der Ausführung des `Query.getReadings`-Felds die DynamoDB-Tabelle `temperatureReadings` den bereitgestellten Durchsatz überschritten hat. DynamoDB hat `ProvisionedThroughputExceededException` beim zweiten Versuch a ausgelöst, AWS AppSync um die verbleibenden Elemente im Stapel zu verarbeiten.

Das folgende JSON stellt den serialisierten Kontext nach dem DynamoDB-Batchaufruf, aber bevor der Antworthandler aufgerufen wurde, dar:

```
{
  "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": []
}
```

Beachten Sie in diesem Kontext die folgenden Dinge:

- Der Aufruffehler wurde für den Kontext bei `ctx.error` by festgelegt AWS AppSync, und der Fehlertyp wurde auf gesetzt. `DynamoDB:ProvisionedThroughputExceededException`
- Die Ergebnisse werden pro Tabelle unter zugeordnet, `ctx.result.data` obwohl ein Fehler vorliegt.
- Schlüssel, die nicht verarbeitet wurden, sind unter verfügbar. `ctx.result.data.unprocessedKeys` Hier AWS AppSync konnte das Element mit dem Schlüssel (sensorId:1, timestamp:2018-02-01T 17:21:05.000 + 08:00) aufgrund des unzureichenden Tabellendurchsatzes nicht abgerufen werden.

> **Note**
>
> Für `BatchPutItem` gilt `ctx.result.data.unprocessedItems`. Für `BatchDeleteItem` gilt `ctx.result.data.unprocessedKeys`.

Sie können mit diesem Fehler auf drei verschiedene Arten umgehen.

1. Übergehen des Aufruffehlers

Wenn Daten ohne Verarbeitung des Aufruffehlers zurückgegeben werden, wird der Fehler effektiv übergangen. Auf diese Weise ist das Ergebnis für das angegebene GraphQL-Feld immer erfolgreich.

Der Code, den wir schreiben, ist vertraut und konzentriert sich nur auf die Ergebnisdaten.

Antworthandler

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

GraphQL-Antwort

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

Der Fehlerantwort werden keine Fehler hinzugefügt, da nur auf Daten reagiert wird.

2. Es wird ein Fehler ausgelöst, um die Ausführung des Response-Handlers abzubrechen

Wenn Teilausfälle aus Sicht des Clients als vollständige Fehler behandelt werden sollen, können Sie die Ausführung des Antworthandlers abbrechen, um zu verhindern, dass Daten zurückgegeben werden. Das Dienstprogramm `util.error(...)` erzielt genau dieses Verhalten.

Code des Antworthandlers

```
export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type, null,
 ctx.result.data.unprocessedKeys);
  }
  return ctx.result.data;
}
```

GraphQL-Antwort

```
{
  "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. (...)"
    }
  ]
}
```

Auch wenn der DynamoDB Stapelvorgang möglicherweise einige Ergebnisse zurückgegeben hat, haben wir uns entschieden, einen Fehler zu melden, indem das GraphQL-Feld `getReadings` mit Null angezeigt und der Fehler dem Fehler-Block der GraphQL-Antwort hinzugefügt wurde.

3. Anfügen eines Fehlers bei der Rückgabe von Daten und Fehlern

In bestimmten Fällen können Anwendungen Teilergebnisse zurückgeben und die Clients über die unverarbeiteten Elemente benachrichtigen und so die Benutzererfahrung verbessern. Die Clients können entscheiden, eine Wiederholung zu implementieren oder den Fehler an den Endbenutzer zu übermitteln. Dies `util.appendError(...)` ist die Utility-Methode, die dieses Verhalten ermöglicht, indem sie es dem Anwendungsdesigner ermöglicht, Fehler an den Kontext anzuhängen, ohne die Auswertung des Antworthandlers zu beeinträchtigen. Nach der Auswertung des Antworthandlers AWS AppSync verarbeitet er alle Kontextfehler, indem er sie an den Fehlerblock der GraphQL-Antwort anhängt.

Code für den Antworthandler

```
export function response(ctx) {
  if (ctx.error) {
    util.appendError(ctx.error.message, ctx.error.type, null,
 ctx.result.data.unprocessedKeys);
  }
  return ctx.result.data;
}
```

Wir haben sowohl den Aufruffehler als auch `unprocessedKeys` das Element innerhalb des Fehlerblocks der GraphQL-Antwort weitergeleitet. Das `getReadings` Feld gibt auch Teildaten aus der `locationReadings` Tabelle zurück, wie Sie in der folgenden Antwort sehen können.

GraphQL-Antwort

```
{
  "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. (...)"
    }
  ]
}
```

Verwendung von HTTP-Resolvern in AWS AppSync

AWS AppSync ermöglicht es Ihnen, unterstützte Datenquellen (d. h. Amazon DynamoDB AWS Lambda, Amazon OpenSearch Service oder Amazon Aurora) zu verwenden, um verschiedene Operationen durchzuführen, zusätzlich zu beliebigen HTTP-Endpunkten zur Auflösung von GraphQL-Feldern. Sobald die HTTP-Endpunkte verfügbar sind, können Sie sie mit einer Datenquelle

verbinden. Anschließend können Sie einen Resolver im Schema konfigurieren, um GraphQL-Operationen wie Abfragen, Mutationen und Abonnements auszuführen. Dieses Tutorial führt Sie durch einige häufig auftretende Beispiele.

In diesem Tutorial verwenden Sie eine REST-API (erstellt mit Amazon API Gateway und Lambda) mit einem AWS AppSync GraphQL-Endpunkt.

Erstellen einer REST-API

Sie können die folgende AWS CloudFormation Vorlage verwenden, um einen REST-Endpunkt einzurichten, der für dieses Tutorial funktioniert:

Der AWS CloudFormation Stack führt die folgenden Schritte aus:

1. Richtet eine Lambda-Funktion ein, die die Geschäftslogik für den Mikroservice enthält.
2. Richtet eine API-Gateway-REST-API mit der folgenden endpoint/method/content Typkombination ein:

API-Ressourcenpfad	HTTP-Methode	Unterstützter Inhaltstyp
/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

Erstellen Sie Ihre GraphQL-API

Um die GraphQL-API zu erstellen in AWS AppSync:

1. Öffnen Sie die AWS AppSync Konsole und wählen Sie Create API.
2. Wählen Sie GraphQL APIs und dann Design from scratch. Wählen Sie Weiter aus.

3. Geben Sie für den API-Namen `UserData` ein. Wählen Sie Weiter aus.
4. Wählen Sie `Create GraphQL resources later`. Wählen Sie Weiter aus.
5. Überprüfen Sie Ihre Eingaben und wählen Sie Create API.

Die AWS AppSync Konsole erstellt mithilfe des API-Schlüsselauthentifizierungsmodus eine neue GraphQL-API für Sie. Sie können die Konsole verwenden, um Ihre GraphQL-API weiter zu konfigurieren und Anfragen auszuführen.

Erstellen eines GraphQL-Schemas

Da wir jetzt eine GraphQL-API haben, erstellen Sie ein GraphQL-Schema. Verwenden Sie im Schema-Editor in der AWS AppSync Konsole das folgende Snippet:

```
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
}
```

Konfigurieren Sie Ihre HTTP-Datenquelle

Gehen Sie wie folgt vor, um die HTTP-Datenquelle zu konfigurieren:

1. Wählen Sie auf der Seite Datenquellen in Ihrer AWS AppSync GraphQL-API die Option Datenquelle erstellen aus.
2. Geben Sie einen Namen für die Datenquelle ein, z. `HTTP_Example`
3. Wählen Sie unter Datenquellentyp die Option HTTP-Endpunkt aus.
4. Stellen Sie den Endpunkt auf den API-Gateway-Endpunkt ein, der zu Beginn des Tutorials erstellt wurde. Sie finden Ihren vom Stack generierten Endpunkt, wenn Sie zur Lambda-Konsole navigieren und Ihre Anwendung unter Anwendungen suchen. In den Einstellungen Ihrer Anwendung sollten Sie einen API-Endpunkt sehen, der Ihr Endpunkt sein wird. AWS AppSync Stellen Sie sicher, dass Sie den Phasennamen nicht als Teil des Endpunkts angeben. Wenn Ihr Endpunkt beispielsweise wäre`https://aaabbbcccd.execute-api.us-east-1.amazonaws.com/v1`, würden Sie Folgendes eingeben`https://aaabbbcccd.execute-api.us-east-1.amazonaws.com`.

> **Note**
>
> Derzeit werden nur öffentliche Endpunkte von AWS AppSync unterstützt.
> Weitere Informationen zu den Zertifizierungsstellen, die vom AWS AppSync Dienst anerkannt werden, finden Sie unter Zertifizierungsstellen (CA) Recognized by AWS AppSync for HTTPS Endpoints.

Konfigurieren von Resolvern

In diesem Schritt verbinden Sie die HTTP-Datenquelle mit den `getUser` `addUser` AND-Abfragen.

So richten Sie den `getUser` Resolver ein:

1. Wählen Sie in Ihrer AWS AppSync GraphQL-API die Registerkarte Schema.
2. Suchen Sie rechts neben dem Schema-Editor im Bereich Resolvers und unter dem Abfragetyp das **`getUser`** Feld und wählen Sie Anhängen aus.
3. Behalten Sie den Resolvertyp bei `Unit` und die Laufzeit bei. `APPSYNC_JS`
4. Wählen Sie unter Datenquellenname den HTTP-Endpunkt aus, den Sie zuvor erstellt haben.

5. Wählen Sie Erstellen aus.
6. Fügen Sie im Resolver-Code-Editor das folgende Snippet als Ihren Request-Handler hinzu:

```
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. Fügen Sie das folgende Snippet als Antworthandler hinzu:

```
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. Wählen Sie auf der Registerkarte Query (Abfrage) und führen Sie dann folgende Abfrage aus:

```
query GetUser{
    getUser(id:1){
        id
        username
    }
}
```

Folgende Antwort sollte zurückgegeben werden:

```
{
```

```
    "data": {
        "getUser": {
            "id": "1",
            "username": "nadia"
        }
    }
}
```

Um den Resolver einzurichten: `addUser`

1. Wählen Sie die Registerkarte Schema aus.
2. Suchen Sie rechts neben dem Schema-Editor im Bereich Resolver unter dem Abfragetyp das **`addUser`** Feld und wählen Sie Anhängen aus.
3. Behalten Sie den Resolvertyp bei `Unit` und die Laufzeit bei. `APPSYNC_JS`
4. Wählen Sie unter Datenquellenname den HTTP-Endpunkt aus, den Sie zuvor erstellt haben.
5. Wählen Sie Erstellen aus.
6. Fügen Sie im Resolver-Code-Editor das folgende Snippet als Ihren Request-Handler hinzu:

```
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. Fügen Sie das folgende Snippet als Antworthandler hinzu:

```
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. Wählen Sie auf der Registerkarte Query (Abfrage) und führen Sie dann folgende Abfrage aus:

```
mutation addUser{
    addUser(userInput:{
        id:"2",
        username:"shaggy"
    }){
        id
        username
    }
}
```

Wenn Sie die `getUser` Abfrage erneut ausführen, sollte sie die folgende Antwort zurückgeben:

```
{
    "data": {
        "getUser": {
        "id": "2",
        "username": "shaggy"
        }
    }
}
```

Dienste werden aufgerufen AWS

Sie können HTTP-Resolver verwenden, um eine GraphQL-API-Schnittstelle für AWS Dienste einzurichten. HTTP-Anfragen an AWS müssen mit dem Signature Version 4-Prozess signiert werden, damit identifiziert werden AWS kann, wer sie gesendet hat. AWS AppSync berechnet die Signatur in Ihrem Namen, wenn Sie der HTTP-Datenquelle eine IAM-Rolle zuordnen.

Sie stellen zwei zusätzliche Komponenten bereit, um AWS Dienste mit HTTP-Resolvern aufzurufen:

- Eine IAM-Rolle mit Berechtigungen zum Aufrufen des Dienstes AWS APIs
- Signierkonfiguration in der Datenquelle

Wenn Sie den ListGraphqlApis Vorgang beispielsweise mit HTTP-Resolvern aufrufen möchten, erstellen Sie zunächst eine IAM-Rolle, der die folgende AWS AppSync Richtlinie zugewiesen ist:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "appsync:ListGraphqlApis"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Als Nächstes erstellen Sie die HTTP-Datenquelle für. AWS AppSync In diesem Beispiel rufen Sie AWS AppSync in der Region USA West (Oregon) auf. Richten Sie die folgende HTTP-Konfiguration in einer Datei mit dem Namen `http.json` ein, die die Signierregion und den Servicenamen enthält:

```
{
    "endpoint": "https://appsync.us-west-2.amazonaws.com/",
    "authorizationConfig": {
        "authorizationType": "AWS_IAM",
        "awsIamConfig": {
            "signingRegion": "us-west-2",
            "signingServiceName": "appsync"
        }
    }
}
```

Verwenden Sie dann die, AWS CLI um die Datenquelle mit einer zugehörigen Rolle wie folgt zu erstellen:

```
aws appsync create-data-source --api-id <API-ID> \
                               --name AWSAppSync \
                               --type HTTP \
                               --http-config file:///http.json \
```

```
                                --service-role-arn <ROLE-ARN>
```

Wenn Sie einen Resolver an das Feld im Schema anhängen, verwenden Sie zum Aufrufen AWS AppSync die folgende Vorlage für die Anforderungszuweisung:

```
{
    "version": "2018-05-29",
    "method": "GET",
    "resourcePath": "/v1/apis"
}
```

Wenn Sie eine GraphQL-Abfrage für diese Datenquelle ausführen, AWS AppSync signiert die Anfrage mit der von Ihnen angegebenen Rolle und fügt die Signatur in die Anfrage ein. Die Abfrage gibt eine Liste von AWS AppSync GraphQL APIs in Ihrem Konto in dieser AWS Region zurück.

Verwenden von Aurora PostgreSQL mit Daten-API in AWS AppSync

Erfahren Sie, wie Sie Ihre GraphQL-API mit Aurora PostgreSQL-Datenbanken verbinden können. AWS AppSync Diese Integration ermöglicht es Ihnen, skalierbare, datengesteuerte Anwendungen zu erstellen, indem Sie SQL-Abfragen und -Mutationen über GraphQL-Operationen ausführen. AWS AppSync bietet eine Datenquelle für die Ausführung von SQL-Anweisungen für Amazon Aurora Aurora-Cluster, die mit einer Daten-API aktiviert sind. Sie können AWS AppSync Resolver verwenden, um SQL-Anweisungen für die Daten-API mit GraphQL-Abfragen, -Mutationen und Abonnements auszuführen.

Bevor Sie mit diesem Tutorial beginnen, sollten Sie über grundlegende Kenntnisse von AWS Diensten und GraphQL-Konzepten verfügen.

> **Note**
>
> Für dieses Tutorial wird die Region `US-EAST-1` verwendet.

Themen

- Richten Sie Ihre Aurora PostgreSQL-Datenbank ein
- Datenbank und Tabelle erstellen

- Erstellen eines GraphQL-Schemas
- Resolver für RDS
- Löschen Sie Ihren Cluster

Richten Sie Ihre Aurora PostgreSQL-Datenbank ein

Bevor Sie eine Amazon RDS-Datenquelle hinzufügen AWS AppSync, gehen Sie wie folgt vor.

1. Aktivieren Sie eine Daten-API auf einem Aurora Serverless v2-Cluster.
2. Konfigurieren Sie ein Geheimnis mit AWS Secrets Manager
3. Erstellen Sie den Cluster mit dem folgenden AWS CLI Befehl.

```
aws rds create-db-cluster \
            --db-cluster-identifier appsync-tutorial \
            --engine aurora-postgresql \
            --engine-version 16.6 \
            --serverless-v2-scaling-configuration MinCapacity=0,MaxCapacity=1 \
            --master-username USERNAME \
            --master-user-password COMPLEX_PASSWORD \
            --enable-http-endpoint
```

Dadurch wird ein ARN für den Cluster zurückgegeben. Nachdem Sie einen Cluster erstellt haben, müssen Sie mit dem folgenden AWS CLI Befehl eine Serverless v2-Instanz hinzufügen.

```
aws rds create-db-instance \
    --db-cluster-identifier appsync-tutorial \
    --db-instance-identifier appsync-tutorial-instance-1 \
    --db-instance-class db.serverless \
    --engine aurora-postgresql
```

> **Note**
>
> Es dauert einige Zeit, bis diese Endpunkte aktiviert werden. Sie können ihren Status in der RDS-Konsole auf der Registerkarte Konnektivität und Sicherheit für den Cluster überprüfen.

Überprüfen Sie den Clusterstatus mit dem folgenden AWS CLI Befehl.

```
aws rds describe-db-clusters \
    --db-cluster-identifier appsync-tutorial \
    --query "DBClusters[0].Status"
```

Erstellen Sie ein Secret über die AWS Secrets Manager Konsole oder AWS CLI mit einer Eingabedatei wie der folgenden, indem Sie die `USERNAME` und `COMPLEX_PASSWORD` aus dem vorherigen Schritt verwenden:

```
{
    "username": "USERNAME",
    "password": "COMPLEX_PASSWORD"
}
```

Übergeben Sie dies als Parameter an AWS CLI:

```
aws secretsmanager create-secret \
    --name appsync-tutorial-rds-secret \
    --secret-string file://creds.json
```

Dadurch wird ein ARN für das Secret zurückgegeben. Notieren Sie sich den ARN Ihres Aurora Serverless v2-Clusters und Secret für später, wenn Sie eine Datenquelle in der AWS AppSync Konsole erstellen.

Datenbank und Tabelle erstellen

Erstellen Sie zunächst eine Datenbank mit dem Namen`TESTDB`. In PostgreSQL ist eine Datenbank ein Container, der Tabellen und andere SQL-Objekte enthält. Stellen Sie sicher, dass Ihr Aurora Serverless v2-Cluster korrekt konfiguriert ist, bevor Sie ihn zu Ihrer AWS AppSync API hinzufügen. Erstellen Sie zunächst eine TESTDB-Datenbank mit dem folgenden `--sql` Parameter.

```
aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:us-east-1:111122223333 ISN:cluster:appsync-tutorial" \
    --secret-arn "arn:aws:secretsmanager:us-east-1:111122223333 ISN:secret:appsync-
tutorial-rds-secret" \
    --sql "create DATABASE \"testdb\"" \
    --database "postgres"
```

Wenn das ohne Fehler läuft, fügen Sie zwei Tabellen mit dem `create table` Befehl hinzu:

```
 aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:us-east-1:111122223333 ISN:cluster:appsync-tutorial" \
    --secret-arn "arn:aws:secretsmanager:us-east-1:111122223333 ISN: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-1:111122223333 ISN:cluster:appsync-tutorial" \
    --secret-arn "arn:aws:secretsmanager:us-east-1:111122223333 ISN: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);'
```

Wenn dies erfolgreich ist, fügen Sie den Cluster als Datenquelle in Ihre API ein.

Erstellen eines GraphQL-Schemas

Jetzt, da Ihre Aurora Serverless v2 Data API mit konfigurierten Tabellen läuft, erstellen wir ein GraphQL-Schema. Sie können Ihre API schnell erstellen, indem Sie mit dem API-Erstellungsassistenten Tabellenkonfigurationen aus einer vorhandenen Datenbank importieren.

Um zu beginnen:

1. Wählen Sie in der AWS AppSync Konsole Create API und dann Start with a Amazon Aurora Cluster aus.
2. Geben Sie API-Details wie den API-Namen an und wählen Sie dann Ihre Datenbank aus, um die API zu generieren.
3. Wählen Sie Ihre Datenbank aus. Aktualisieren Sie bei Bedarf die Region und wählen Sie dann Ihren Aurora-Cluster und Ihre TESTDB-Datenbank aus.
4. Wählen Sie Ihr Secret und dann Import.
5. Sobald die Tabellen erkannt wurden, aktualisieren Sie die Typnamen. Wechseln Sie `Todos` zu `Todo` und `Tasks` zu`Task`.
6. Zeigen Sie eine Vorschau des generierten Schemas an, indem Sie Schemavorschau wählen. Ihr Schema wird ungefähr so aussehen:

```
type Todo {
  id: Int!
  description: String!
  due: AWSDate!
  createdAt: String
}

type Task {
  id: Int!
  todoId: Int!
  description: String
}
```

7. Für die Rolle können Sie entweder eine neue Rolle AWS AppSync erstellen lassen oder eine mit einer Richtlinie erstellen, die der folgenden ähnelt:

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-data:ExecuteStatement"
            ],
            "Resource": [
                "arn:aws:rds:us-east-1:111122223333 ISN:cluster:appsync-
tutorial",
                "arn:aws:rds:us-east-1:111122223333 ISN:cluster:appsync-
tutorial:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-east-1:111122223333
 ISN:secret:your:secret:arn:appsync-tutorial-rds-secret",
                "arn:aws:secretsmanager:us-east-1:111122223333
 ISN:secret:your:secret:arn:appsync-tutorial-rds-secret:*"
```

```
            ]
        }
    ]
}
```

Beachten Sie, dass diese Richtlinie zwei Aussagen enthält, für die Sie Rollenzugriff gewähren. Die erste Ressource ist Ihr Aurora-Cluster und die zweite ist Ihr AWS Secrets Manager ARN.

Wählen Sie Weiter, überprüfen Sie die Konfigurationsdetails und wählen Sie dann Create API. Sie haben jetzt eine voll funktionsfähige API. Sie können die vollständigen Details Ihrer API auf der Schema-Seite überprüfen.

Resolver für RDS

Der API-Erstellungsablauf hat automatisch die Resolver für die Interaktion mit unseren Typen erstellt. Wenn Sie sich die Schema-Seite ansehen, finden Sie Resolver einiger der folgenden Resolver.

- Erstellen Sie eine über das Feld`todo`. `Mutation.createTodo`
- Aktualisieren Sie ein `todo` über das `Mutation.updateTodo` Feld.
- Löschen Sie ein `todo` über das `Mutation.deleteTodo` Feld.
- Holen Sie sich eine Single `todo` über das `Query.getTodo` Feld.
- Alle `todos` über das `Query.listTodos` Feld auflisten.

Sie werden ähnliche Felder und Resolver für den `Task` Typ finden. Schauen wir uns einige der Resolver genauer an.

Mutation.CreateToDo

Wählen Sie im Schema-Editor in der AWS AppSync Konsole auf der rechten Seite die Option neben. `testdb createTodo(...): Todo` Der Resolver-Code verwendet die `insert` Funktion aus dem `rds` Modul, um dynamisch eine Insert-Anweisung zu erstellen, die der `todos` Tabelle Daten hinzufügt. Da wir mit Postgres arbeiten, können wir die `returning` Anweisung nutzen, um die eingefügten Daten zurückzuholen.

Aktualisieren Sie den folgenden Resolver, um den `DATE` Feldtyp korrekt anzugeben. 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]
}
```

Speichern Sie den Resolver. Der Typhinweis markiert das in unserem Eingabeobjekt `due` korrekt als `DATE` Typ. Dadurch kann die Postgres-Engine den Wert richtig interpretieren. Aktualisieren Sie als Nächstes Ihr Schema, um das `id` aus der `CreateTodo` Eingabe zu entfernen. Da unsere Postgres-Datenbank die generierte ID zurückgeben kann, können Sie sich darauf verlassen, dass sie erstellt und das Ergebnis wie folgt in einer einzigen Anfrage zurückgibt.

```
input CreateTodoInput {
    due: AWSDate!
    createdAt: String
    description: String!
}
```

Nehmen Sie die Änderung vor und aktualisieren Sie Ihr Schema. Gehen Sie zum Abfrage-Editor, um der Datenbank wie folgt ein Element hinzuzufügen.

```
mutation CreateTodo {
  createTodo(input: {description: "Hello World!", due: "2023-12-31"}) {
    id
    due
    description
    createdAt
  }
}
```

Sie erhalten das folgende Ergebnis.

```
{
  "data": {
    "createTodo": {
      "id": 1,
      "due": "2023-12-31",
      "description": "Hello World!",
      "createdAt": "2023-11-14 20:47:11.875428"
    }
  }
}
```

Query.ListToDos

Wählen Sie im Schema-Editor in der Konsole auf der rechten Seite die Option `testdb` neben. `listTodos(id: ID!): Todo` Der Request-Handler verwendet die Select Utility-Funktion, um eine Anforderung zur Laufzeit dynamisch zu erstellen.

```
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)
```

```
}
```

Wir möchten `todos` nach dem `due` Datum filtern. Lassen Sie uns den Resolver aktualisieren, in den `due` Werte umgewandelt werden sollen`DATE`. Aktualisieren Sie die Liste der Importe und den Request-Handler wie folgt.

```
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 };
}
```

Gehen Sie im Query-Editor wie folgt vor.

```
query LIST {
  listTodos(limit: 10, filter: {due: {between: ["2021-01-01", "2025-01-02"]}}) {
    items {
      id
      due
      description
    }
  }
}
```

mutation.updateTodo

Sie können auch ein. `update` `Todo` Lassen Sie uns im Query-Editor unser erstes `Todo` Element von aktualisieren `id1`.

```
mutation UPDATE {
  updateTodo(input: {id: 1, description: "edits"}) {
    description
    due
    id
  }
}
```

Beachten Sie, dass Sie das `id` Element angeben müssen, das Sie aktualisieren möchten. Sie können auch eine Bedingung angeben, um nur ein Element zu aktualisieren, das bestimmte Bedingungen erfüllt. Beispielsweise möchten wir das Element möglicherweise nur bearbeiten, wenn die Beschreibung `edits` wie folgt beginnt.

```
mutation UPDATE {
  updateTodo(input: {id: 1, description: "edits: make a change"}, condition:
 {description: {beginsWith: "edits"}}) {
    description
```

```
    due
    id
  }
}
```

Genau wie wir mit unseren `list` Operationen `create` und umgegangen sind, können wir unseren Resolver so aktualisieren, dass er das `due` Feld in a `DATE` umwandelt. Speichern Sie diese Änderungen `updateTodo` wie folgt.

```
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];
}
```

Versuchen Sie jetzt ein Update mit einer Bedingung:

```
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

Du kannst `delete` mit der Mutation weitermachen. `Todo` `deleteTodo` Dies funktioniert wie die `updateTodo` Mutation, und Sie müssen das `id` Element, das Sie löschen möchten, wie folgt angeben.

```
mutation DELETE {
  deleteTodo(input: {id: 1}) {
    description
    due
    id
  }
}
```

Schreiben von benutzerdefinierten Abfragen

Wir haben die `rds` Modul-Dienstprogramme verwendet, um unsere SQL-Anweisungen zu erstellen. Wir können auch unsere eigene benutzerdefinierte statische Anweisung schreiben, um mit unserer Datenbank zu interagieren. Aktualisieren Sie zunächst das Schema, um das `id` Feld aus der `CreateTask` Eingabe zu entfernen.

```
input CreateTaskInput {
    todoId: Int!
    description: String
}
```

Erstellen Sie als Nächstes einige Aufgaben. Eine Aufgabe hat Todo wie folgt eine Fremdschlüsselbeziehung.

```
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 }
}
```

Erstellen Sie ein neues Feld Ihres `Query` Typs mit dem folgenden Namen`getTodoAndTasks`.

```
getTodoAndTasks(id: Int!): Todo
```

Fügen Sie dem Todo Typ wie folgt ein `tasks` Feld hinzu.

```
type Todo {
    due: AWSDate!
    id: Int!
    createdAt: String
    description: String!
    tasks:TaskConnection
}
```

Speichern Sie das Schema. Wählen Sie im Schema-Editor in der Konsole auf der rechten Seite Attach Resolver for `getTodosAndTasks(id: Int!): Todo` aus. Wählen Sie Ihre Amazon RDS-Datenquelle. Aktualisieren Sie Ihren Resolver mit dem folgenden Code.

```
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;
}
```

In diesem Code verwenden wir die `sql` Tag-Vorlage, um eine SQL-Anweisung zu schreiben, an die wir zur Laufzeit sicher einen dynamischen Wert übergeben können. `createPgStatement`kann bis zu zwei SQL-Anfragen gleichzeitig annehmen. Wir verwenden das, um eine Anfrage für uns `todo` und eine weitere für unsere zu senden`tasks`. Sie hätten dies mit einer `JOIN` Anweisung oder einer anderen Methode tun können. Die Idee ist, Ihre eigene SQL-Anweisung schreiben zu können, um Ihre Geschäftslogik zu implementieren. Gehen Sie wie folgt vor, um die Abfrage im Query-Editor zu verwenden.

```
query TodoAndTasks {
  getTodosAndTasks(id: 2) {
    id
    due
    description
    tasks {
      items {
        id
        description
      }
    }
  }
}
```

Löschen Sie Ihren Cluster

> **Important**
>
> Das Löschen eines Clusters ist dauerhaft. Überprüfen Sie Ihr Projekt gründlich, bevor Sie diese Aktion ausführen.

Um Ihren Cluster zu löschen:

```
$ aws rds delete-db-cluster \
    --db-cluster-identifier appsync-tutorial \
    --skip-final-snapshot
```

VTL-Resolver-Tutorials für AWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Datenquellen und Resolver werden verwendet, AWS AppSync um GraphQL-Anfragen zu übersetzen und Informationen aus Ihren Ressourcen abzurufen. AWS AWS AppSync unterstützt die automatische Bereitstellung und Verbindungen mit bestimmten Datenquellentypen. AWS AppSync unterstützt AWS Lambda auch Amazon DynamoDB, relationale Datenbanken (Amazon Aurora Serverless), Amazon OpenSearch Service und HTTP-Endpunkte als Datenquellen. Sie können eine GraphQL-API mit Ihren vorhandenen AWS Ressourcen verwenden oder Datenquellen und Resolver von Grund auf neu erstellen. Die folgenden Abschnitte sollen einige der gängigsten GraphQL-Anwendungsfälle in Form von Tutorials erläutern.

AWS AppSync verwendet in Apache Velocity Template Language (VTL) geschriebene Mapping-Vorlagen für Resolver. Weitere Informationen zur Verwendung von Mapping-Vorlagen finden Sie in der Referenz zu Resolver-Mapping-Vorlagen. Weitere Informationen zur Arbeit mit VTL finden Sie im Programmierleitfaden für Resolver-Mapping-Vorlagen.

AWS AppSync unterstützt die automatische Bereitstellung von DynamoDB-Tabellen aus einem GraphQL-Schema, wie unter Bereitstellung aus Schema (optional) und Beispielschema starten beschrieben. Sie können auch aus einer vorhandenen DynamoDB-Tabelle importieren, wodurch das Schema erstellt wird und Resolver verbunden werden. Dies wird unter Import aus Amazon DynamoDB (optional) beschrieben.

Themen

- Erstellen einer einfachen Post-Anwendung mit DynamoDB-Resolvern
- Verwenden von Resolvern in AWS LambdaAWS AppSync
- Verwendung von Amazon OpenSearch Service-Resolvern in AWS AppSync
- Verwendung lokaler Resolver in AWS AppSync
- Kombinieren von GraphQL-Resolvern in AWS AppSync
- Verwenden von DynamoDB-Batchoperationen in AWS AppSync
- Durchführen von DynamoDB-Transaktionen in AWS AppSync

- Verwendung von HTTP-Resolvern in AWS AppSync
- Verwenden von Aurora Serverless v2 mit AWS AppSync
- Verwendung von Pipeline-Resolvern in AWS AppSync
- Verwenden von Delta Sync-Vorgängen für versionierte Datenquellen in AWS AppSync

Erstellen einer einfachen Post-Anwendung mit DynamoDB-Resolvern

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Dieses Tutorial zeigt, wie Sie Ihre eigenen Amazon DynamoDB-Tabellen zu einer GraphQL-API bringen AWS AppSync und sie mit dieser verbinden können.

Sie können DynamoDB-Ressourcen in Ihrem Namen AWS AppSync bereitstellen lassen. Alternativ können Sie auch vorhandene Tabellen mit einem GraphQL-Schema verbinden, indem Sie eine Datenquelle und einen Resolver erstellen. In beiden Fällen können Sie Lese- und Schreibvorgänge in Ihrer DynamoDB-Datenbank über GraphQL-Anweisungen ausführen – und Echtzeit-Daten abonnieren.

Es müssen bestimmte Konfigurationsschritte ausgeführt werden, damit GraphQL-Anweisungen in DynamoDB-Operationen und die Antworten wieder in GraphQL übersetzt werden können. In diesem Tutorial wird der Konfigurationsprozess anhand mehrerer Szenarien und Datenzugriffsmuster aus der Praxis veranschaulicht.

DynamoDB-Tabellen einrichten

Um mit diesem Tutorial zu beginnen, müssen Sie zunächst die folgenden Schritte ausführen, um Ressourcen bereitzustellen AWS .

1. Stellen Sie AWS Ressourcen mithilfe der folgenden AWS CloudFormation Vorlage in der CLI bereit:

```
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
```

Alternativ können Sie den folgenden AWS CloudFormation Stack in der Region US-West 2 (Oregon) in Ihrem AWS Konto starten.

Dadurch wird Folgendes erstellt:

- Eine DynamoDB-Tabelle namens`AppSyncTutorial-Post`, die Daten enthalten `Post` wird.
- Eine IAM-Rolle und die zugehörige, von IAM verwaltete Richtlinie, um die Interaktion mit der Tabelle AWS AppSync zu ermöglichen. `Post`

2. Wenn Sie weitere Informationen zum Stack und den erstellten Ressourcen benötigen, führen Sie den folgenden CLI-Befehl aus:

```
aws cloudformation describe-stacks --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

3. Wenn Sie die Ressourcen später löschen möchten, führen Sie folgenden Befehl aus:

```
aws cloudformation delete-stack --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

Erstellen Sie Ihre GraphQL-API

Um die GraphQL-API zu erstellen in AWS AppSync:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
   - Wählen Sie im APIs Dashboard Create API aus.
2. Wählen Sie im Fenster Passen Sie Ihre API an oder importieren Sie aus Amazon DynamoDB die Option Von Grund auf neu erstellen.
   - Wählen Sie rechts neben demselben Fenster Start aus.
3. Stellen Sie im Feld API-Name den Namen der API auf ein`AWSAppSyncTutorial`.
4. Wählen Sie Erstellen aus.

Die AWS AppSync Konsole erstellt mithilfe des API-Schlüsselauthentifizierungsmodus eine neue GraphQL-API für Sie. Sie können in der Konsole den Rest der GraphQL-API einrichten und im weiteren Verlauf dieses Tutorials abfragen.

Definition einer grundlegenden Post-API

Nachdem Sie eine AWS AppSync GraphQL-API erstellt haben, können Sie ein grundlegendes Schema einrichten, das das grundlegende Erstellen, Abrufen und Löschen von Post-Daten ermöglicht.

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
   - Wählen Sie im APIs Dashboard die API aus, die Sie gerade erstellt haben.
2. Wählen Sie in der Seitenleiste Schema aus.
   - Ersetzen Sie im Schemabereich den Inhalt durch den folgenden Code:

```
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. Wählen Sie Speichern.

Dieses Schema definiert einen `Post`-Typ und Operationen zum Hinzufügen und Abrufen von `Post`-Objekten.

Konfiguration der Datenquelle für die DynamoDB-Tabellen

Verknüpfen Sie als Nächstes die im Schema definierten Abfragen und Mutationen mit der `AppSyncTutorial-Post` DynamoDB-Tabelle.

Zunächst AWS AppSync müssen Sie sich Ihrer Tabellen bewusst sein. Hierzu richten Sie eine Datenquelle in AWS AppSync ein:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.
   b. Wählen Sie in der Seitenleiste Datenquellen aus.
2. Klicken Sie auf Create data source.
   a. Geben Sie als Namen der Datenquelle in `PostDynamoDBTable` ein.
   b. Wählen Sie als Datenquellentyp Amazon DynamoDB-Tabelle aus.
   c. Wählen Sie für Region US-WEST-2 aus.
   d. Wählen Sie als Tabellenname die DynamoDB-Tabelle AppSyncTutorial-Post aus.
   e. Erstellen Sie eine neue IAM-Rolle (empfohlen) oder wählen Sie eine vorhandene Rolle aus, die über die IAM-Berechtigung verfügt. `lambda:invokeFunction` Für bestehende Rollen ist eine Vertrauensrichtlinie erforderlich, wie im Abschnitt Eine Datenquelle anhängen beschrieben.

      Im Folgenden finden Sie ein Beispiel für eine IAM-Richtlinie, die über die erforderlichen Berechtigungen für die Ausführung von Vorgängen auf der Ressource verfügt:

JSON

```
{
     "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. Wählen Sie Erstellen aus.

Den AddPost-Resolver einrichten (DynamoDB) PutItem

Nachdem AWS AppSync Sie die DynamoDB-Tabelle erkannt haben, können Sie sie mit einzelnen Abfragen und Mutationen verknüpfen, indem Sie Resolver definieren. Der erste Resolver, den Sie erstellen, ist der `addPost` Resolver, mit dem Sie einen Beitrag in der `AppSyncTutorial-Post` DynamoDB-Tabelle erstellen können.

Resolver bestehen aus folgenden Komponenten:

- Die Stelle im GraphQL-Schema, an der der Resolver angefügt werden soll. In diesem Fall richten Sie einen Resolver im Feld `addPost` im Typ `Mutation` ein. Diesen Resolver können Sie mit `mutation { addPost(...){...} }` aufrufen.
- Die Datenquelle für diesen Resolver. In diesem Fall soll die zuvor definierte Datenquelle `PostDynamoDBTable` verwendet werden, damit Sie Einträge in der DynamoDB-Tabelle `AppSyncTutorial-Post` hinzufügen können.
- Die Zuweisungsvorlage für Anforderungen. Der Zweck der Vorlage für die Anforderungszuweisung besteht darin, die eingehende Anfrage des Aufrufers in Anweisungen für die Ausführung gegen DynamoDB AWS AppSync zu übersetzen.

- Die Zuweisungsvorlage für Antworten. Eine Zuweisungsvorlage für Antworten übersetzt die Antwort aus DynamoDB zurück in eine Eingabe, die GraphQL erwartet. Dies ist nützlich, wenn sich das Format der Daten in DynamoDB vom `Post`-Typ in GraphQL unterscheidet. In diesem Fall stimmt die Form jedoch überein, daher werden die Daten direkt übergeben.

Richten Sie den Resolver wie folgt ein:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Konsole. AppSync
   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.
   b. Wählen Sie in der Seitenleiste Datenquellen aus.
2. Klicken Sie auf Create data source.
   a. Geben Sie als Namen der Datenquelle in `PostDynamoDBTable` ein.
   b. Wählen Sie als Datenquellentyp Amazon DynamoDB-Tabelle aus.
   c. Wählen Sie für Region US-WEST-2 aus.
   d. Wählen Sie als Tabellenname die DynamoDB-Tabelle AppSyncTutorial-Post aus.
   e. Erstellen Sie eine neue IAM-Rolle (empfohlen) oder wählen Sie eine vorhandene Rolle aus, die über die IAM-Berechtigung verfügt. `lambda:invokeFunction` Für bestehende Rollen ist eine Vertrauensrichtlinie erforderlich, wie im Abschnitt Eine Datenquelle anhängen beschrieben.

      Im Folgenden finden Sie ein Beispiel für eine IAM-Richtlinie, die über die erforderlichen Berechtigungen für die Ausführung von Vorgängen auf der Ressource verfügt:

      JSON

```
{
     "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. Wählen Sie Erstellen aus.
4. Wählen Sie die Registerkarte Schema aus.
5. Suchen Sie rechts im Bereich Data types (Datentypen) das Feld addPost des Typs Mutation und wählen Sie dann Attach (Anhängen).
6. Wählen Sie im Menü Aktion die Option Laufzeit aktualisieren und anschließend Unit Resolver (nur VTL) aus.
7. Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
8. Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

```
{
    "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 }
    }
}
```

Hinweis: Für alle Schlüssel und Attributwerte wird ein Typ angegeben. So legen Sie beispielsweise das Feld `author` auf `{ "S" : "${context.arguments.author}" }` fest. Der `S` Teil gibt an AWS AppSync und DynamoDB an, dass der Wert ein Zeichenkettenwert sein wird. Der tatsächliche Wert wird aus dem `author`-Argument eingefügt. Das Feld `version` ist ein Zahlenfeld, da als Typ `N` verwendet wird. Schließlich initialisieren Sie auch die Felder `ups`, `downs` und `version`.

Für dieses Tutorial haben Sie angegeben, dass der `ID!` GraphQL-Typ, der das neue Element, das in DynamoDB eingefügt wird, indexiert, Teil der Client-Argumente ist. AWS AppSync enthält ein Hilfsprogramm zur automatischen ID-Generierung namens`$utils.autoId()`, das Sie auch in der Form von hätten verwenden können. `"id" : { "S" : "${$utils.autoId()}" }` Dann würden Sie `id: ID!` nicht in der Schemadefinition von `addPost()` angeben, sondern die ID würde automatisch eingefügt. Sie werden diese Technik in diesem Tutorial nicht verwenden, sollten sie jedoch als bewährte Methode beim Schreiben in DynamoDB-Tabellen betrachten.

Weitere Informationen zu Lambda-Zuweisungsvorlagen finden Sie in der Referenzdokumentation Übersicht über die Resolver-Zuweisungsvorlage. Weitere Informationen zur GetItem Anforderungszuordnung finden Sie in der GetItemReferenzdokumentation. Weitere Informationen zu Typen enthält die Referenzdokumentation Typsystem (Anforderungszuweisung).

9. Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
$utils.toJson($context.result)
```

Hinweis: Da das Format der Daten in der Tabelle `AppSyncTutorial-Post` genau mit dem Format des `Post`-Typs in GraphQL übereinstimmt, übergibt die Zuweisungsvorlage für die Antwort die Ergebnisse einfach direkt. Beachten Sie auch, dass in allen Beispielen in diesem Tutorial die gleiche Zuweisungsvorlage für Antworten verwendet wird, sodass Sie nur eine Datei erstellen müssen.

10. Wählen Sie Speichern.

Aufrufen der API zum Hinzufügen eines Posts

Jetzt, da der Resolver eingerichtet ist, AWS AppSync kann er eine eingehende `addPost` Mutation in eine PutItem DynamoDB-Operation übersetzen. Sie können jetzt eine Mutation ausführen, um Inhalte in der Tabelle hinzuzufügen.

- Wählen Sie die Registerkarte Queries aus.
- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein:

```
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
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Die Ergebnisse des neu erstellten Posts sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

Folgendes wurden ausgeführt:

- AWS AppSync hat eine Mutationsanfrage erhalten. `addPost`

- AWS AppSync hat die Anfrage und die Vorlage für die Anforderungszuweisung entgegengenommen und ein Dokument zur Anforderungszuweisung generiert. Dies sieht wie folgt aus:

```
{
    "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 hat das Anforderungszuordnungsdokument verwendet, um eine `PutItem` DynamoDB-Anforderung zu generieren und auszuführen.
- AWS AppSync nahm die Ergebnisse der `PutItem` Anfrage und konvertierte sie wieder in GraphQL-Typen.

```
{
    "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
}
```

- Dann wurden sie über das Antwortzuweisungsdokument unverändert weiter übergeben.
- Das neu erstellte Objekt wurde in der GraphQL-Antwort zurückgegeben.

Einrichtung des GetPost-Resolvers (DynamoDB) GetItem

Jetzt, da Sie der `AppSyncTutorial-Post` DynamoDB-Tabelle Daten hinzufügen können, müssen Sie die `getPost` Abfrage so einrichten, dass sie diese Daten aus der `AppSyncTutorial-Post` Tabelle abrufen kann. Hierzu richten Sie einen weiteren Resolver ein.

- Wählen Sie die Registerkarte Schema aus.
- Suchen Sie rechts im Bereich Data types (Datentypen) das Feld getPost des Typs Query (Abfrage) und wählen Sie dann Attach (Anhängen).
- Wählen Sie im Menü Aktion die Option Laufzeit aktualisieren und anschließend Unit Resolver (nur VTL) aus.
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

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

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
$utils.toJson($context.result)
```

- Wählen Sie Speichern.

Aufrufen der API zum Abrufen eines Posts

Jetzt ist der Resolver eingerichtet und AWS AppSync weiß, wie man eine eingehende `getPost` Abfrage in eine `GetItem` DynamoDB-Operation übersetzt. Sie können jetzt eine Abfrage ausführen, um den zuvor erstellten Post abzurufen.

- Wählen Sie die Registerkarte Queries aus.
- Fügen Sie im Bereich Abfragen Folgendes ein:

```
query getPost {
  getPost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Der von DynamoDB abgerufene Beitrag sollte im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

Folgendes wurden ausgeführt:

- AWS AppSync hat eine `getPost` Abfrageanforderung erhalten.
- AWS AppSync hat die Anfrage und die Vorlage für die Anforderungszuweisung entgegengenommen und ein Dokument zur Anforderungszuweisung generiert. Dies sieht wie folgt aus:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "id" : { "S" : "123" }
    }
}
```

- AWS AppSync hat das Anforderungszuordnungsdokument verwendet, um eine GetItem DynamoDB-Anforderung zu generieren und auszuführen.
- AWS AppSync nahm die Ergebnisse der `GetItem` Anfrage und konvertierte sie wieder in GraphQL-Typen.

```
{
    "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
}
```

- Dann wurden sie über das Antwortzuweisungsdokument unverändert weiter übergeben.
- Das abgerufene Objekt wurde in der Antwort zurückgegeben.

Nehmen Sie alternativ das folgende Beispiel:

```
query getPost {
  getPost(id:123) {
    id
    author
    title
  }
}
```

Wenn Ihre `getPost` Abfrage nur das`id`, und benötigt`author`, können Sie Ihre Anforderungszuordnungsvorlage so ändern`title`, dass Projektionsausdrücke verwendet

werden, um nur die gewünschten Attribute aus Ihrer DynamoDB-Tabelle anzugeben, um unnötige Datenübertragungen von DynamoDB zu zu zu vermeiden. AWS AppSync Die Vorlage für die Anforderungszuweisung könnte beispielsweise wie der folgende Ausschnitt aussehen:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },
    "projection" : {
     "expression" : "#author, id, title",
     "expressionNames" : { "#author" : "author"}
    }
}
```

Eine UpdatePost-Mutation erstellen (DynamoDB) UpdateItem

Bisher können Sie `Post` Objekte in DynamoDB erstellen und abrufen. Als Nächstes richten Sie eine neue Mutation ein, um ein Objekt zu aktualisieren. Dazu verwenden Sie die UpdateItem DynamoDB-Operation.

- Wählen Sie die Registerkarte Schema aus.
- Ändern Sie im Schemabereich den `Mutation` Typ, um eine neue `updatePost` Mutation hinzuzufügen, wie folgt:

```
type Mutation {
    updatePost(
        id: ID!,
        author: String!,
        title: String!,
        content: String!,
        url: String!
    ): Post
    addPost(
        author: String!
        title: String!
        content: String!
        url: String!
    ): Post!
}
```

- Wählen Sie Speichern.
- Suchen Sie rechts im Bereich Data types (Datentypen) das neue Feld updatePost des Typs Mutation und wählen Sie dann Attach (Anhängen).
- Wählen Sie im Menü Aktion die Option Laufzeit aktualisieren und anschließend Unit Resolver (nur VTL) aus.
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

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

Hinweis: Dieser Resolver verwendet DynamoDB UpdateItem, was sich erheblich von der Operation unterscheidet. PutItem Anstatt das gesamte Element zu schreiben, bitten Sie DynamoDB lediglich, bestimmte Attribute zu aktualisieren. Dies erfolgt mithilfe von DynamoDB-Aktualisierungsausdrücken. Der Ausdruck selbst wird im Feld `expression` im Abschnitt `update` angegeben. Er gibt an, dass die Attribute `author`, `title`, `content` und "url" festgelegt werden sollen und dann das Feld `version` inkrementiert werden soll. Die zu verwendenden Werte werden nicht im Ausdruck selbst angegeben. Der Ausdruck enthält Platzhalter, deren Namen mit einem Doppelpunkt beginnen. Diese werden dann im Feld `expressionValues` definiert.

Schließlich hat DynamoDB reservierte Wörter, die nicht in der vorkommen dürfen. `expression` `url` ist beispielsweise ein reservierter Begriff. Zum Aktualisieren des Felds `url` können Sie daher Platzhalter für Namen verwenden und sie im Feld `expressionNames` definieren.

Weitere Informationen zur `UpdateItem` Anforderungszuordnung finden Sie in der UpdateItemReferenzdokumentation. Weitere Informationen zum Schreiben von Aktualisierungsausdrücken finden Sie in der DynamoDB-Dokumentation UpdateExpressions .

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
$utils.toJson($context.result)
```

Aufrufen der API zum Aktualisieren eines Posts

Jetzt ist der Resolver eingerichtet und AWS AppSync weiß, wie man eine eingehende `update` Mutation in eine `Update` DynamoDB-Operation übersetzt. Sie können jetzt eine Mutation ausführen, um das zuvor geschriebene Element zu aktualisieren.

- Wählen Sie die Registerkarte Queries aus.
- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein. Außerdem müssen Sie das `id`-Argument mit dem zuvor notierten Wert aktualisieren.

```
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
  }
```

```
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Der aktualisierte Beitrag in DynamoDB sollte im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

In diesem Beispiel wurden die `downs` Felder `ups` und nicht geändert, da die Vorlage für die Anforderungszuweisung DynamoDB nicht AWS AppSync aufforderte, irgendetwas mit diesen Feldern zu tun. Außerdem wurde das `version` Feld um 1 erhöht, weil Sie DynamoDB gebeten AWS AppSync haben, dem Feld 1 hinzuzufügen. `version`

Den updatePost Resolver (DynamoDB) ändern UpdateItem

Dies war ein erfolgreicher Start für die `updatePost`-Mutation, es gibt aber zwei wesentliche Probleme:

- Auch wenn nur ein einzelnes Feld aktualisiert werden soll, müssen alle Felder aktualisiert werden.
- Wenn zwei Benutzer das Objekt ändern, können Informationen verloren gehen.

Um diese Probleme zu beheben, ändern wir die `updatePost`-Mutation so, dass nur Argumente geändert werden, die in der Anforderung angegeben wurden. Dann fügen wir der `UpdateItem`-Operation eine Bedingung hinzu.

1. Wählen Sie die Registerkarte Schema aus.

2. Ändern Sie im Bereich Schema das Feld `updatePost` im Typ `Mutation`, um die Ausrufungszeichen aus den Argumenten `author`, `title`, `content` und `url` zu entfernen. Achten Sie dabei darauf, das Feld `id` nicht zu ändern. Hierdurch werden die Argumente optional. Fügen Sie außerdem ein neues erforderliches `expectedVersion`-Argument hinzu.

```
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. Wählen Sie Speichern.
4. Suchen Sie im Bereich Data types (Datentypen) auf der rechten Seite das Feld updatePost des Typs Mutation.
5. Wählen Sie, ob der vorhandene Resolver geöffnet PostDynamoDBTablewerden soll.
6. Ändern Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) folgendermaßen die Anforderungszuweisungsvorlage:

```
{
    "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. Wählen Sie Speichern.

Diese Vorlage ist eines der komplexeren Beispiele. Es zeigt die Leistung und Flexibilität der Zuweisungsvorlagen. Es durchläuft alle Argumente und springt dann zu `id` und `expectedVersion`. Wenn das Argument auf etwas gesetzt ist, fordert AWS AppSync es DynamoDB auf, dieses Attribut für das Objekt in DynamoDB zu aktualisieren. Wenn das Attribut auf Null gesetzt ist, fordert AWS AppSync es DynamoDB auf, dieses Attribut aus dem Post-Objekt zu entfernen. Wenn ein Argument nicht angegeben wurde, wird das Attribut nicht geändert. Außerdem wird das Feld `version` inkrementiert.

Außerdem gibt es einen neuen `condition`-Abschnitt. Ein Bedingungsausdruck ermöglicht es Ihnen AWS AppSync , DynamoDB anhand des Status des Objekts, das sich bereits vor der Ausführung des Vorgangs in DynamoDB befand, mitzuteilen, ob die Anforderung erfolgreich sein soll oder nicht. In diesem Fall möchten Sie, dass die `UpdateItem` Anfrage nur erfolgreich ist, wenn das `version` Feld des Elements, das sich derzeit in DynamoDB befindet, genau mit dem `expectedVersion` Argument übereinstimmt.

Weitere Informationen zu Bedingungsausdrücken finden Sie in der Bedingungsausdrücke-Referenzdokumentation.

Aufrufen der API zum Aktualisieren eines Posts

Aktualisieren wir nun unser `Post`-Objekt mit dem neuen Resolver:

- Wählen Sie die Registerkarte Queries aus.
- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein. Außerdem müssen Sie das `id`-Argument mit dem zuvor notierten Wert aktualisieren.

```
mutation updatePost {
  updatePost(
    id:123
    title: "An empty story"
    content: null
    expectedVersion: 2
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
```

```
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Der aktualisierte Beitrag in DynamoDB sollte im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

In dieser Anfrage haben Sie DynamoDB gebeten AWS AppSync , nur das `content` Feld `title` und zu aktualisieren. Alle anderen Felder bleiben unverändert (nur das Feld `version` wird inkrementiert). Das `title`-Attribut wurde auf einen neuen Wert festgelegt und das `content`-Attribut wurde aus dem Post entfernt. Die Felder `author`, `url`, `ups` und `downs` blieben unverändert.

Führen Sie die Mutationsanforderung erneut aus und belassen Sie sie dabei unverändert. Es wird eine Antwort ähnlich der folgenden angezeigt:

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

Die Anforderung schlägt fehl, weil der Bedingungsausdruck als "false" ausgewertet wird:

- Als Sie die Anfrage zum ersten Mal ausgeführt haben, war der Wert des `version` Felds des Beitrags in DynamoDB2, der `expectedVersion` dem Argument entsprach. Die Anfrage war erfolgreich, was bedeutete, dass das `version` Feld in DynamoDB auf inkrementiert wurde. `3`
- Als Sie die Anfrage zum zweiten Mal ausgeführt haben, war der Wert des `version` Felds des Beitrags in DynamoDB3, der nicht mit dem `expectedVersion` Argument übereinstimmte.

Dieses Muster wird normalerweise als optimistische Sperre bezeichnet.

Ein Merkmal eines AWS AppSync DynamoDB-Resolvers besteht darin, dass er den aktuellen Wert des Post-Objekts in DynamoDB zurückgibt. Sie finden dies im Feld `data` im Abschnitt `errors` der GraphQL-Antwort. Ihre Anwendung kann anhand dieser Informationen entscheiden, wie weiter vorgegangen werden soll. In diesem Fall können Sie sehen, dass das `version` Feld des Objekts in DynamoDB auf `3` gesetzt ist. Sie könnten also das `expectedVersion` Argument einfach auf aktualisieren `3` und die Anfrage wäre wieder erfolgreich.

Weitere Informationen zum Behandeln von Fehlern bei Bedingungsprüfungen finden Sie in der Zuweisungsvorlagen-Referenzdokumentation für Bedingungsausdrücke.

UpvotePost- und DownvotePost-Mutationen erstellen (DynamoDB) UpdateItem

Der Typ `Post` verfügt über die Felder `ups` und `downs`, mit denen Upvotes und Downvotes erfasst werden können. In der API können sie allerdings derzeit nicht verwendet werden. Fügen Sie daher einige Mutationen hinzu, um Upvotes und Downvotes für die Posts zu ermöglichen.

- Wählen Sie die Registerkarte Schema aus.
- Ändern Sie im Schemabereich den `Mutation` Typ, um neue Mutationen hinzuzufügen, wie folgt: `upvotePost` `downvotePost`

```
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!
}
```

- Wählen Sie Speichern.
- Suchen Sie rechts im Bereich Data types (Datentypen) das neue Feld upvotePost des Typs Mutation und wählen Sie dann Attach (Anhängen).
- Wählen Sie im Menü Aktion die Option Laufzeit aktualisieren und anschließend Unit Resolver (nur VTL) aus.
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

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

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
$utils.toJson($context.result)
```

- Wählen Sie Speichern.
- Suchen Sie rechts im Bereich Data types (Datentypen) das neue Feld `downvotePost` des Typs Mutation und wählen Sie dann Attach (Anhängen).
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

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

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
$utils.toJson($context.result)
```

- Wählen Sie Speichern.

Aufrufen der API für ein Upvote oder Downvote eines Posts

Jetzt wurden die neuen Resolver eingerichtet, AWS AppSync weiß, wie man eine eingehende Operation `upvotePost` oder eine `downvote` Mutation in eine UpdateItem DynamoDB-Operation übersetzt. Sie können jetzt Mutationen ausführen, um Upvotes oder Downvotes für den zuvor erstellten Post auszuführen.

- Wählen Sie die Registerkarte Queries aus.
- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein. Außerdem müssen Sie das `id`-Argument mit dem zuvor notierten Wert aktualisieren.

```
mutation votePost {
  upvotePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Der Beitrag wurde in DynamoDB aktualisiert und sollte im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) einige weitere Male. Jedes Mal, wenn Sie die Abfrage ausführen, sollten die Felder ups und `version` um 1 erhöht werden.
- Ändern Sie folgendermaßen die Abfrage zum Aufrufen der `downvotePost`-Mutation:

```
mutation votePost {
  downvotePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche). Jedes Mal, wenn Sie die Abfrage ausführen, sollten jetzt die Felder downs und `version` um 1 erhöht werden.

```
{
  "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
```

```
    }
  }
}
```

Den DeletePost-Resolver einrichten (DynamoDB) DeleteItem

Mit der nächsten Mutation, die Sie einrichten möchten, soll ein Post gelöscht werden. Dazu verwenden Sie die `DeleteItem` DynamoDB-Operation.

- Wählen Sie die Registerkarte Schema aus.
- Ändern Sie im Schemabereich den `Mutation` Typ, um eine neue `deletePost` Mutation hinzuzufügen, wie folgt:

```
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!
}
```

  Dieses Mal wurde das Feld `expectedVersion` als optional konfiguriert. Dies wird später erläutert, wenn Sie die Anforderungszuweisungsvorlage hinzufügen.

- Wählen Sie Speichern.
- Suchen Sie rechts im Bereich Data types (Datentypen) das neue Feld delete des Typs Mutation und wählen Sie dann Attach (Anhängen).

- Wählen Sie im Menü Aktion die Option Laufzeit aktualisieren und anschließend Unit Resolver (nur VTL) aus.
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

```
{
    "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
}
```

  Hinweis: Das Argument `expectedVersion` ist ein optionales Argument. Wenn der Aufrufer ein `expectedVersion` Argument in der Anfrage festlegt, fügt die Vorlage eine Bedingung hinzu, die nur dann den Erfolg der `DeleteItem` Anfrage ermöglicht, wenn das Element bereits gelöscht wurde oder wenn das `version` Attribut des Beitrags in DynamoDB genau dem entspricht. `expectedVersion` Wenn es ausgelassen wird, wird kein Bedingungsausdruck für die `DeleteItem`-Anforderung angegeben. Es ist erfolgreich`version`, unabhängig vom Wert von oder ob das Element in DynamoDB vorhanden ist oder nicht.

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
$utils.toJson($context.result)
```

  Hinweis: Auch wenn Sie ein Element löschen, können Sie das gelöschte Element zurückgeben, sofern es nicht bereits vorher gelöscht wurde.

- Wählen Sie Speichern.

Weitere Informationen zur `DeleteItem` Anforderungszuordnung finden Sie in der DeleteItemReferenzdokumentation.

Aufrufen der API zum Löschen eines Posts

Jetzt ist der Resolver eingerichtet und AWS AppSync weiß, wie man eine eingehende `delete` Mutation in eine `DeleteItem` DynamoDB-Operation übersetzt. Sie können jetzt eine Mutation ausführen, um Inhalte in der Tabelle zu löschen.

- Wählen Sie die Registerkarte Queries aus.
- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein. Außerdem müssen Sie das `id`-Argument mit dem zuvor notierten Wert aktualisieren.

```
mutation deletePost {
  deletePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Der Beitrag wurde aus DynamoDB gelöscht. Beachten Sie, dass der Wert des Elements AWS AppSync zurückgegeben wird, das aus DynamoDB gelöscht wurde und der im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden sollte. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

Der Wert wird nur zurückgegeben, wenn dieser Aufruf von `deletePost` derjenige war, der ihn tatsächlich aus DynamoDB gelöscht hat.

- Wählen Sie erneut Execute query (Abfrage ausführen).
- Der Aufruf ist weiterhin erfolgreich, es wird jedoch kein Wert zurückgegeben.

```
{
  "data": {
    "deletePost": null
  }
}
```

Löschen wir jetzt einen Post. Dieses Mal geben wir aber einen `expectedValue` an. Zuerst müssen Sie jedoch einen neuen Post erstellen, da der Post, mit dem Sie bisher gearbeitet haben, gerade gelöscht wurde.

- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein:

```
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
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Die Ergebnisse des neu erstellten Posts sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Notieren Sie die `id` des neu erstellten Objekts, da diese gleich benötigt wird. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

Jetzt versuchen wir, diesen Post zu löschen, geben aber einen falschen Wert für `expectedVersion` an:

- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein. Außerdem müssen Sie das `id`-Argument mit dem zuvor notierten Wert aktualisieren.

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 9999
  ) {
    id
    author
    title
    content
```

```
    url
    ups
    downs
    version
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).

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

Die Anforderung ist fehlgeschlagen, weil der Bedingungsausdruck als falsch ausgewertet wird: Der Wert für `version` des Beitrags in DynamoDB entspricht nicht dem in den Argumenten `expectedValue` angegebenen Wert. Der aktuelle Wert des Objekts wird im Feld `data` im Abschnitt `errors` der GraphQL-Antwort zurückgegeben.

- Wiederholen Sie die Anforderung und korrigieren Sie dabei `expectedVersion`:

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Diesmal ist die Anfrage erfolgreich und der Wert, der aus DynamoDB gelöscht wurde, wird zurückgegeben:

```
{
  "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
    }
  }
}
```

- Wählen Sie erneut Execute query (Abfrage ausführen).
- Der Aufruf ist immer noch erfolgreich, aber diesmal wird kein Wert zurückgegeben, da der Beitrag bereits in DynamoDB gelöscht wurde.

```
{
  "data": {
    "deletePost": null
  }
}
```

Einrichten des allPost-Resolvers (DynamoDB Scan)

Bisher ist die API nur nützlich, wenn Sie die `id` eines jeden Beitrags kennen, den Sie betrachten möchten. Fügen wir jetzt einen neuen Resolver hinzu, der alle Posts in der Tabelle zurückgibt.

- Wählen Sie die Registerkarte Schema aus.
- Ändern Sie im Schemabereich den `Query` Typ, um eine neue `allPost` Abfrage hinzuzufügen, wie folgt:

```
type Query {
    allPost(count: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
}
```

- Fügen Sie einen neuen `PaginationPosts`-Typ hinzu:

```
type PaginatedPosts {
    posts: [Post!]!
    nextToken: String
}
```

- Wählen Sie Speichern.
- Suchen Sie rechts im Bereich Data types (Datentypen) das neue Feld allPost des Typs Query (Abfrage) und wählen Sie dann Attach (Anhängen).
- Wählen Sie im Menü Aktion die Option Laufzeit aktualisieren und anschließend Unit Resolver (nur VTL) aus.
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.

- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

```
{
    "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
}
```

  Dieser Resolver verfügt über zwei optionale Argumente: `count` gibt die maximale Anzahl von Elementen an, die in einem Aufruf zurückgegeben werden sollen, und mit `nextToken` kann der nächste Ergebnissatz abgerufen werden. (Später wird erläutert, woher der Wert für `nextToken` stammt.)

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
{
    "posts": $utils.toJson($context.result.items)
    #if( ${context.result.nextToken} )
        ,"nextToken": $util.toJson($context.result.nextToken)
    #end
}
```

  Hinweis: Diese Zuweisungsvorlage für Antworten unterscheidet sich von allen bisher betrachteten Zuweisungsvorlagen. Das Ergebnis der `allPost`-Abfrage ist `PaginatedPosts` mit einer Liste von Posts und einem Paginierungs-Token. Die Form dieses Objekts unterscheidet sich von dem, was vom AWS AppSync DynamoDB-Resolver zurückgegeben wird: Die Liste der Beiträge wird `items` in den AWS AppSync DynamoDB-Resolver-Ergebnissen aufgerufen, aber sie wird aufgerufen. `posts` `PaginatedPosts`

- Wählen Sie Speichern.

Weitere Informationen zur Zuordnung von `Scan` Anfragen finden Sie in der Referenzdokumentation zum Scannen.

Aufrufen der API zum Scannen aller Posts

Jetzt ist der Resolver eingerichtet und AWS AppSync weiß, wie man eine eingehende `allPost` Abfrage in eine `Scan` DynamoDB-Operation übersetzt. Sie können jetzt die Tabelle scannen, um alle Posts abzurufen.

Bevor Sie dies ausprobieren können, muss allerdings die Tabelle mit Daten gefüllt werden, da Sie alle Daten gelöscht haben, mit denen Sie bislang gearbeitet haben.

- Wählen Sie die Registerkarte Queries aus.
- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein:

```
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 }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).

Scannen wir jetzt die Tabelle so, dass jeweils fünf Ergebnisse zurückgegeben werden.

- Fügen Sie im Bereich Queries (Abfragen) die folgende Abfrage ein:

```
query allPost {
  allPost(count: 5) {
```

```
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Die ersten fünf Posts sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

Sie haben fünf Ergebnisse und ein `nextToken`, das zum Abrufen des nächsten Ergebnissatzes verwendet werden kann.

- Aktualisieren Sie die `allPost`-Abfrage, um das `nextToken` aus dem vorherigen Ergebnissatz einzuschließen:

```
query allPost {
  allPost(
    count: 5
    nextToken:
 "eyJ2ZXJzaW9uIjoxLCJ0b2tlbiI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eThOSmRvcG9ad2RHYjI
  ) {
    posts {
      id
      author
    }
    nextToken
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Die verbleibenden vier Posts sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Dieser Ergebnissatz enthält kein `nextToken`, weil Sie alle neun Posts durchlaufen haben und keine mehr übrig sind. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

Den allPostsBy Author Resolver einrichten (DynamoDB Query)

Sie können DynamoDB nicht nur nach allen Beiträgen durchsuchen, sondern auch DynamoDB abfragen, um Beiträge abzurufen, die von einem bestimmten Autor erstellt wurden. Die zuvor erstellte DynamoDB-Tabelle hat bereits einen `GlobalSecondaryIndex` Aufruf, den `author-index` Sie mit einer `Query` DynamoDB-Operation verwenden können, um alle Beiträge abzurufen, die von einem bestimmten Autor erstellt wurden.

- Wählen Sie die Registerkarte Schema aus.
- Ändern Sie im Schemabereich den `Query` Typ, um eine neue `allPostsByAuthor` Abfrage hinzuzufügen, wie folgt:

```
type Query {
    allPostsByAuthor(author: String!, count: Int, nextToken: String): PaginatedPosts!
    allPost(count: Int, nextToken: String): PaginatedPosts!
    getPost(id: ID): Post
}
```

  Hinweis: Hier wird der gleiche `PaginatedPosts`-Typ wie in der `allPost`-Abfrage verwendet.
- Wählen Sie Speichern.
- Suchen Sie im Bereich Datentypen auf der rechten Seite nach dem neu erstellten allPostsByAutorenfeld für den Abfragetyp, und wählen Sie dann Anhängen aus.
- Wählen Sie im Menü Aktion die Option Laufzeit aktualisieren und anschließend Unit Resolver (nur VTL) aus.
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

```
{
    "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
}
```

Dieser Resolver verfügt wie der `allPost`-Resolver über zwei optionale Argumente: `count` gibt die maximale Anzahl von Elementen an, die in einem Aufruf zurückgegeben werden sollen, und mit `nextToken` kann der nächste Ergebnissatz abgerufen werden. (Der Wert für `nextToken` kann aus einem vorherigen Abruf übernommen werden.)

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
{
    "posts": $utils.toJson($context.result.items)
    #if( ${context.result.nextToken} )
        ,"nextToken": $util.toJson($context.result.nextToken)
    #end
}
```

Hinweis: Dies ist die gleiche Zuweisungsvorlage für Antworten, die auch im `allPost`-Resolver verwendet wurde.

- Wählen Sie Speichern.

Weitere Informationen zur `Query` Anforderungszuordnung finden Sie in der Query-Referenzdokumentation.

Aufrufen der API zum Abfragen aller Posts von einem Autor

Jetzt ist der Resolver eingerichtet und AWS AppSync weiß, wie man eine eingehende `allPostsByAuthor` Mutation in eine `Query` DynamoDB-Operation gegen den Index übersetzt. `author-index` Sie können nun die Tabelle abfragen, um alle Posts von einem bestimmten Autor abzurufen.

Zuvor müssen Sie jedoch die Tabelle mit weiteren Post-Daten füllen, da bisher alle Posts den gleichen Autor haben.

- Wählen Sie die Registerkarte Queries aus.
- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein:

```
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 }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).

Wir fragen jetzt die Tabelle ab, so dass alle Posts von `Nadia` zurückgegeben werden.

- Fügen Sie im Bereich Queries (Abfragen) die folgende Abfrage ein:

```
query allPostsByAuthor {
  allPostsByAuthor(author: "Nadia") {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).

- Alle Posts von `Nadia` sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you know...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

Die Paginierung funktioniert für `Query` genauso wie für `Scan`. Suchen wir z. B. nach allen Posts von `AUTHORNAME`, wobei jeweils fünf Posts abgerufen werden.

- Fügen Sie im Bereich Queries (Abfragen) die folgende Abfrage ein:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    count: 5
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).

- Alle Posts von AUTHORNAME sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

- Aktualisieren Sie das nextToken-Argument mit dem Wert, der in der vorherigen Abfrage zurückgegeben wurde:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    count: 5
    nextToken:
 "eyJ2ZXJzaW9uIjoxLCJ0b2tlbiI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eThOSmRvcG9ad2RHYjI
```

```
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Alle verbleibenden Posts von AUTH0RNAME sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "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
    }
  }
}
```

Verwenden von Sätzen

Bis zu diesem Zeitpunkt war der `Post` Typ ein flaches Objekt. key/value Sie können mit dem AWS AppSyncDynamo DB-Resolver auch komplexe Objekte wie Gruppen, Listen und Maps modellieren.

Aktualisieren wir jetzt den `Post`-Typ, so dass er Tags enthält. Ein Beitrag kann 0 oder mehr Tags haben, die in DynamoDB als String-Set gespeichert werden. Außerdem richten Sie einige Mutationen ein, um Tags hinzuzufügen und zu entfernen, sowie eine neue Abfrage zum Scannen nach Posts mit einem bestimmten Tag.

- Wählen Sie die Registerkarte Schema aus.
- Ändern Sie im Schemabereich den `Post` Typ wie folgt, um ein neues `tags` Feld hinzuzufügen:

```
type Post {
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
  tags: [String!]
}
```

- Ändern Sie im Schemabereich den `Query` Typ, um eine neue `allPostsByTag` Abfrage hinzuzufügen, wie folgt:

```
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
}
```

- Ändern Sie im Schemabereich den `Mutation` Typ, um neue `addTag` und `removeTag` Mutationen hinzuzufügen, wie folgt:

```
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!
}
```

- Wählen Sie Speichern.
- Suchen Sie im Bereich Datentypen auf der rechten Seite nach dem neu erstellten allPostsByTag-Feld für den Abfragetyp, und wählen Sie dann Anhängen aus.
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

```
{
    "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
```

```
}
```

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
{
    "posts": $utils.toJson($context.result.items)
    #if( ${context.result.nextToken} )
        ,"nextToken": $util.toJson($context.result.nextToken)
    #end
}
```

- Wählen Sie Speichern.
- Suchen Sie rechts im Bereich Data types (Datentypen) das neue Feld addTag des Typs Mutation und wählen Sie dann Attach (Anhängen).
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

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

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
$utils.toJson($context.result)
```

- Wählen Sie Speichern.

- Suchen Sie rechts im Bereich Data types (Datentypen) das neue Feld removeTag des Typs Mutation und wählen Sie dann Attach (Anhängen).
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

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

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
$utils.toJson($context.result)
```

- Wählen Sie Speichern.

Aufrufen der API zum Arbeiten mit Tags

Nachdem Sie die Resolver eingerichtet haben, AWS AppSync weiß er, wie eingehende `addTag` `allPostsByTag` Anfragen und Anfragen in DynamoDB `UpdateItem` und Operationen übersetzt werden. `removeTag` `Scan`

Wählen Sie zum Ausprobieren einen der zuvor erstellten Posts aus. Verwenden wir als Beispiel einen Post, der von `Nadia` verfasst wurde.

- Wählen Sie die Registerkarte Queries aus.
- Fügen Sie im Bereich Queries (Abfragen) die folgende Abfrage ein:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "Nadia"
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Alle Posts von Nadia sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you known...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

- Verwenden wir den Post mit dem Titel `"The cutest dog in the world"`. Notieren Sie dessen `id`, da diese später benötigt wird.

Fügen wir jetzt den Tag `dog` hinzu.

- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein. Außerdem müssen Sie das `id`-Argument mit dem zuvor notierten Wert aktualisieren.

```
mutation addTag {
  addTag(id:10 tag: "dog") {
    id
    title
    tags
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Der Post wird mit dem neuen Tag aktualisiert.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

Sie können weitere Tags folgendermaßen hinzufügen:

- Aktualisieren Sie die Mutation und ändern Sie das `tag`-Argument in `puppy`.

```
mutation addTag {
  addTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Der Post wird mit dem neuen Tag aktualisiert.

```
{
  "data": {
```

```
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog",
        "puppy"
      ]
    }
  }
}
```

Tags können auch gelöscht werden:

- Fügen Sie die folgende Mutation in den Bereich Queries (Abfragen) ein. Außerdem müssen Sie das `id`-Argument mit dem zuvor notierten Wert aktualisieren.

```
mutation removeTag {
  removeTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Der Post wird aktualisiert und das Tag `puppy` wird gelöscht.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

Sie können auch alle Posts mit einem Tag suchen:

- Fügen Sie im Bereich Queries (Abfragen) die folgende Abfrage ein:

```
query allPostsByTag {
  allPostsByTag(tag: "dog") {
    posts {
      id
      title
      tags
    }
    nextToken
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Alle Posts mit dem Tag dog werden folgendermaßen zurückgegeben:

```
{
  "data": {
    "allPostsByTag": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world",
          "tags": [
            "dog",
            "puppy"
          ]
        }
      ],
      "nextToken": null
    }
  }
}
```

Verwenden von Listen und Zuordnungen

Neben DynamoDB-Sets können Sie auch DynamoDB-Listen und -Maps verwenden, um komplexe Daten in einem einzigen Objekt zu modellieren.

Fügen wir nun die Möglichkeit hinzu, Posts Kommentare hinzuzufügen. Dies wird als Liste von Kartenobjekten für das `Post` Objekt in DynamoDB modelliert.

Hinweis: In einer realen Anwendung würden Sie Kommentare in der eigenen Tabelle modellieren. Für dieses Tutorial fügen Sie sie einfach in die `Post`-Tabelle ein.

- Wählen Sie die Registerkarte Schema aus.
- Fügen Sie im Bereich Schema folgendermaßen einen neuen `Comment`-Typ hinzu:

```
type Comment {
    author: String!
    comment: String!
}
```

- Ändern Sie im Schemabereich den `Post` Typ wie folgt, um ein neues `comments` Feld hinzuzufügen:

```
type Post {
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
  tags: [String!]
  comments: [Comment!]
}
```

- Ändern Sie im Schemabereich den `Mutation` Typ wie folgt, um eine neue `addComment` Mutation hinzuzufügen:

```
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!
}
```

- Wählen Sie Speichern.
- Suchen Sie rechts im Bereich Data types (Datentypen) das neue Feld addComment des Typs Mutation und wählen Sie dann Attach (Anhängen).
- Wählen Sie für Data source name (Datenquellenname) die Option PostDynamoDBTable aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) Folgendes ein:

```
{
  "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)
    }
  }
}
```

Dieser Aktualisierungsausdruck fügt eine Liste mit dem neuen Kommentar an die vorhandene `comments`-Liste an. Wenn die Liste noch nicht vorhanden ist, wird sie erstellt.

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) Folgendes ein:

```
$utils.toJson($context.result)
```

- Wählen Sie Speichern.

Aufrufen der API zum Hinzufügen eines Kommentars

Jetzt, wo Sie die Resolver eingerichtet haben, AWS AppSync weiß, wie eingehende `addComment` Anfragen in `UpdateItem` DynamoDB-Operationen übersetzt werden.

Fügen Sie zum Ausprobieren einen Kommentar zu dem Post hinzu, dem auch die Tags hinzugefügt wurden.

- Wählen Sie die Registerkarte Queries aus.
- Fügen Sie im Bereich Queries (Abfragen) die folgende Abfrage ein:

```
mutation addComment {
  addComment(
    id:10
    author: "Steve"
    comment: "Such a cute dog."
  ) {
    id
    comments {
      author
      comment
    }
  }
}
```

- Wählen Sie Execute query (Abfrage ausführen) (orangefarbene Wiedergabeschaltfläche).
- Alle Posts von Nadia sollten im Ergebnisbereich rechts neben dem Abfragebereich angezeigt werden. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
{
```

```
  "data": {
    "addComment": {
      "id": "10",
      "comments": [
        {
          "author": "Steve",
          "comment": "Such a cute dog."
        }
      ]
    }
  }
}
```

Wenn Sie die Anforderung mehrmals ausführen, werden mehrere Kommentare an die Liste angefügt.

Schlussfolgerung

In diesem Tutorial haben Sie eine API erstellt, mit der wir Post-Objekte in DynamoDB mithilfe AWS AppSync von GraphQL bearbeiten können. Weitere Informationen finden Sie unter Referenz für Resolver-Zuweisungsvorlagen.

Zum Aufräumen können Sie die AppSync GraphQL-API von der Konsole löschen.

Um die DynamoDB-Tabelle und die IAM-Rolle zu löschen, die Sie für dieses Tutorial erstellt haben, können Sie Folgendes ausführen, um den `AWSAppSyncTutorialForAmazonDynamoDB` Stack zu löschen, oder die AWS CloudFormation Konsole aufrufen und den Stack löschen:

```
aws cloudformation delete-stack \
    --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

Verwenden von Resolvern in AWS LambdaAWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Sie können AWS Lambda with verwenden AWS AppSync , um jedes GraphQL-Feld aufzulösen. Beispielsweise könnte eine GraphQL-Abfrage einen Aufruf an eine Amazon Relational Database Service (Amazon RDS) -Instance senden, und eine GraphQL-Mutation könnte in einen Amazon Kinesis Kinesis-Stream schreiben. In diesem Abschnitt zeigen wir Ihnen, wie Sie eine Lambda-Funktion schreiben, die Geschäftslogik auf der Grundlage des Aufrufs einer GraphQL-Feldoperation ausführt.

Erstellen einer Lambda-Funktion

Das folgende Beispiel zeigt eine eingeschriebene Lambda-FunktionNode.js, die als Teil einer Blogpost-Anwendung verschiedene Operationen an Blogbeiträgen ausführt.

```
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;
    }
};
```

Diese Lambda-Funktion ruft einen Beitrag nach ID ab, fügt einen Beitrag hinzu, ruft eine Liste von Beiträgen ab und ruft verwandte Beiträge für einen bestimmten Beitrag ab.

Hinweis: Die Lambda-Funktion verwendet die `switch` Anweisung on`event.field`, um zu ermitteln, welches Feld gerade aufgelöst wird.

Erstellen Sie diese Lambda-Funktion mit der AWS Management Console oder einem AWS CloudFormation Stack. Um die Funktion aus einem CloudFormation Stack zu erstellen, können Sie den folgenden Befehl AWS Command Line Interface (AWS CLI) verwenden:

```
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
```

Sie können den AWS CloudFormation Stack auch in der AWS Region USA West (Oregon) in Ihrem AWS Konto von hier aus starten:

Eine Datenquelle für Lambda konfigurieren

Nachdem Sie die Lambda-Funktion erstellt haben, navigieren Sie in der AWS AppSync Konsole zu Ihrer GraphQL-API und wählen Sie dann die Registerkarte Datenquellen aus.

Wählen Sie Datenquelle erstellen, geben Sie einen benutzerfreundlichen Datenquellennamen ein (z. B.**`Lambda`**), und wählen Sie AWS Lambda dann als Datenquellentyp die Option Funktion aus. Wählen Sie für Region dieselbe Region wie Ihre Funktion aus. (Wenn Sie die Funktion aus dem bereitgestellten CloudFormation Stack erstellt haben, befindet sich die Funktion wahrscheinlich in US-WEST-2.) Wählen Sie für Function ARN den Amazon Resource Name (ARN) Ihrer Lambda-Funktion.

Nachdem Sie Ihre Lambda-Funktion ausgewählt haben, können Sie entweder eine neue AWS Identity and Access Management (IAM-) Rolle erstellen (für die die entsprechenden Berechtigungen AWS AppSync zugewiesen werden) oder eine vorhandene Rolle mit der folgenden Inline-Richtlinie auswählen:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:REGION:ACCOUNTNUMBER:function/
LAMBDA_FUNCTION"
        }
```

```
    ]
}
```

Sie müssen außerdem wie folgt eine Vertrauensbeziehung mit AWS AppSync der IAM-Rolle einrichten:

JSON

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

Erstellen Sie ein GraphQL-Schema

Nachdem die Datenquelle mit Ihrer Lambda-Funktion verbunden ist, erstellen Sie ein GraphQL-Schema.

Stellen Sie im Schema-Editor in der AWS AppSync Konsole sicher, dass Ihr Schema dem folgenden Schema entspricht:

```
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]
}
```

Resolver konfigurieren

Nachdem Sie nun eine Lambda-Datenquelle und ein gültiges GraphQL-Schema registriert haben, können Sie Ihre GraphQL-Felder mithilfe von Resolvern mit Ihrer Lambda-Datenquelle verbinden.

Um einen Resolver zu erstellen, benötigen Sie Mapping-Vorlagen. Weitere Informationen zu Mapping-Vorlagen finden Sie unterResolver Mapping Template Overview.

Weitere Informationen zu Lambda-Mapping-Vorlagen finden Sie unterResolver mapping template reference for Lambda.

In diesem Schritt fügen Sie der Lambda-Funktion einen Resolver für die folgenden Felder hinzu:`getPost(id:ID!): Post`, `allPosts: [Post]``addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!`, und. `Post.relatedPosts: [Post]`

Wählen Sie im Schema-Editor in der AWS AppSync Konsole auf der rechten Seite Attach Resolver for aus. `getPost(id:ID!): Post`

Wählen Sie dann im Menü Aktion die Option Laufzeit aktualisieren und anschließend Unit Resolver (nur VTL) aus.

Wählen Sie anschließend Ihre Lambda-Datenquelle aus. Wählen Sie im Bereich request mapping template (Anforderungszuweisungsvorlage) Invoke And Forward Arguments (Argumente aufrufen und weiterleiten).

Ändern Sie das Objekt `payload`, um den Feldnamen hinzuzufügen. Ihre Vorlage sollte jetzt wie folgt aussehen:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments":  $utils.toJson($context.arguments)
    }
}
```

Wählen Sie im Bereich response mapping template (Antwortzuweisungsvorlage) Return Lambda Result (Lambda-Ergebnis zurückgeben) aus.

Verwenden Sie in diesem Fall die Basisvorlage in unveränderter Form. Sie sollte wie folgt aussehen:

```
$utils.toJson($context.result)
```

Wählen Sie Speichern. Sie haben nun den ersten Resolver angehängt. Wiederholen Sie diese Operation wie folgt für die verbleibenden Felder:

Für die `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!`-Anforderungszuweisungsvorlage:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "addPost",
        "arguments":  $utils.toJson($context.arguments)
    }
}
```

Für die `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!`-Antwortzuweisungsvorlage:

```
$utils.toJson($context.result)
```

Für die `allPosts: [Post]`-Anforderungszuweisungsvorlage:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "allPosts"
    }
}
```

Für die `allPosts: [Post]`-Antwortzuweisungsvorlage:

```
$utils.toJson($context.result)
```

Für die `Post.relatedPosts: [Post]`-Anforderungszuweisungsvorlage:

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "relatedPosts",
        "source":  $utils.toJson($context.source)
    }
}
```

Für die `Post.relatedPosts: [Post]`-Antwortzuweisungsvorlage:

```
$utils.toJson($context.result)
```

Testen Sie Ihre GraphQL-API

Jetzt ist Ihre Lambda-Funktion mit GraphQL-Resolvern verbunden und Sie können einige Mutationen und Abfragen mithilfe der Konsole oder einer Client-Anwendung ausführen.

Wählen Sie auf der linken Seite der AWS AppSync Konsole Abfragen aus und fügen Sie dann den folgenden Code ein:

addPost-Mutation

```
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
    }
}
```

getPost-Abfrage

```
query getPost {
    getPost(id: "2") {
        id
        author
        title
        content
        url
        ups
        downs
    }
}
```

allPosts-Abfrage

```
query allPosts {
    allPosts {
        id
        author
        title
        content
        url
        ups
        downs
        relatedPosts {
            id
```

```
            title
        }
    }
}
```

Zurückkehrende Fehler

Jede gegebene Feldauflösung kann zu einem Fehler führen. Mit AWS AppSync können Sie Fehler aus den folgenden Quellen melden:

- Anforderungs- oder Antwortzuweisungsvorlage
- Lambda-Funktion

Von der Zuweisungsvorlage

Um absichtliche Fehler auszulösen, können Sie die `$utils.error` Hilfsmethode aus der Vorlage Velocity Template Language (VTL) verwenden. Das Argument enthält die `errorMessage`, den `errorType` und einen optionalen `data`-Wert. `data` ist hilfreich, um zusätzliche Daten an den Client zurückzugeben, wenn ein Fehler auftritt. Das Objekt `data` wird `errors` der endgültigen GraphQL-Antwort hinzugefügt.

Das folgende Beispiel zeigt, wie Sie es in der Antwortzuweisungsvorlage `Post.relatedPosts: [Post]` verwenden:

```
$utils.error("Failed to fetch relatedPosts", "LambdaFailure", $context.result)
```

Dies löst eine GraphQL-Antwort ähnlich der Folgenden aus:

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

Dabei ist `allPosts[0].relatedPosts` gleich null, weil ein Fehler aufgetreten ist und `errorMessage`, `errorType` und `data` im Objekt `data.errors[0]` vorhanden sind.

Von der Lambda-Funktion

AWS AppSync versteht auch Fehler, die die Lambda-Funktion auslöst. Mit dem Lambda-Programmiermodell können Sie behandelte Fehler melden. Wenn die Lambda-Funktion einen Fehler ausgibt, AWS AppSync kann das aktuelle Feld nicht aufgelöst werden. Nur die von Lambda zurückgegebene Fehlermeldung ist in der Antwort enthalten. Derzeit können Sie keine überflüssigen Daten an den Client zurückgeben, indem Sie einen Fehler in der Lambda-Funktion auslösen.

Hinweis: Wenn Ihre Lambda-Funktion einen unbehandelten Fehler auslöst, AWS AppSync verwendet sie die von Lambda festgelegte Fehlermeldung.

Die folgenden Lambda-Funktion löst einen Fehler aus:

```
exports.handler = (event, context, callback) => {
    console.log("Received event {}", JSON.stringify(event, 3));
    callback("I fail. Always.");
};
```

Dadurch wird eine GraphQL-Antwort ähnlich der folgenden ausgegeben:

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

Anwendungsfall für Fortgeschrittene: Batching

Die Lambda-Funktion in diesem Beispiel hat ein `relatedPosts` Feld, das eine Liste verwandter Beiträge für einen bestimmten Beitrag zurückgibt. In den Beispielabfragen gibt der `allPosts` Feldaufruf der Lambda-Funktion fünf Beiträge zurück. Da wir angegeben haben, dass wir auch

`relatedPosts` für jeden zurückgegebenen Beitrag eine Auflösung durchführen möchten, wird die `relatedPosts` Feldoperation fünfmal aufgerufen.

```
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
        }
    }
}
```

Das klingt in diesem speziellen Beispiel vielleicht nicht wesentlich, aber dieses verstärkte Überabrufen kann die Anwendung schnell untergraben.

Wenn wir z. B. in derselben Abfrage erneut `relatedPosts` für die zurückgegebenen verwandten `Posts` abrufen würden, würde die Anzahl der Abrufe dramatisch ansteigen.

```
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
            }
```

```
        }
    }
}
```

In dieser relativ einfachen Abfrage AWS AppSync würde die Lambda-Funktion 1 + 5 + 25 = 31 mal aufgerufen.

Dies ist eine häufig vorkommende Herausforderung, die oft als N+1-Problem (in diesem Fall N = 5) bezeichnet wird und zu höherer Latenz und höheren Kosten für die Anwendung führen kann.

Ein Ansatz zur Lösung dieses Problems besteht darin, ähnliche Feld-Resolver-Anforderungen zu bündeln. In diesem Beispiel könnte die Lambda-Funktion, anstatt eine Liste verwandter Beiträge für einen bestimmten Beitrag auflösen zu lassen, stattdessen eine Liste verwandter Beiträge für einen bestimmten Stapel von Beiträgen auflösen.

Um dies zu demonstrieren, müssen wir den Resolver `Post.relatedPosts: [Post]` in einen stapelfähigen Resolver ändern.

Wählen Sie auf der rechten Seite der AWS AppSync Konsole den vorhandenen `Post.relatedPosts: [Post]` Resolver aus. Ändern Sie die Zuweisungsvorlage für Anforderungen wie folgt:

```
{
    "version": "2017-02-28",
    "operation": "BatchInvoke",
    "payload": {
        "field": "relatedPosts",
        "source":  $utils.toJson($context.source)
    }
}
```

Nur das Feld `operation` wurde von `Invoke` in `BatchInvoke` geändert. Das Payload-Feld wird jetzt zu einem Array mit allem, was in der Vorlage angegeben ist. In diesem Beispiel erhält die Lambda-Funktion Folgendes als Eingabe:

```
[
    {
        "field": "relatedPosts",
        "source": {
            "id": 1
        }
```

```
    },
    {
        "field": "relatedPosts",
        "source": {
            "id": 2
        }
    },
    ...
]
```

Wenn in der Vorlage für die Anforderungszuweisung angegeben `BatchInvoke` ist, empfängt die Lambda-Funktion eine Liste von Anfragen und gibt eine Ergebnisliste zurück.

Insbesondere muss die Ergebnisliste der Größe und Reihenfolge der Payload-Einträge der Anfrage entsprechen, damit die Ergebnisse entsprechend abgeglichen werden AWS AppSync können.

In diesem Batching-Beispiel gibt die Lambda-Funktion eine Reihe von Ergebnissen wie folgt zurück:

```
[
    [{"id":"2","title":"Second book"}, {"id":"3","title":"Third book"}],   //
 relatedPosts for id=1
    [{"id":"3","title":"Third book"}]
           // relatedPosts for id=2
]
```

Die folgende Lambda-Funktion in Node.js demonstriert diese Batching-Funktionalität für das `Post.relatedPosts` Feld wie folgt:

```
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;
    }
};
```

Rückgabe einzelner Fehler

Die vorherigen Beispiele zeigen, dass es möglich ist, einen einzelnen Fehler von der Lambda-Funktion zurückzugeben oder einen Fehler aus den Mapping-Vorlagen auszulösen. Bei Batch-Aufrufen wird durch das Auslösen eines Fehlers über die Lambda-Funktion ein ganzer Batch als fehlgeschlagen gekennzeichnet. Dies kann für bestimmte Szenarien akzeptabel sein, in denen ein nicht behebbarer Fehler auftritt, z. B. eine fehlgeschlagene Verbindung zu einem Datenspeicher. In

Fällen, in denen einige Elemente im Stapel erfolgreich sind und andere fehlschlagen, ist es jedoch möglich, sowohl Fehler als auch gültige Daten zurückzugeben. Da die Batch-Antwort AWS AppSync erfordert, um Elemente aufzulisten, die der ursprünglichen Größe des Stapels entsprechen, müssen Sie eine Datenstruktur definieren, die gültige Daten von fehlerhaften Daten unterscheiden kann.

Wenn beispielsweise erwartet wird, dass die Lambda-Funktion einen Stapel verwandter Beiträge zurückgibt, könnten Sie sich dafür entscheiden, eine Liste von *Response* Objekten zurückzugeben, in der jedes Objekt optionale Daten -, errorMessage - und ErrorType-Felder hat. Wenn das Feld errorMessage angezeigt wird, bedeutet dies, dass ein Fehler aufgetreten ist.

Der folgende Code zeigt, wie Sie die Lambda-Funktion aktualisieren können:

```
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;
    }
};
```

In diesem Beispiel analysiert die folgende Antwortzuordnungsvorlage jedes Element der Lambda-Funktion und löst alle auftretenden Fehler aus:

```
#if( $context.result && $context.result.errorMessage )
    $utils.error($context.result.errorMessage, $context.result.errorType,
 $context.result.data)
#else
    $utils.toJson($context.result.data)
#end
```

Dieses Beispiel gibt eine GraphQL-Antwort ähnlich der folgenden aus:

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

Konfiguration der maximalen Batchgröße

Standardmäßig werden bei der Verwendung `BatchInvoke` Anfragen AWS AppSync an Ihre Lambda-Funktion in Stapeln von bis zu fünf Elementen gesendet. Sie können die maximale Batchgröße Ihrer Lambda-Resolver konfigurieren.

Um die maximale Batchgröße auf einem Resolver zu konfigurieren, verwenden Sie den folgenden Befehl in der (): 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**
>
> Wenn Sie eine Vorlage für die Anforderungszuweisung bereitstellen, müssen Sie den `BatchInvoke` Vorgang verwenden, um Batching zu verwenden.

Sie können auch den folgenden Befehl verwenden, um Batching auf Direct Lambda Resolvern zu aktivieren und zu konfigurieren:

```
$ aws appsync create-resolver --api-id <api-id> --type-name Query --field-name
 relatedPosts \
 --data-source-name "<lambda-datasource>" \
 --max-batch-size X
```

Konfiguration der maximalen Batchgröße mit VTL-Vorlagen

Bei Lambda-Resolvern mit VTL-In-Request-Vorlagen hat die maximale Batchgröße keine Auswirkung, es sei denn, sie haben sie direkt als `BatchInvoke` Operation in VTL angegeben. Ebenso wird, wenn Sie eine Mutation der obersten Ebene durchführen, kein Batching für Mutationen durchgeführt, da die GraphQL-Spezifikation verlangt, dass parallel Mutationen sequentiell ausgeführt werden.

Nehmen wir zum Beispiel die folgenden Mutationen:

```
type Mutation {
    putItem(input: Item): Item
    putItems(inputs: [Item]): [Item]
}
```

Mit der ersten Mutation können wir 10 erstellen, `Items` wie im folgenden Ausschnitt gezeigt:

```
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
    }
}
```

In diesem Beispiel `Items` werden die nicht in einer Gruppe von 10 gebündelt, auch wenn die maximale Batchgröße im Lambda Resolver auf 10 gesetzt ist. Stattdessen werden sie sequentiell gemäß der GraphQL-Spezifikation ausgeführt.

Um eine tatsächliche Batch-Mutation durchzuführen, können Sie dem folgenden Beispiel folgen und die zweite Mutation verwenden:

```
mutation MyMutation {
```

```
    putItems([$someItem1, $someItem2, $someItem3,$someItem4, $someItem5, $someItem6,
    $someItem7, $someItem8, $someItem9, $someItem10]) {
    id,
    name
    }
}
```

Weitere Informationen zur Verwendung von Batching mit Direct Lambda Resolvern finden Sie unter. Direkte Lambda-Resolver

Verwendung von Amazon OpenSearch Service-Resolvern in AWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync unterstützt die Nutzung von Amazon OpenSearch Service von Domains aus, die Sie in Ihrem eigenen AWS Konto bereitgestellt haben, sofern sie nicht in einer VPC existieren. Nachdem Ihre Domänen bereitgestellt wurden, können Sie sich über eine Datenquelle mit ihnen verbinden. Zu diesem Zeitpunkt können Sie auch einen Resolver im Schema konfigurieren, um GraphQL-Operationen, wie z. B. Abfragen, Mutationen und Abonnements, auszuführen. Diese Anleitung führt Sie durch einige häufig auftretende Beispiele.

Weitere Informationen finden Sie in der Resolver Mapping Template Reference für. OpenSearch

One-Click-Setup

Um automatisch einen GraphQL-Endpunkt AWS AppSync mit konfiguriertem Amazon OpenSearch Service einzurichten, können Sie diese AWS CloudFormation Vorlage verwenden:

Nach Abschluss der AWS CloudFormation Bereitstellung können Sie direkt mit der Ausführung von GraphQL-Abfragen und -Mutationen fortfahren.

Erstellen Sie eine neue OpenSearch Dienstdomäne

Um mit diesem Tutorial beginnen zu können, benötigen Sie eine bestehende OpenSearch Dienstdomäne. Wenn Sie noch keine haben, können Sie das folgende Bespiel verwenden. Beachten Sie, dass es bis zu 15 Minuten dauern kann, bis eine OpenSearch Dienstdomäne erstellt ist, bevor Sie mit der Integration in eine AWS AppSync Datenquelle fortfahren können.

```
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
```

Sie können den folgenden AWS CloudFormation Stack in der Region USA West 2 (Oregon) in Ihrem AWS Konto starten:

Launch Stack

Datenquelle für den OpenSearch Service konfigurieren

Nachdem die OpenSearch Service-Domain erstellt wurde, navigieren Sie zu Ihrer AWS AppSync GraphQL-API und wählen Sie die Registerkarte Datenquellen. Wählen Sie Neu und geben Sie einen benutzerfreundlichen Namen für die Datenquelle ein, z. B. „oss“. Wählen Sie dann OpenSearch Amazon-Domain als Datenquellentyp, wählen Sie die entsprechende Region aus, und Ihre OpenSearch Service-Domain sollte aufgeführt sein. Nachdem Sie sie ausgewählt haben, können Sie entweder eine neue Rolle erstellen und ihnen AWS AppSync die entsprechenden Berechtigungen zuweisen, oder Sie können eine bestehende Rolle auswählen, für die die folgende Inline-Richtlinie gilt:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1234234",
            "Effect": "Allow",
            "Action": [
```

```
                "es:ESHttpDelete",
                "es:ESHttpHead",
                "es:ESHttpGet",
                "es:ESHttpPost",
                "es:ESHttpPut"
            ],
            "Resource": [
                "arn:aws:es:us-east-1:ACCOUNTNUMBER:domain/democluster/*"
            ]
        }
    ]
}
```

Sie müssen außerdem eine Vertrauensbeziehung AWS AppSync für diese Rolle einrichten mit:

JSON

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

Darüber hinaus hat die OpenSearch Service-Domain ihre eigene Zugriffsrichtlinie, die Sie über die Amazon OpenSearch Service-Konsole ändern können. Sie müssen eine Richtlinie hinzufügen, die der folgenden ähnelt und die entsprechenden Aktionen und Ressourcen für die OpenSearch Service-Domain enthält. Beachten Sie, dass der Principal die AppSync Datenquellenrolle sein wird. Wenn Sie diese Rolle von der Konsole erstellen lassen, finden Sie diese Rolle in der IAM-Konsole.

JSON

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/service-role/
APPSYNC_DATASOURCE_ROLE"
            },
            "Action": [
                "es:ESHttpDelete",
                "es:ESHttpHead",
                "es:ESHttpGet",
                "es:ESHttpPost",
                "es:ESHttpPut"
            ],
            "Resource": "arn:aws:es:us-east-1:ACCOUNTNUMBER:domain/DOMAIN_NAME/*"
        }
    ]
}
```

Verbinden eines Resolvers

Nachdem die Datenquelle nun mit Ihrer OpenSearch Service-Domain verbunden ist, können Sie sie mit einem Resolver mit Ihrem GraphQL-Schema verbinden, wie im folgenden Beispiel gezeigt:

```
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
}
...
```

Beachten Sie, dass es einen benutzerdefinierten `Post`-Typ mit einem `id`-Feld gibt. In den folgenden Beispielen gehen wir davon aus, dass es einen Prozess gibt (der automatisiert werden kann), um diesen Typ in Ihre OpenSearch Service-Domain einzufügen, der einem Pfadstamm von zugeordnet würde`/post/_doc`, wo sich der Index `post` befindet. Von diesem Stammpfad aus können Sie einzelne Dokumente, Platzhaltersuchen mit `/id/post*` oder Suchen in mehreren Dokumenten mit dem Pfad von durchführen. `/post/_search` Wenn Sie beispielsweise einen anderen Typ haben`User`, können Sie Dokumente unter einem neuen Index namens `user` indexieren und dann Suchen mit dem Pfad von durchführen. `/user/_search`

Ändern Sie im Schema-Editor in der AWS AppSync Konsole das vorherige `Posts` Schema, sodass es eine `searchPosts` Abfrage enthält:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  searchPosts: [Post]
}
```

Speichern Sie das Schema. Wählen Sie auf der rechten Seite unter `searchPosts` die Option Resolver anfügen (Attach resolver) aus. Wählen Sie im Menü Aktion die Option Laufzeit aktualisieren und anschließend Unit Resolver (nur VTL) aus. Wählen Sie dann Ihre OpenSearch Service-Datenquelle aus. Wählen Sie im Bereich Zuweisungsvorlage für Anforderungen (request mapping template) die Dropdown-Liste für Abfrageposten (Query posts) aus, um eine Basisvorlage zu erhalten. Ändern Sie den `path` so ab, dass er `/post/_search` anzeigt. Sie sollte wie folgt aussehen:

```
{
    "version":"2017-02-28",
    "operation":"GET",
    "path":"/post/_search",
    "params":{
```

```
        "headers":{},
        "queryString":{},
        "body":{
            "from":0,
            "size":50
        }
    }
}
```

Dabei wird davon ausgegangen, dass das obige Schema Dokumente enthält, die in OpenSearch Service unter dem `post` Feld indexiert wurden. Wenn Sie Ihre Daten anders strukturieren, müssen Sie eine entsprechende Anpassung durchführen.

Im Abschnitt Antwortzuordnungsvorlage müssen Sie den entsprechenden `_source` Filter angeben, wenn Sie die Datenergebnisse einer OpenSearch Serviceabfrage abrufen und in GraphQL übersetzen möchten. Verwenden Sie die folgende Vorlage:

```
[
    #foreach($entry in $context.result.hits.hits)
    #if( $velocityCount > 1 ) , #end
    $utils.toJson($entry.get("_source"))
    #end
]
```

Ändern Ihrer Suchvorgänge

Die vorherige Zuweisungsvorlage für Anforderungen führt eine einfache Abfrage aller Datensätze aus. Angenommen, Sie möchten nach einem bestimmten Autor suchen. Nehmen wir darüber hinaus noch an, Sie möchten, dass dieser Autor in Ihrer GraphQL-Abfrage als Argument definiert ist. Fügen Sie nun im Schema-Editor der AWS AppSync-Konsole eine `allPostsByAuthor`-Abfrage hinzu:

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  allPostsByAuthor(author: String!): [Post]
  searchPosts: [Post]
}
```

Wählen Sie nun Attach Resolver und wählen Sie die OpenSearch Service-Datenquelle aus. Verwenden Sie jedoch das folgende Beispiel in der Antwortzuordnungsvorlage:

```
{
    "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)
                }
            }
        }
    }
}
```

Beachten Sie, dass der body mit einer Begriffsabfrage für das `author`-Feld ausgefüllt wurde und vom Client als Argument übergeben wurde. Sie könnten optional bereits ausgefüllte Informationen, wie z. B. einen Standardtext, oder sogar andere Dienstprogramme verwenden.

Wenn Sie diesen Resolver verwenden möchten, füllen Sie die Zuweisungsvorlage für Antworten mit den gleichen Informationen aus, wie im vorherigen Beispiel gezeigt.

Daten zum OpenSearch Dienst hinzufügen

Möglicherweise möchten Sie Ihrer OpenSearch Service-Domain aufgrund einer GraphQL-Mutation Daten hinzufügen. Hierbei handelt es sich um einen äußerst effektiven Mechanismus für Suchvorgänge und andere Zwecke. Da Sie GraphQL-Abonnements verwenden können, um Ihre Daten in Echtzeit zu erstellen, dient es als Mechanismus, um Clients über Aktualisierungen von Daten in Ihrer OpenSearch Service-Domain zu informieren.

Kehren Sie zur Schemaseite in der AWS AppSync Konsole zurück und wählen Sie Attach Resolver für die Mutation aus. `addPost()` Wählen Sie erneut die OpenSearch Service-Datenquelle aus und verwenden Sie die folgende Antwortzuordnungsvorlage für das `Posts` Schema:

```
{
    "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)
        }
    }
}
```

Wie zuvor ist dies nur ein Beispiel dafür, wie Ihre Daten strukturiert sein könnten. Wenn Sie über andere Feldnamen oder Indizes verfügen, müssen Sie den `path` und `body` entsprechend anpassen. Dieses Beispiel zeigt auch, wie Sie `$context.arguments` verwenden, um die Vorlage mit den GraphQL-Mutationsargumenten zu füllen.

Bevor Sie fortfahren, verwenden Sie die folgende Antwortzuordnungsvorlage, die das Ergebnis der Mutationsoperation oder Fehlerinformationen als Ausgabe zurückgibt:

```
#if($context.error)
    $util.toJson($ctx.error)
#else
    $util.toJson($context.result)
#end
```

Abrufen eines einzelnen Dokuments

Sofern Sie die `getPost(id:ID)`-Abfrage im Schema verwenden möchten, um ein einzelnes Dokument zurückzugeben, suchen Sie diese Abfrage im Schema-Editor der AWS AppSync-Konsole und wählen Sie Attach resolver (Resolver anfügen) aus. Wählen Sie erneut die OpenSearch Service-Datenquelle aus und verwenden Sie die folgende Zuordnungsvorlage:

```
{
    "version":"2017-02-28",
    "operation":"GET",
```

```
    "path": $util.toJson("post/_doc/$context.arguments.id"),
    "params":{
        "headers":{},
        "queryString":{},
        "body":{}
    }
}
```

Da der oben genannte `path` das `id`-Argument ohne Text verwendet, wird auf diese Weise ein einzelnes Dokument zurückgegeben. Jedoch müssen Sie die folgende Zuweisungsvorlage für Antworten verwenden, da Sie jetzt ein einzelnes Element und keine Liste zurückgeben:

```
$utils.toJson($context.result.get("_source"))
```

Ausführen von Abfragen und Mutationen

Sie sollten jetzt in der Lage sein, GraphQL-Operationen für Ihre OpenSearch Service-Domain durchzuführen. Navigieren Sie zur Registerkarte Abfragen der AWS AppSync Konsole und fügen Sie einen neuen Datensatz hinzu:

```
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
       )
}
```

Sie werden das Ergebnis der Mutation auf der rechten Seite sehen. In ähnlicher Weise können Sie jetzt eine `searchPosts` Abfrage für Ihre OpenSearch Service-Domain ausführen:

```
query searchPosts {
    searchPosts {
        id
        title
        author
```

```
        content
    }
}
```

Bewährte Methoden

- OpenSearch Der Dienst sollte zum Abfragen von Daten dienen, nicht als primäre Datenbank. Möglicherweise möchten Sie OpenSearch Service in Verbindung mit Amazon DynamoDB verwenden, wie unter Kombinieren von GraphQL-Resolvern beschrieben.
- Gewähren Sie nur Zugriff auf Ihre Domain, indem Sie der AWS AppSync Servicerolle den Zugriff auf den Cluster gestatten.
- Sie können mit dieser Entwicklung klein anfangen, indem Sie zunächst nur den preisgünstigsten Cluster verwenden, und später während der Produktion zu einem größeren Cluster überwechseln, der über eine hohe Verfügbarkeit verfügt.

Verwendung lokaler Resolver in AWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync ermöglicht es Ihnen AWS Lambda, unterstützte Datenquellen (Amazon DynamoDB oder Amazon OpenSearch Service) für verschiedene Operationen zu verwenden. Doch in bestimmten Szenarien ist ein Aufruf an eine unterstützte Datenquelle möglicherweise nicht erforderlich.

Hier ist der lokale Resolver nützlich. Anstatt eine Remote-Datenquelle aufzurufen, leitet der lokale Resolver das Ergebnis der Zuweisungsvorlage für Anforderungen einfach an die Zuweisungsvorlage für Antworten weiter. Das Feldauflösung verlässt AWS AppSync nicht.

Lokale Resolver sind für verschiedene Anwendungsfälle nützlich. Der beliebtest Anwendungsfall ist das Veröffentlichen von Benachrichtigungen ohne Auslösen eines Datenquellenaufrufs. Um diesen Anwendungsfall zu demonstrieren, erstellen wir eine Paging-Anwendung; bei der Benutzer sich gegenseitig pagen können. In diesem Beispiel werden Abonnements verwendet. Wenn Sie noch nicht mit Abonnements vertraut sind, sehen Sie sich das Tutorial Echtzeitdaten an.

Erstellen der Paging-Anwendung

In unserer Paging-Anwendung können Clients einen Posteingang abonnieren und Seiten an andere Clients senden. Jede Seite enthält eine Nachricht. Hier wird das Schema gezeigt:

```
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
}
```

Hängen wir nun einen Resolver an das `Mutation.page`-Feld an. Klicken Sie im Bereich Schema auf Attach Resolver (Resolver anhängen) neben der Felddefinition auf dem rechten Bereich. Erstellen Sie eine neue Datenquelle vom Typ Keine und geben Sie ihr einen Namen. PageDataSource

Geben Sie für die Zuweisungsvorlage für Anforderungen Folgendes ein:

```
{
  "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()"
  }
}
```

Und wählen Sie für die Zuweisungsvorlage für Antworten die Standardoption Forward the result (Ergebnis weiterleiten) aus. Speichern Sie Ihren Resolver. Ihre Anwendung ist nun bereit, beginnen wir mit dem Pagen!

Senden an und Abonnieren von Seiten

Damit Clients Seiten empfangen können, müssen sie zuerst einen Posteingang abonnieren.

Führen wir im Bereich Queries (Abfragen) das `inbox`-Abonnement aus:

```
subscription Inbox {
    inbox(to: "Nadia") {
        body
        to
        from
        sentAt
    }
}
```

Nadia erhält Seiten, wann immer die `Mutation.page`-Mutation aufgerufen wird. Rufen wir nun die Mutation durch Ausführen der Mutation auf:

```
mutation Page {
    page(to: "Nadia", body: "Hello, World!") {
        body
        to
        from
        sentAt
    }
}
```

Wir haben gerade die Verwendung von lokalen Resolvern gezeigt, indem wir eine Seite gesendet und sie empfangen haben, ohne AWS AppSync zu verlassen.

Kombinieren von GraphQL-Resolvern in AWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Resolver und Felder in einem GraphQL-Schema zeichnen sich durch 1-zu-1-Beziehungen mit einem hohen Maß an Flexibilität aus. Da eine Datenquelle auf einem Resolver unabhängig vom Schema konfiguriert wird, haben Sie die Möglichkeit, GraphQL-Typen über verschiedene Datenquellen aufzulösen oder zu manipulieren. Dabei spielt die Kombination des Schemas keine Rolle und kann sich ganz nach Ihren Anforderungen richten.

Die folgenden Beispielszenarien zeigen, wie Sie Datenquellen in Ihrem Schema kombinieren und abgleichen können. Bevor Sie beginnen, empfehlen wir, dass Sie mit der Einrichtung von Datenquellen und Resolvern für AWS Lambda Amazon DynamoDB und Amazon OpenSearch Service vertraut sind, wie in den vorherigen Tutorials beschrieben.

Beispielschema

Das folgende Schema hat einen Typ `Post` mit 3 `Query` Operationen und 3 definierten `Mutation` Operationen:

```
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
}
```

In diesem Beispiel müssten Sie insgesamt 6 Resolver anfügen. Eine Möglichkeit wäre, all diese Daten aus einer Amazon DynamoDB-Tabelle mit dem Namen`Posts`, zu beziehen, in der ein Scan und `searchPosts` eine Abfrage `AllPosts` ausgeführt werden, wie in der DynamoDB Resolver Mapping Template Reference beschrieben. Es gibt jedoch Alternativen, um Ihre Geschäftsanforderungen zu erfüllen, z. B. die Auflösung dieser GraphQL-Abfragen von Lambda oder OpenSearch Service.

Ändern von Daten mithilfe von Resolvern

Möglicherweise müssen Sie Ergebnisse aus einer Datenbank wie DynamoDB (oder Amazon Aurora) an Clients zurückgeben, wobei einige der Attribute geändert wurden. Dies kann mit der Formatierung der Datentypen (z. B. Zeitstempeldifferenzen der Clients) oder mit Problemen bei der Handhabung der Rückwärtskompatibilität in Zusammenhang stehen. Zur Veranschaulichung manipuliert im folgenden Beispiel eine AWS Lambda Funktion die positiven und negativen Stimmen für Blogbeiträge, indem sie ihnen bei jedem Aufruf des GraphQL-Resolvers Zufallszahlen zuweist:

```
'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);
    });
};
```

Hierbei handelt es sich um eine absolut gültige Lambda-Funktion, die dem `AllPosts`-Feld im GraphQL-Schema zugewiesen werden könnte. Auf diese Weise würde jeder Abfrage, die alle Ergebnisse zurückgibt, Zufallszahlen für die positiven und negativen Stimmen vergeben.

DynamoDB und Service OpenSearch

Bei einigen Anwendungen führen Sie möglicherweise Mutationen oder einfache Suchabfragen für DynamoDB durch und lassen einen Hintergrundprozess Dokumente an OpenSearch Service übertragen. Anschließend können Sie den `searchPosts` Resolver einfach an die OpenSearch Service-Datenquelle anhängen und Suchergebnisse (aus Daten, die ihren Ursprung in DynamoDB haben) mithilfe einer GraphQL-Abfrage zurückgeben. Dies kann äußerst hilfreich sein, wenn Sie Ihren Anwendungen erweiterte Suchfunktionen hinzufügen, beispielsweise Schlüsselwort-, Fuzzy- oder sogar raumbezogene Suchen. Die Übertragung von Daten aus DynamoDB kann über einen ETL-Prozess erfolgen, oder Sie können alternativ mithilfe von Lambda aus DynamoDB streamen. Sie können ein vollständiges Beispiel dafür starten, indem Sie den folgenden AWS CloudFormation Stack in der Region USA West 2 (Oregon) in Ihrem Konto verwenden: AWS

Mit dem Schema in diesem Beispiel können Sie Beiträge mithilfe eines DynamoDB-Resolvers wie folgt hinzufügen:

```
mutation add {
```

```
    putPost(author:"Nadia"
        title:"My first post"
        content:"This is some test content"
        url:"https://aws.amazon.com/appsync/"
    ){
        id
        title
    }
}
```

Dadurch werden Daten in DynamoDB geschrieben, das dann Daten über Lambda an Amazon OpenSearch Service streamt, wo Sie nach allen Beiträgen anhand verschiedener Felder suchen können. Da sich die Daten beispielsweise in Amazon OpenSearch Service befinden, können Sie entweder die Autoren- oder Inhaltsfelder mit Freiformtext, auch mit Leerzeichen, wie folgt durchsuchen:

```
query searchName{
    searchAuthor(name:"   Nadia   "){
        id
        title
        content
    }
}

query searchContent{
    searchContent(text:"test"){
        id
        title
        content
    }
}
```

Da die Daten direkt in DynamoDB geschrieben werden, können Sie mit den Abfragen `allPosts{...}` und `singlePost{...}` dennoch effiziente Listen- oder Element-Suchvorgänge für die Tabelle durchführen. Dieser Stack verwendet den folgenden Beispielcode für DynamoDB-Streams:

Hinweis: Dieser Code dient nur der Veranschaulichung.

```
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);
};
```

Sie können dies dann mithilfe von DynamoDB-Streams an eine DynamoDB-Tabelle mit dem Primärschlüssel von anhängen`id`, und alle Änderungen an der DynamoDB-Quelle würden in Ihre Service-Domain gestreamt. OpenSearch Weitere Informationen zum Konfigurieren dieser Einstellung finden Sie in der Dokumentation zu DynamoDB Streams.

Verwenden von DynamoDB-Batchoperationen in AWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync unterstützt die Verwendung von Amazon DynamoDB-Batchoperationen für eine oder mehrere Tabellen in einer einzigen Region. Zu den unterstützten Vorgängen gehören `BatchGetItem`, `BatchPutItem` und `BatchDeleteItem`. Durch die Verwendung dieser Funktionen in AWS AppSync können Sie beispielsweise folgende Aufgaben ausführen:

- Übermitteln einer Liste von Schlüsseln in einer einzigen Abfrage und Zurückgabe der Ergebnisse aus einer Tabelle
- Lesen der Datensätze aus einer oder mehrerer Tabellen in einer einzigen Abfrage
- Schreiben von Datensätzen in großen Mengen in einer oder mehreren Tabellen
- Bedingtes Schreiben oder Löschen von Datensätzen in mehreren Tabellen, zwischen denen möglicherweise eine Beziehung besteht

Die Verwendung von Batch-Operationen mit DynamoDB AWS AppSync ist eine fortgeschrittene Technik, die etwas mehr Nachdenken und Wissen über Ihre Backend-Operationen und

Tabellenstrukturen erfordert. Darüber hinaus AWS AppSync weisen Batch-Operationen zwei wesentliche Unterschiede zu Vorgängen ohne Batch auf:

- Die Rolle der Datenquelle muss über Berechtigungen für alle Tabellen verfügen, auf die Resolver zugreifen wird.
- Die Tabellenangaben für einen Resolver sind Teil der Zuweisungsvorlage.

Berechtigungen

Wie bei anderen Resolvern müssen Sie eine Datenquelle in erstellen AWS AppSync und entweder eine Rolle erstellen oder eine vorhandene verwenden. Da Batchoperationen unterschiedliche Berechtigungen für DynamoDB-Tabellen erfordern, müssen Sie den konfigurierten Rollen Berechtigungen für Lese- oder Schreibaktionen gewähren:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-east-1:account:table/TABLENAME",
                "arn:aws:dynamodb:region:account:table/TABLENAME/*"
            ]
        }
    ]
}
```

Hinweis: Rollen sind an Datenquellen in einer Datenquelle gebunden AWS AppSync, und Resolver für Felder werden für eine Datenquelle aufgerufen. Für Datenquellen, die für den Abruf von DynamoDB konfiguriert sind, ist nur eine Tabelle angegeben, um die Konfiguration zu vereinfachen. Möchten Sie jedoch einen Stapelvorgang an mehreren Tabellen mit nur einem Resolver ausführen, stellt dies eine erweiterte Aufgabe dar und Sie müssen der Rolle dieser Datenquelle Zugriff auf

alle Tabellen gewähren, mit denen der Resolver interagieren wird. Dies kann im Feld Ressource (Resource) in der IAM-Richtlinie weiter oben eingerichtet werden. Die Konfiguration der Tabellen, um Stapelaufrufe auszuführen, erfolgt in der Resolver-Vorlage und wird nachfolgend beschrieben.

Datenquelle

Der Einfachheit halber verwenden wir die gleiche Datenquelle für alle in diesem Tutorial verwendeten Resolver. Erstellen Sie auf der Registerkarte Datenquellen eine neue DynamoDB-Datenquelle und geben Sie ihr einen Namen. BatchTutorial Sie können den Namen der Tabelle frei wählen, da Tabellennamen als Teil der Zuweisungsvorlage für Anforderungen bei Stapelvorgängen angegeben werden. Wir geben der Tabelle den Namen `empty`.

Für diese Anleitung kann jede Rolle mit der folgenden eingebundene Richtlinie verwendet werden:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-east-1:account:table/Posts",
                "arn:aws:dynamodb:region:account:table/Posts/*",
                "arn:aws:dynamodb:us-east-1:account:table/locationReadings",
                "arn:aws:dynamodb:region:account:table/locationReadings/*",
                "arn:aws:dynamodb:us-east-1:account:table/temperatureReadings",
                "arn:aws:dynamodb:region:account:table/temperatureReadings/*"
            ]
        }
    ]
}
```

Stapelvorgang mit einer Tabelle

> **Warning**
>
> `BatchPutItem`und `BatchDeleteItem` werden nicht unterstützt, wenn sie zusammen mit der Konflikterkennung und -lösung verwendet werden. Diese Einstellungen müssen deaktiviert werden, um mögliche Fehler zu vermeiden.

In diesem Beispiel gehen wir davon aus, dass Sie über eine einzelne Tabelle mit dem Namen Posts verfügen, zu der Sie anhand von Stapelvorgängen Elemente hinzufügen bzw. von der Sie Elemente entfernen möchten. Verwenden Sie das folgende Schema und beachten Sie, dass wir für die Abfrage eine Liste mit folgenden Elementen übergeben IDs:

```
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
}
```

Fügen Sie dem Feld `batchAdd()` mithilfe der folgenden Zuweisungsvorlage für Anforderungen einen Resolver an. Hierdurch wird automatisch jedes Element im GraphQL- `input PostInput`-Typ berücksichtigt und eine Zuordnung erstellt, wie sie für die `BatchPutItem`-Operation erforderlich ist:

```
#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)
    }
}
```

In diesem Fall handelt es sich bei der Zuweisungsvorlage für Antworten um einen einfachen Pass-Through. Beachten Sie jedoch, dass der Tabellenname als `..data.Posts` an das Kontextobjekt angefügt wird:

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

Navigieren Sie jetzt zur Seite Queries (Abfragen) der AWS AppSync-Konsole und führen Sie die folgende batchAdd-Mutation aus:

```
mutation add {
    batchAdd(posts:[{
            id: 1 title: "Running in the Park"},{
            id: 2 title: "Playing fetch"
        }]){
            id
            title
    }
}
```

Sie sollten die Ergebnisse auf dem Bildschirm sehen und können unabhängig voneinander über die DynamoDB-Konsole überprüfen, ob beide Werte in die Tabelle Posts geschrieben wurden.

Fügen Sie als Nächstes dem Feld `batchGet()` mithilfe der folgenden Zuweisungsvorlage für Anforderungen einen Resolver an. Hierdurch wird automatisch jedes Element im GraphQL `ids:[]`-Typ berücksichtigt und eine Zuordnung erstellt, wie sie für die `BatchGetItem`-Operation erforderlich ist:

```
#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"}
                }
        }
    }
}
```

In diesem Fall handelt es sich bei der Zuweisungsvorlage für Anforderungen wiederum um einen einfachen Pass-Through, dem erneut dem Context-Objekt des Tabellennamens `..data.Posts` angefügt wurde:

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

Navigieren Sie wieder zur Seite Queries (Abfragen) der AWS AppSync-Konsole und führen Sie die folgende batchGet-Abfrage aus:

```
query get {
    batchGet(ids:[1,2,3]){
        id
        title
    }
}
```

Dies sollte die Ergebnisse für die beiden `id`-Werte, die Sie zuvor hinzugefügt haben, zurückgeben. Beachten Sie, dass ein `null`-Wert für die `id` mit einem Wert von `3` zurückgegeben wurde. Das liegt daran, dass bisher noch kein Datensatz mit diesem Wert in Ihrer Posts-Tabelle vorhanden war. Beachten Sie auch, dass die Ergebnisse in derselben Reihenfolge AWS AppSync zurückgegeben

werden wie die an die Abfrage übergebenen Schlüssel. Dies ist eine zusätzliche Funktion, die in Ihrem AWS AppSync Namen ausgeführt wird. Wenn Sie jetzt zu `batchGet(ids:[1,3,2)` wechseln, sehen Sie, dass sich die Reihenfolge geändert hat. Sie können darüber hinaus erkennen, welche `id` einen `null`-Wert zurückgegeben hat.

Fügen Sie zum Schluss dem Feld `batchDelete()` mithilfe der folgenden Zuweisungsvorlage für Anforderungen einen Resolver an. Hierdurch wird automatisch jedes Element im GraphQL `ids:[]`-Typ berücksichtigt und eine Zuordnung erstellt, wie sie für die `BatchGetItem`-Operation erforderlich ist:

```
#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)
    }
}
```

In diesem Fall handelt es sich bei der Zuweisungsvorlage für Anforderungen wiederum um einen einfachen Pass-Through, dem erneut dem Context-Objekt des Tabellennamens `..data.Posts` angefügt wurde:

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

Navigieren Sie jetzt zurück zur Seite Queries (Abfragen) der AWS AppSync-Konsole und führen Sie die folgende batchDelete-Mutation aus:

```
mutation delete {
    batchDelete(ids:[1,2]){ id }
}
```

Die Datensätze mit der `id` 1 und 2 sollten jetzt gelöscht sein. Wenn Sie die frühere `batchGet()`-Abfrage ausführen, sollten diese `null` zurückgeben.

Stapelvorgang mit mehreren Tabellen

> **Warning**
>
> `BatchPutItem`und `BatchDeleteItem` werden nicht unterstützt, wenn sie zusammen mit der Konflikterkennung und -lösung verwendet werden. Diese Einstellungen müssen deaktiviert werden, um mögliche Fehler zu vermeiden.

AWS AppSync ermöglicht es Ihnen auch, tabellenübergreifende Batch-Operationen durchzuführen. Lassen Sie uns hierzu eine komplexere Anwendung erstellen. Stellen Sie sich vor, wir entwickeln eine Gesundheits-App für Haustiere, bei der Sensoren den Standort und die Körpertemperatur der Tiere aufzeichnen. Diese Sensoren sind batteriebetrieben und versuchen, alle paar Minuten eine Verbindung mit dem Netzwerk aufzubauen. Wenn ein Sensor eine Verbindung herstellt, sendet er seine Messwerte an unsere AWS AppSync API. Die Daten werden von Auslösern analysiert und dem Besitzer des Haustiers in einem Dashboard angezeigt. Lassen Sie uns jetzt die Interaktionen zwischen dem Sensor und dem Backend-Datenspeicher näher untersuchen.

Als Voraussetzung erstellen wir zunächst zwei DynamoDB-Tabellen. LocationReadings speichert Sensorstandortwerte und TemperatureReadings speichert Sensortemperaturwerte. Beide Tabellen haben zufällig die gleiche Primärschlüsselstruktur: `sensorId (String)` als der Partitionsschlüssel und `timestamp (String)` als der Sortierschlüssel.

Verwenden wir das folgende GraphQL-Schema:

```
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 - Aufzeichnen von Sensormesswerten

Nachdem eine Verbindung zum Internet hergestellt wurde, müssen unsere Sensoren in der Lage sein, ihre Messwerte zu senden. Sie verwenden dazu die API GraphQL-Feld `Mutation.recordReadings`. Um unsere API in Betrieb nehmen zu können, müssen wir zunächst einen Resolver anfügen.

Wählen Sie hierzu Anfügen (Attach) neben dem Feld `Mutation.recordReadings` aus. Wählen Sie auf dem nächsten Bildschirm die gleiche `BatchTutorial`-Datenquelle wie zu Beginn dieser Anleitung aus.

Fügen Sie die folgende Zuweisungsvorlage für Anforderungen hinzu.

Zuweisungsvorlage für Anforderungen

```
## 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)
    }
}
```

Wie Sie sehen, können wir über die `BatchPutItem`-Operation mehrere Tabellen angeben.

Verwenden Sie die folgende Zuweisungsvorlage für Antworten.

Zuweisungsvorlage für Antworten

```
## 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)
```

Bei Stapelvorgängen können durch den Aufruf sowohl Fehler als auch Ergebnisse zurückgegeben werden. In diesem Fall können wir eine zusätzliche Fehlerbehandlung vornehmen.

Hinweis: Die Verwendung von `$utils.appendError()` lässt sich mit `$util.error()` vergleichen, mit dem großen Unterschied, dass die Bewertung der Zuweisungsvorlage nicht unterbrochen wird. Stattdessen wird signalisiert, dass das Feld zwar eine Fehlermeldung hervorrief, die Beurteilung der Vorlage jedoch abgeschlossen wurde und infolgedessen die Daten wieder an den Aufrufer zurückgegeben werden konnten. Wir empfehlen Ihnen die Verwendung von `$utils.appendError()`, wenn Ihrer Anwendung zumindest Teilergebnisse zurückgegeben werden müssen.

Speichern Sie den Resolver und navigieren Sie zur Abfrageseite der AWS AppSync Konsole. Senden wir jetzt einige Sensormesswerte.

Führen Sie die folgende Mutation aus:

```
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
    }
  }
}
```

Wir werden 10 Sensormesswerte in einer Mutation senden, wobei die Messwerte in zwei Tabellen aufgeteilt sind. Verwenden Sie die DynamoDB-Konsole, um zu überprüfen, ob Daten sowohl in den LocationReadings - als auch in den TemperatureReadings-Tabellen angezeigt werden.

BatchDeleteItem - Löschen von Sensormesswerten

Gleichzeitig müssen wir auch Stapel mit Sensormesswerten löschen können. Dazu verwenden wir das GraphQL-Feld `Mutation.deleteReadings`. Wählen Sie hierzu Anfügen (Attach) neben dem Feld `Mutation.recordReadings` aus. Wählen Sie auf dem nächsten Bildschirm die gleiche `BatchTutorial`-Datenquelle wie zu Beginn dieser Anleitung aus.

Verwenden Sie die folgende Zuweisungsvorlage für Anforderungen.

Zuweisungsvorlage für Anforderungen

```
## 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)
    }
}
```

Die Zuweisungsvorlage für Antworten ist die gleiche, die wir bereits für `Mutation.recordReadings` verwendet haben.

Zuweisungsvorlage für Antworten

```
## 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)
```

Speichern Sie den Resolver und navigieren Sie zur Abfrageseite der AWS AppSync Konsole. Lassen Sie uns jetzt einige der Sensormesswerte löschen!

Führen Sie die folgende Mutation aus:

```
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
    }
  }
}
```

Überprüfen Sie über die DynamoDB-Konsole, ob diese beiden Messwerte aus den Tabellen LocationReadings und TemperatureReadings gelöscht wurden.

BatchGetItem - Messwerte abrufen

Eine weitere gängige Operation für unsere Gesundheits-App für Haustiere bestünde darin, die Messwerte eines Sensors zu einem bestimmten Zeitpunkt abzurufen. Fügen wir dazu einen Resolver zum GraphQL-Feld `Query.getReadings` unseres Schemas hinzu. Wählen Sie Anfügen (Attach) und auf dem nächsten Bildschirm die gleiche `BatchTutorial`-Datenquelle wie zu Beginn dieser Anleitung aus.

Fügen wir die folgende Zuweisungsvorlage für Anforderungen hinzu.

Zuweisungsvorlage für Anforderungen

```
## 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
        }
    }
```

```
}
```

Beachten Sie, dass wir die BatchGetItemOperation jetzt verwenden.

Unsere Zuweisungsvorlage für Antworten unterscheidet sich etwas, da wir uns für die Rückgabe einer `SensorReading`-Liste entschieden haben. Ordnen wir jetzt das Aufrufergebnis der gewünschten Form zu.

Zuweisungsvorlage für Antworten

```
## 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)
```

Speichern Sie den Resolver und navigieren Sie zur Seite Abfragen der AWS AppSync Konsole. Rufen wir jetzt die Sensormesswerte ab!

Führen Sie die folgende Abfrage aus:

```
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
    }
  }
}
```

Wir haben erfolgreich die Verwendung von DynamoDB-Batchoperationen unter Verwendung von demonstriert. AWS AppSync

Fehlerbehandlung

In AWS AppSync Datenquellenoperationen können manchmal Teilergebnisse zurückgegeben werden. Wir verwenden den Begriff Teilergebnis, wenn bei der Ausgabe einer Operation einige Daten fehlen und darin ein Fehler enthalten ist. Da die Fehlerbehandlung grundsätzlich anwendungsspezifisch erfolgt, bietet Ihnen AWS AppSync die Möglichkeit, Fehler in der Zuweisungsvorlage für Antworten zu handhaben. Sollte beim Resolver ein Aufruffehler vorliegen, lässt sich dieser im Kontext an `$ctx.error` erkennen. Aufruffehler umfassen immer eine Nachricht und eine Art, auf die über die Eigenschaften `$ctx.error.message` und `$ctx.error.type`zugegriffen werden kann. Während die Zuweisungsvorlagen für Antworten aufgerufen werden, können Sie Teilergebnisse auf drei verschiedene Arten handhaben:

1. Übergehen des Aufruffehlers, indem nur Daten zurückgegeben werden
2. Melden des Fehlers (unter Verwendung von `$util.error(...)`), wodurch die Bewertung der Zuweisungsvorlage für Antworten unterbrochen wird und keine Daten zurückgegeben werden
3. Anfügen eines Fehler (unter Verwendung von `$util.appendError(...)`) und der gleichzeitigen Zurückgabe von Daten

Wir veranschaulichen jetzt jeden der drei oben genannten Punkte mit den DynamoDB-Stapelvorgängen.

DynamoDB-Stapelvorgänge

Mit DynamoDB-Stapelvorgängen ist es möglich, dass ein Stapel teilweise abgeschlossen wird. Das bedeutet, dass einige der angeforderten Elemente oder Schlüssel nicht verarbeitet werden. Wenn AWS AppSync ein Stapel nicht abgeschlossen werden kann, werden unverarbeitete Elemente und ein Aufruffehler für den Kontext angezeigt.

Wir werden die Fehlerbehandlung mithilfe der `Query.getReadings`-Feldkonfiguration der `BatchGetItem`-Operation aus dem vorherigen Abschnitt dieser Anleitung implementieren.

Dieses Mal nehmen wir jedoch an, dass bei der Ausführung des `Query.getReadings`-Felds die DynamoDB-Tabelle `temperatureReadings` den bereitgestellten Durchsatz überschritten hat. DynamoDB hat ProvisionedThroughputExceededExceptionbeim zweiten Versuch a ausgelöst, AWS AppSync um die verbleibenden Elemente im Stapel zu verarbeiten.

Der folgende JSON-Code stellt den serialisierten Kontext dar, nachdem der DynamoDB-Stapel aufgerufen, aber noch bevor die Zuweisungsvorlage für Antworten ausgewertet wurde.

```
{
  "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": []
}
```

Beachten Sie in diesem Kontext die folgenden Dinge:

- der Aufruffehler wurde für den Kontext bei `$ctx.error` by festgelegt AWS AppSync, und der Fehlertyp wurde auf DynamoDB: gesetzt. ProvisionedThroughputExceededException
- die Ergebnisse werden pro Tabelle unter `$ctx.result.data` zugeordnet, auch wenn ein Fehler vorhanden ist
- unverarbeitete Schlüssel finden Sie unter `$ctx.result.data.unprocessedKeys`. Hier AWS AppSync konnte das Element mit dem Schlüssel (sensorId:1, timestamp:2018-02-01T 17:21:05.000 + 08:00) aufgrund des unzureichenden Tabellendurchsatzes nicht abgerufen werden.

Hinweis: Für `BatchPutItem` gilt `$ctx.result.data.unprocessedItems`. Für `BatchDeleteItem` gilt `$ctx.result.data.unprocessedKeys`.

Sie können mit diesem Fehler auf drei verschiedene Arten umgehen.

1. Übergehen des Aufruffehlers

Wenn Daten ohne Verarbeitung des Aufruffehlers zurückgegeben werden, wird der Fehler effektiv übergangen. Auf diese Weise ist das Ergebnis für das angegebene GraphQL-Feld immer erfolgreich.

Die von uns geschriebene Zuweisungsvorlage für Antworten ist vertraut und konzentriert sich auf die Ergebnisdaten.

Zuweisungsvorlage für Antworten:

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

GraphQL-Antwort:

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

Der Fehlerantwort werden keine Fehler hinzugefügt, da nur auf Daten reagiert wird.

2. Melden eines Fehlers, um die Vorlagenausführung abzubrechen

Wenn Teilausfälle aus der Perspektive des Clients als vollständige Ausfälle behandelt werden sollten, können Sie die Ausführung der Vorlage abbrechen, um die Rückgabe von Daten zu verhindern. Das Dienstprogramm `$util.error(...)` erzielt genau dieses Verhalten.

Zuweisungsvorlage für Antworten:

```
## 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)
```

GraphQL-Antwort:

```
{
  "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. (...)"
    }
  ]
}
```

Auch wenn der DynamoDB Stapelvorgang möglicherweise einige Ergebnisse zurückgegeben hat, haben wir uns entschieden, einen Fehler zu melden, indem das GraphQL-Feld `getReadings` mit Null angezeigt und der Fehler dem Fehler-Block der GraphQL-Antwort hinzugefügt wurde.

3. Anfügen eines Fehlers bei der Rückgabe von Daten und Fehlern

In bestimmten Fällen können Anwendungen Teilergebnisse zurückgeben und die Clients über die unverarbeiteten Elemente benachrichtigen und so die Benutzererfahrung verbessern. Die Clients können entscheiden, eine Wiederholung zu implementieren oder den Fehler an den Endbenutzer zu übermitteln. Über das Dienstprogramm `$util.appendError(...)` kann der Anwendungs-Designer Fehler im Hinblick auf den Kontext anfügen, ohne dass sich dies auf die Bewertung der Vorlage auswirkt. Nach der Auswertung der Vorlage AWS AppSync werden alle Kontextfehler verarbeitet, indem sie an den Fehlerblock der GraphQL-Antwort angehängt werden.

Zuweisungsvorlage für Antworten:

```
#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)
```

Wir haben sowohl den Aufruffehler als auch das unprocessedKeys-Element an den Fehlerblock der GraphQL-Antwort weitergeleitet. Das `getReadings`-Feld gibt auch partielle Daten aus der locationReadings-Tabelle zurück, wie Sie in der nachstehenden Antwort sehen können.

GraphQL-Antwort:

```
{
  "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. (...)"
```

```
    }
  ]
}
```

Durchführen von DynamoDB-Transaktionen in AWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync unterstützt die Verwendung von Amazon DynamoDB-Transaktionsoperationen für eine oder mehrere Tabellen in einer einzigen Region. Zu den unterstützten Operationen gehören `TransactGetItems` und `TransactWriteItems`. Mithilfe dieser Funktionen können Sie Aufgaben wie die folgenden ausführen: AWS AppSync

- Übermitteln einer Liste von Schlüsseln in einer einzigen Abfrage und Zurückgabe der Ergebnisse aus einer Tabelle
- Lesen der Datensätze aus einer oder mehrerer Tabellen in einer einzigen Abfrage
- Schreiben Sie Datensätze in einer Transaktion auf irgendeine all-or-nothing Weise in eine oder mehrere Tabellen
- Ausführen von Transaktionen, wenn einige Bedingungen erfüllt sind

Berechtigungen

Wie bei anderen Resolvern müssen Sie eine Datenquelle in erstellen AWS AppSync und entweder eine Rolle erstellen oder eine vorhandene verwenden. Da Transaktionsvorgänge unterschiedliche Berechtigungen für DynamoDB-Tabellen erfordern, müssen Sie den konfigurierten Rollen Berechtigungen für Lese- oder Schreibaktionen gewähren:

JSON

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

```
        {
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-east-1:accountId:table/TABLENAME",
                "arn:aws:dynamodb:region:accountId:table/TABLENAME/*"
            ]
        }
    ]
}
```

Hinweis: Rollen sind an Datenquellen in einer Datenquelle gebunden AWS AppSync, und Resolver für Felder werden für eine Datenquelle aufgerufen. Für Datenquellen, die für den Abruf von DynamoDB konfiguriert sind, ist nur eine Tabelle angegeben, um die Konfiguration zu vereinfachen. Möchten Sie jedoch einen Transaktionsvorgang an mehreren Tabellen mit nur einem Resolver ausführen, stellt dies eine erweiterte Aufgabe dar, und Sie müssen der Rolle dieser Datenquelle Zugriff auf alle Tabellen gewähren, mit denen der Resolver interagieren wird. Dies kann im Feld Ressource (Resource) in der IAM-Richtlinie weiter oben eingerichtet werden. Die Konfiguration der Tabellen, um Transaktionsaufrufe auszuführen, erfolgt in der Resolver-Vorlage und wird nachfolgend beschrieben.

Datenquelle

Der Einfachheit halber verwenden wir die gleiche Datenquelle für alle in diesem Tutorial verwendeten Resolver. Erstellen Sie auf der Registerkarte Datenquellen eine neue DynamoDB-Datenquelle und geben Sie ihr einen Namen. TransactTutorial Sie können den Namen der Tabelle frei wählen, da Tabellennamen als Teil der Zuweisungsvorlage für Anforderungen bei Transaktionsvorgängen angegeben werden. Wir geben der Tabelle den Namen `empty`.

Wir haben zwei Tabellen mit den Bezeichnungen savingAccounts und checkingAccounts, beide mit `accountNumber` als Partitionsschlüssel, und eine transactionHistory-Tabelle mit `transactionId` als Partitionsschlüssel.

Für dieses Tutorial kann jede Rolle mit der folgenden eingebundenen Richtlinie verwendet werden: Ersetzen Sie `region` und `accountId` durch Ihre Region und Konto-ID:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-east-1:accountId:table/savingAccounts",
                "arn:aws:dynamodb:region:accountId:table/savingAccounts/*",
                "arn:aws:dynamodb:us-east-1:accountId:table/checkingAccounts",
                "arn:aws:dynamodb:region:accountId:table/checkingAccounts/*",
                "arn:aws:dynamodb:us-east-1:accountId:table/transactionHistory",
                "arn:aws:dynamodb:region:accountId:table/transactionHistory/*"
            ]
        }
    ]
}
```

Transaktionen

Für dieses Beispiel ist der Kontext eine klassische Banktransaktion, bei der wir `TransactWriteItems` für folgende Aktionen verwenden:

- Überweisen von Geld von Sparkonten auf Girokonten
- Generieren neuer Transaktionsdatensätze für jede Transaktion

Anschließend verwenden wir `TransactGetItems`, um Details der Sparkonten und Girokonten abzurufen.

> **Warning**
>
> `TransactWriteItems`wird nicht unterstützt, wenn es zusammen mit Konflikterkennung und Konfliktlösung verwendet wird. Diese Einstellungen müssen deaktiviert werden, um mögliche Fehler zu vermeiden.

Wir definieren unser GraphQL-Schema wie folgt:

```
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 - Konten auffüllen

Um Geld zwischen Konten zu übertragen, müssen wir die Tabelle mit den Details füllen. Wir verwenden dazu die GraphQL-Operation `Mutation.populateAccounts`.

Klicken Sie im Abschnitt Schema neben dem `Mutation.populateAccounts` Vorgang auf Anhängen. Gehen Sie zu VTL Unit Resolvers und wählen Sie dann dieselbe `TransactTutorial` Datenquelle aus.

Verwenden Sie die folgende Zuweisungsvorlage für Anforderungen:

Zuweisungsvorlage für Anforderungen

```
#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)
}
```

Und die folgende Zuweisungsvorlage für Antworten:

Zuweisungsvorlage für Antworten

```
#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)
```

Speichern Sie den Resolver und navigieren Sie zum Abschnitt Abfragen der AWS AppSync Konsole, um die Konten aufzufüllen.

Führen Sie die folgende Mutation aus:

```
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
    }
  }
}
```

Wir haben drei Sparkonten und drei Girokonten in einer Mutation gefüllt.

Verwenden Sie die DynamoDB-Konsole, um zu überprüfen, ob Daten sowohl in den Tabellen SavingAccounts als auch in checkingAccounts angezeigt werden.

TransactWriteItems - Geld überweisen

Fügen Sie der `transferMoney`-Mutation mithilfe der folgenden Zuweisungsvorlage für Anforderungen einen Resolver an. Beachten Sie, dass die Werte von `amounts`, `savingAccountNumbers` und `checkingAccountNumbers` identisch sind.

```
#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)
}
```

Wir werden drei Banktransaktionen in einem einzigen `TransactWriteItems`-Vorgang durchführen. Verwenden Sie die folgende Vorlage für die Zuordnung von Antworten:

```
#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)
```

Navigieren Sie nun zum Abschnitt Queries (Abfragen) der AWS AppSync -Konsole, und führen Sie die transferMoney-Mutation wie folgt aus:

```
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
    }
  }
}
```

Wir haben zwei Banktransaktionen in einer Mutation gesendet. Verwenden Sie die DynamoDB-Konsole, um zu überprüfen, ob Daten in den Tabellen SavingAccounts, checkingAccounts und TransactionHistory angezeigt werden.

TransactGetItems - Konten abrufen

Um die Details aus dem Sparkonto und dem Girokonto in einer einzigen Transaktionsanforderung abzurufen, hängen wir einen Resolver an die `Query.getAccounts`-GraphQL-Operation in unserem Schema an. Wählen Sie Anhängen, gehen Sie zu VTL Unit Resolvers und wählen Sie dann auf dem nächsten Bildschirm dieselbe `TransactTutorial` Datenquelle aus, die Sie zu Beginn des Tutorials erstellt haben. Konfigurieren Sie die Vorlagen wie folgt:

Zuweisungsvorlage für Anforderungen

```
#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)
}
```

Zuweisungsvorlage für Antworten

```
#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)
```

Speichern Sie den Resolver und navigieren Sie zu den Query-Abschnitten der Konsole. AWS AppSync Um die Sparkonten und die Girokonten abzurufen, führen Sie die folgende Abfrage aus:

```
query getAccounts {
  getAccounts(
    savingAccountNumbers: ["1", "2", "3"],
    checkingAccountNumbers: ["1", "2"]
  ) {
    savingAccounts {
      accountNumber
      username
      balance
    }
    checkingAccounts {
      accountNumber
      username
      balance
    }
  }
}
```

Wir haben erfolgreich die Verwendung von DynamoDB-Transaktionen unter Verwendung von demonstriert. AWS AppSync

Verwendung von HTTP-Resolvern in AWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync ermöglicht es Ihnen, unterstützte Datenquellen (d. h. Amazon DynamoDB AWS Lambda, Amazon OpenSearch Service oder Amazon Aurora) zu verwenden, um verschiedene Operationen durchzuführen, zusätzlich zu beliebigen HTTP-Endpunkten zur Auflösung von GraphQL-Feldern. Sobald die HTTP-Endpunkte verfügbar sind, können Sie sie mit einer Datenquelle verbinden. Anschließend können Sie einen Resolver im Schema konfigurieren, um GraphQL-Operationen wie Abfragen, Mutationen und Abonnements auszuführen. Dieses Tutorial führt Sie durch einige häufig auftretende Beispiele.

In diesem Tutorial verwenden Sie eine REST-API (erstellt mit Amazon API Gateway und Lambda) mit einem AWS AppSync GraphQL-Endpunkt.

One-Click-Setup

Wenn Sie automatisch einen GraphQL-Endpunkt AWS AppSync mit einem konfigurierten HTTP-Endpunkt einrichten möchten (mit Amazon API Gateway und Lambda), können Sie die folgende AWS CloudFormation Vorlage verwenden:

Erstellen einer REST-API

Sie können die folgende AWS CloudFormation Vorlage verwenden, um einen REST-Endpunkt einzurichten, der für dieses Tutorial funktioniert:

Der AWS CloudFormation Stack führt die folgenden Schritte aus:

1. Richtet eine Lambda-Funktion ein, die die Geschäftslogik für den Mikroservice enthält.
2. Richtet eine API-Gateway-REST-API mit der folgenden endpoint/method/content Typkombination ein:

API-Ressourcenpfad	HTTP-Methode	Unterstützter Inhaltstyp
/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

Erstellen der GraphQL-API

Um die GraphQL-API zu erstellen in AWS AppSync:

- Öffnen Sie die AWS AppSync Konsole und wählen Sie Create API.
- Geben Sie für den API-Namen `UserData` ein.
- Wählen Sie Custom schema (Benutzerdefiniertes Schema).
- Wählen Sie Erstellen aus.

Die AWS AppSync Konsole erstellt mithilfe des API-Schlüsselauthentifizierungsmodus eine neue GraphQL-API für Sie. Sie können in der Konsole den Rest der GraphQL-API einrichten und im weiteren Verlauf dieses Tutorials abfragen.

Erstellen eines GraphQL-Schemas

Da wir jetzt eine GraphQL-API haben, erstellen Sie ein GraphQL-Schema. Stellen Sie im Schema-Editor in der AWS AppSync Konsole sicher, dass Ihr Schema dem folgenden Schema entspricht:

```
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
}
```

Konfigurieren der HTTP-Datenquelle

Gehen Sie wie folgt vor, um die HTTP-Datenquelle zu konfigurieren:

- Wählen Sie auf der DataSourcesRegisterkarte Neu aus, und geben Sie dann einen benutzerfreundlichen Namen für die Datenquelle ein (z. B.HTTP).
- Wählen Sie für Data source type (Datenquellentyp) die Option HTTP aus.
- Legen Sie den Endpunkt auf den erstellten API-Gateway-Endpunkt fest. Stellen Sie sicher, dass Sie den Namen der Stufe nicht in den Namen des Endpunkts einschließen.

Hinweis: Derzeit werden nur öffentliche Endpunkte von AWS AppSync unterstützt.

Hinweis: Weitere Informationen zu den Zertifizierungsstellen, die vom AWS AppSync Dienst anerkannt werden, finden Sie unter Zertifizierungsstellen (CA) Recognized by AWS AppSync for HTTPS Endpoints.

Konfigurieren von Resolvern

In diesem Schritt verbinden Sie die HTTP-Datenquelle mit der Abfrage getUser.

Richten Sie den Resolver wie folgt ein:

- Wählen Sie die Registerkarte Schema aus.
- Suchen Sie im Bereich Data types (Datentypen) rechts unter dem Query type (Abfragetyp) das Feld getUser und wählen Sie Attach(Anfügen) aus.
- Wählen Sie für Data source name (Datenquellenname) die Option HTTP aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) den folgenden Code ein:

```
{
    "version": "2018-05-29",
    "method": "GET",
    "params": {
        "headers": {
            "Content-Type": "application/json"
        }
    },
    "resourcePath": $util.toJson("/v1/users/${ctx.args.id}")
}
```

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) den folgenden Code ein:

```
## 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
```

- Wählen Sie auf der Registerkarte Query (Abfrage) und führen Sie dann folgende Abfrage aus:

```
query GetUser{
    getUser(id:1){
        id
        username
    }
}
```

Folgende Antwort sollte zurückgegeben werden:

```
{
    "data": {
        "getUser": {
            "id": "1",
            "username": "nadia"
```

```
        }
    }
}
```

- Wählen Sie die Registerkarte Schema aus.
- Suchen Sie im Bereich Data types (Datentypen) rechts unter Mutation das Feld addUser und wählen Sie Attach (Anfügen) aus.
- Wählen Sie für Data source name (Datenquellenname) die Option HTTP aus.
- Fügen Sie unter Configure the request mapping template (Zuweisungsvorlage für Anforderungen konfigurieren) den folgenden Code ein:

```
{
    "version": "2018-05-29",
    "method": "POST",
    "resourcePath": "/v1/users",
    "params":{
      "headers":{
        "Content-Type": "application/json",
      },
      "body": $util.toJson($ctx.args.userInput)
    }
}
```

- Fügen Sie unter Configure the response mapping template (Zuweisungsvorlage für Antworten konfigurieren) den folgenden Code ein:

```
## 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
```

- Wählen Sie auf der Registerkarte Query (Abfrage) und führen Sie dann folgende Abfrage aus:

```
mutation addUser{
    addUser(userInput:{
        id:"2",
        username:"shaggy"
    }){
        id
        username
    }
}
```

Folgende Antwort sollte zurückgegeben werden:

```
{
    "data": {
        "getUser": {
        "id": "2",
        "username": "shaggy"
        }
    }
}
```

Dienste aufrufen AWS

Sie können HTTP-Resolver verwenden, um eine GraphQL-API-Schnittstelle für AWS Dienste einzurichten. HTTP-Anfragen an AWS müssen mit dem Signature Version 4-Prozess signiert werden, damit identifiziert werden AWS kann, wer sie gesendet hat. AWS AppSync berechnet die Signatur in Ihrem Namen, wenn Sie der HTTP-Datenquelle eine IAM-Rolle zuordnen.

Sie stellen zwei zusätzliche Komponenten bereit, um AWS Dienste mit HTTP-Resolvern aufzurufen:

- Eine IAM-Rolle mit Berechtigungen zum Aufrufen des Dienstes AWS APIs
- Signierkonfiguration in der Datenquelle

Wenn Sie den ListGraphqlApis Vorgang beispielsweise mit HTTP-Resolvern aufrufen möchten, erstellen Sie zunächst eine IAM-Rolle, der die folgende AWS AppSync Richtlinie zugewiesen ist:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "appsync:ListGraphqlApis"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Als Nächstes erstellen Sie die HTTP-Datenquelle für. AWS AppSync In diesem Beispiel rufen Sie AWS AppSync in der Region USA West (Oregon) auf. Richten Sie die folgende HTTP-Konfiguration in einer Datei mit dem Namen `http.json` ein, die die Signierregion und den Servicenamen enthält:

```
{
    "endpoint": "https://appsync.us-west-2.amazonaws.com/",
    "authorizationConfig": {
        "authorizationType": "AWS_IAM",
        "awsIamConfig": {
            "signingRegion": "us-west-2",
            "signingServiceName": "appsync"
        }
    }
}
```

Verwenden Sie dann die, AWS CLI um die Datenquelle mit einer zugehörigen Rolle wie folgt zu erstellen:

```
aws appsync create-data-source --api-id <API-ID> \
                               --name AWSAppSync \
                               --type HTTP \
                               --http-config file:///http.json \
                               --service-role-arn <ROLE-ARN>
```

Wenn Sie einen Resolver an das Feld im Schema anhängen, verwenden Sie zum Aufrufen AWS AppSync die folgende Vorlage für die Anforderungszuweisung:

```
{
    "version": "2018-05-29",
    "method": "GET",
    "resourcePath": "/v1/apis"
}
```

Wenn Sie eine GraphQL-Abfrage für diese Datenquelle ausführen, AWS AppSync signiert die Anfrage mit der von Ihnen angegebenen Rolle und fügt die Signatur in die Anfrage ein. Die Abfrage gibt eine Liste von AWS AppSync GraphQL APIs in Ihrem Konto in dieser AWS Region zurück.

Verwenden von Aurora Serverless v2 mit AWS AppSync

Connect Ihre GraphQL-API mit Aurora Serverless-Datenbanken mithilfe von. AWS AppSync Mit dieser Integration können Sie SQL-Anweisungen über GraphQL-Abfragen, -Mutationen und -Subskriptionen ausführen und so flexibel mit Ihren relationalen Daten interagieren.

> **Note**
>
> Für dieses Tutorial wird die Region `US-EAST-1` verwendet.

Vorteile

- Nahtlose Integration zwischen GraphQL und relationalen Datenbanken
- Fähigkeit, SQL-Operationen über GraphQL-Schnittstellen durchzuführen
- Serverlose Skalierbarkeit mit Aurora Serverless v2
- Sicherer Datenzugriff über AWS Secrets Manager
- Schutz vor SQL-Injection durch Eingabebereinigung
- Flexible Abfragefunktionen, einschließlich Filter- und Bereichsoperationen

Häufige Anwendungsfälle

- Entwicklung skalierbarer Anwendungen mit relationalen Datenanforderungen

- Für deren Erstellung APIs sind sowohl GraphQL-Flexibilität als auch SQL-Datenbankfunktionen erforderlich
- Verwaltung von Datenoperationen durch GraphQL-Mutationen und Abfragen
- Implementierung sicherer Datenbankzugriffsmuster

In diesem Tutorial lernen Sie Folgendes.

- Richten Sie einen Aurora Serverless v2-Cluster ein
- Aktivieren Sie die Daten-API-Funktionalität
- Datenbankstrukturen erstellen und konfigurieren
- Definieren Sie GraphQL-Schemas für Datenbankoperationen
- Implementieren Sie Resolver für Abfragen und Mutationen
- Schützen Sie Ihren Datenzugriff durch eine angemessene Eingabebereinigung
- Führen Sie verschiedene Datenbankoperationen über GraphQL-Schnittstellen aus

Themen

- Richten Sie Ihren Datenbankcluster ein
- Aktivieren der Daten-API
- Datenbank und Tabelle erstellen
- GraphQL-Schema
- Connect Sie Ihre API mit Datenbankoperationen
- Ändern Sie Ihre Daten über die API
- Rufen Sie Ihre Daten ab
- Schützen Sie Ihren Datenzugriff

Richten Sie Ihren Datenbankcluster ein

Bevor Sie eine Amazon RDS-Datenquelle hinzufügen AWS AppSync, müssen Sie zunächst eine Daten-API auf einem Aurora Serverless v2-Cluster aktivieren und einen geheimen Schlüssel mithilfe von AWS Secrets Managerkonfigurieren. Sie können einen Aurora Serverless v2-Cluster erstellen, indem Sie: AWS CLI

```
aws rds create-db-cluster \
```

```
    --db-cluster-identifier appsync-tutorial \
    --engine aurora-mysql \
    --engine-version 8.0 \
    --serverless-v2-scaling-configuration MinCapacity=0,MaxCapacity=1 \
    --master-username USERNAME \
    --master-user-password COMPLEX_PASSWORD \
    --enable-http-endpoint
```

Dadurch wird ein ARN für den Cluster zurückgegeben.

Nachdem Sie den Cluster erstellt haben, müssen Sie mit dem folgenden Befehl eine Aurora Serverless v2-Instance hinzufügen.

```
aws rds create-db-instance \
    --db-cluster-identifier appsync-tutorial \
    --db-instance-identifier appsync-tutorial-instance-1 \
    --db-instance-class db.serverless \
    --engine aurora-mysql
```

> **Note**
>
> Die Aktivierung dieser Endpunkte benötigt Zeit. Sie können ihren Status in der Amazon RDS-Konsole auf der Registerkarte Konnektivität und Sicherheit für den Cluster überprüfen. Sie können den Status Ihres Clusters auch mit dem folgenden AWS CLI Befehl überprüfen.
>
> ```
> aws rds describe-db-clusters \
>     --db-cluster-identifier appsync-tutorial \
>     --query "DBClusters[0].Status"
> ```

Sie können ein Secret mithilfe der AWS Secrets Manager Konsole oder AWS CLI mit einer Eingabedatei wie der folgenden erstellen, indem Sie die `USERNAME` und `COMPLEX_PASSWORD` aus dem vorherigen Schritt verwenden.

```
{
    "username": "USERNAME",
    "password": "COMPLEX_PASSWORD"
}
```

Übergeben Sie dies als Parameter an AWS CLI:

```
aws secretsmanager create-secret --name HttpRDSSecret --secret-string file://creds.json
 --region us-east-1
```

Dadurch wird ein ARN für das Secret zurückgegeben.

Notieren Sie sich den ARN Ihres Aurora Serverless Clusters und Secret für die spätere Verwendung in der AppSync Konsole, wenn Sie eine Datenquelle erstellen.

Aktivieren der Daten-API

Die Daten-API können Sie auf Ihrem Cluster aktivieren, indem Sie die folgende Anleitung aus der RDS-Dokumentation ausführen. Die Daten-API muss aktiviert werden, bevor sie als AppSync Datenquelle hinzugefügt werden kann.

Datenbank und Tabelle erstellen

Sobald Sie Ihre Daten-API aktiviert haben, können Sie sicherstellen, dass sie mit dem `aws rds-data execute-statement` Befehl in der funktioniert AWS CLI. Dadurch wird sichergestellt, dass Ihr Aurora Serverless-Cluster korrekt konfiguriert ist, bevor Sie ihn zu Ihrer AppSync API hinzufügen. Erstellen Sie zunächst eine Datenbank namens TESTDB mit dem folgenden `--sql` Parameter:

```
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"
```

Wenn dies fehlerfrei ausgeführt wird, fügen Sie mit dem Befehl create table eine Tabelle hinzu:

```
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"
```

Wenn alles ohne Probleme gelaufen ist, können Sie den Cluster als Datenquelle zu Ihrer AppSync API hinzufügen.

GraphQL-Schema

Ihre Aurora Serverless-Daten-API ist eingerichtet und wird mit einer Tabelle ausgeführt. Jetzt erstellen wir ein GraphQL-Schema und fügen Resolver an, um Mutationen und Abonnements durchzuführen. Erstellen Sie eine neue API in der AWS AppSync Konsole, navigieren Sie zur Schemaseite und geben Sie Folgendes ein:

```
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
}
```

Speichern Sie Ihr Schema. Navigieren Sie zur Seite Data Sources (Datenquellen) und erstellen Sie eine neue Datenquelle. Wählen Sie für den Typ der Datenquelle Relationale Datenbank aus und geben Sie einen aussagekräftigen Namen ein. Verwenden Sie den im letzten Schritt erstellten Datenbanknamen und den darin erstellen Cluster ARN (Cluster-ARN). Für die Rolle können Sie entweder eine neue Rolle AppSync erstellen lassen oder eine mit einer Richtlinie erstellen, die der folgenden ähnelt:

JSON

```
{
    "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:*"
            ]
        }
    ]
}
```

Beachten Sie, dass diese Richtlinie zwei Anweisungen enthält, die den Zugriff der Rolle gewähren. Die erste Ressource ist Ihr Aurora Serverless-Cluster und die zweite ist Ihr AWS Secrets Manager ARN. Sie müssen BEIDE ARNs in der AppSync Datenquellenkonfiguration angeben, bevor Sie auf Erstellen klicken.

Übergeben Sie dies als Parameter an den AWS CLI.

```
aws secretsmanager create-secret \
  --name HttpRDSSecret \
  --secret-string file://creds.json \
  --region us-east-1
```

Dadurch wird ein ARN für das Secret zurückgegeben. Notieren Sie sich den ARN Ihres Aurora Serverless Clusters und Secret für später, wenn Sie eine Datenquelle in der AWS AppSync Konsole erstellen.

Erstellen Sie Ihre Datenbankstruktur

Sobald Sie Ihre Daten-API aktiviert haben, können Sie sicherstellen, dass sie mit dem `aws rds-data execute-statement` Befehl in der funktioniert AWS CLI. Dadurch wird sichergestellt, dass Ihr Aurora Serverless v2-Cluster korrekt konfiguriert ist, bevor Sie ihn zu Ihrer AWS AppSync API hinzufügen. Erstellen Sie zunächst eine Datenbank namens TESTDB mit dem folgenden `--sql` Parameter.

```
aws rds-data execute-statement \
                --resource-arn "arn:aws:rds:us-east-1:111122223333:cluster:appsync-
tutorial" \
                --secret-arn "arn:aws:secretsmanager:us-
east-1:111122223333:secret:appsync-tutorial-rds-secret"  \
                --region us-east-1 \
                --sql "create DATABASE TESTDB"
```

Wenn dies ohne Fehler läuft, fügen Sie mit dem folgenden Befehl create table eine Tabelle hinzu.

```
aws rds-data execute-statement \
      --resource-arn "arn:aws:rds:us-east-1:111122223333:cluster:http-endpoint-test" \
      --secret-arn "arn:aws:secretsmanager:us-east-1:111122223333:secret:testHttp2-
AmNvc1" \
      --region us-east-1 \
      --sql "create table Pets(id varchar(200), type varchar(200), price float)" \
      --database "TESTDB"
```

Entwerfen Sie Ihre API-Schnittstelle

Nachdem die Aurora Serverless v2 Data API mit einer Tabelle betriebsbereit ist, erstellen Sie ein GraphQL-Schema und hängen Sie Resolver an, um Mutationen und Abonnements durchzuführen. Erstellen Sie eine neue API in der AWS AppSync Konsole, navigieren Sie zur Schemaseite in der Konsole und geben Sie Folgendes ein.

```
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
}
```

Speichern Sie Ihr Schema. Navigieren Sie zur Seite Data Sources (Datenquellen) und erstellen Sie eine neue Datenquelle. Wählen Sie Relationale Datenbank als Datenquellentyp und geben Sie einen benutzerfreundlichen Namen ein. Verwenden Sie den im letzten Schritt erstellten Datenbanknamen und den darin erstellen Cluster ARN (Cluster-ARN). Für die Rolle können Sie entweder eine neue Rolle AWS AppSync erstellen lassen oder eine mit einer Richtlinie erstellen, die der folgenden ähnelt.

JSON

```
{
        "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-1:111122223333:cluster:mydbcluster",
                    "arn:aws:rds:us-east-1:111122223333:cluster:mydbcluster:*"
                ]
            },
            {
                "Effect": "Allow",
                "Action": [
                    "secretsmanager:GetSecretValue"
                ],
                "Resource": [
                    "arn:aws:secretsmanager:us-
east-1:111122223333:secret:mysecret",
                    "arn:aws:secretsmanager:us-
east-1:111122223333:secret:mysecret:*"
                ]
            }
        ]
    }
```

Beachten Sie, dass diese Richtlinie zwei Anweisungen enthält, die den Zugriff der Rolle gewähren. Die erste Ressource ist Ihr Aurora Serverless v2-Cluster und die zweite ist Ihr AWS Secrets Manager ARN. Sie müssen BEIDE ARNs in der AWS AppSync Datenquellenkonfiguration angeben, bevor Sie auf Erstellen klicken.

Connect Sie Ihre API mit Datenbankoperationen

Da wir nun ein gültiges GraphQL-Schema und eine RDS-Datenquelle haben, können Sie Resolver an die GraphQL-Felder Ihres Schemas anhängen. Unsere API bietet folgende Funktionen:

1. erstellen Sie mit dem Feld Mutation.createPet ein Haustier
2. aktualisieren Sie ein Haustier mithilfe des Felds Mutation.UpdatePet
3. lösche ein Haustier mithilfe des Felds Mutation.deletePet
4. Holen Sie sich eine Single mithilfe des Felds Query.getPet
5. listet alle mithilfe des Felds query.listPets auf
6. listet mithilfe der Abfrage Haustiere in einer Preisklasse auf. listPetsByPriceRangeFeld

Mutation.createPet

Wählen Sie im Schema-Editor in der AWS AppSync Konsole auf der rechten Seite Attach Resolver for `createPet(input: CreatePetInput!): Pet` aus. Wählen Sie Ihre RDS-Datenquelle aus. Fügen Sie die folgende Vorlage im Abschnitt request mapping template (Zuweisungsvorlage für Anforderungen) ein:

```
#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)
    }
}
```

Das System führt SQL-Anweisungen sequentiell aus, basierend auf der Reihenfolge im Anweisungsarray. Die Ergebnisse werden wieder in derselben Reihenfolge zurückgegeben. Da es sich um eine Mutation handelt, führen Sie nach dem Einfügen eine SELECT-Anweisung aus, um die festgeschriebenen Werte abzurufen und die GraphQL-Antwortzuordnungsvorlage zu füllen.

Fügen Sie die folgende Vorlage im Abschnitt response mapping template (Zuweisungsvorlage für Antworten) ein:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[1][0])
```

Da die Anweisungen zwei SQL-Abfragen enthalten, müssen wir das zweite Ergebnis in der Matrix, die aus der Datenbank zurückgegeben wird, wie folgt angeben: `$utils.rds.toJsonString($ctx.result))[1][0])`.

Mutation.updatePet

Wählen Sie im Schema-Editor in der AWS AppSync Konsole Attach Resolver for aus. `updatePet(input: UpdatePetInput!): Pet` Wählen Sie Ihre RDS-Datenquelle aus. Fügen Sie im Abschnitt Vorlage für die Anforderungszuweisung die folgende Vorlage hinzu.

```
{
"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)
    }
}
```

Fügen Sie im Abschnitt Vorlage für die Antwortzuordnung die folgende Vorlage hinzu.

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[1][0])
```

Mutation.deletePet

Wählen Sie im Schema-Editor in der AWS AppSync Konsole Attach Resolver for `deletePet(input: DeletePetInput!): Pet` aus. Wählen Sie Ihre RDS-Datenquelle aus. Fügen Sie im Abschnitt Vorlage für die Anforderungszuweisung die folgende Vorlage hinzu.

```
{
"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"
    }
}
```

Fügen Sie im Abschnitt Vorlage für die Antwortzuordnung die folgende Vorlage hinzu.

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0][0])
```

Query.getPet

Nachdem die Mutationen für Ihr Schema erstellt wurden, können Sie die drei Abfragen miteinander verbinden, um zu zeigen, wie Sie einzelne Elemente und Listen abrufen und SQL-Filter anwenden können. Wählen Sie im Schema-Editor in der AWS AppSync Konsole Attach Resolver for `getPet(id: ID!): Pet` aus. Wählen Sie Ihre RDS-Datenquelle aus. Fügen Sie im Abschnitt Vorlage für die Anforderungszuweisung die folgende Vorlage hinzu.

```
{
"version": "2018-05-29",
        "statements": [
            $util.toJson("select * from Pets WHERE id=:ID")
    ],
    "variableMap": {
        ":ID": "$ctx.args.id"
    }
}
```

Fügen Sie die folgende Vorlage im Abschnitt response mapping template (Zuweisungsvorlage für Antworten) ein:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0][0])
```

Query.listPets

Wählen Sie im Schema-Editor in der AWS AppSync Konsole auf der rechten Seite Attach Resolver for `getPet(id: ID!): Pet` aus. Wählen Sie Ihre RDS-Datenquelle aus. Fügen Sie im Abschnitt Vorlage für die Anforderungszuweisung die folgende Vorlage hinzu.

```
{
    "version": "2018-05-29",
    "statements": [
        "select * from Pets"
    ]
}
```

Fügen Sie im Abschnitt Vorlage für die Antwortzuordnung die folgende Vorlage hinzu.

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0])
```

Abfrage. listPetsByPriceRange

Wählen Sie im Schema-Editor in der AWS AppSync Konsole auf der rechten Seite Attach Resolver for `getPet(id: ID!): Pet` aus. Wählen Sie Ihre RDS-Datenquelle aus. Fügen Sie im Abschnitt Vorlage für die Anforderungszuweisung die folgende Vorlage hinzu.

```
{
    "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)
    }
}
```

Fügen Sie die folgende Vorlage im Abschnitt response mapping template (Zuweisungsvorlage für Antworten) ein:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0])
```

Ändern Sie Ihre Daten über die API

Nachdem Sie nun alle Resolver mit SQL-Anweisungen konfiguriert und Ihre GraphQL-API Ihrer Aurora Serverless-Daten-API zugewiesen haben, können Sie damit beginnen, Mutationen und Abfragen durchzuführen. Wählen Sie in der AWS AppSync Konsole den Tab Abfragen und geben Sie Folgendes ein, um ein Haustier zu erstellen:

```
mutation add {
    createPet(input : { type:fish, price:10.0 }){
        id
        type
        price
    }
}
```

Die Antwort enthält die ID, den Typ und den Preis. Beispiel:

```
{
```

```
  "data": {
    "createPet": {
      "id": "c6fedbbe-57ad-4da3-860a-ffe8d039882a",
      "type": "fish",
      "price": "10.0"
    }
  }
}
```

Sie können dieses Element verändern, indem Sie die Mutation updatePet ausführen:

```
mutation update {
    updatePet(input : {
        id: ID_PLACEHOLDER,
        type:bird,
        price:50.0
    }){
        id
        type
        price
    }
}
```

Beachten Sie, dass wir die id verwendet haben, die von der vorherigen createPet-Operation zurückgegeben wurde. Dies ist ein eindeutiger Wert für Ihren Datensatz, da der Resolver `$util.autoId()` verwendet hat. Auf ähnliche Weise können Sie einen Datensatz löschen:

```
mutation delete {
    deletePet(input : {id:ID_PLACEHOLDER}){
        id
        type
        price
    }
}
```

Erstellen Sie mit der ersten Mutation einige Datensätze mit unterschiedlichen Werten für price und führen Sie dann einige Abfragen aus.

Rufen Sie Ihre Daten ab

Verwenden Sie weiterhin auf der Registerkarte Abfragen der Konsole die folgende Anweisung, um alle von Ihnen erstellten Datensätze aufzulisten.

```
query allpets {
    listPets {
        id
        type
        price
    }
}
```

Nutzen Sie das SQL WHERE-Prädikat, das `where price > :MIN and price < :MAX` in unserer Zuordnungsvorlage für Query enthalten war. listPetsByPriceRangemit der folgenden GraphQL-Abfrage:

```
query petsByPriceRange {
    listPetsByPriceRange(min:1, max:11) {
        id
        type
        price
    }
}
```

Es werden nur Datensätze mit einem Preis über 1 $ und unter 10 $ angezeigt. Abschließend können Sie wie folgt Abfragen ausführen, um einzelne Datensätze abzurufen:

```
query onePet {
    getPet(id:ID_PLACEHOLDER){
        id
        type
        price
    }
}
```

Schützen Sie Ihren Datenzugriff

SQL Injection ist eine Sicherheitslücke in Datenbankanwendungen. Sie tritt auf, wenn Angreifer bösartigen SQL-Code über Benutzereingabefelder einfügen. Dies kann unbefugten Zugriff auf Datenbankdaten ermöglichen. Wir empfehlen Ihnen, alle Benutzereingaben vor der Verarbeitung sorgfältig zu überprüfen und zu bereinigen, um sich vor SQL-Injection-Angriffen zu schützen. `variableMap` Wenn Variablenzuordnungen nicht verwendet werden, sind Sie dafür verantwortlich, die Argumente ihrer GraphQL-Operationen zu bereinigen. Hierzu können Sie in der Anforderungszuweisungsvorlage eingabespezifische Validierungsschritte festlegen,

bevor eine SQL-Anweisung für Ihre Daten-API ausgeführt wird. Sehen wir uns an, wie wir die Anforderungszuweisungsvorlage des Beispiels `listPetsByPriceRange` anpassen können. Statt sich ausschließlich auf die Benutzereingabe zu verlassen, können Sie wie folgt verfahren:

```
#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)
    }
}
```

Eine weitere Möglichkeit zum Schutz vor nicht autorisierten Eingaben beim Ausführen der Resolver gegen Ihre Daten-API bieten vorbereitete Anweisungen mit gespeicherter Prozedur und parametrisierten Eingaben. Beispiel: Definieren Sie im Resolver `listPets` die folgende Prozedur, die Select als vorbereitete Anweisung ausführt:

```
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
```

Erstellen Sie dies in Ihrer Aurora Serverless v2-Instance.

```
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"
```

Der resultierende Resolver-Code für listPets wird vereinfacht, da wir jetzt einfach die gespeicherte Prozedur aufrufen. Als Mindestanforderung müssen bei jeder Zeichenfolgeneingabe einfache Anführungszeichen durch Escape-Zeichen geschützt sein.

```
#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("'", "''"))
    }
}
```

Verwenden von Escape-Zeichenketten

Verwenden Sie einfache Anführungszeichen, um den Anfang und das Ende von Zeichenkettenliteralen in einer SQL-Anweisung zu markieren, z. `'some string value'`. Damit Zeichenfolgenwerte mit einem oder mehreren einfachen Anführungszeichen (`'`) innerhalb einer Zeichenfolge verwendet werden können, muss jedes durch zwei einfache Anführungszeichen (`''`) ersetzt werden. Lautet die Eingabezeichenfolge beispielsweise `Nadia's dog`, würden Sie sie für die SQL-Anweisung wie folgt durch Escape-Zeichen schützen:

```
update Pets set type='Nadia''s dog' WHERE id='1'
```

Verwendung von Pipeline-Resolvern in AWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync bietet eine einfache Möglichkeit, ein GraphQL-Feld über Unit-Resolver mit einer einzelnen Datenquelle zu verbinden. Eine einzige Operation auszuführen, reicht jedoch möglicherweise nicht aus. Pipeline-Resolver bieten die Möglichkeit einer seriellen Ausführung von Operationen auf Datenquellen. Erstellen Sie Funktionen in Ihrer API und verbinden Sie diese mit einem Pipeline-Resolver. Jedes Ergebnis einer Funktionsausführung wird an die nächste Funktion weitergeleitet, bis keine weitere Funktion mehr auszuführen ist. Mit Pipeline-Resolvern können Sie jetzt direkt in AWS AppSync komplexere Workflows erstellen. In diesem Tutorial erstellen Sie eine einfache App für die Anzeige von Bildern, in der Benutzer Bilder veröffentlichen und von ihren Freunden veröffentlichte Bilder anzeigen können.

One-Click-Setup

Wenn Sie den GraphQL-Endpunkt automatisch AWS AppSync mit allen konfigurierten Resolvern und den erforderlichen AWS Ressourcen einrichten möchten, können Sie die folgende AWS CloudFormation Vorlage verwenden:

Dieser Stack erstellt die folgenden Ressourcen in Ihrem Konto:

- IAM-Rolle für den Zugriff AWS AppSync auf die Ressourcen in Ihrem Konto
- 2 DynamoDB-Tabellen
- 1 Amazon Cognito-Benutzerpool
- 2 Amazon Cognito-Benutzerpoolgruppen
- 3 Amazon Cognito-Benutzerpoolbenutzer
- 1 API AWS AppSync

Am Ende des AWS CloudFormation Stack-Erstellungsprozesses erhalten Sie eine E-Mail für jeden der drei Amazon Cognito Cognito-Benutzer, die erstellt wurden. Jede E-Mail enthält ein temporäres Passwort, mit dem Sie sich als Amazon Cognito Cognito-Benutzer an der AWS AppSync Konsole anmelden. Speichern Sie die Passwörter, damit Sie während des Tutorials darauf zurückgreifen können.

Manuelle Einrichtung

Wenn Sie es vorziehen, einen step-by-step Vorgang manuell über die AWS AppSync Konsole durchzuführen, folgen Sie dem unten stehenden Einrichtungsprozess.

Richten Sie Ihre AWS AppSync Nicht-Ressourcen ein

Die API kommuniziert mit zwei DynamoDB-Tabellen: einer Bildertabelle, in der Bilder gespeichert werden, und einer Freundes-Tabelle, in der Beziehungen zwischen Benutzern gespeichert werden. Die API wird für die Verwendung des Amazon Cognito-Benutzerpools als Authentifizierungstyp konfiguriert. Der folgende AWS CloudFormation Stack richtet diese Ressourcen im Konto ein.

Am Ende des AWS CloudFormation Stack-Erstellungsprozesses erhalten Sie eine E-Mail für jeden der drei Amazon Cognito Cognito-Benutzer, die erstellt wurden. Jede E-Mail enthält ein temporäres Passwort, mit dem Sie sich als Amazon Cognito-Benutzer bei der AWS AppSync-Konsole anmelden. Speichern Sie die Passwörter, damit Sie während des Tutorials darauf zurückgreifen können.

Erstellen der GraphQL-API

Um die GraphQL-API zu erstellen in AWS AppSync:

1. Öffnen Sie die AWS AppSync Konsole und wählen Sie Build From Scratch und dann Start.
2. Legen Sie den Namen der API auf `AppSyncTutorial-PicturesViewer` fest.
3. Wählen Sie Create (Erstellen) aus.

Die AWS AppSync Konsole erstellt mithilfe des API-Schlüsselauthentifizierungsmodus eine neue GraphQL-API für Sie. Sie können in der Konsole den Rest der GraphQL-API einrichten und im weiteren Verlauf dieses Tutorials abfragen.

Konfigurieren der GraphQL-API

Sie müssen die AWS AppSync API mit dem Amazon Cognito Cognito-Benutzerpool konfigurieren, den Sie gerade erstellt haben.

1. Wählen Sie die Registerkarte Einstellungen.
2. Wählen Sie im Abschnitt Authorization Type (Autorisierungstyp) Amazon Cognito User Pool (Amazon Cognito-Benutzerpool) aus.
3. Wählen Sie unter Benutzerpool-Konfiguration die Option US-WEST-2 für die Region aus.AWS
4. Wählen Sie den UserPool Benutzerpool AppSyncTutorial- aus.
5. Wählen Sie DENY (VERWEIGERN) als Default Action (Standardaktion) aus.
6. Lassen Sie das AppId Client-Regex-Feld leer.
7. Wählen Sie Save (Speichern) aus.

Die API ist jetzt für die Verwendung des Amazon Cognito-Benutzerpools als Autorisierungstyp eingerichtet.

Konfiguration von Datenquellen für die DynamoDB-Tabellen

Nachdem die DynamoDB-Tabellen erstellt wurden, navigieren Sie in der Konsole zu Ihrer AWS AppSync GraphQL-API und wählen Sie die Registerkarte Datenquellen. Jetzt erstellen Sie eine Datenquelle AWS AppSync für jede der DynamoDB-Tabellen, die Sie gerade erstellt haben.

1. Klicken Sie auf die Registerkarte Data source (Datenquelle).
2. Wählen Sie New (Neu) aus, um eine neue Datenquelle zu erstellen.
3. Geben Sie als Namen für die Datenquelle `PicturesDynamoDBTable` ein.
4. Wählen Sie als Typ der Datenquelle Amazon DynamoDB table (Amazon DynamoDB-Tabelle) aus.
5. Wählen Sie US-WEST-2 als Region aus.
6. Wählen Sie aus der Tabellenliste die AppSyncTutorialDynamoDB-Tabelle -Pictures aus.
7. Wählen Sie im Abschnitt Eine bestehende Rolle erstellen oder verwenden die Option Bestehende Rolle aus.
8. Wählen Sie die Rolle aus, die gerade aus der CloudFormation Vorlage erstellt wurde. Wenn Sie die nicht geändert haben ResourceNamePrefix, sollte der Name der Rolle AppSyncTutorial-Dynamo lauten DBRole.
9. Wählen Sie Create (Erstellen) aus.

Wiederholen Sie den Vorgang für die Friends-Tabelle. Der Name der DynamoDB-Tabelle sollte AppSyncTutorial-Friends lauten, wenn Sie den ResourceNamePrefixParameter bei der Erstellung des Stacks nicht geändert haben. CloudFormation

Erstellen des GraphQL-Schemas

Nachdem die Datenquellen nun mit Ihren DynamoDB-Tabellen verbunden sind, erstellen wir ein GraphQL-Schema. Stellen Sie im Schema-Editor in der AWS AppSync Konsole sicher, dass Ihr Schema dem folgenden Schema entspricht:

```
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!
}
```

Klicken Sie auf Save Schema (Schema speichern), um Ihr Schema zu speichern.

Einige Schemafelder wurden mit der Richtlinie @aws_auth kommentiert. Da für die Konfiguration der Standardaktion der API DENY (VERWEIGERN) festgelegt ist, lehnt die API alle Benutzer ab, die

kein Mitglied der in der Richtlinie @aws_auth aufgeführten Gruppen sind. Weitere Informationen zum Schutz Ihrer API finden Sie unter Security (Sicherheit). In diesem Fall haben nur Admin-Benutzer Zugriff auf die Felder Mutation.CreatePicture und Mutation.CreateFriendship, während Benutzer, die entweder Mitglieder der Admins - oder Viewers-Gruppen sind, auf die Query zugreifen können. getPicturesByFeld „Besitzer“. Alle anderen Benutzer haben keinen Zugriff.

Konfigurieren von Resolvern

Nachdem Sie jetzt ein gültiges GraphQL-Schema und zwei Datenquellen besitzen, können Sie Resolver mit den GraphQL-Feldern auf dem Schema verbinden. Die API bietet folgende Funktionen:

- Erstellen eines Bildes über das Feld Mutation.createPicture
- Erstellen einer Freundschaft über das Feld Mutation.createFriendship
- Abrufen eines Bildes über das Feld Query.getPicture

Mutation.createPicture

Wählen Sie im Schema-Editor in der AWS AppSync Konsole auf der rechten Seite Attach Resolver for `createPicture(input: CreatePictureInput!): Picture!` aus. Wählen Sie die DynamoDB-Datenquelle PicturesDynamoDBTableaus. Fügen Sie die folgende Vorlage im Abschnitt request mapping template (Zuweisungsvorlage für Anforderungen) ein:

```
#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)
}
```

Fügen Sie die folgende Vorlage im Abschnitt response mapping template (Zuweisungsvorlage für Antworten) ein:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end
$util.toJson($ctx.result)
```

Die Funktion zum Erstellen eines Bildes ist fertig. Ein Bild wird in der Tabelle Pictures mit einer zufällig generierte UUID als ID des Bildes und unter Verwendung des Cognito-Benutzernamens für den Eigentümer des Bildes gespeichert.

Mutation.createFriendship

Wählen Sie im Schema-Editor in der AWS AppSync Konsole auf der rechten Seite Attach Resolver for aus. `createFriendship(id: ID!, target: ID!): Boolean` Wählen Sie die DynamoDB-Datenquelle FriendsDynamoDBTableaus. Fügen Sie die folgende Vorlage im Abschnitt request mapping template (Zuweisungsvorlage für Anforderungen) ein:

```
#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)
    }
}
```

Wichtig: In der BatchPutItemAnforderungsvorlage sollte der genaue Name der DynamoDB-Tabelle vorhanden sein. Der Standardtabellenname ist AppSyncTutorial -Friends. Wenn Sie den falschen Tabellennamen verwenden, erhalten Sie eine Fehlermeldung, wenn Sie AppSync versuchen, die angegebene Rolle anzunehmen.

Gehen Sie in diesem Tutorial der Einfachheit halber so vor, als ob die Freundschaftsanfrage genehmigt worden wäre, und speichern Sie den Beziehungseintrag direkt in der AppSyncTutorialFriendsTabelle.

Da die Beziehung bidirektional ist, werden für jede Freundschaft faktisch zwei Elemente gespeichert. Weitere Informationen zu den bewährten Methoden von Amazon DynamoDB zur Darstellung von many-to-many Beziehungen finden Sie unter Bewährte Methoden für DynamoDB.

Fügen Sie die folgende Vorlage im Abschnitt response mapping template (Zuweisungsvorlage für Antworten) ein:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end
true
```

Hinweis: Stellen Sie sicher, dass Ihre Anforderungsvorlage den richtigen Tabellennamen enthält. Der Standardname ist AppSyncTutorial-Friends, aber Ihr Tabellenname könnte abweichen, wenn Sie den Parameter ändern. CloudFormation ResourceNamePrefix

Abfrage. getPicturesByBesitzer

Nachdem Sie jetzt Freundschaften und Bilder besitzen, müssen Sie Benutzern die Möglichkeit bieten, die Bilder ihrer Freunde anzuzeigen. Um diese Anforderung zu erfüllen, müssen Sie zuerst prüfen, ob der Anforderer mit dem Eigentümer befreundet ist und anschließend die Bilder abrufen.

Da für diese Funktionalität zwei Datenquellenoperationen erforderlich sind, müssen Sie zwei Funktionen erstellen. Die erste Funktion, isFriend, prüft, ob der Anforderer und der Eigentümer Freunde sind. Die zweite Funktion, getPicturesByBesitzer, ruft die angeforderten Bilder mit einer Besitzer-ID ab. Schauen wir uns unten den Ausführungsablauf für den vorgeschlagenen Resolver in der Abfrage an. getPicturesByFeld „Besitzer“:

1. Vorher-Zuweisungsvorlage: Bereitet den Kontext und die Eingabeargumente für das Feld vor.
2. isFriend-Funktion: Prüft, ob der Anforderer der Eigentümer des Bildes ist. Wenn nicht, überprüft es, ob die Benutzer des Anforderers und des Besitzers Freunde sind, indem es eine GetItem DynamoDB-Operation in der Friends-Tabelle ausführt.
3. getPicturesByOwner-Funktion: Ruft mithilfe einer DynamoDB-Abfrageoperation für den globalen Sekundärindex des Eigentümerindexes Bilder aus der Tabelle Pictures ab.
4. Nachher-Zuweisungsvorlage: Weist die resultierenden Bilder so zu, dass die DynamoDB-Attribute den erwarteten Feldern des GraphQL-Typs korrekt zugewiesen sind.

Erstellen wir zuerst die Funktionen.

isFriend-Funktion

1. Wählen Sie die Registerkarte Functions (Funktionen) aus.
2. Wählen Sie Create Function (Funktion erstellen) aus, um eine Funktion zu erstellen.
3. Geben Sie als Namen für die Datenquelle `FriendsDynamoDBTable` ein.
4. Geben Sie für den Namen der Funktion isFriend ein.
5. Fügen Sie die folgende Vorlage in den Textbereich der Zuweisungsvorlage für Anforderungen ein:

```
#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. Fügen Sie die folgende Vorlage in den Textbereich der Zuweisungsvorlage für Antworten ein:

```
#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. Wählen Sie Create Function.

Ergebnis: Sie haben die Funktion isFriend erstellt.

getPicturesByFunktion „Besitzer“

1. Wählen Sie die Registerkarte Functions (Funktionen) aus.
2. Wählen Sie Create Function (Funktion erstellen) aus, um eine Funktion zu erstellen.
3. Geben Sie als Namen für die Datenquelle `PicturesDynamoDBTable` ein.
4. Geben Sie für den Namen der Funktion `getPicturesByOwner` ein.
5. Fügen Sie die folgende Vorlage in den Textbereich der Zuweisungsvorlage für Anforderungen ein:

```
{
    "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. Fügen Sie die folgende Vorlage in den Textbereich der Zuweisungsvorlage für Antworten ein:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end

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

7. Wählen Sie Create Function.

Ergebnis: Sie haben die Funktion getPicturesByBesitzer erstellt. Nachdem die Funktionen erstellt wurden, fügen Sie der Abfrage einen Pipeline-Resolver hinzu. getPicturesByFeld „Besitzer“.

Wählen Sie im Schema-Editor in der AWS AppSync Konsole auf der rechten Seite Attach Resolver for `Query.getPicturesByOwner(id: ID!): [Picture]` aus. Klicken Sie auf der folgenden Seite auf den Link Convert to pipeline resolver (In Pipeline-Resolver konvertieren), der unter der Dropdown-Liste der Datenquellen angezeigt wird. Verwenden Sie Folgendes für die Zuweisungsvorlage für Antworten:

```
#set($result = { "owner": $ctx.args.id, "callerId": $ctx.identity.username })
$util.toJson($result)
```

Verwenden Sie Folgendes für die Nachher-Zuweisungsvorlage:

```
#foreach($picture in $ctx.result.items)
    ## prepend "src://" to picture.src property
    #set($picture['src'] = "src://${picture['src']}")
#end
$util.toJson($ctx.result.items)
```

Wählen Sie Create Resolver (Resolver erstellen) aus. Sie haben erfolgreich Ihren ersten Pipeline-Resolver angefügt. Fügen Sie auf derselben Seite die beiden zuvor erstellten Funktionen hinzu. Wählen Sie im Abschnitt für Funktionen Add A Funktion (Funktion hinzufügen) aus. Wählen Sie dann den Namen der ersten Funktion, isFriend, aus oder geben Sie ihn ein. Fügen Sie die zweite Funktion hinzu, indem Sie den gleichen Vorgang für die Funktion getPicturesByOwner ausführen. Stellen Sie sicher, dass die Funktion IsFriend zuerst in der Liste erscheint, gefolgt von der Funktion getPicturesByBesitzer. Mit den Pfeilen nach oben und unten können Sie die Ausführungsreihenfolge der Funktionen in der Pipeline verändern.

Nachdem der Pipeline-Resolver erstellt wurde und Sie die Funktionen angehängt haben, wollen wir die neu erstellte GraphQL-API testen.

Testen der GraphQL-API

Zuerst müssen Sie Bilder und Freundschaften auffüllen, indem Sie mit dem erstellten Admin-Benutzer ein paar Mutationen ausführen. Wählen Sie auf der linken Seite der AWS AppSync Konsole die Registerkarte Abfragen aus.

createPicture-Mutation

1. Wählen Sie in der AWS AppSync Konsole die Registerkarte Abfragen aus.
2. Wählen Sie Login With User Pools (Anmeldung mit Benutzerpools) aus.
3. Geben Sie im Modal die Cognito Sample Client ID ein, die vom CloudFormation Stack erstellt wurde (z. B. 37solo6mmhh7k4v63cqdfgdg5d).
4. Geben Sie den Benutzernamen ein CloudFormation , den Sie als Parameter an den Stack übergeben haben. Der Standardwert lautet nadia.
5. Verwenden Sie das temporäre Passwort, das an die von Ihnen angegebene E-Mail gesendet wurde, als Parameter für den CloudFormation Stack (z. B. UserPoolUserEmail).
6. Wählen Sie Login (Anmelden) aus. Sie sollten jetzt sehen, dass die Schaltfläche in Logout nadia umbenannt wurde, oder in einen anderen Benutzernamen, den Sie bei der Erstellung des CloudFormation Stacks gewählt haben (d. h. UserPoolUsername).

Lassen Sie uns nun einige createPicture-Mutationen senden, um die Tabelle mit Bildern zu füllen. Führen Sie die folgende GraphQL-Abfrage in der Konsole aus:

```
mutation {
  createPicture(input:{
    owner: "nadia"
    src: "nadia.jpg"
  }) {
    id
    owner
    src
  }
}
```

Die Antwort sollte wie folgt aussehen:

```
{
  "data": {
    "createPicture": {
      "id": "c6fedbbe-57ad-4da3-860a-ffe8d039882a",
      "owner": "nadia",
      "src": "nadia.jpg"
    }
  }
```

```
}
```

Lassen Sie uns noch einige weitere Bilder hinzufügen:

```
mutation {
  createPicture(input:{
    owner: "shaggy"
    src: "shaggy.jpg"
  }) {
    id
    owner
    src
  }
}
```

```
mutation {
  createPicture(input:{
    owner: "rex"
    src: "rex.jpg"
  }) {
    id
    owner
    src
  }
}
```

Sie haben mit nadia als Admin-Benutzer drei Bilder hinzugefügt.

createFriendship-Mutation

Lassen Sie uns einen Freundschaftseintrag hinzufügen. Führen Sie die folgenden Mutationen in der Konsole aus.

Hinweis: Sie müssen weiterhin als Admin-Benutzer angemeldet sein (der standardmäßige Admin-Benutzer ist nadia).

```
mutation {
  createFriendship(id: "nadia", target: "shaggy")
}
```

Die Antwort sollte wie folgt aussehen:

```
{
  "data": {
    "createFriendship": true
  }
}
```

nadia und shaggy sind Freunde. rex ist mit niemandem befreundet.

getPicturesByBesitzer-Anfrage

Melden Sie sich für diesen Schritt mithilfe von Cognito-Benutzerpools als Benutzer nadia an, indem Sie die zu Beginn dieses Tutorials eingerichteten Anmeldeinformationen verwenden. Rufen Sie als nadia die Bilder im Besitz von shaggy ab.

```
query {
    getPicturesByOwner(id: "shaggy") {
        id
        owner
        src
    }
}
```

Da nadia und shaggy Freunde sind, gibt die Abfrage das entsprechende Bild zurück.

```
{
  "data": {
    "getPicturesByOwner": [
      {
        "id": "05a16fba-cc29-41ee-a8d5-4e791f4f1079",
        "owner": "shaggy",
        "src": "src://shaggy.jpg"
      }
    ]
  }
}
```

nadia kann auch ihre eigenen Bilder erfolgreich abrufen. Der Pipeline-Resolver wurde so optimiert, dass in diesem Fall die GetItem IsFriend-Operation nicht ausgeführt wird. Versuchen Sie, die folgende Abfrage auszuführen:

```
query {
```

```
    getPicturesByOwner(id: "nadia") {
        id
        owner
        src
    }
}
```

Wenn Sie die Protokollierung auf Ihrer API aktiviert haben (im Bereich Settings (Einstellungen)), legen Sie für die Debugging-Stufe ALL fest und führen Sie die gleiche Abfrage erneut, damit Protokolle für die Feldausführung zurückgegeben werden. In den Protokollen können Sie prüfen, ob die Rückgabe der Funktion isFriend zu einem frühen Zeitpunkt während der Phase der Zuweisungsvorlage für Anforderungen erfolgt ist:

```
{
  "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
}
```

Der earlyReturnedValueSchlüssel stellt die Daten dar, die von der #return -Direktive zurückgegeben wurden.

Schließlich wird Rex, obwohl er Mitglied der Viewers Cognito UserPool Group ist und weil Rex mit niemandem befreundet ist, auf keines der Bilder zugreifen können, die Shaggy oder Nadia gehören. Wenn Sie sich als rex in der Konsole anmelden und die folgende Abfrage ausführen:

```
query {
    getPicturesByOwner(id: "nadia") {
        id
        owner
        src
    }
}
```

Sie erhalten folgenden Fehler wegen fehlender Autorisierung:

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

Sie haben erfolgreich eine komplexe Autorisierung mit Pipeline-Resolvern implementiert.

Verwenden von Delta Sync-Vorgängen für versionierte Datenquellen in AWS AppSync

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Client-Anwendungen AWS AppSync speichern Daten, indem sie GraphQL-Antworten lokal auf der Festplatte in einer Mobil-/Webanwendung zwischenspeichern. Versionierte Datenquellen und `Sync`-Vorgänge bieten Kunden die Möglichkeit, den Synchronisierungsprozess mit einem einzigen Resolver durchzuführen. Auf diese Weise können Clients ihren lokalen Cache mit Ergebnissen aus einer Basis-Abfrage, die möglicherweise viele Datensätze enthalten, hydratisieren und anschließend nur die Daten empfangen, die seit der letzten Abfrage geändert wurden (die Delta-Updates). Indem Clients berechtigt werden, die Basis-Hydration des Caches mit einer Abfrage und inkrementelle Updates mit einer anderen Abfrage separat durchzuführen, können die Rechenvorgänge von der Client-Anwendung in das Backend verlagert werden. Dies ist wesentlich effizienter für Client-Anwendungen, die häufig zwischen Online- und Offline-Status wechseln.

Um Delta Sync zu implementieren, verwendet die `Sync`-Abfrage den `Sync`-Vorgang für eine versionierte Datenquelle. Wenn eine AWS AppSync Mutation ein Element in einer versionierten Datenquelle ändert, wird ein Datensatz dieser Änderung ebenfalls in der Delta-Tabelle gespeichert. Sie können wählen, ob Sie verschiedene Delta-Tabellen (z. B. eine pro Typ, eine pro Domainbereich) für andere versionierte Datenquellen oder eine einzelne Delta-Tabelle für Ihre API verwenden möchten. AWS AppSync empfiehlt, keine einzige Delta-Tabelle für mehrere APIs zu verwenden, um die Kollision von Primärschlüsseln zu vermeiden.

Darüber hinaus können Delta Sync-Clients auch ein Abonnement als Argument erhalten. Der Client koordiniert dann das Wiederherstellen von Abonnementverbindungen und Schreibvorgänge während des Übergangs vom Offline- zum Online-Zustand. Delta Sync führt hierzu eine automatische Wiederaufnahme der Abonnements durch, einschließlich exponentiellem Backoff und Neuversuch mit Jitter in verschiedenen Netzwerkfehlerszenarien und Speichern der Ereignisse in einer Warteschlange. Die entsprechende Delta- oder Basisabfrage wird dann ausgeführt, bevor Ereignisse aus der Warteschlange zusammengeführt und letztlich Abonnements wie gewohnt verarbeitet werden.

Die Dokumentation zu den Client-Konfigurationsoptionen, einschließlich Amplify DataStore, ist auf der Amplify Framework-Website verfügbar. In dieser Dokumentation wird beschrieben, wie Sie versionierte DynamoDB-Datenquellen und Sync-Vorgänge so einrichten, dass sie mit dem Delta Sync-Client für optimalen Datenzugriff arbeiten.

One-Click-Setup

Verwenden Sie diese Vorlage, um den GraphQL-Endpunkt AWS AppSync mit allen konfigurierten Resolvern und den erforderlichen AWS Ressourcen automatisch einzurichten: AWS CloudFormation

Dieser Stack erstellt die folgenden Ressourcen in Ihrem Konto:

- 2 DynamoDB-Tabellen (Base und Delta)
- 1 AWS AppSync API mit API-Schlüssel
- 1 IAM-Rolle mit Richtlinie für DynamoDB-Tabellen

Es werden zwei Tabellen verwendet, um Ihre Sync-Abfragen in eine zweite Tabelle zu partitionieren, die als Journal der Ereignisse fungiert, welche verpasst wurden, während die Clients offline waren. Um die Abfragen in der Delta-Tabelle effizient zu halten, TTLs werden Amazon DynamoDB verwendet, um die Ereignisse bei Bedarf automatisch zu bereinigen. Die TTL-Zeit ist für Ihre Anforderungen an die Datenquelle konfigurierbar (Sie können dies als 1 Stunde, 1 Tag usw. verwenden).

Schema

Um Delta Sync zu demonstrieren, erstellt die Beispielanwendung ein Posts-Schema, das von einer Base - und Delta-Tabelle in DynamoDB unterstützt wird. AWS AppSync schreibt die Mutationen automatisch in beide Tabellen. Die Sync-Abfrage ruft Datensätze entsprechend aus der Basis- oder der Delta-Tabelle ab. Außerdem ist ein einziges Abonnement definiert, das zeigt, wie die Clients dies in ihrer Logik für erneute Verbindungen nutzen können.

```
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
}
```

Das GraphQL-Schema ist Standard. Einige Aspekte sind jedoch der Erwähnung wert, bevor Sie fortfahren. Zuerst werden alle Mutationen automatisch in die Basis-Tabelle und dann in die Delta-Tabelle geschrieben. Die Basis-Tabelle ist die zentrale Datenquelle (Single Source of Truth) für den Status, während die Delta-Tabelle Ihr Journal ist. Wenn Sie die `lastSync: AWSTimestamp` nicht übergeben, wird die `syncPosts`-Abfrage über die Basis-Tabelle ausgeführt. Sie hydratisiert den Cache und wird regelmäßig als globaler Catchup-Prozess für Sonderfälle ausgeführt, in denen Clients länger als die in der Delta-Tabelle konfigurierte TTL-Zeit offline sind. Wenn Sie die `lastSync: AWSTimestamp`-Abfrage jedoch übergeben, wird die `syncPosts`-Abfrage über die Delta-Tabelle ausgeführt und von Clients verwendet, um Ereignisse abzurufen, die seit ihrem letzte Offline-Zeitpunkt geändert wurden. Amplify-Clients übergeben den `lastSync: AWSTimestamp`-Wert automatisch und halten ihn entsprechend dauerhaft auf der Festplatte.

Das Feld _deleted auf Post wird für DELETE-Vorgänge verwendet. Wenn Clients offline sind und Datensätze aus der Basis-Tabelle entfernt werden, benachrichtigt dieses Attribut Clients, die eine

Synchronisierung durchführen, dass sie Elemente aus ihrem lokalen Cache entfernen müssen. In Fällen, in denen Clients für längere Zeiträume offline sind und das Element entfernt wurde, bevor der Client diesen Wert mit einer Delta Sync-Abfrage abrufen kann, wird das globale Catchup-Ereignis in der Basis-Abfrage (im Client konfigurierbar) ausgeführt und entfernt das Element aus dem Cache. Dieses Feld ist als optional gekennzeichnet, weil es nur dann einen Wert zurückgibt, wenn es eine Sync-Abfrage ausführt, in der gelöschte Elemente vorhanden sind.

Mutationen

AWS AppSync Führt für alle Mutationen eine Create/Update/Delete Standardoperation in der Basistabelle aus und zeichnet die Änderung auch automatisch in der Delta-Tabelle auf. Sie können den Aufbewahrungszeitraum für Datensätze verlängern oder verkürzen, indem Sie den `DeltaSyncTableTTL`-Wert in der Datenquelle anpassen. Für Unternehmen, deren Daten häufigen Änderungen unterliegen, ist es sinnvoll, einen kurzen Zeitraum anzugeben. Wenn Ihre Clients für längere Zeiträume offline sind, kann es ratsam sein, einen längeren Zeitraum anzugeben.

Sync-Abfragen

Die Basisabfrage ist ein DynamoDB-Synchronisierungsvorgang ohne Angabe eines `lastSync` Werts. Dies funktioniert in vielen Organisationen, weil die Basis-Abfrage nur beim Start und danach in regelmäßigen Abständen ausgeführt wird.

Die Deltaabfrage ist ein DynamoDB-Synchronisierungsvorgang mit einem angegebenen `lastSync` Wert. Die Delta-Abfrage wird jedes Mal ausgeführt, wenn der Client wieder in den Online-Zustand wechselt (sofern nicht die periodische Basis-Abfrage die Ausführung ausgelöst hat). Clients verfolgen automatisch das letzte Mal nach, als sie eine Abfrage zum Synchronisieren von Daten erfolgreich ausgeführt haben.

Wenn eine Delta-Abfrage ausgeführt wird, verwendet der Resolver der Abfrage `ds_pk` und `ds_sk`, um nur die Datensätze abzufragen, die seit der letzten Synchronisierung durch den Client geändert wurden. Der Client speichert die entsprechende GraphQL-Antwort.

Weitere Informationen zum Ausführen von Sync-Abfragen finden Sie in der Dokumentation zu Synchronisierungsoperationen.

Beispiel

Beginnen wir zunächst mit dem Aufruf einer `createPost`-Mutation, um ein Element zu erstellen:

```
mutation create {
  createPost(input: {author: "Nadia", title: "My First Post", content: "Hello World"})
 {
    id
    author
    title
    content
    _version
    _lastChangedAt
    _deleted
  }
}
```

Der Rückgabewert dieser Mutation sieht wie folgt aus:

```
{
  "data": {
    "createPost": {
      "id": "81d36bbb-1579-4efe-92b8-2e3f679f628b",
      "author": "Nadia",
      "title": "My First Post",
      "content": "Hello World",
      "_version": 1,
      "_lastChangedAt": 1574469356331,
      "_deleted": null
    }
  }
}
```

Wenn Sie den Inhalt der Basis-Tabelle untersuchen, wird ein Datensatz angezeigt, der wie folgt aussieht:

```
{
  "_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"
  }
}
```

Wenn Sie den Inhalt der Delta-Tabelle untersuchen, wird ein Datensatz angezeigt, der wie folgt aussieht:

```
{
  "_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"
  }
}
```

Jetzt können wir eine Basis-Abfrage simulieren, die von einem Client ausgeführt wird, um seinen lokalen Datenspeicher mit einer `syncPosts`-Abfrage zu hydratisieren, die wie folgt aussieht:

```
query baseQuery {
  syncPosts(limit: 100, lastSync: null, nextToken: null) {
    items {
      id
      author
      title
      content
      _version
      _lastChangedAt
    }
    startedAt
    nextToken
  }
}
```

Der Rückgabewert dieser Basis-Abfrage sieht wie folgt aus:

```
{
  "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
    }
  }
}
```

Wir speichern den `startedAt`-Wert später, um eine Delta-Abfrage zu simulieren, aber zuerst müssen wir eine Änderung an unserer Tabelle vornehmen. Verwenden wir die `updatePost`-Mutation zum Ändern unseres vorhandenen Beitrags:

```
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
  }
}
```

Der Rückgabewert dieser Mutation sieht wie folgt aus:

```
{
  "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
    }
  }
}
```

Wenn Sie den Inhalt der Basis-Tabelle jetzt untersuchen, sollten Sie das aktualisierte Element sehen:

```
{
  "_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"
  }
}
```

Wenn Sie den Inhalt der Delta-Tabelle jetzt untersuchen, sollten Sie zwei Datensätze sehen:

1. Ein Datensatz zum Zeitpunkt der Erstellung des Elements
2. Ein Datensatz zum Zeitpunkt der Aktualisierung des Elements.

Das neue Element sieht folgendermaßen aus:

```
{
  "_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"
  }
}
```

Jetzt können wir eine Delta-Abfrage simulieren, um Änderungen abzurufen, die aufgetreten sind, als ein Client offline war. Wir verwenden den `startedAt`-Wert, der von unserer Basis-Abfrage zurückgegeben wird, um folgende Anfrage zu stellen:

```
query delta {
  syncPosts(limit: 100, lastSync: 1574469602238, nextToken: null) {
    items {
      id
      author
      title
      content
      _version
    }
    startedAt
    nextToken
  }
}
```

Der Rückgabewert dieser Delta-Abfrage sieht wie folgt aus:

```
{
  "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
    }
  }
}
```

Konfiguration und Einstellungen in AWS AppSync

AWS AppSync ermöglicht Ihnen:

- Daten zwischenspeichern, die häufig angefordert werden, sich aber wahrscheinlich nicht von Anfrage zu Anfrage ändern. Dies kann die Belastung Ihrer Resolver reduzieren. Weitere Informationen finden Sie unter the section called “Konfiguration von serverseitigem Caching und API-Nutzlastkomprimierung”.
- Versionieren Sie GraphQL-Objekte, um Konflikte zwischen mehreren Clients zu behandeln und zu vermeiden. Weitere Informationen finden Sie unter the section called “Versionierung, Konflikterkennung und Synchronisierungsvorgänge für DynamoDB”.
- Verwenden Sie benutzerdefinierte Domainnamen, um eine einzelne, einprägsame Domain zu konfigurieren, die sowohl für Ihr GraphQL als auch für Echtzeit APIs funktioniert. Weitere Informationen finden Sie unter Konfiguration benutzerdefinierter Domainnamen.
- Erlauben Sie den Zugriff auf Ihr GraphQL APIs über eine VPC. Weitere Informationen finden Sie unter Private verwenden AWS AppSync. APIs
- Teilen Sie Ihr GraphQL APIs durch eine Integration mit AWS Resource Access Manager. Weitere Informationen finden Sie unter Teilen Ihres AWS AppSync APIs.
- Aktivieren Sie die Introspektion und legen Sie die Abfragetiefe und die Auflösungsgrenzen pro Abfrage fest. Weitere Informationen finden Sie unter Konfigurationsgrenzen.
- Verwenden Sie Umgebungsvariablen, um das Verhalten Ihrer AWS AppSync Resolver und Funktionen anzupassen, ohne Ihren Code zu aktualisieren. Weitere Informationen finden Sie unter Verwenden von Umgebungsvariablen in. AWS AppSync

AWS AppSync Enthält außerdem die folgenden AWS Standardtools für Protokollierung, Überwachung und Ablaufverfolgung:

- Einloggen AWS CloudTrail
- Überwachung mit Amazon CloudWatch
- Rückverfolgung mit AWS X-Ray

Konfiguration von serverseitigem Caching und API-Nutzlastkomprimierung in AWS AppSync

AWS AppSync Die serverseitigen Funktionen zum Zwischenspeichern von Daten stellen Daten in einem In-Memory-Cache mit hoher Geschwindigkeit zur Verfügung, wodurch die Leistung verbessert und die Latenz verringert wird. Dies reduziert die Notwendigkeit, direkt auf Datenquellen zuzugreifen. Caching ist sowohl für Unit- als auch für Pipeline-Resolver verfügbar.

AWS AppSync ermöglicht es Ihnen auch, API-Antworten zu komprimieren, sodass Nutzdateninhalte schneller geladen und heruntergeladen werden. Dies reduziert potenziell die Belastung Ihrer Anwendungen und senkt möglicherweise auch Ihre Datenübertragungsgebühren. Das Komprimierungsverhalten ist konfigurierbar und kann nach eigenem Ermessen festgelegt werden.

In diesem Abschnitt finden Sie Hilfe bei der Definition des gewünschten Verhaltens für serverseitiges Caching und Komprimierung in Ihrer AWS AppSync API.

Instance-Typen

AWS AppSync hostet Amazon-Instances ElastiCache (Redis OSS) in demselben AWS Konto und derselben AWS Region wie Ihre AWS AppSync API.

Die folgenden Instance-Typen ElastiCache (Redis OSS) sind verfügbar:

small

1 vCPU, 1,5 GiB RAM, geringe bis mäßige Netzwerkleistung

Medium

2 vCPU, 3 GiB RAM, geringe bis mäßige Netzwerkleistung

large

2 vCPU, 12,3 GiB RAM, bis zu 10 Gigabit Netzwerkleistung

xlarge

4 vCPU, 25,05 GiB RAM, bis zu 10 Gigabit Netzwerkleistung

2xlarge

8 vCPU, 50,47 GiB RAM, bis zu 10 Gigabit Netzwerkleistung

4xlarge

16 vCPU, 101,38 GiB RAM, bis zu 10 Gigabit Netzwerkleistung

8xlarge

32 vCPU, 203,26 GiB RAM, 10-Gigabit-Netzwerkleistung (nicht in allen Regionen verfügbar)

12xlarge

48 vCPU, 317,77 GiB RAM, 10 Gigabit-Netzwerkleistung

> **Note**
>
> In der Vergangenheit haben Sie einen bestimmten Instance-Typ (z. B.) angegeben. `t2.medium` Seit Juli 2020 sind diese älteren Instanztypen weiterhin verfügbar, ihre Verwendung ist jedoch veraltet und es wird davon abgeraten. Wir empfehlen, die hier beschriebenen generischen Instanztypen zu verwenden.

Caching-Verhalten

Im Folgenden sind die Verhaltensweisen im Zusammenhang mit dem Caching aufgeführt:

Keine

Keine serverseitige Zwischenspeicherung.

Vollständige Zwischenspeicherung von Anforderungen

Das vollständige Zwischenspeichern von Anfragen ist ein Mechanismus, bei dem die Ausführungsergebnisse des Resolvers einzeln zwischengespeichert werden. Bei dieser Einstellung wird die Ausführung aller Resolver, die während einer Anfrage aufgerufen wurden, AWS AppSync zwischengespeichert, wobei jeder Resolver separat zwischengespeichert wird. Die Daten für jeden Resolver werden aus seiner Datenquelle abgerufen und füllen den Cache, bis die Gültigkeitsdauer (TTL) abgelaufen ist. Bei nachfolgenden API-Anfragen werden Ergebnisse für jeden spezifischen Resolver aus dem Cache zurückgegeben. Das bedeutet, dass Datenquellen nicht direkt kontaktiert werden, es sei denn, die TTL ist abgelaufen. AWS AppSync verwendet den Inhalt der `context.arguments` und `context.identity` -Maps als Caching-Schlüssel für jeden Resolver.

Zwischenspeicherung pro Resolver

Bei dieser Einstellung muss jeder Resolver explizit aktiviert werden, damit er Antworten zwischenspeichern kann. Sie können eine TTL und Caching-Schlüssel auf dem Resolver angeben. Bei den Caching-Schlüsseln, die Sie angeben können, handelt es sich um die Zuordnungen `context.arguments` der obersten Ebene und um `context.source` and/or Zeichenkettenfelder aus diesen `context.identity` Zuordnungen. Der TTL-Wert ist obligatorisch, die Zwischenspeicherungsschlüssel sind jedoch optional. Wenn Sie keine Caching-Schlüssel angeben, sind die Standardwerte der Inhalt der Zuordnungen`context.arguments`, und`context.source`. `context.identity`

Sie könnten beispielsweise die folgenden Kombinationen verwenden:

- context.arguments und context.source
- context.arguments und context.identity.sub
- context.arguments.id oder context.arguments. InputType.id
- context.source.id und context.identity.sub
- context.identity.claims.username

Wenn Sie nur eine TTL und keine Caching-Schlüssel angeben, verhält sich der Resolver genauso wie beim vollständigen Zwischenspeichern von Anfragen.

Caching auf Betriebsebene

Das Caching auf Betriebsebene speichert die gesamten Antworten auf GraphQL-Abfrageoperationen als Ganzes. Wenn diese Option aktiviert ist, werden erfolgreiche Abfrageantworten zwischengespeichert, bis ihre TTL abläuft. Die maximale Größe der zwischenspeicherbaren Antwort beträgt 15 MB. Bei nachfolgenden Abfrageanfragen mit demselben Cache-Schlüssel werden Antworten direkt aus dem Cache bereitgestellt, ohne dass Resolver ausgeführt werden, solange die TTL noch nicht abgelaufen ist.

Der Cache-Schlüssel für das Caching auf Operationsebene wird mithilfe einer Kombination der folgenden Elemente generiert:

- Bestimmte Attribute aus der JSON-Nutzlast der Anfrage:
  - Die `query` Zeichenfolge
  - Die `operationName` Schnur
  - Die `variables` Karte
- Die `context.identity` Karte (außer `context.identity.sourceIp` für IAM- und Amazon Cognito Cognito-Anfragen)

- Die `context.request.headers` Map (mit Ausnahme bestimmter reservierter Header, die im nächsten Abschnitt aufgeführt sind)

Der von der Anfrage verwendete Autorisierungstyp wirkt sich auch auf den Cache-Schlüssel aus. Bei IAM-autorisierten Anfragen enthält der Cache-Schlüssel zusätzlich die Liste der erlaubten und verweigerten Ressourcen. Bei Lambda-autorisierten Anfragen enthält der Cache-Schlüssel zusätzlich die Liste der abgelehnten Felder.

Der Cache-Schlüssel berücksichtigt alle Anforderungsheader, die sich in befinden`context.request.headers`, mit Ausnahme der folgenden reservierten Header, die normalerweise nur für bestimmte Anfragen gelten:

- authorization
- cloudfront-forwarded-proto
- cloudfront-is-desktop-viewer
- cloudfront-is-mobile-viewer
- cloudfront-is-smarttv-viewer
- cloudfront-is-tablet-viewer
- cloudfront-viewer-asn
- cloudfront-viewer-country
- content-length
- host
- priority
- sec-ch-ua
- sec-ch-ua-mobile
- sec-ch-ua-platform
- via
- x-amz-cf-id
- x-amz-date
- x-amz-security-token
- x-amzn-appsync-is-vpce-request
- x-amzn-remote-ip
- x-amzn-requestid
- x-amzn-trace-id

- x-forwarded-for

Gültigkeitsdauer im Cache

Diese Einstellung legt fest, wie lange zwischengespeicherte Einträge im Speicher gespeichert werden sollen. Die maximale TTL beträgt 3.600 Sekunden (1 Stunde). Danach werden Einträge automatisch gelöscht.

Cache-Verschlüsselung

Wenn Sie die serverseitige Datencaching-Funktion verwenden AWS AppSync, ist die Verschlüsselung im Ruhezustand und bei der Übertragung für neue Caches immer aktiviert und kann nicht deaktiviert werden.

Um die Verschlüsselung für einen vorhandenen API-Cache zu aktivieren, löschen Sie den Cache und erstellen Sie ihn anschließend neu.

Um Cache-Einträge ungültig zu machen, können Sie entweder mit der AWS AppSync Konsole oder mit der () einen Flush-Cache-API-Aufruf ausführen. AWS Command Line Interface AWS CLI

Weitere Informationen zum ApiCacheDatentyp finden Sie in der AWS AppSync API-Referenz.

Räumung des Caches

Wenn Sie AWS AppSync das serverseitige Caching einrichten, können Sie eine maximale TTL konfigurieren. Dieser Wert definiert die Zeitspanne, für die zwischengespeicherte Einträge im Speicher gespeichert werden. In Situationen, in denen Sie bestimmte Einträge aus Ihrem Cache entfernen müssen, können Sie AWS AppSync das `evictFromApiCache` Erweiterungsprogramm in der Anfrage oder Antwort Ihres Resolvers verwenden. (Zum Beispiel, wenn sich Ihre Daten in Ihren Datenquellen geändert haben und Ihr Cache-Eintrag jetzt veraltet ist.) Um ein Element aus dem Cache zu entfernen, müssen Sie seinen Schlüssel kennen. Wenn Sie Elemente dynamisch entfernen müssen, empfehlen wir daher, das Zwischenspeichern pro Resolver zu verwenden und explizit einen Schlüssel zu definieren, mit dem Einträge zu Ihrem Cache hinzugefügt werden.

Einen Cache-Eintrag löschen

Verwenden Sie das Erweiterungsprogramm, um ein Element aus dem Cache zu entfernen. `evictFromApiCache` Geben Sie den Typ- und Feldnamen an und stellen Sie dann ein Objekt mit Schlüsselwertelementen bereit, um den Schlüssel für den Eintrag zu erstellen, den Sie entfernen

möchten. Im Objekt steht jeder Schlüssel für einen gültigen Eintrag aus dem `context` Objekt, das in der Liste des zwischengespeicherten Resolvers verwendet wird. `cachingKey` Jeder Wert ist der tatsächliche Wert, der verwendet wurde, um den Wert des Schlüssels zu ermitteln. Sie müssen die Elemente im Objekt in derselben Reihenfolge platzieren wie die Caching-Schlüssel in der Liste des zwischengespeicherten Resolvers. `cachingKey`

Sehen Sie sich zum Beispiel das folgende Schema an:

```
type Note {
  id: ID!
  title: String
  content: String!
}

type Query {
  getNote(id: ID!): Note
}

type Mutation {
  updateNote(id: ID!, content: String!): Note
}
```

In diesem Beispiel können Sie das Caching pro Resolver aktivieren und es dann für die Abfrage aktivieren. `getNote` Anschließend können Sie den Caching-Schlüssel so konfigurieren, dass er aus besteht. `[context.arguments.id]`

Wenn Sie versuchen`Note`, einen abzurufen, um den Cache-Schlüssel zu erstellen, AWS AppSync führt er mithilfe des `id` Arguments der Abfrage eine Suche in seinem serverseitigen Cache durch. `getNote`

Wenn Sie eine aktualisieren`Note`, müssen Sie den Eintrag für die spezifische Notiz entfernen, um sicherzustellen, dass sie bei der nächsten Anforderung aus der Backend-Datenquelle abgerufen wird. Dazu müssen Sie einen Request-Handler erstellen.

Das folgende Beispiel zeigt eine Möglichkeit, die Räumung mit dieser Methode zu handhaben:

```
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;
```

Alternativ können Sie die Räumung auch im Response-Handler abwickeln.

Wenn die `updateNote` Mutation verarbeitet ist, wird AWS AppSync versucht, den Eintrag zu entfernen. Wenn ein Eintrag erfolgreich gelöscht wurde, enthält die Antwort einen `apiCacheEntriesDeleted` Wert im `extensions` Objekt, der angibt, wie viele Einträge gelöscht wurden:

```
"extensions": {  "apiCacheEntriesDeleted": 1}
```

Löschen eines Cache-Eintrags auf der Grundlage seiner Identität

Sie können Caching-Schlüssel auf der Grundlage mehrerer Werte aus dem Objekt erstellen. `context`

Nehmen wir zum Beispiel das folgende Schema, das Amazon Cognito Cognito-Benutzerpools als Standard-Authentifizierungsmodus verwendet und von einer Amazon DynamoDB DynamoDB-Datenquelle unterstützt wird:

```
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
}
```

Die `Note` Objekttypen werden in einer DynamoDB-Tabelle gespeichert. Die Tabelle hat einen zusammengesetzten Schlüssel, der den Amazon Cognito Cognito-Benutzernamen als Primärschlüssel und den `id` (einen Slug) von `Note` als Partitionsschlüssel verwendet. Dies ist ein

Mehrmandantensystem, das es mehreren Benutzern ermöglicht, ihre privaten `Note` Objekte zu hosten und zu aktualisieren, die niemals gemeinsam genutzt werden.

Da es sich um ein leseintensives System handelt, wird die `getNote` Abfrage per Resolver-Caching zwischengespeichert, wobei der Caching-Schlüssel aus folgenden Komponenten besteht: `[context.identity.username, context.arguments.id]` Wenn a aktualisiert `Note` wird, können Sie den entsprechenden Eintrag löschen. `Note` Sie müssen die Komponenten dem Objekt in derselben Reihenfolge hinzufügen, in der sie in der Liste Ihres Resolvers angegeben sind. `cachingKeys`

Das folgende Beispiel zeigt dies:

```
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;
```

Ein Backend-System kann den Eintrag auch aktualisieren `Note` und entfernen. Nehmen wir zum Beispiel diese Mutation:

```
type Mutation {
  updateNoteFromBackend(id: ID!, content: String!, username: ID!): Note @aws_iam
}
```

Sie können den Eintrag entfernen, aber die Komponenten des Caching-Schlüssels zum Objekt hinzufügen. `cachingKeys`

Im folgenden Beispiel erfolgt die Räumung in der Antwort des Resolvers:

```
import { util, Context } from '@aws-appsync/utils';
import { update } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
```

```
    return update({ key: { id: ctx.args.id }, update: { context: ctx.args.content } });
}

export function response(ctx) {
    extensions.evictFromApiCache('Query', 'getNote', {
        'ctx.identity.username': ctx.args.username,
        'ctx.args.id': ctx.args.id,
    });
    return ctx.result;
}
```

In Fällen, in denen Ihre Backend-Daten außerhalb von aktualisiert wurden AWS AppSync, können Sie ein Element aus dem Cache entfernen, indem Sie eine Mutation aufrufen, die eine Datenquelle verwendet. `NONE`

API-Antworten komprimieren

AWS AppSync ermöglicht es Clients, komprimierte Payloads anzufordern. Auf Anfrage werden API-Antworten komprimiert und als Antwort auf Anfragen zurückgegeben, die angeben, dass komprimierter Inhalt bevorzugt wird. Komprimierte API-Antworten werden schneller geladen, Inhalte werden schneller heruntergeladen und Ihre Datenübertragungsgebühren können ebenfalls reduziert werden.

> **Note**
>
> Die Komprimierung ist für alle neuen Versionen verfügbar, die nach dem 1. Juni 2020 APIs erstellt wurden.
> AWS AppSync komprimiert Objekte nach bestem Wissen. In seltenen Fällen AWS AppSync kann die Komprimierung aufgrund einer Vielzahl von Faktoren, einschließlich der aktuellen Kapazität, übersprungen werden.

AWS AppSync kann GraphQL-Abfrage-Payloadgrößen zwischen 1.000 und 10.000.000 Byte komprimieren. Um die Komprimierung zu aktivieren, muss ein Client den `Accept-Encoding` Header mit dem Wert senden. `gzip` Die Komprimierung kann überprüft werden, indem der Wert des `Content-Encoding` Headers in der Antwort (`gzip`) überprüft wird.

Der Abfrage-Explorer in der AWS AppSync Konsole legt standardmäßig automatisch den Header-Wert in der Anfrage fest. Wenn Sie eine Abfrage ausführen, die eine ausreichend große Antwort hat, kann die Komprimierung mithilfe der Entwicklertools Ihres Browsers bestätigt werden.

Konfiguration benutzerdefinierter Domainnamen für GraphQL und Echtzeit APIs

Mit können Sie benutzerdefinierte Domainnamen verwenden AWS AppSync, um eine einzelne, einprägsame Domain zu konfigurieren, die sowohl für Ihr GraphQL als auch für Echtzeit APIs funktioniert.

Mit anderen Worten, Sie können einen einfachen und einprägsamen Endpunkt URLs mit Domainnamen Ihrer Wahl verwenden, indem Sie benutzerdefinierte Domainnamen erstellen, die Sie mit den AWS AppSync APIs in Ihrem Konto verknüpfen.

Wenn Sie eine AWS AppSync API konfigurieren, werden zwei Endpunkte bereitgestellt:

AWS AppSync GraphQL-Endpunkt:

```
https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql
```

AWS AppSync Echtzeit-Endpunkt:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Mit benutzerdefinierten Domainnamen können Sie über eine einzige Domain mit beiden Endpunkten interagieren. Wenn Sie beispielsweise `api.example.com` als Ihre benutzerdefinierte Domain konfigurieren, können Sie mit diesen sowohl mit Ihren GraphQL- als auch mit Echtzeit-Endpunkten interagieren: URLs

AWS AppSync GraphQL-Endpunkt für benutzerdefinierte Domänen:

```
https://api.example.com/graphql
```

AWS AppSync Echtzeit-Endpunkt für benutzerdefinierte Domänen:

```
wss://api.example.com/graphql/realtime
```

> **Note**
>
> AWS AppSync APIs unterstützt nur TLS 1.2 und TLS 1.3 für benutzerdefinierte Domainnamen.

Registrierung und Konfiguration eines Domainnamens

Um benutzerdefinierte Domainnamen für Sie einzurichten AWS AppSync APIs, benötigen Sie einen registrierten Internet-Domainnamen. Sie können eine Internetdomain bei einem Domain-Registrar Ihrer Wahl Amazon Route 53 domain registration oder bei einem Domain-Registrar eines Drittanbieters registrieren. Weitere Informationen zu Route 53 finden Sie unter Was ist Amazon Route 53? im Amazon Route 53 Developer Guide.

Der benutzerdefinierte Domainname einer API kann der Name einer Subdomain oder der Root-Domain (auch als „Zone-Apex“ bezeichnet) einer registrierten Internet-Domain sein. Nachdem Sie einen benutzerdefinierten Domainnamen in erstellt haben AWS AppSync, müssen Sie den Ressourceneintrag Ihres DNS-Anbieters erstellen oder aktualisieren, um ihn Ihrem API-Endpunkt zuzuordnen. Ohne diese Zuordnung können API-Anfragen, die für den benutzerdefinierten Domainnamen bestimmt sind, nicht erreicht werden AWS AppSync.

Einen benutzerdefinierten Domainnamen erstellen in AWS AppSync

Durch das Erstellen eines benutzerdefinierten Domainnamens für eine AWS AppSync API wird eine Amazon CloudFront Verteilung eingerichtet. Sie müssen einen DNS-Eintrag einrichten, um den benutzerdefinierten Domainnamen dem CloudFront Distributionsdomänennamen zuzuordnen. Diese Zuordnung ist erforderlich, um API-Anfragen, die für den benutzerdefinierten Domainnamen gebunden sind, AWS AppSync über die zugeordnete CloudFront Verteilung weiterzuleiten. Außerdem müssen Sie ein Zertifikat für den benutzerdefinierten Domänennamen bereitstellen.

Um den benutzerdefinierten Domainnamen einzurichten oder das zugehörige Zertifikat zu aktualisieren, müssen Sie berechtigt sein, CloudFront Distributionen zu aktualisieren und das AWS Certificate Manager (ACM) -Zertifikat zu beschreiben, das Sie verwenden möchten. Um diese Berechtigungen zu gewähren, fügen Sie einem IAM-Benutzer, einer Gruppe oder einer IAM-Rolle in Ihrem Konto die folgende AWS Identity and Access Management (IAM-) Richtlinienerklärung bei:

JSON

```
{
    "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 AppSync unterstützt benutzerdefinierte Domainnamen, indem Server Name Indication (SNI) bei der Distribution genutzt wird. CloudFront Weitere Informationen zur Verwendung benutzerdefinierter Domainnamen in einer CloudFront Distribution, einschließlich des erforderlichen Zertifikatsformats und der maximalen Zertifikatsschlüssellänge, finden Sie unter Using HTTPS with CloudFront im Amazon CloudFront Developer Guide.

Um einen benutzerdefinierten Domainnamen als Hostnamen der API einzurichten, muss der API-Besitzer ein öffentliches oder importiertes ACM-Zertifikat in der `us-east-1` AWS Region (USA Ost (Nord-Virginia)) bereitstellen, das den benutzerdefinierten Domainnamen abdeckt. Weitere Informationen zu ACM finden Sie unter Was ist? AWS Certificate Manager im AWS Certificate Manager Benutzerhandbuch.

Benutzerdefinierte Wildcard-Domainnamen in AWS AppSync

AWS AppSync unterstützt benutzerdefinierte Domainnamen mit Platzhaltern. Um einen benutzerdefinierten Domainnamen mit Platzhaltern zu konfigurieren, geben Sie ein Platzhalterzeichen (*) als erste Subdomain einer benutzerdefinierten Domäne an. Dies steht für alle möglichen Subdomänen der Stammdomäne. Beispielsweise `*.example.com` führt der benutzerdefinierte Domänenname mit Platzhaltern zu Subdomänen wie`a.example.com`, und`b.example.com`. `c.example.com` Alle diese Subdomänen werden zu derselben Domain weitergeleitet.

Um einen benutzerdefinierten Domainnamen mit Platzhalter verwenden zu können AWS AppSync, müssen Sie ein von ACM ausgestelltes Zertifikat bereitstellen, das einen Platzhalternamen enthält,

der mehrere Websites in derselben Domain schützen kann. Weitere Informationen finden Sie unter Eigenschaften des ACM-Zertifikats im Benutzerhandbuch.AWS Certificate Manager

Versionierung, Konflikterkennung und Synchronisierungsvorgänge für DynamoDB-Datenquellen in AWS AppSync

AWS AppSyncDie erweiterten Datenverwaltungsfunktionen für DynamoDB nutzen drei Hauptfunktionen: versionierte Datenquellen, Konflikterkennung und -lösung sowie Synchronisierungsvorgänge. Diese Tools ermöglichen robuste, skalierbare Anwendungen, die gleichzeitige Datenänderungen und Synchronisierungen in verteilten Umgebungen effizient handhaben.

Versionierte Datenquellen bilden die Grundlage AWS AppSync des Datenverwaltungssystems. Diese Funktion erweitert DynamoDB-Elemente automatisch mit Versionsmetadaten, zeichnet Änderungen auf, die durch AWS AppSync Mutationen an einer Delta-Tabelle vorgenommen wurden, und verwaltet „Tombstones“ für gelöschte Elemente. Entwickler können Aufbewahrungsfristen für gelöschte Elemente und Änderungsprotokolle konfigurieren, wodurch der Speicherplatz optimiert und gleichzeitig die Datenkonsistenz gewährleistet wird. Versionierte Datenquellen optimieren die Implementierung von Konflikterkennungs- und Synchronisierungsvorgängen und bieten so eine solide Grundlage für eine erweiterte Datenverarbeitung.

Mechanismen zur Konflikterkennung und -lösung gewährleisten die Datenintegrität bei gleichzeitigen Schreibvorgängen. AWS AppSync bietet drei Strategien: Optimistic Concurrency, Automerge und Lambda-basierte Lösung. Optimistic Concurrency lehnt widersprüchliche Mutationen ab, sodass Clients es mit aktualisierten Daten erneut versuchen können. Automerge löst automatisch Konflikte auf der Grundlage von Datentypen, führt Listen zusammen, führt Mengenvereinigungen durch und behält bestehende Skalarwerte bei. Die Lambda-basierte Lösung ermöglicht eine benutzerdefinierte Logik für komplexe Konfliktszenarien. Diese Optionen bieten Entwicklern Flexibilität beim Umgang mit Datenkonflikten und gewährleisten so die Konsistenz zwischen verteilten Systemen.

Synchronisierungsvorgänge ermöglichen das effiziente Abrufen und Aktualisieren von Daten in Client-Anwendungen. Mit dieser Funktion können Clients alle Ergebnisse aus einer DynamoDB-Tabelle abrufen und anschließend nur Daten abrufen, die seit ihrer letzten Abfrage geändert wurden. AWS AppSync bestimmt anhand des bereitgestellten Synchronisierungstokens, ob auf die Basistabelle oder die Delta-Tabelle zugegriffen werden soll, wodurch die Leistung optimiert und die Datenübertragung reduziert wird.

Themen

- Versionierung von DynamoDB-Datenquellen in AWS AppSync
- Erkennung und Lösung von Konflikten in AWS AppSync
- Verwenden von DynamoDB-Synchronisierungsvorgängen für versionierte Datenquellen in AWS AppSync

Versionierung von DynamoDB-Datenquellen in AWS AppSync

AWS AppSync unterstützt derzeit die Versionierung von DynamoDB-Datenquellen. Konflikterkennungs-, Konfliktlösungs- und Synchronisierungsvorgänge erfordern eine `Versioned`-Datenquelle. Wenn Sie die Versionierung für eine Datenquelle aktivieren, wird automatisch: AWS AppSync

- Erweitern von Elementen mit Metadaten zur Objektversionierung.
- Zeichnet Änderungen an Elementen mit AWS AppSync Mutationen in einer Delta-Tabelle auf.
- Verwalten gelöschter Elemente in der Basis-Tabelle mit einem „Tombstone“ für einen konfigurierbaren Zeitraum.

Konfiguration der versionierten Datenquelle

Wenn Sie die Versionierung für eine DynamoDB-Datenquelle aktivieren, geben Sie die folgenden Felder an:

`BaseTableTTL`

Die Anzahl der Minuten, für die gelöschte Elemente in der Basis-Tabelle mit einem „Tombstone“ - einem Metadatenfeld, das angibt, dass das Element gelöscht wurde - beibehalten werden. Sie können diesen Wert auf 0 setzen, wenn Elemente sofort entfernt werden sollen, wenn sie gelöscht werden. Dies ist ein Pflichtfeld.

`DeltaSyncTableName`

Der Name der Tabelle, in der Änderungen an Elementen mit AWS AppSync Mutationen gespeichert werden. Dies ist ein Pflichtfeld.

`DeltaSyncTableTTL`

Die Anzahl der Minuten, in denen Elemente in der Delta-Tabelle aufbewahrt werden sollen. Dies ist ein Pflichtfeld.

Protokollierung der Delta-Sync-Tabelle

AWS AppSync unterstützt derzeit Delta Sync Logging für Mutationen mit `PutItem`, `UpdateItem`, und `DeleteItem` DynamoDB-Operationen.

Wenn eine AWS AppSync Mutation ein Element in einer versionierten Datenquelle ändert, wird ein Datensatz dieser Änderung in einer Delta-Tabelle gespeichert, die für inkrementelle Aktualisierungen optimiert ist. Sie können wählen, ob Sie verschiedene Delta-Tabellen (z. B. eine pro Typ, eine pro Domainbereich) für andere versionierte Datenquellen oder eine einzelne Delta-Tabelle für Ihre API verwenden möchten. AWS AppSync empfiehlt, keine einzige Delta-Tabelle für mehrere APIs zu verwenden, um die Kollision von Primärschlüsseln zu vermeiden.

Das für diese Tabelle erforderliche Schema ist wie folgt:

`ds_pk`

Ein Zeichenfolgenwert, der als Partitionsschlüssel verwendet wird. Sie besteht aus der Verkettung des Namens der Basisdatenquelle und des ISO-8601-Formats des Datums, an dem die Änderung eingetreten ist (z. B.). `Comments:2019-01-01`

Wenn das *`customPartitionKey`* Flag aus der VTL-Mapping-Vorlage als Spaltenname des Partitionsschlüssels gesetzt ist (siehe Resolver Mapping Template Reference for DynamoDB im AWS AppSync Developer Guide), ändert sich das Format der *`ds_pk`* Änderungen und die Zeichenfolge wird erstellt, indem ihr der Wert des Partitionsschlüssels im neuen Datensatz in der Basistabelle angehängt wird. Wenn der Datensatz in der Basistabelle beispielsweise den Partitionsschlüsselwert `1a` und den Sortierschlüsselwert von hat2b, lautet der neue Wert der Zeichenfolge:`Comments:2019-01-01:1a`.

`ds_sk`

Ein Zeichenfolgenwert, der als Sortierschlüssel verwendet wird. Es wird erstellt, indem das ISO 8601-Format des Zeitpunkts der Änderung, des Primärschlüssels des Elements und der Version des Elements verkettet wird. Die Kombination dieser Felder garantiert die Einzigartigkeit jedes Eintrags in der Delta-Tabelle (z. B. für eine Uhrzeit und eine ID von `09:30:00` und eine Version von `1a2`, das wäre). `09:30:00:1a:2`

Wenn das *`customPartitionKey`* Flag der VTL-Mapping-Vorlage auf den Spaltennamen des Partitionsschlüssels gesetzt ist (siehe Resolver Mapping Template Reference for DynamoDB im AWS AppSync Developer Guide), ändert sich das Format der *`ds_sk`* Änderungen und die Zeichenfolge wird erstellt, indem der Wert des Kombinationsschlüssels durch den Wert des

Sortierschlüssels in der Basistabelle ersetzt wird. Wenn der Datensatz in der Basistabelle gemäß dem vorherigen Beispiel den Partitionsschlüsselwert `1a` und den Sortierschlüsselwert von `hat2b`, lautet der neue Wert der Zeichenfolge:`09:30:00:2b:3`.

`_ttl`

Ein numerischer Wert, der den Zeitstempel in Epochensekunden speichert, zu dem ein Element aus der Delta-Tabelle entfernt werden soll. Dieser Wert wird festgelegt, indem der in der Datenquelle konfigurierte `DeltaSyncTableTTL`-Wert zum Zeitpunkt der Änderung hinzugefügt wird. Dieses Feld sollte als DynamoDB TTL-Attribut konfiguriert werden.

Die für die Verwendung mit der Basis-Tabelle konfigurierte IAM-Rolle muss auch die Berechtigung zum Betrieb für die Delta-Tabelle enthalten. In diesem Beispiel wird die Berechtigungsscheine für eine Basis-Tabelle mit dem Namen `Comments` und eine Delta-Tabelle mit dem Namen `ChangeLog` angezeigt:

JSON

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

Versionierte Datenquellen-Metadaten

AWS AppSync verwaltet in Ihrem Namen Metadatenfelder in `Versioned` Datenquellen. Wenn Sie diese Felder selbst ändern, kann dies zu Fehlern in Ihrer Anwendung oder zu Datenverlusten führen. Zu diesen Feldern gehören:

`_version`

Ein monoton steigender Zähler, der jedes Mal aktualisiert wird, wenn eine Änderung an einem Element auftritt.

`_lastChangedAt`

Ein numerischer Wert, der den Zeitstempel in Epochenmillisekunden speichert, an dem ein Element zuletzt geändert wurde.

`_deleted`

Ein boolescher „Tombstone“-Wert, der angibt, dass ein Element gelöscht wurde. Dies kann von Anwendungen verwendet werden, um gelöschte Elemente aus lokalen Datenspeichern zu entfernen.

`_ttl`

Ein numerischer Wert, der den Zeitstempel in Epochensekunden speichert, an dem ein Element aus der zugrunde liegenden Datenquelle entfernt werden soll.

`ds_pk`

Ein Zeichenfolgenwert, der als Partitionsschlüssel für Delta-Tabellen verwendet wird.

`ds_sk`

Ein Zeichenfolgenwert, der als Sortierschlüssel für Delta-Tabellen verwendet wird.

`gsi_ds_pk`

Ein Zeichenkettenwertattribut, das generiert wurde, um einen globalen sekundären Index als Partitionsschlüssel zu unterstützen. Es wird nur aufgenommen, wenn *`customPartitionKey`* sowohl die *`populateIndexFields`* Flags als auch in der VTL-Mapping-Vorlage aktiviert sind (siehe Resolver Mapping Template Reference for DynamoDB im AWS AppSync Developer Guide). Wenn diese Option aktiviert ist, wird der Wert durch Verkettung des Namens der Basisdatenquelle und des ISO 8601-Formats des Datums, an dem die Änderung vorgenommen wurde, gebildet (wenn die Basistabelle beispielsweise Comments heißt, wird dieser Datensatz als gesetzt). `Comments:2019-01-01`

`gsi_ds_sk`

Ein Zeichenkettenwertattribut, das generiert wurde, um einen globalen sekundären Index als Sortierschlüssel zu unterstützen. Es wird nur aufgenommen, wenn *`customPartitionKey`* sowohl die *`populateIndexFields`* Flags als auch in der VTL-Mapping-Vorlage aktiviert sind (siehe Resolver Mapping Template Reference for DynamoDB im AWS AppSync Developer Guide). Wenn diese Option aktiviert ist, wird der Wert durch Verkettung des ISO 8601-Formats des Zeitpunkts, zu dem die Änderung erfolgte, des Partitionsschlüssels des Elements in der Basistabelle, des Sortierschlüssels des Elements in der Basistabelle und der Version des Elements (z. B. für eine Zeit von`09:30:00`, ein Partitionsschlüsselwert von`1a`, ein Sortierschlüsselwert von `2b` und die Version von`3`, das wäre`09:30:00:1a#2b:3`) konstruiert.

Diese Metadatenfelder wirken sich auf die Gesamtgröße der Elemente in der zugrunde liegenden Datenquelle aus. AWS AppSync empfiehlt, beim Entwerfen Ihrer Anwendung 500 Byte oder mehr als die maximale Primärschlüsselgröße des Speichers für versionierte Datenquellenmetadaten zu reservieren. Um diese Metadaten in Clientanwendungen zu verwenden, schließen Sie die Felder `_version`, `_lastChangedAt` und `_deleted` in Ihre GraphQL-Typen und in den Auswahlsatz für Mutationen ein.

Erkennung und Lösung von Konflikten in AWS AppSync

Bei gleichzeitigen Schreibvorgängen mit AWS AppSync können Sie Strategien zur Konflikterkennung und Konfliktlösung konfigurieren, um Aktualisierungen entsprechend zu handhaben. Die Konflikterkennung bestimmt, ob die Mutation mit dem tatsächlich geschriebenen Element in der Datenquelle in Konflikt steht. Die Konflikterkennung wird aktiviert, indem der Wert im `conflictDetection` Feld SyncConfig für auf `VERSION` gesetzt wird.

Die Konfliktlösung ist die Aktion, die für den Fall ausgeführt wird, dass ein Konflikt erkannt wird. Dies wird bestimmt, indem das Feld Conflict Handler im festgelegt wird SyncConfig. Es gibt drei Strategien zur Konfliktlösung:

- OPTIMISTIC_CONCURRENCY
- AUTOMERGE
- LAMBDA

Versionen werden AWS AppSync bei Schreibvorgängen automatisch inkrementiert und sollten nicht von Clients oder außerhalb eines Resolvers geändert werden, der mit einer versionsfähigen

Datenquelle konfiguriert ist. Dadurch ändert sich das Konsistenzverhalten des Systems, was zu Datenverlusten führen kann.

Optimistische Parallelität

Optimistische Parallelität ist eine Konfliktlösungsstrategie, die versionierte Datenquellen AWS AppSync vorsieht. Wenn der Konflikt-Resolver auf „Optimistische Parallelität“ gesetzt ist und eine eingehende Mutation eine Version hat, die sich von der tatsächlichen Version des Objekts unterscheidet, weist der Conflict Handler die eingehende Anforderung einfach zurück. Innerhalb der GraphQL-Antwort wird das vorhandene Element auf dem Server mit der neuesten Version bereitgestellt. Vom Client wird dann erwartet, dass er diesen Konflikt lokal behandelt und die Mutation mit der aktualisierten Version des Elements erneut versucht.

Automatische Zusammenführungen

Automerge bietet Entwicklern eine einfache Möglichkeit, eine Konfliktlösungsstrategie zu konfigurieren, ohne clientseitige Logik zu schreiben, um Konflikte manuell zusammenzuführen, die nicht von anderen Strategien behandelt werden konnten. Automerge hält sich an einen strengen Regelsatz, wenn Daten zusammengeführt werden, um Konflikte zu lösen. Die Grundsätze des Automerge-Verfahrens drehen sich um den zugrunde liegenden Datentyp des GraphQL-Feldes. Diese sind:

- Konflikt in einem Skalarfeld: GraphQL-Skalar oder jedes Feld, das keine Sammlung ist (d. h. List, Set, Map). Zurückweisen des eingehenden Werts für das skalare Feld und Auswahl des auf dem Server vorhandenen Werts.
- Konflikt in einer Liste: GraphQL-Typ und Datenbanktyp sind Listen. Verketten der eingehenden Liste mit der vorhandenen Liste auf dem Server. Die Listenwerte in der eingehenden Mutation werden an das Ende der Liste auf dem Server angehängt. Doppelte Werte werden beibehalten.
- Konflikt auf einem Set: Der GraphQL-Typ ist eine Liste, und der Datenbanktyp ist ein Set. Anwenden einer Set-Vereinigung mit dem eingehenden und dem vorhandenen Set auf dem Server. Dies entspricht den Eigenschaften eines Sets, d. h. es gibt keine doppelten Einträge.
- Wenn eine eingehende Mutation dem Element ein neues Feld hinzufügt oder auf ein Feld mit dem Wert von angewendet wird`null`, fügen Sie dieses Feld mit dem vorhandenen Element zusammen.
- Konflikt auf einer Map: Wenn der zugrunde liegende Datentyp in der Datenbank eine Map ist (d.h ein Schlüssel-Wert-Dokument), werden die die oben genannten Regeln angewendet, wenn jede Eigenschaft der Map analysiert und verarbeitet wird.

Automerge wurde entwickelt, um Anforderungen automatisch mit einer aktualisierten Version zu erkennen, zusammenzuführen und erneut zu versuchen, so dass der Client keine Konflikte manuell zusammenführen muss.

Um ein Beispiel dafür zu zeigen, wie Automerge einen Konflikt auf einem Scalar-Typ verarbeitet, verwenden wir den folgenden Datensatz als Ausgangspunkt.

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "_version" : 4
}
```

Jetzt versucht eine eingehende Mutation möglicherweise, das Element zu aktualisieren, jedoch mit einer älteren Version, da der Client noch nicht mit dem Server synchronisiert wurde. Dies sollte wie folgt aussehen:

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 55,
  "_version" : 2
}
```

Beachten Sie die veraltete Version von 2 in der eingehenden Anforderung. Während dieses Vorgangs führt Automerge die Daten zusammen, indem die Aktualisierung des Feldes „jersey“ auf „55“ abgelehnt wird und der Wert bei „5“ bleibt, was dazu führt, dass das folgende Abbild des Elements auf dem Server gespeichert wird.

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "_version" : 5 # version is incremented every time automerge performs a merge that is
 stored on the server.
}
```

Angesichts des Status des oben gezeigten Elements mit Version 5, nehmen wir jetzt an, es geht eine Mutation ein, die versucht, das Element mit dem folgenden Abbild zu mutieren:

```
{
  "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
}
```

Es gibt drei Punkte zu dieser eingehenden Mutation drei wichtige Punkte. Der Name, ein Skalar, wurde geändert, aber zwei neue Felder „interests“, ein Satz und „points“, eine Liste, wurden hinzugefügt. In diesem Szenario wird ein Konflikt aufgrund fehlender Versionsübereinstimmungen erkannt. Automerge hält sich an seine Eigenschaften und weist die Namensänderung zurück, da es sich um einen Skalar und eine Hinzufügung zu den nicht in Konflikt stehenden Feldern handelt. Dies führt dazu, dass das Element, das auf dem Server gespeichert wird, wie folgt aussieht.

```
{
  "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
}
```

Bei dem aktualisierten Abbild des Elements mit Version 6 nehmen wir nur an, dass eine eingehende Mutation (mit einer anderen fehlenden Versionsübereinstimmung) versucht, das Element in Folgendes umzuwandeln:

```
{
  "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
}
```

Hier beobachten wir, dass das eingehende Feld für „interests“ einen doppelten Wert hat, der auf dem Server vorhanden ist, sowie zwei neue Werte. Da der zugrunde liegende Datentyp ein Set

ist, kombiniert Automerge in diesem Fall die Werte, die auf dem Server vorhanden sind, mit den Werten in der eingehenden Anforderung und entfernt alle Duplikate. Ähnlich besteht ein Konflikt im Feld „points“, in dem ein doppelter Wert und ein neuer Wert vorhanden ist. Da der zugrunde liegende Datentyp hier jedoch eine Liste ist, hängt Automerge einfach alle Werte in der eingehenden Anforderung an das Ende der bereits auf dem Server vorhandenen Werte an. Das resultierende zusammengeführte Abbild, das auf dem Server gespeichert wird, sieht wie folgt aus:

```
{
  "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
}
```

Nehmen wir an, dass das Element, das auf dem Server gespeichert ist, in Version 8 wie folgt aussieht.

```
{
  "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
}
```

Eine eingehende Anforderung versucht jedoch, das Element mit dem folgenden Abbild zu aktualisieren, erneut mit einer fehlenden Versionsübereinstimmung:

```
{
  "id" : 1,
  "name" : "Nadia",
  "stats": {
```

```
      "ppg": "25.7",
      "rpg": "6.9"
  }
  "_version" : 3
}
```

In diesem Szenario können wir sehen, dass die Felder, die bereits auf dem Server vorhanden sind, fehlen (interests, points, jersey). Zusätzlich wird der Wert für „ppg“ innerhalb der Map „stats“ bearbeitet, ein neuer Wert „rpg“ wird hinzugefügt, und „apg“ wird weggelassen. Automerge behält die Felder, die weggelassen wurden (Hinweis: Wenn Felder entfernt werden sollen, muss die Anforderung erneut mit der übereinstimmenden Version versucht werden), damit sie nicht verloren gehen. Es wendet auch die gleichen Regeln auf Felder in Maps an, weshalb die Änderung zu „ppg“ abgelehnt wird, während „apg“ erhalten bleibt und „rpg“, ein neues Feld, hinzugefügt wird. Das resultierende Element, das auf dem Server gespeichert wird, sieht jetzt wie folgt aus:

```
{
  "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
}
```

Lambdas

Es stehen mehrere Strategien zur Lambda-Auflösung zur Auswahl:

- `RESOLVE`: Ersetzt den vorhandenen Artikel durch einen neuen Artikel, der als Antwort-Payload geliefert wird. Sie können denselben Vorgang nur für ein einzelnes Element gleichzeitig wiederholen. Derzeit unterstützt für DynamoDB `PutItem` und `UpdateItem`.
- `REJECT`: Lehnt die Mutation ab und gibt einen Fehler mit dem vorhandenen Element in der GraphQL-Antwort zurück. Derzeit unterstützt für DynamoDB `PutItem`, `UpdateItem` und `DeleteItem`.

- `REMOVE`: Entfernt das vorhandene Element. Derzeit unterstützt für DynamoDB `DeleteItem`.

Die Lambda-Aufrufanforderung

Der AWS AppSync DynamoDB-Resolver ruft die in der angegebene Lambda-Funktion auf. `LambdaConflictHandlerArn` Er verwendet die gleiche `service-role-arn`-Konfiguration für die Datenquelle. Die Nutzlast des Aufrufs hat die folgende Struktur:

```
{
    "newItem": { ... },
    "existingItem": {... },
    "arguments": { ... },
    "resolver": { ... },
    "identity": { ... }
}
```

Die Felder sind wie folgt definiert:

`newItem`

Das Vorschauelement, wenn die Mutation erfolgreich war.

`existingItem`

Das Element, das sich derzeit in der DynamoDB-Tabelle befindet.

`arguments`

Die Argumente aus der GraphQL-Mutation.

`resolver`

Informationen über den Resolver. AWS AppSync

`identity`

Informationen über den Aufrufer. Dieses Feld wird auf null gesetzt, wenn der Zugriff mit API-Schlüssel erfolgt.

Beispiel-Nutzlast:

```
{
```

```
    "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"
    }
}
```

Die Lambda-Aufrufantwort

Für `PutItem`- und `UpdateItem`-Konfliktlösung

Die Mutation `RESOLVE`. Die Antwort muss das folgende Format haben.

```
{
    "action": "RESOLVE",
    "item": { ... }
}
```

Das `item`-Feld stellt ein Objekt dar, das verwendet wird, um das vorhandene Element in der zugrunde liegenden Datenquelle zu ersetzen. Der Primärschlüssel und die Synchronisierungsmetadaten werden ignoriert, wenn sie in `item` enthalten sind.

Die Mutation `REJECT`. Die Antwort muss das folgende Format haben.

```
{
    "action": "REJECT"
}
```

Für `DeleteItem`-Konfliktlösung

Das Element `REMOVE` Die Antwort muss das folgende Format haben.

```
{
    "action": "REMOVE"
}
```

Die Mutation `REJECT`. Die Antwort muss das folgende Format haben.

```
{
    "action": "REJECT"
}
```

Die folgende Lambda-Beispielfunktion überprüft, wer den Aufruf tätigt, sowie den Namen des Resolvers. Falls es sich um das Objekt handelt`jeffTheAdmin`, `REMOVE` das DeletePost gelöst werden soll, oder `RESOLVE` der Konflikt mit dem neuen Objekt für Update/Put die Problembehebung. Wenn nicht, ist die Mutation `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;
}
```

Fehler

Im Folgenden finden Sie eine Liste möglicher Fehler, die während eines Konfliktlösungsprozesses auftreten können:

`ConflictUnhandled`

Die Konflikterkennung findet eine fehlende Versionsübereinstimmung, und der Conflict Handler lehnt die Mutation ab.

Beispiel: Konfliktlösung mit einem Conflict Handler für optimistische Parallelität. Oder Lambda-Conflict Handler, zurückgegeben mit `REJECT`.

`ConflictError`

Beim Versuch, einen Konflikt zu lösen, tritt ein interner Fehler auf.

Beispiel: Lambda-Conflict Handler gibt eine nicht wohlgeformte Antwort zurück. Oder: Lambda-Conflict Handler kann nicht aufgerufen werden, da die bereitgestellte Ressource `LambdaConflictHandlerArn` nicht gefunden wurde.

`MaxConflicts`

Für die Konfliktlösung wurde die maximale Zahl der Wiederholungsversuche erreicht.

Beispiel: Zu viele gleichzeitige Anfragen für dasselbe Objekt. Bevor der Konflikt gelöst wird, wird das Objekt von einem anderen Client auf eine neue Version aktualisiert.

`BadRequest`

Client versucht, Metadatenfelder (`_version`, `_ttl`, `_lastChangedAt`, `_deleted`) zu aktualisieren.

Beispiel: Der Client versucht, ein Objekt mit `_version` einer Aktualisierungsmutation zu aktualisieren.

`DeltaSyncWriteError`

Fehler beim Schreiben des Delta-Synchronisierungsdatensatzes.

Beispiel: Mutation erfolgreich, aber ein interner Fehler trat auf, als versucht wurde, in die Delta-Synchronisationstabelle zu schreiben.

`InternalFailure`

Es ist ein interner Fehler aufgetreten.

`UnsupportedOperation`

Der Vorgang '*X*' wird nicht unterstützt. Die Versionierung von Datenquellen unterstützt nur die folgenden Operationen (TransactGetItems,,, ScanPutItem, QueryBatchGetItem,,, GetItemDeleteItem, UpdateItem Sync).

Beispiel: Verwendung bestimmter Transaktions- und Batchoperationen bei aktiviertem Konflikt. detection/resolution Diese Operationen werden derzeit nicht unterstützt.

CloudWatch Logs

Wenn eine AWS AppSync API CloudWatch Protokolle aktiviert hat und die Protokollierungseinstellungen auf Protokolle auf Feldebene `enabled` und Protokollebene für die Protokolle auf Feldebene auf gesetzt sind`ALL`, AWS AppSync werden Informationen zur Konflikterkennung und -lösung an die Protokollgruppe ausgegeben. Informationen zum Format der Protokollmeldungen finden Sie in der Dokumentation zu Konflikterkennung und Synchronisierungsprotokollierung.

Verwenden von DynamoDB-Synchronisierungsvorgängen für versionierte Datenquellen in AWS AppSync

Versionierte Datenquellen unterstützen `Sync` Operationen, mit denen Sie alle Ergebnisse aus einer DynamoDB-Tabelle abrufen und dann nur die Daten empfangen können, die seit Ihrer letzten Abfrage geändert wurden (die Delta-Updates). Wenn es eine Anforderung für einen `Sync` Vorgang AWS AppSync erhält, verwendet es die in der Anforderung angegebenen Felder, um zu bestimmen, ob auf die Basistabelle oder die Delta-Tabelle zugegriffen werden soll.

- Wenn das `lastSync`-Feld nicht angegeben ist, wird ein `Scan` auf der Basis-Tabelle durchgeführt.
- Wenn das `lastSync`-Feld angegeben ist, der Wert jedoch vor dem `current moment - DeltaSyncTTL` liegt, wird ein `Scan` auf der Basis-Tabelle durchgeführt.
- Wenn das `lastSync`-Feld angegeben ist und der Wert auf oder nach dem `current moment - DeltaSyncTTL` liegt, wird ein `Query` auf der Delta-Tabelle ausgeführt.

AWS AppSync gibt das `startedAt` Feld an die Antwortzuordnungsvorlage für alle `Sync` Operationen zurück. Das `startedAt`-Feld ist der Moment, in Epochenmillisekunden, an dem der `Sync`-Vorgang gestartet wurde, den Sie lokal speichern und in einer anderen Anforderung verwenden können. Wenn ein Paginierungstoken in der Anforderung enthalten war, entspricht dieser Wert dem Wert, der von der Anforderung für die erste Ergebnisseite zurückgegeben wird.

Informationen zum Format für `Sync`-Zuweisungsvorlagen finden Sie in der Zuweisungsvorlagen-Referenz.

Verwendung CloudWatch zur Überwachung und Protokollierung von GraphQL-API-Daten

Sie können Ihre GraphQL-API mithilfe von CloudWatch Metriken und Protokollen protokollieren und CloudWatch debuggen. Diese Tools ermöglichen es Entwicklern, die Leistung zu überwachen, Probleme zu beheben und ihre GraphQL-Operationen effektiv zu optimieren.

CloudWatch metrics ist ein Tool, das eine breite Palette von Metriken zur Überwachung der API-Leistung und -Nutzung bietet. Diese Metriken lassen sich in zwei Hauptkategorien einteilen:

1. Allgemeine API-Metriken: Dazu gehören `4XXError` und `5XXError` zur Nachverfolgung von Client- und Serverfehlern, `Latency` zur Messung von Antwortzeiten, `Requests` zur Überwachung der gesamten API-Aufrufe und `TokensConsumed` zur Nachverfolgung der Ressourcennutzung.

2. Abonnement-Metriken in Echtzeit: Diese Metriken konzentrieren sich auf WebSocket Verbindungen und Abonnementaktivitäten. Sie umfassen Messwerte für Verbindungsanfragen, erfolgreiche Verbindungen, Abonnementregistrierungen, die Veröffentlichung von Nachrichten sowie aktive Verbindungen und Abonnements.

In dem Leitfaden werden auch Enhanced Metrics vorgestellt, die detailliertere Daten zur Resolverleistung, zu Datenquelleninteraktionen und zu einzelnen GraphQL-Vorgängen bieten. Diese Metriken bieten tiefere Einblicke, sind jedoch mit zusätzlichen Kosten verbunden.

CloudWatch Logs ist ein Tool, das Logging-Funktionen für Ihr GraphQL APIs ermöglicht. Protokolle können auf zwei Ebenen der API eingerichtet werden:

1. Protokolle auf Anforderungsebene: Diese erfassen allgemeine Anforderungsinformationen, einschließlich HTTP-Header, GraphQL-Abfragen, Betriebsübersichten und Abonnementregistrierungen.
2. Protokolle auf Feldebene: Diese enthalten detaillierte Informationen zu einzelnen Feldauflösungen, einschließlich der Zuordnungen von Anfragen und Antworten sowie Ablaufverfolgungsinformationen für jedes Feld.

Sie können die Protokollierung konfigurieren, Protokolleinträge interpretieren und Protokolldaten zur Fehlerbehebung und Optimierung verwenden. AWS AppSync bietet verschiedene Protokolltypen, die Aufschluss über die Ausführungs-, Analyse-, Validierungs- und Feldauflösungsdaten Ihrer Abfrage geben.

Einrichtung und Konfiguration

Verwenden Sie die AWS AppSync Konsole, um die automatische Protokollierung auf einer GraphQL-API zu aktivieren.

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.
2. Wählen Sie auf der APIsSeite den Namen einer GraphQL-API aus.
3. Wählen Sie auf der Startseite Ihrer API im Navigationsbereich Einstellungen aus.
4. Führen Sie unter Logging (Protokollierung) folgende Schritte aus:

    a. Aktivieren Sie die Option „Protokolle aktivieren“.

    b. Für eine detaillierte Protokollierung auf Anforderungsebene aktivieren Sie das Kontrollkästchen unter Ausführlichen Inhalt einbeziehen. (optional)

c. Wählen Sie unter Field Resolver-Protokollebene Ihre bevorzugte Protokollierungsebene auf Feldebene aus (Keine, Fehler, Info, Debug oder Alle). (optional)

d. Wählen Sie unter Eine bestehende Rolle erstellen oder verwenden die Option Neue Rolle aus, um eine neue Rolle AWS Identity and Access Management (IAM) AWS AppSync zu erstellen, in die Logs geschrieben werden können. CloudWatch Oder wählen Sie Bestehende Rolle, um den Amazon-Ressourcennamen (ARN) einer vorhandenen IAM-Rolle in Ihrem AWS Konto auszuwählen.

5. Wählen Sie Speichern.

Manuelle IAM-Rollenkonfiguration

Wenn Sie sich dafür entscheiden, eine bestehende IAM-Rolle zu verwenden, muss die Rolle AWS AppSync die erforderlichen Berechtigungen zum Schreiben von Protokollen gewähren. CloudWatch Um dies manuell zu konfigurieren, müssen Sie eine Dienstrolle ARN angeben, damit diese Rolle beim Schreiben der Protokolle übernommen werden AWS AppSync kann.

Erstellen Sie in der IAM-Konsole eine neue Richtlinie mit dem Namen`AWSAppSyncPushToCloudWatchLogsPolicy`, der die folgende Definition hat:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

Erstellen Sie als Nächstes eine neue Rolle mit dem Namen AWSAppSyncPushToCloudWatchLogsRoleund fügen Sie der Rolle die neu erstellte Richtlinie hinzu. Bearbeiten Sie die Vertrauensstellung für diese Rolle wie folgt:

JSON

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

Kopieren Sie den Rollen-ARN und verwenden Sie ihn, wenn Sie die Protokollierung für eine AWS AppSync GraphQL-API einrichten.

CloudWatch Metriken

Sie können CloudWatch Metriken verwenden, um bestimmte Ereignisse zu überwachen und Warnmeldungen zu senden, die zu HTTP-Statuscodes oder Latenz führen können. Die folgenden Metriken werden ausgegeben:

Liste der Metriken

`4XXError`

Fehler aufgrund von Anfragen, die aufgrund einer falschen Client-Konfiguration nicht gültig sind. In der Regel treten diese Fehler überall außerhalb der GraphQL-Verarbeitung auf. Diese Fehler können beispielsweise auftreten, wenn die Anfrage eine falsche JSON-Payload oder eine falsche Abfrage enthält, wenn der Dienst gedrosselt ist oder wenn die Autorisierungseinstellungen falsch konfiguriert sind.

Einheit: Anzahl. Verwenden Sie die Summen-Statistik, um die Gesamtanzahl der aufgetretenen Fehler zu erhalten.

`5XXError`

Beim Ausführen einer GraphQL-Abfrage sind Fehler aufgetreten. Dies kann beispielsweise auftreten, wenn eine Abfrage für ein leeres oder falsches Schema aufgerufen wird. Es kann auch vorkommen, wenn die Amazon Cognito Cognito-Benutzerpool-ID oder AWS Region nicht gültig ist. Alternativ kann dies auch passieren, wenn AWS AppSync bei der Bearbeitung einer Anfrage ein Problem auftritt.

Einheit: Anzahl. Verwenden Sie die Summen-Statistik, um die Gesamtanzahl der aufgetretenen Fehler zu erhalten.

`Latency`

Die Zeit zwischen dem AWS AppSync Empfang einer Anfrage von einem Kunden und der Rückgabe einer Antwort an den Client. Dies beinhaltet nicht die Netzwerklatenz, die auftritt, bis eine Antwort die Endgeräte erreicht hat.

Einheit: Millisekunden Verwenden Sie die Durchschnittsstatistik, um erwartete Latenzen zu bewerten.

`Requests`

Die Anzahl der Anfragen (Abfragen + Mutationen), die alle APIs in Ihrem Konto bearbeitet haben, aufgeschlüsselt nach Region.

Einheit: Anzahl. Die Anzahl aller Anfragen, die in einer bestimmten Region bearbeitet wurden.

`TokensConsumed`

Tokens werden auf der `Requests` Grundlage der Menge an Ressourcen (Verarbeitungszeit und verwendeter Speicherplatz) zugewiesen, die a `Request` verbraucht. Normalerweise `Request` verbraucht jedes Exemplar ein Token. Einem `Request` Benutzer, der große Mengen an Ressourcen verbraucht, werden jedoch nach Bedarf zusätzliche Token zugewiesen.

Einheit: Anzahl. Die Anzahl der Token, die Anfragen zugewiesen wurden, die in einer bestimmten Region verarbeitet wurden.

`NetworkBandwidthOutAllowanceExceeded`

> **Note**
>
> In der AWS AppSync Konsole können Sie auf der Seite mit den Cache-Einstellungen mit der Option Cache Health Metrics diese Cache-bezogene Integritätsmetrik aktivieren.

Die Netzwerkpakete wurden verworfen, weil der Durchsatz das aggregierte Bandbreitenlimit überschritten hat. Dies ist nützlich für die Diagnose von Engpässen in einer Cache-Konfiguration. Daten werden für eine bestimmte API aufgezeichnet, indem die `API_Id` in der Metrik angegeben wird. `appsyncCacheNetworkBandwidthOutAllowanceExceeded`

Einheit: Anzahl. Die Anzahl der Pakete, die nach Überschreitung des Bandbreitenlimits für eine durch die ID angegebene API verworfen wurden.

`EngineCPUUtilization`

> **Note**
>
> In der AWS AppSync Konsole können Sie auf der Seite mit den Cache-Einstellungen mit der Option Cache Health Metrics diese Cache-bezogene Integritätsmetrik aktivieren.

Die CPU-Auslastung (in Prozent), die dem Redis OSS-Prozess zugewiesen ist. Dies ist nützlich für die Diagnose von Engpässen in einer Cache-Konfiguration. Daten werden für eine bestimmte API aufgezeichnet, indem die `API_Id` in der Metrik angegeben wird. `appsyncCacheEngineCPUUtilization`

Einheit: Prozent. Der CPU-Prozentsatz, der derzeit vom Redis OSS-Prozess für eine durch ID angegebene API verwendet wird.

Abonnements in Echtzeit

Alle Metriken werden in einer Dimension ausgegeben: Grafik QLAPIId. Dies bedeutet, dass alle Metriken mit der GraphQL-API IDs gekoppelt sind. Die folgenden Metriken beziehen sich auf GraphQL-Abonnements im Vergleich zu Pure WebSockets:

Liste der Metriken

`ConnectRequests`

Die Anzahl der WebSocket Verbindungsanfragen an AWS AppSync, einschließlich erfolgreicher und erfolgloser Versuche.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtzahl der Verbindungsanfragen zu ermitteln.

`ConnectSuccess`

Die Anzahl der erfolgreichen WebSocket Verbindungen zu AWS AppSync. Verbindungen ohne Abonnement sind möglich.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der erfolgreichen Verbindungen zu erhalten.

`ConnectClientError`

Die Anzahl der WebSocket Verbindungen, die AWS AppSync aufgrund von clientseitigen Fehlern abgelehnt wurden. Dies könnte bedeuten, dass der Dienst gedrosselt ist oder dass die Autorisierungseinstellungen falsch konfiguriert sind.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der clientseitigen Verbindungsfehler zu erhalten.

`ConnectServerError`

Die Anzahl der Fehler, die auf die Verarbeitung von Verbindungen zurückzuführen sind AWS AppSync . Diese Fehler treten normalerweise bei einem unerwarteten serverseitigen Problem auf.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der serverseitigen Verbindungsfehler zu erhalten.

`DisconnectSuccess`

Die Anzahl der erfolgreichen WebSocket Verbindungsabbrüche von AWS AppSync.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der erfolgreichen Verbindungstrennungen zu erhalten.

`DisconnectClientError`

Die Anzahl der Client-Fehler, die auf das Trennen von AWS AppSync Verbindungen zurückzuführen sind WebSocket .

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der Fehler bei Verbindungstrennungen zu erhalten.

`DisconnectServerError`

Die Anzahl der Serverfehler, die auf das Trennen von AWS AppSync Verbindungen zurückzuführen sind WebSocket .

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der Fehler bei Verbindungstrennungen zu erhalten.

`SubscribeSuccess`

Die Anzahl der Abonnements, für die erfolgreich registriert wurden AWS AppSync . WebSocket Es ist möglich, Verbindungen ohne Abonnements zu haben, aber es ist nicht möglich, Abonnements ohne Verbindungen zu haben.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der erfolgreichen Abonnements zu erhalten.

`SubscribeClientError`

Die Anzahl der Abonnements, die von AWS AppSync aufgrund von clientseitigen Fehlern abgelehnt wurden. Dies kann der Fall sein, wenn eine JSON-Payload falsch ist, der Dienst gedrosselt ist oder die Autorisierungseinstellungen falsch konfiguriert sind.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der clientseitigen Abonnementfehler zu erhalten.

`SubscribeServerError`

Die Anzahl der Fehler, die auf die Verarbeitung von AWS AppSync Abonnements zurückzuführen sind. Diese Fehler treten normalerweise bei einem unerwarteten serverseitigen Problem auf.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der serverseitigen Abonnementfehler zu erhalten.

`UnsubscribeSuccess`

Die Anzahl der Abmeldeanfragen, die erfolgreich verarbeitet wurden.

Einheit: Anzahl. Verwenden Sie die Sum-Statistik, um die Gesamtzahl der erfolgreichen Abmeldeanfragen zu ermitteln.

`UnsubscribeClientError`

Die Anzahl der Abmeldeanfragen, die von AWS AppSync aufgrund von clientseitigen Fehlern abgelehnt wurden.

Einheit: Anzahl. Verwenden Sie die Sum-Statistik, um die Gesamtzahl der Fehler bei der clientseitigen Abmeldeanfrage zu ermitteln.

`UnsubscribeServerError`

Die Anzahl der Fehler, die auf die Verarbeitung von Abmeldeanfragen zurückzuführen sind AWS AppSync . Diese Fehler treten normalerweise bei einem unerwarteten serverseitigen Problem auf.

Einheit: Anzahl. Verwenden Sie die Sum-Statistik, um die Gesamtzahl der Fehler bei serverseitigen Abmeldeanfragen zu ermitteln.

`PublishDataMessageSuccess`

Die Anzahl der Abonnementereignismeldungen, die erfolgreich veröffentlicht wurden.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der erfolgreich veröffentlichten Abonnementereignismeldungen zu erhalten.

`PublishDataMessageClientError`

Die Anzahl der Abonnementereignismeldungen, die aufgrund von clientseitigen Fehlern nicht veröffentlicht werden konnten.

`Unit`: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der clientseitigen Fehler bei der Veröffentlichung von Abonnementereignissen zu erhalten.

`PublishDataMessageServerError`

Die Anzahl der Fehler, die AWS AppSync beim Veröffentlichen von Abonnement-Ereignismeldungen entstanden sind. Diese Fehler treten normalerweise bei einem unerwarteten serverseitigen Problem auf.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der serverseitigen Fehler bei der Veröffentlichung von Abonnementereignissen zu erhalten.

`PublishDataMessageSize`

Die Größe der veröffentlichten Abonnementereignismeldungen.

Einheit: Byte.

`ActiveConnections`

Die Anzahl gleichzeitiger WebSocket Verbindungen von Clients zu AWS AppSync in 1 Minute.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der geöffneten Verbindungen zu erhalten.

`ActiveSubscriptions`

Die Anzahl der gleichzeitigen Abonnements von Clients in 1 Minute.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtanzahl der aktiven Abonnements zu erhalten.

`ConnectionDuration`

Die Zeitspanne, in der die Verbindung offen bleibt.

Einheit: Millisekunden. Verwenden Sie die Durchschnittsstatistik, um die Verbindungsdauer auszuwerten.

`OutboundMessages`

Die Anzahl der erfolgreich veröffentlichten Messnachrichten. Eine gemessene Nachricht entspricht 5 kB gelieferter Daten.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtzahl der erfolgreich veröffentlichten Messnachrichten zu ermitteln.

`InboundMessageSuccess`

Die Anzahl der erfolgreich verarbeiteten eingehenden Nachrichten. Jeder Abonnementtyp, der durch eine Mutation aufgerufen wird, generiert eine eingehende Nachricht.

Einheit: Anzahl. Verwenden Sie die Sum-Statistik, um die Gesamtzahl der erfolgreich verarbeiteten eingehenden Nachrichten abzurufen.

`InboundMessageError`

Die Anzahl der eingehenden Nachrichten, die aufgrund ungültiger API-Anfragen nicht verarbeitet werden konnten, z. B. weil die maximale Größe der Abonnementnutzlast von 240 kB überschritten wurde.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtzahl der eingehenden Nachrichten mit API-bezogenen Verarbeitungsfehlern abzurufen.

`InboundMessageFailure`

Die Anzahl der eingehenden Nachrichten, deren Verarbeitung aufgrund von Fehlern von fehlgeschlagen ist. AWS AppSync

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtzahl der eingehenden Nachrichten mit entsprechenden Verarbeitungsfehlern abzurufen AWS AppSync.

`InboundMessageDelayed`

Die Anzahl verzögerter eingehender Nachrichten. Eingehende Nachrichten können verzögert werden, wenn entweder die Quote für eingehende Nachrichten oder die Quote für ausgehende Nachrichten überschritten wird.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtzahl der verspäteten eingehenden Nachrichten zu ermitteln.

`InboundMessageDropped`

Die Anzahl der verworfenen eingehenden Nachrichten. Eingehende Nachrichten können gelöscht werden, wenn entweder die Quote für eingehende Nachrichten oder die Quote für ausgehende Nachrichten überschritten wird.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtzahl der verworfenen eingehenden Nachrichten zu ermitteln.

`InvalidationSuccess`

Die Anzahl der Abonnements, die durch eine Mutation mit erfolgreich für ungültig erklärt (abgemeldet) wurden. `$extensions.invalidateSubscriptions()`

Einheit: Anzahl. Verwenden Sie die Sum-Statistik, um die Gesamtzahl der Abonnements abzurufen, die erfolgreich abgemeldet wurden.

`InvalidationRequestSuccess`

Die Anzahl der erfolgreich verarbeiteten Invalidierungsanfragen.

Einheit: Anzahl. Verwenden Sie die Sum-Statistik, um die Gesamtzahl der erfolgreich verarbeiteten Invalidierungsanforderungen zu ermitteln.

`InvalidationRequestError`

Die Anzahl der Invalidierungsanforderungen, die aufgrund ungültiger API-Anfragen nicht verarbeitet werden konnten.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtzahl der Invalidierungsanfragen mit API-bezogenen Verarbeitungsfehlern zu ermitteln.

`InvalidationRequestFailure`

Die Anzahl der Invalidierungsanforderungen, die aufgrund von Fehlern von nicht verarbeitet werden konnten. AWS AppSync

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtzahl der Invalidierungsanforderungen mit entsprechenden Verarbeitungsfehlern abzurufen AWS AppSync.

`InvalidationRequestDropped`

Die Anzahl der verworfenen Ungültigungsanforderungen, als das Kontingent für Invalidierungsanfragen überschritten wurde.

Einheit: Anzahl. Verwenden Sie die Summenstatistik, um die Gesamtzahl der verworfenen Invalidierungsanfragen zu ermitteln.

Vergleich eingehender und ausgehender Nachrichten

Wenn eine Mutation ausgeführt wird, werden Abonnementfelder mit der @aws_subscribe -Direktive für diese Mutation aufgerufen. Jeder Abonnementaufruf generiert eine eingehende Nachricht. Wenn beispielsweise zwei Abonnementfelder dieselbe Mutation in @aws_subscribe angeben, werden zwei eingehende Nachrichten generiert, wenn diese Mutation aufgerufen wird.

Eine ausgehende Nachricht entspricht 5 kB an Daten, die an WebSocket Clients geliefert werden. Das Senden von 15 kB Daten an 10 Clients führt beispielsweise zu 30 ausgehenden Nachrichten (15 kB * 10 Clients/5 kB pro Nachricht = 30 Nachrichten).

Sie können eine Erhöhung des Kontingents für eingehende oder ausgehende Nachrichten beantragen. Weitere Informationen finden Sie unter AWS AppSync Endpunkte und Kontingente im AWS Allgemeinen Referenzhandbuch und in den Anweisungen zur Beantragung einer Kontingenterhöhung im Servicekontingents-Benutzerhandbuch.

Verbesserte Metriken

Verbesserte Metriken geben detaillierte Daten zur API-Nutzung und -Leistung aus, z. B. Anzahl von AWS AppSync Anfragen und Fehlern, Latenz und Cache-Treffern/Fehlschläge. Alle erweiterten Metrikdaten werden an Ihr CloudWatch Konto gesendet, und Sie können die zu sendenden Datentypen konfigurieren.

> **Note**
>
> Bei der Verwendung erweiterter Messwerte fallen zusätzliche Gebühren an. Weitere Informationen finden Sie in den detaillierten Preisstufen für die Überwachung in den CloudWatchAmazon-Preisen.

Diese Messwerte finden Sie auf verschiedenen Einstellungsseiten in der AWS AppSync Konsole. Auf der Seite mit den API-Einstellungen können Sie im Bereich „Erweiterte Metriken“ die folgenden Elemente aktivieren oder deaktivieren:

Verhalten der Resolver-Metriken: Diese Optionen steuern, wie zusätzliche Metriken für Resolver erfasst werden. Sie können wählen, ob Sie vollständige Request Resolver-Metriken (Metriken, die für alle Resolver in Anfragen aktiviert sind) oder Metriken pro Resolver (Metriken, die nur für Resolver aktiviert sind, bei denen die Konfiguration aktiviert ist) aktivieren möchten. Verfügbar sind die nachfolgend aufgeführten Optionen:

Liste des Verhaltens von Resolver-Metriken

`GraphQL errors per resolver (GraphQLError)`

Die Anzahl der pro Resolver aufgetretenen GraphQL-Fehler.

Metrische Dimension:, `API_Id` `Resolver`

Einheit: Anzahl.

`Requests per resolver (Request)`

Die Anzahl der Aufrufe, die während einer Anfrage erfolgten. Dies wird pro Resolver aufgezeichnet.

Metrische Abmessung:, `API_Id` `Resolver`

Einheit: Anzahl.

`Latency per resolver (Latency)`

Die Zeit, bis ein Resolver-Aufruf abgeschlossen ist. Die Latenz wird in Millisekunden gemessen und pro Resolver aufgezeichnet.

`API_Id`Metrische Dimension:, `Resolver`

Einheit: Millisekunden

`Cache hits per resolver (CacheHit)`

Die Anzahl der Cache-Treffer während einer Anfrage. Dies wird nur ausgegeben, wenn ein Cache verwendet wird. Cache-Treffer werden pro Resolver aufgezeichnet.

Metrische Dimension:, `API_Id` `Resolver`

Einheit: Anzahl.

`Cache misses per resolver (CacheMiss)`

Die Anzahl der Cache-Fehlschläge während einer Anfrage. Dies wird nur ausgegeben, wenn ein Cache verwendet wird. Cache-Fehlschläge werden pro Resolver aufgezeichnet.

Metrische Dimension:, `API_Id` `Resolver`

Einheit: Anzahl.

Verhalten von Datenquellenmetriken: Diese Optionen steuern, wie zusätzliche Metriken für Datenquellen erfasst werden. Sie können wählen, ob Sie Metriken für vollständige Anfragen (Metriken, die für alle Datenquellen in Anfragen aktiviert sind) oder Metriken pro Datenquelle (Metriken, die nur für Datenquellen aktiviert sind, für die die Konfiguration aktiviert ist) aktivieren möchten. Verfügbar sind die nachfolgend aufgeführten Optionen:

Liste des Verhaltens von Datenquellen-Metriken

`Requests per data source (Request)`

Die Anzahl der Aufrufe, die während einer Anfrage erfolgt sind. Anfragen werden pro Datenquelle aufgezeichnet. Wenn vollständige Anfragen aktiviert sind, hat jede Datenquelle ihren eigenen Eintrag in CloudWatch.

Metrische Dimension:`API_Id`, `Datasource`

Einheit: Anzahl.

`Latency per data source (Latency)`

Die Zeit, bis ein Datenquellenaufruf abgeschlossen ist. Die Latenz wird pro Datenquelle aufgezeichnet.

Metrische Dimension:`API_Id`, `Datasource`

Einheit: Millisekunden

`Errors per data source (GraphQLError)`

Die Anzahl der Fehler, die während eines Datenquellenaufrufs aufgetreten sind.

Metrische Dimension:`API_Id`, `Datasource`

Einheit: Anzahl.

Betriebsmetriken: Aktiviert GraphQL-Metriken auf Betriebsebene.

Liste des Verhaltens von Betriebsmetriken

`Requests per operation (Request)`

Die Häufigkeit, mit der eine angegebene GraphQL-Operation aufgerufen wurde.

Metrische Abmessung:`API_Id`, `Operation`

Einheit: Anzahl.

`GraphQL errors per operation (GraphQLError)`

Die Anzahl der GraphQL-Fehler, die während eines angegebenen GraphQL-Vorgangs aufgetreten sind.

Metrische Abmessung:, `API_Id` `Operation`

Einheit: Anzahl.

CloudWatch Protokolle

Sie können zwei Arten von Protokollierung auf jeder neuen oder vorhandenen GraphQL-API konfigurieren: auf dem Anforderungslevel und auf dem Feldlevel.

Protokolle auf Anforderungsebene

Wenn die Protokollierung auf Anforderungsebene (Ausführlichen Inhalt einbeziehen) konfiguriert ist, werden die folgenden Informationen protokolliert:

- Die Anzahl der verbrauchten Token
- HTTP-Header für Anforderungen und Antworten
- Die GraphQL-Abfrage, die in der Anfrage ausgeführt wird
- Die allgemeine Zusammenfassung der Operation

- Neue und bestehende GraphQL-Abonnements, die registriert sind

Protokolle auf Feldebene

Wenn die Protokollierung auf Feldebene konfiguriert ist, werden die folgenden Informationen protokolliert:

- Generierte Anforderungszuordnung mit Quelle und Argumenten für jedes Feld
- Die transformierte Antwortzuordnung für jedes Feld, die die Daten enthält, die sich aus der Auflösung dieses Felds ergeben
- Verfolgen von Informationen für jedes Feld

AWS AppSync Verwaltet die Protokolle, wenn Sie die CloudWatch Protokollierung aktivieren. Der Prozess umfasst das Erstellen von Protokollgruppen und -Streams sowie die Meldung an Protokoll-Streams mit diesen Protokollen.

Wenn Sie die Protokollierung auf einer GraphQL-API aktivieren und Anfragen stellen, AWS AppSync erstellt eine Protokollgruppe und Protokollstreams unter der Protokollgruppe. Die Protokollgruppe wird nach dem `/aws/appsync/apis/{graphql_api_id}`-Format benannt. In jeder Protokollgruppe werden die Protokolle weiter in Protokoll-Streams unterteilt. Diese werden anhand der Last Event Time (Letzte Ereigniszeit) sortiert, das heißt anhand des Zeitpunkts, zu dem die protokollierten Daten gemeldet werden.

Jedes Protokollereignis ist mit dem X-amzn- RequestId der Anfrage gekennzeichnet. Auf diese Weise können Sie Protokollereignisse filtern CloudWatch , um alle protokollierten Informationen zu dieser Anfrage abzurufen. Sie können das RequestId aus den Antwortheadern jeder AWS AppSync GraphQL-Anfrage abrufen.

Die Feld-Level-Protokollierung wird mit den folgenden Protokollebenen konfiguriert:

- Keine — Es werden keine Protokolle auf Feldebene erfasst.
- Fehler — Protokolliert die folgenden Informationen nur für die Felder, die zur Fehlerkategorie gehören:
    - Der Fehlerabschnitt in der Serverantwort
    - Feld-Level-Fehler
    - Die generierten request/response Funktionen, die für Fehlerfelder behoben wurden

- Info — Protokolliert die folgenden Informationen nur für die Felder, die zu den Kategorien Info und Fehler gehören:
    - Nachrichten auf Informationsebene
    - Die Benutzernachrichten, die über und gesendet wurden `$util.log.info` `console.log`
    - Nachverfolgungs- und Zuordnungsprotokolle auf Feldebene werden nicht angezeigt.
    - Wenn die Protokollierung auf Feldebene auf `INFO` oder höher gesetzt ist und ausführliche Inhalte enthalten sind, werden die Protokollmeldungen der transformierten Mapping-Vorlage AWS AppSync hinzugefügt. Dies enthält alle Informationen, die der transformierten Zuordnungsvorlage hinzugefügt wurden, oder die Ausgabe des vom Resolver oder der Funktion ausgeführten JavaScript Codes. Sie sollte nicht verwendet werden, wenn Sie beabsichtigen, vertrauliche Informationen wie Passwörter oder Autorisierungsheader an nachgeschaltete Datenquellen zu senden und diese Informationen nicht in Ihren Protokollen haben möchten.
- Debug — Protokolliert die folgenden Informationen nur für die Felder, die in den Kategorien Debug, Info und Fehler enthalten sind:
    - Meldungen auf Debug-Ebene
    - Die Benutzernachrichten, die über`$util.log.info`,, und `$util.log.debug` gesendet wurden `console.log` `console.debug`
    - Nachverfolgungs- und Zuordnungsprotokolle auf Feldebene werden nicht angezeigt.
- Alle — Protokolliert die folgenden Informationen für alle Felder in der Abfrage:
    - Feld-Level-Rückverfolgungsinformationen
    - Die generierten request/response Funktionen, die für jedes Feld aufgelöst wurden

Vorteile der Überwachung

Sie können die Protokollierung und Metriken verwenden, um Ihre GraphQL-Abfragen zu identifizieren, Fehler darin zu beheben und sie zu optimieren. Diese werden Ihnen beispielsweise dabei helfen, Latenzprobleme zu debuggen und zwar mithilfe der Rückverfolgungsinformationen, die für jedes Feld in der Abfrage protokolliert werden. Um dies zu demonstrieren, nehmen Sie einmal an, dass Sie einen oder mehrere Resolver in einer GraphQL-Abfrage verwenden. Ein Beispiel für einen Feldvorgang in CloudWatch Logs könnte wie folgt aussehen:

```
{
    "path": [
        "singlePost",
```

```
        "authors",
        0,
        "name"
    ],
    "parentType": "Post",
    "returnType": "String!",
    "fieldName": "name",
    "startOffset": 416563350,
    "duration": 11247
}
```

Dies kann einem GraphQL-Schema entsprechen, das wie folgt aussieht:

```
type Post {
  id: ID!
  name: String!
  authors: [Author]
}

type Author {
  id: ID!
  name: String!
}

type Query {
  singlePost(id:ID!): Post
}
```

In den vorherigen Protokollergebnissen zeigt path ein einzelnes Element in Ihren Daten, das beim Ausführen einer Abfrage zurückgegeben wurde, mit dem Namen`singlePost()`. In diesem Beispiel steht es für das Namensfeld am ersten Index (0). Das StartOffset gibt einen Offset vom Start des GraphQL-Abfragevorgangs an. Die Dauer ist die Gesamtzeit für die Auflösung des Felds. Diese Werte können nützlich sein, um herauszufinden, warum Daten aus einer bestimmten Datenquelle möglicherweise langsamer als erwartet ausgeführt werden oder ob ein bestimmtes Feld die gesamte Abfrage drosselt. Sie könnten sich beispielsweise dafür entscheiden, den bereitgestellten Durchsatz für eine Amazon DynamoDB-Tabelle zu erhöhen oder ein bestimmtes Feld aus einer Abfrage zu entfernen, wodurch der gesamte Vorgang schlecht ausgeführt wird.

AWS AppSync Generiert seit dem 8. Mai 2019 Protokollereignisse als vollständig strukturiertes JSON. Dies kann Ihnen helfen, Protokollanalysedienste wie CloudWatch Logs Insights und Amazon OpenSearch Service zu verwenden, um die Leistung Ihrer GraphQL-Anfragen und die

Nutzungsmerkmale Ihrer Schemafelder zu verstehen. Auf diese Weise können Sie z. B. Resolver mit hohen Wartezeiten identifizieren, die möglicherweise die Ursache eines Leistungsproblems sind. Zudem können Sie die am häufigsten und am seltesten verwendeten Felder in Ihrem Schema identifizieren und die Auswirkungen von veralteten GraphQL-Feldern bewerten.

Konflikterkennung und Synchronisierungsprotokollierung

Wenn bei einer AWS AppSync API die Protokollierung in CloudWatch Logs so konfiguriert ist, dass die Protokollebene des Field Resolvers auf Alle gesetzt ist, werden Informationen zur Konflikterkennung und -lösung an die Protokollgruppe ausgegeben. AWS AppSync Dies bietet einen detaillierten Einblick, wie die AWS AppSync API auf einen Konflikt reagiert hat. Um Ihnen bei der Interpretation der Antwort zu helfen, sind die folgenden Informationen in den Protokollen enthalten:

Liste der Metriken

`conflictType`

Gibt an, ob ein Konflikt aufgrund einer fehlenden Versionsübereinstimmung oder der vom Kunden bereitgestellten Bedingung aufgetreten ist.

`conflictHandlerConfigured`

Gibt den Conflict Handler an, der zum Zeitpunkt der Anforderung auf dem Resolver konfiguriert war.

`message`

Enthält Informationen darüber, wie der Konflikt erkannt und gelöst wurde.

`syncAttempt`

Die Anzahl der Versuche, die der Server unternommen hat, um die Daten zu synchronisieren, bevor die Anforderung letztlich abgelehnt wurde.

`data`

Wenn der Konflikthandler konfiguriert ist`Automerge`, wird dieses Feld ausgefüllt, um anzuzeigen, welche Entscheidung `Automerge` für jedes Feld getroffen wurde. Die Aktionen können sein:

- ABGELEHNT — `Automerge` When lehnt den eingehenden Feldwert zugunsten des Werts auf dem Server ab.
- HINZUGEFÜGT — Wenn das eingehende Feld `Automerge` hinzugefügt wird, weil auf dem Server noch kein Wert vorhanden ist.

- ANGEFÜGT — Wenn die eingehenden Werte `Automerge` an die Werte für die Liste angehängt werden, die auf dem Server vorhanden ist.
- **`Automerge`**MERGED — Wenn die eingehenden Werte mit den Werten für das Set zusammengeführt werden, das auf dem Server vorhanden ist.

Verwenden Sie Token-Zählungen, um Ihre Anfragen zu optimieren

Anfragen, die weniger als oder gleich 1.500 KB-Sekunden Arbeitsspeicher und vCPU-Zeit verbrauchen, wird ein Token zugewiesen. Anfragen mit einem Ressourcenverbrauch von mehr als 1.500 KB-Sekunden erhalten zusätzliche Token. Wenn eine Anfrage beispielsweise 3.350 KB-Sekunden verbraucht, werden der Anfrage drei Token (AWS AppSync auf die nächste Ganzzahl aufgerundet) zugewiesen. AWS AppSync Ordnet dem in Ihrem Konto standardmäßig maximal 5.000 oder 10.000 Anforderungstoken pro Sekunde zu, abhängig von der Region, APIs in der sie bereitgestellt wird. AWS Wenn Sie APIs jeweils durchschnittlich zwei Token pro Sekunde verwenden, sind Sie auf 2.500 bzw. 5.000 Anfragen pro Sekunde beschränkt. Wenn Sie mehr Token pro Sekunde als die zugewiesene Menge benötigen, können Sie eine Anfrage einreichen, um das Standardkontingent für die Rate der Anforderungstoken zu erhöhen. Weitere Informationen finden Sie unter AWS AppSync Endpunkte und Kontingente im Allgemeine AWS-Referenz Handbuch und Beantragung einer Kontingenterhöhung im Servicekontingents-Benutzerhandbuch.

Eine hohe Token-Anzahl pro Anfrage könnte darauf hindeuten, dass die Möglichkeit besteht, Ihre Anfragen zu optimieren und die Leistung Ihrer API zu verbessern. Zu den Faktoren, die Ihre Token-Anzahl pro Anfrage erhöhen können, gehören:

- Die Größe und Komplexität Ihres GraphQL-Schemas.
- Die Komplexität von Vorlagen für die Zuordnung von Anfragen und Antworten.
- Die Anzahl der Resolver-Aufrufe pro Anfrage.
- Die Menge der von Resolvern zurückgegebenen Daten.
- Die Latenz von Downstream-Datenquellen.
- Schema- und Abfrageentwürfe, die aufeinanderfolgende Datenquellenaufrufe erfordern (im Gegensatz zu parallel oder gebündelten Aufrufen).
- Konfiguration der Protokollierung, insbesondere Protokollinhalte auf Feldebene und ausführliche Protokollinhalte.

> **Note**
>
> Zusätzlich zu den AWS AppSync Metriken und Protokollen können Clients über den Antwort-Header auf die Anzahl der in einer Anfrage verbrauchten Token zugreifen. `x-amzn-appsync-TokensConsumed`

Größenbeschränkungen für Protokolle

Wenn die Protokollierung aktiviert wurde, AWS AppSync werden standardmäßig bis zu 1 MB an Protokollen pro Anfrage gesendet. Protokolle, die diese Größe überschreiten, werden gekürzt. Um die Protokollgröße zu reduzieren, wählen Sie die `ERROR` Protokollierungsebene für Protokolle auf Feldebene und deaktivieren Sie die `VERBOSE` Protokollierung oder deaktivieren Sie die Protokolle auf Feldebene vollständig, falls sie nicht benötigt werden. Als Alternative zur `ALL` Protokollebene können Sie Enhanced Metrics verwenden, um Metriken zu bestimmten Resolvern, Datenquellen oder GraphQL-Vorgängen abzurufen, oder die von bereitgestellten Protokollierungsdienstprogramme verwenden, um nur die erforderlichen Informationen AppSync zu protokollieren.

Referenz zum Protokolltyp

RequestSummary

- requestId: Eindeutige Bezeichnung für die Anforderung.
- graphQLAPIId: ID der GraphQL-API, die die Anfrage stellt.
- statusCode: Antwort auf den HTTP-Statuscode.
- Latenz: End-to-end Latenz der Anfrage in Nanosekunden als Ganzzahl.

```
{
     "logType": "RequestSummary",
     "requestId": "dbe87af3-c114-4b32-ae79-8af11f3f96f1",
     "graphQLAPIId": "pmo28inf75eepg63qxq4ekoeg4",
     "statusCode": 200,
     "latency": 242000000
}
```

ExecutionSummary

- requestId: Eindeutige Bezeichnung für die Anforderung.

- graphQLAPIId: ID der GraphQL-API, die die Anfrage stellt.
- startTime: Der Startzeitstempel der GraphQL-Verarbeitung für die Anfrage im RFC 3339-Format.
- EndTime: Der Endzeitstempel der GraphQL-Verarbeitung für die Anfrage im RFC 3339-Format.
- Dauer: Die gesamte verstrichene GraphQL-Verarbeitungszeit in Nanosekunden als Ganzzahl.
- Version: Die Schemaversion von. ExecutionSummary
- Parsing:
    - startOffset: Der Start-Offset für das Parsen in Nanosekunden relativ zum Aufruf als Ganzzahl.
    - duration: Die für die Analyse aufgewendete Zeit in Nanosekunden als Ganzzahl.
- Validierung:
    - startOffset: Der Start-Offset für die Validierung, in Nanosekunden, relativ zum Aufruf, als Ganzzahl.
    - duration: Die für die Validierung aufgewendete Zeit in Nanosekunden als Ganzzahl.

```
{
    "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"
}
```

Nachverfolgung

- requestId: Eindeutige Bezeichnung für die Anforderung.
- graphQLAPIId: ID der GraphQL-API, die die Anfrage stellt.

- startOffset: Der Start-Offset für die Feldauflösung in Nanosekunden relativ zum Aufruf als Ganzzahl.
- duration: Die für die Auflösung des Felds aufgewendete Zeit in Nanosekunden als Ganzzahl.
- fieldName: Der Name des Feldes, das aufgelöst wird.
- parentType: Der übergeordnete Typ des Feldes, das aufgelöst wird.
- returnType: Der Rückgabetyp des Feldes, das aufgelöst wird.
- path: Eine Liste von Pfadsegmenten, die am Stamm der Antwort beginnt und mit dem aufzulösenden Feld endet.
- resolverArn: Der ARN des Resolvers, der für die Feldauflösung verwendet wird. Möglicherweise nicht in verschachtelten Feldern vorhanden.

```
{
    "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"
}
```

Analysieren Sie Ihre Logs mit Logs Insights CloudWatch

Es folgen Beispiele für Abfragen, die Sie ausführen können, um umsetzbare Einblicke in die Leistung und den Zustand Ihrer GraphQL-Operationen zu erhalten. Diese Beispiele sind als Beispielabfragen in der CloudWatch Logs Insights-Konsole verfügbar. Wählen Sie in der CloudWatchKonsole Logs Insights, wählen Sie die AWS AppSync Protokollgruppe für Ihre GraphQL-API aus und wählen Sie dann AWS AppSync Abfragen unter Beispielabfragen aus.

Die folgende Abfrage gibt die 10 häufigsten GraphQL-Anfragen mit der maximalen Anzahl verbrauchter Token zurück:

```
filter @message like "Tokens Consumed"
| parse @message "* Tokens Consumed: *" as requestId, tokens
| sort tokens desc
| display requestId, tokens
| limit 10
```

Die folgende Abfrage gibt die 10 wichtigsten Resolver mit maximaler Latenz zurück:

```
  fields resolverArn, duration
| filter logType = "Tracing"
| limit 10
| sort duration desc
```

Die folgende Abfrage gibt die am häufigsten aufgerufenen Resolver zurück:

```
  fields ispresent(resolverArn) as isRes
| stats count() as invocationCount by resolverArn
| filter isRes and logType = "Tracing"
| limit 10
| sort invocationCount desc
```

Die folgende Abfrage gibt Resolver mit den meisten Fehlern in Zuweisungsvorlagen zurück:

```
  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
```

Die folgende Abfrage gibt Resolver-Latenzstatistiken zurück:

```
  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
```

Die folgende Abfrage gibt Feldlatenzstatistiken zurück:

```
  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
```

Die Ergebnisse von CloudWatch Logs Insights-Abfragen können in CloudWatch Dashboards exportiert werden.

Analysieren Sie Ihre Logs mit Service OpenSearch

Sie können Ihre AWS AppSync Protokolle mit Amazon OpenSearch Service durchsuchen, analysieren und visualisieren, um Leistungsengpässe und Ursachen für Betriebsprobleme zu identifizieren. Sie können Resolver mit der maximalen Latenz und Fehlern identifizieren. Darüber hinaus können Sie OpenSearch Dashboards verwenden, um Dashboards mit aussagekräftigen Visualisierungen zu erstellen. OpenSearch Dashboards ist ein Open-Source-Tool zur Datenvisualisierung und -erkundung, das im Service verfügbar ist. OpenSearch Mithilfe von OpenSearch Dashboards können Sie die Leistung und den Zustand Ihrer GraphQL-Operationen kontinuierlich überwachen. Sie können beispielsweise Dashboards erstellen, um die P90-Latenz Ihrer GraphQL-Anfragen zu visualisieren und die P90-Latenzen jedes Resolvers detailliert zu untersuchen.

Wenn Sie OpenSearch Service verwenden, verwenden Sie „cwl*“ als Filtermuster für die Suche nach Indizes. OpenSearch OpenSearch Der Dienst indexiert die aus Logs gestreamten CloudWatch Protokolle mit dem Präfix „cwl-“. Um AWS AppSync API-Protokolle von anderen an OpenSearch Service gesendeten CloudWatch Protokollen zu unterscheiden, empfehlen wir, Ihrer Suche einen zusätzlichen Filterausdruck von `graphQLAPIID.keyword=`*`YourGraphQLAPIID`* hinzuzufügen.

Migration des Protokollformats

Protokollereignisse, die am oder nach dem 8. Mai 2019 AWS AppSync generiert werden, sind als vollständig strukturiertes JSON formatiert. Um GraphQL-Anfragen vor dem 8. Mai 2019 zu analysieren, können Sie ältere Protokolle mithilfe eines im GitHub Beispiel verfügbaren Skripts zu vollständig strukturiertem JSON migrieren. Wenn Sie das Protokollformat vor dem 8. Mai 2019 verwenden müssen, erstellen Sie ein Support-Ticket mit den folgenden Einstellungen: Legen Sie Type (Typ) auf Account Management (Kontoverwaltung) und dann Category (Kategorie) auf General Account Question (Allgemeine Frage zum Konto) fest.

Sie können auch Metrikfilter verwenden CloudWatch , um Protokolldaten in numerische CloudWatch Metriken umzuwandeln, sodass Sie sie grafisch darstellen oder einen Alarm auslösen können.

Wird verwendet AWS X-Ray , um Anfragen zu verfolgen in AWS AppSync

Sie können AWS X-Raydamit Anfragen verfolgen, während sie in AWS AppSync ausgeführt werden. Sie können X-Ray AWS AppSync in allen AWS Regionen verwenden, in denen X-Ray verfügbar ist. X-Ray gibt Ihnen einen detaillierten Überblick über eine gesamte GraphQL-Anfrage. Auf diese Weise können Sie Latenzen in Ihren APIs und den ihnen zugrunde liegenden Resolvern und Datenquellen analysieren. Sie können eine X-Ray-Servicekarte verwenden, um die Latenz einer Anfrage anzuzeigen, einschließlich aller AWS Dienste, die in X-Ray integriert sind. Sie können auch Sampling-Regeln konfigurieren, um X-Ray gemäß den von Ihnen angegebenen Kriterien mitzuteilen, welche Anfragen aufgezeichnet werden sollen und mit welchen Abtastraten.

Weitere Informationen zur Probennahme in X-Ray finden Sie unter Konfiguration von Sampling-Regeln in der AWS X-Ray Konsole.

Einrichtung und Konfiguration

Sie können X-Ray-Tracing für eine GraphQL-API über die AWS AppSync Konsole aktivieren.

1. Melden Sie sich bei der Konsole an AWS AppSync .
2. Wählen Sie im Navigationsbereich die Option Settings (Einstellungen) aus.
3. Aktivieren Sie unter X-Ray die Option Enable X-Ray (X-Ray aktivieren).
4. Wählen Sie Save (Speichern) aus. X-Ray-Tracing ist jetzt für Ihre API aktiviert.

Wenn Sie das AWS CLI oder verwenden AWS CloudFormation, können Sie das X-Ray-Tracing auch aktivieren, wenn Sie eine neue AWS AppSync API erstellen oder eine bestehende AWS AppSync API aktualisieren, indem Sie die `xrayEnabled` Eigenschaft auf `true` setzen.

Wenn X-Ray Tracing für eine AWS AppSync API aktiviert ist, wird in Ihrem Konto automatisch eine AWS Identity and Access Management serviceverknüpfte Rolle mit den entsprechenden Berechtigungen erstellt. Dies ermöglicht es AWS AppSync , Spuren auf sichere Weise an X-Ray zu senden.

Verfolgen Sie Ihre API mit X-Ray

Sampling

Mithilfe von Samplingregeln können Sie die Menge der von Ihnen in AWS AppSync aufgezeichneten Daten steuern und das Samplingverhalten bei laufendem Betrieb ändern, ohne Ihren Code ändern oder neu implementieren zu müssen. Diese Regel führt beispielsweise ein Sampling von Anforderungen an die GraphQL-API mit der API-ID `3n572shhcpfokwhdnq1ogu59v6` durch.

- Regelname – `test-sample`
- Priorität – `10`
- Reservoirgröße – `10`
- Bestimmtes Zeitintervall – `10`
- Service-Name – `*`
- Servicetyp – `AWS::AppSync::GraphQLAPI`
- HTTP-Methode – `*`
- Ressourcen-ARN – `arn:aws:appsync:us-west-2:123456789012:apis/3n572shhcpfokwhdnq1ogu59v6`
- Host – `*`

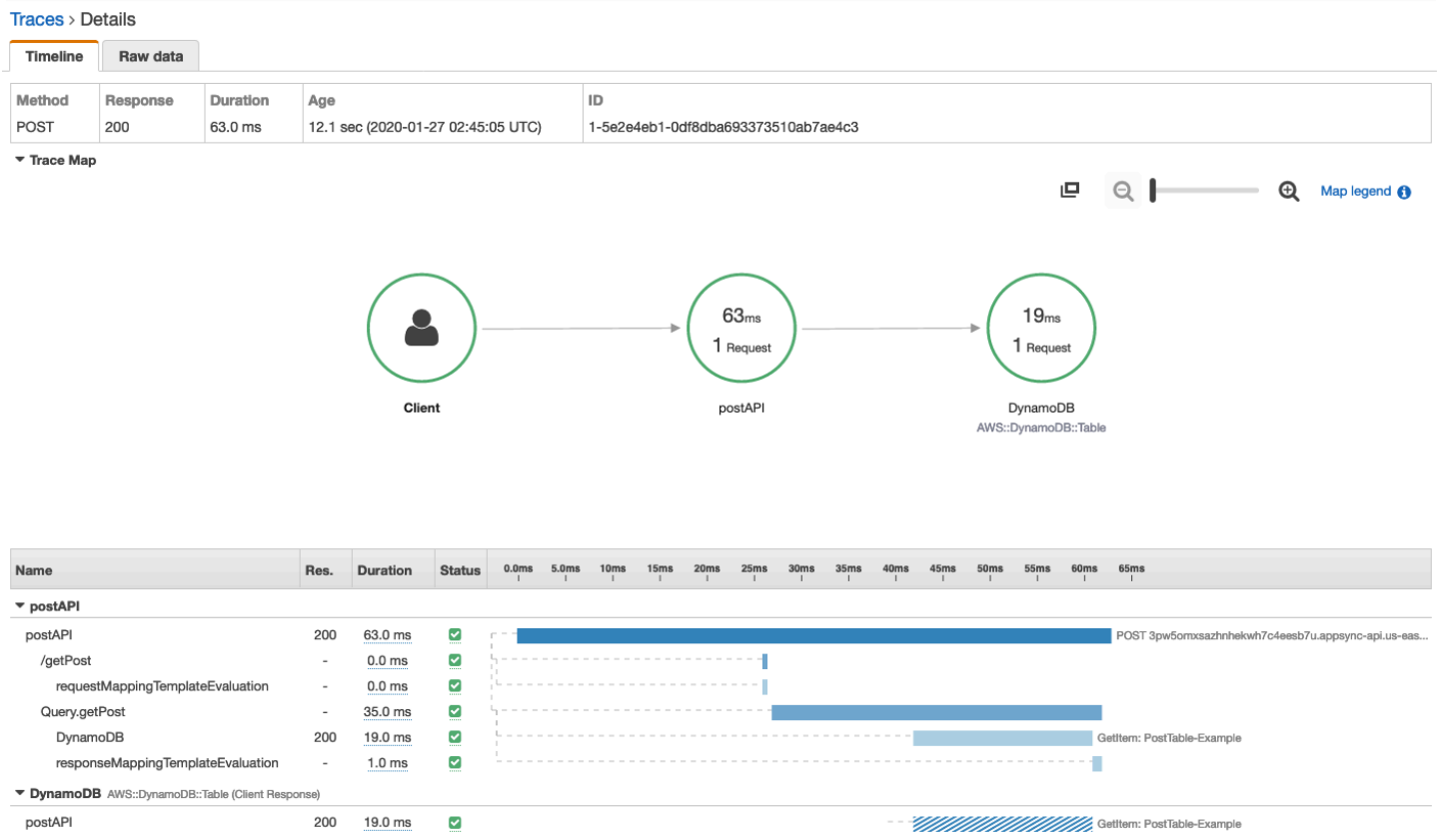
Grundlegendes zu Ablaufverfolgungen

Wenn Sie X-Ray Tracing für Ihre GraphQL-API aktivieren, können Sie die X-Ray-Trace-Detailseite verwenden, um detaillierte Latenzinformationen zu Anfragen an Ihre API zu überprüfen. Das folgende Beispiel zeigt die Ablaufverfolgungsansicht zusammen mit der Servicezuordnung für diese spezielle Anforderung. Die Anfrage wurde an eine API gestellt, die `postAPI` mit einem Post-Typ aufgerufen wurde und deren Daten in einer Amazon DynamoDB-Tabelle namens enthalten sind. `PostTable-Example`

Das folgende Ablaufverfolgungsbild entspricht der folgenden GraphQL-Abfrage:

```
query getPost {
    getPost(id: "1") {
      id
      title
    }
}
```

Der Resolver für die `getPost` Abfrage verwendet die zugrunde liegende DynamoDB-Datenquelle. Die folgende Trace-Ansicht zeigt den Aufruf von DynamoDB sowie die Latenzen verschiedener Teile der Abfrageausführung:



- Im vorhergehenden Bild repräsentiert `/getPost` den vollständigen Pfad zu dem Element, das aufgelöst wird. Da es sich in diesem Fall bei `getPost` um ein Feld für den Stammtyp `Query` handelt, wird es direkt nach dem Stamm des Pfades angezeigt.
- `requestMappingTemplateEvaluation` repräsentiert die Zeit, die von AWS AppSync für die Auswertung der Anforderungszuweisungsvorlage für dieses Element in der Abfrage aufgewendet wurde.
- `Query.getPost` repräsentiert einen Typ und ein Feld (im Format `Type.field`). Es kann mehrere Untersegmente enthalten, abhängig von der Struktur der API und der Anforderung, die verfolgt wird.
  - `DynamoDB` repräsentiert die Datenquelle, die mit diesem Resolver verknüpft ist. Es enthält die Latenz für den Netzwerkaufruf an DynamoDB zur Auflösung des Felds.
  - `responseMappingTemplateEvaluation` repräsentiert die Zeit, die von AWS AppSync für die Auswertung der Antwortzuweisungsvorlage für dieses Element in der Abfrage aufgewendet wurde.

Wenn Sie Traces in X-Ray anzeigen, können Sie zusätzliche Kontext- und Metadateninformationen zu den Untersegmenten im AWS AppSync Segment abrufen, indem Sie die Untersegmente auswählen und die Detailansicht erkunden.

Beachten Sie bei bestimmten tief verschachtelten oder komplexen Abfragen, dass das von an X-Ray gelieferte Segment größer sein AWS AppSync kann als die maximale Größe, die für Segmentdokumente zulässig ist, wie in AWS X-Ray Segmentdokumente definiert. X-Ray zeigt keine Segmente an, die den Grenzwert überschreiten.

Protokollieren von AWS AppSync API-Aufrufen mit AWS CloudTrail

AWS AppSync ist in einen Dienst integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in ausgeführt wurden AWS AppSync. CloudTrail erfasst alle API-Aufrufe AWS AppSync als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der AWS AppSync Konsole und von Codeaufrufen an die AWS AppSync APIs. Anhand der gesammelten Informationen können Sie ermitteln, CloudTrail an wen die Anfrage gestellt wurde AWS AppSync, die IP-Adresse des Anfragenden, wer die Anfrage gestellt hat, wann die Anfrage gestellt wurde, und weitere Informationen.

Sie können einen Trail erstellen, um die kontinuierliche Übermittlung von CloudTrail Ereignissen an einen Amazon Simple Storage Service (Amazon S3) -Bucket zu ermöglichen, einschließlich Ereignissen für AWS AppSync. Wenn Sie keinen Trail konfigurieren, können Sie trotzdem die neuesten Ereignisse in der CloudTrail Konsole anzeigen.

Weitere Informationen zu CloudTrail finden Sie im AWS CloudTrail Benutzerhandbuch.

AWS AppSync Informationen in CloudTrail

CloudTrail ist in Ihrem AWS Konto aktiviert, wenn Sie das Konto erstellen. In der CloudTrail Konsole im Event-Verlauf können Sie aktuelle Ereignisse in Ihrem AWS Konto ansehen, suchen und herunterladen. Weitere Informationen finden Sie im AWS CloudTrail Benutzerhandbuch unter Ereignisse mit CloudTrail Ereignisverlauf anzeigen.

Für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem AWS Konto, einschließlich der Ereignisse für AWS AppSync, erstellen Sie einen Trail. Wenn Sie einen Trail in der Konsole anlegen, gilt dieser für alle AWS -Regionen. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3 S3-Bucket. Darüber hinaus können Sie andere AWS Dienste konfigurieren, um die in den CloudTrail Protokollen

gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie in folgenden Themen im AWS CloudTrail -Benutzerhandbuch:

- Erstellen Sie einen Trail für Ihr AWS Konto
- AWS Serviceintegrationen mit Protokollen CloudTrail
- Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen
- Empfangen von CloudTrail Protokolldateien von mehreren Konten

CloudTrail protokolliert alle AWS AppSync API-Operationen. Aufrufe von, und `ListResolvers` APIs generieren beispielsweise Einträge in den CloudTrail Protokolldateien. `CreateGraphqlApi` `CreateDataSource` Diese und andere Operationen sind in der AWS AppSync API-Referenz dokumentiert.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen helfen Ihnen beim Bestimmen der Folgenden Elemente:

- Ob die Anfrage mit Root- oder AWS Identity and Access Management (IAM-) Benutzeranmeldedaten gestellt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie unter CloudTrail UserIdentity Element im AWS CloudTrail Benutzerhandbuch.

AWS AppSync Datenereignisse in CloudTrail

Datenereignisse liefern Informationen über die Ressourcenoperationen, die auf oder in einer Ressource ausgeführt werden (z. B. Lesen oder Schreiben in ein Amazon-S3-Objekt). Sie werden auch als Vorgänge auf Datenebene bezeichnet. Datenereignisse sind oft Aktivitäten mit hohem Volume. Protokolliert standardmäßig CloudTrail keine Datenereignisse. Der CloudTrail Ereignisverlauf zeichnet keine Datenereignisse auf.

Für Datenereignisse werden zusätzliche Gebühren fällig. Weitere Informationen zur CloudTrail Preisgestaltung finden Sie unter AWS CloudTrail Preisgestaltung.

Sie können Datenereignisse für den `AWS::AppSync::GraphQLApi` Ressourcentyp mithilfe der CloudTrail Konsole oder CloudTrail API-Operationen protokollieren (dazu gehören Abfrage-, Mutations- und Abonnementvorgänge, Verbindungsvorgänge mit Ihrem WebSocket Echtzeit-Endpunkt, aber keine Nachrichten, die über Ihren WebSocket Echtzeit-Endpunkt gesendet werden). AWS CLI Weitere Informationen zum Protokollieren von Datenereignissen finden Sie unter Protokollieren von Datenereignissen mit dem AWS Management Console und Protokollieren von Datenereignissen mit dem AWS Command Line Interface im AWS CloudTrail -Benutzerhandbuch.

In der folgenden Tabelle ist der AWS AppSync Ressourcentyp aufgeführt, für den Sie Datenereignisse protokollieren können. In der Spalte Datenereignistyp (Konsole) wird der Wert angezeigt, der aus der Liste Datenereignistyp in der CloudTrail Konsole ausgewählt werden kann. In der Wertspalte resources.type wird der `resources.type` Wert angezeigt, den Sie bei der Konfiguration erweiterter Event-Selektoren mithilfe von oder angeben würden. AWS CLI CloudTrail APIs In der CloudTrail Spalte APIs Protokollierte Daten werden die API-Aufrufe angezeigt, die CloudTrail für den Ressourcentyp protokolliert wurden.

Typ des Datenereignisses (Konsole)	resources.type-Wert	Daten, die APIs protokolliert wurden CloudTrail
AppSync GraphQL	`AWS::AppSync::GraphQLApi`	GraphQL

Sie können erweiterte Event-Selektoren so konfigurieren, dass sie nach den Feldern `eventName`, `readOnly` und `resources.ARN` filtern, sodass nur die Ereignisse protokolliert werden, die für Sie wichtig sind. Weitere Informationen zu diesen Feldern finden Sie unter AdvancedFieldSelector in der API-Referenz zu AWS CloudTrail

```
[
  {
    "name": "Only 1 AppSync API",
    "fieldSelectors": [
      {
        "field": "eventCategory",
        "equals": [
          "Data"
        ]
      },
      {
```

```
        "field": "resources.type",
        "equals": [
          "AWS::AppSync::GraphQLApi"
        ]
      },
      {
        "field": "resources.ARN",
        "equals": [
          "arn:aws:appsync:us-east-1:111122223333:apis/YourGraphQLApiId"
        ]
      }
    ]
  }
]
```

Grundlegendes zu AWS AppSync Einträgen in Protokolldateien

CloudTrail übermittelt Ereignisse als Protokolldateien, die einen oder mehrere Protokolleinträge enthalten. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält Informationen über den angeforderten Vorgang, das Datum und die Uhrzeit des Vorgangs, die Anforderungsparameter usw. Da es sich bei diesen Protokolldateien nicht um einen geordneten Stack-Trace der öffentlichen API-Aufrufe handelt, werden sie nicht in einer bestimmten Reihenfolge angezeigt.

> **Note**
>
> Das `requestID` ist keine verbindliche eindeutige ID für Protokolle, die von AWS AppSync gesendet werden. Die `requestID` kann vom Client überschrieben werden. Daher sollten Sie Vorsicht walten lassen, wenn Sie Entscheidungen auf der Grundlage dieser Informationen treffen.

Der folgende CloudTrail Beispielprotokolleintrag veranschaulicht den `CreateApiKey` Vorgang.

```
{
  "Records": [{
    "eventVersion": "1.05",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "A1B2C3D4E5F6G7EXAMPLE",
      "arn": "arn:aws:iam::111122223333:user/Alice",
```

```
      "accountId": "111122223333",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "userName": "diego_ramirez"
    },
    "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"
    }
  ]
}
```

Der folgende CloudTrail Beispielprotokolleintrag veranschaulicht den `ListApiKeys` Vorgang.

```
{
  "Records": [{
    "eventVersion": "1.05",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "A1B2C3D4E5F6G7EXAMPLE",
      "arn": "arn:aws:iam::111122223333:user/diego_ramirez",
      "accountId": "111122223333",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "userName": "diego_ramirez"
    },
    "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"
    }
  ]
}
```

Der folgende CloudTrail Beispielprotokolleintrag veranschaulicht den `DeleteApiKey` Vorgang.

```
{
  "Records": [{
    "eventVersion": "1.05",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "A1B2C3D4E5F6G7EXAMPLE",
      "arn": "arn:aws:iam::111122223333:user/diego_ramirez",
      "accountId": "111122223333",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "userName": "diego_ramirez"
    },
    "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"
    }
  ]
}
```

Der folgende CloudTrail Beispielprotokolleintrag zeigt eine erfolgreiche GraphQL-Mutation, die mit einem benutzerdefinierten Lambda-Funktionsautorisierer autorisiert wurde.

```
{
  "eventVersion": "1.10",
    "userIdentity": {
      "type": "Unknown"
    },
    "eventTime": "2024-11-06T15:42:30Z",
    "eventSource": "appsync.amazonaws.com",
    "eventName": "GraphQL",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "15.248.1.214",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:131.0)
 Gecko/20100101 Firefox/131.0",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData": {
      "operationName": "MyMutation",
      "authType": [
        "AWS_LAMBDA"
      ],
      "fieldAuthorizationResults": {
        "deniedFields": []
      }
```

```
    },
    "requestID": "c2d3768b-3446-40a1-bd95-8399fe776f96",
    "eventID": "21568be1-a1a8-4f43-b978-63cb4cc02a96",
    "readOnly": false,
    "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::AppSync::GraphQLApi",
      "ARN": "arn:aws:appsync:us-west-2:123456789012:apis/rxfqcxzi3nbvza2hsq4njqqq6u"
    }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "123456789012",
    "eventCategory": "Data"
}
```

Der folgende CloudTrail Beispielprotokolleintrag zeigt einen teilweise erfolgreichen GraphQL-Vorgang, der mit einem benutzerdefinierten Lambda-Funktionsautorisierer autorisiert wurde. Beachten Sie die `fieldAuthorizationResults.deniedFields` Eigenschaft, die die verweigerten Felder angibt.

```
{
  "eventVersion": "1.10",
  "userIdentity": {
    "type": "Unknown"
  },
  "eventTime": "2024-11-06T16:11:49Z",
  "eventSource": "appsync.amazonaws.com",
  "eventName": "GraphQL",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "15.248.1.214",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:131.0) Gecko/20100101
 Firefox/131.0",
  "requestParameters": null,
  "responseElements": null,
  "additionalEventData": {
    "operationName": "MyMutation",
    "authType": [
      "AWS_LAMBDA"
    ],
    "fieldAuthorizationResults": {
      "deniedFields": [
```

```
        "arn:aws:appsync:us-west-2:123456789012:apis/rxfqcxzi3nbvza2hsq4njqqq6u/types/
Mutation/fields/createPost",
        "arn:aws:appsync:us-west-2:123456789012:apis/rxfqcxzi3nbvza2hsq4njqqq6u/types/
Subscription/fields/onCreatePost",
        "arn:aws:appsync:us-west-2:123456789012:apis/rxfqcxzi3nbvza2hsq4njqqq6u/types/
Post/fields/status"
      ]
    }
  },
  "requestID": "ae817c4c-66ba-4f64-92a5-ba9c9c341dcd",
  "eventID": "30109698-7605-476a-9dff-b7ed78d134dc",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::AppSync::GraphQLApi",
      "ARN": "arn:aws:appsync:us-west-2:123456789012:apis/rxfqcxzi3nbvza2hsq4njqqq6u"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data"
}
```

Der folgende CloudTrail Beispielprotokolleintrag zeigt einen fehlgeschlagenen GraphQL-Vorgang.

```
{
  "eventVersion": "1.10",
  "userIdentity": {
    "type": "Unknown"
  },
  "eventTime": "2024-11-06T15:51:11Z",
  "eventSource": "appsync.amazonaws.com",
  "eventName": "GraphQL",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "15.248.1.214",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:131.0) Gecko/20100101
 Firefox/131.0",
  "errorCode": "AccessDenied",
  "errorMessage": "{\n \"errors\" : [ {\n \"errorType\" : \"UnauthorizedException\",\n
 \"message\" : \"You are not authorized to make this call.\"\n } ]\n}",
  "requestParameters": null,
```

```
  "responseElements": null,
  "additionalEventData": {
    "operationName": "MyFullyDeniedLambdaMutation"
  },
  "requestID": "0bef3cf3-a48b-4de9-8b1f-038afb563516",
  "eventID": "b738651f-4ec0-4548-8fec-200c6b42842b",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::AppSync::GraphQLApi",
      "ARN": "arn:aws:appsync:us-west-2:123456789012:apis/rxfqcxzi3nbvza2hsq4njqqq6u"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data"
}
```

Das folgende Beispiel zeigt eine erfolgreiche GraphQL-Anfrage.

```
{
  "eventVersion": "1.10",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:jane_doe",
    "arn": "arn:aws:sts::123456789012:assumed-role/admin/jane_doe",
    "accountId": "123456789012",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/admin",
        "accountId": "123456789012",
        "userName": "jane_doe"
      },
      "attributes": {
        "creationDate": "2024-11-06T15:40:09Z",
        "mfaAuthenticated": "false"
      }
    }
  },
```

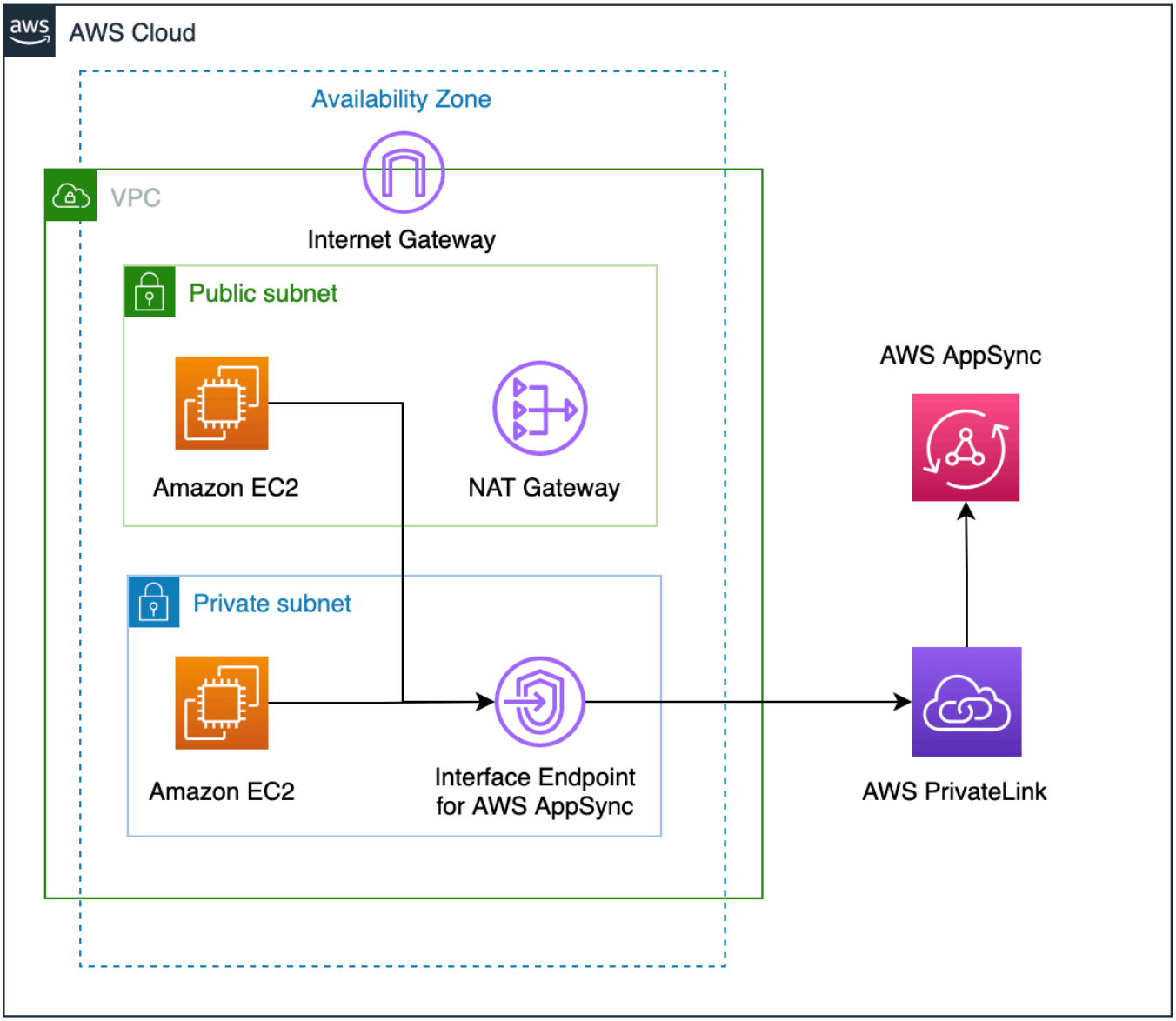
```
  "eventTime": "2024-11-06T16:03:43Z",
  "eventSource": "appsync.amazonaws.com",
  "eventName": "GraphQL",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "15.248.1.214",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:131.0) Gecko/20100101
 Firefox/131.0",
  "requestParameters": null,
  "responseElements": null,
  "additionalEventData": {
    "operationName": "IamFullSuccess",
    "authType": [
      "AWS_IAM"
    ],
    "fieldAuthorizationResults": {
      "allowedFields": [
        "arn:aws:appsync:us-west-2:123456789012:apis/rxfqcxzi3nbvza2hsq4njqqq6u/types/
Mutation/fields/createSecondPostAllowed"
      ],
      "deniedFields": []
    }
  },
  "requestID": "edc6bbbf-6bf2-40f5-820f-ef444f12e0c1",
  "eventID": "524656a5-0925-4370-9e7e-08888e9c299f",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::AppSync::GraphQLApi",
      "ARN": "arn:aws:appsync:us-west-2:123456789012:apis/rxfqcxzi3nbvza2hsq4njqqq6u"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data"
}
```


AWS AppSync Private verwenden APIs

Wenn Sie Amazon Virtual Private Cloud (Amazon VPC) verwenden, können Sie AWS AppSync Private erstellen, d. h. APIs, auf die nur von einer VPC aus zugegriffen werden kann. APIs Mit

einer privaten API können Sie den API-Zugriff auf Ihre internen Anwendungen einschränken und eine Verbindung zu Ihren GraphQL- und Realtime-Endpunkten herstellen, ohne Daten öffentlich zugänglich zu machen.

Um eine private Verbindung zwischen Ihrer VPC und dem AWS AppSync Service herzustellen, müssen Sie einen VPC-Schnittstellen-Endpunkt erstellen. Schnittstellenendpunkte werden mit Strom versorgt AWS PrivateLink, sodass Sie AWS AppSync APIs ohne Internet-Gateway, NAT-Gerät, VPN-Verbindung oder Verbindung privat darauf zugreifen können. AWS Direct Connect Instances in Ihrer VPC benötigen keine öffentlichen IP-Adressen für die Kommunikation. AWS AppSync APIs Datenverkehr zwischen Ihrer VPC und verlässt das AWS Netzwerk AWS AppSync nicht.



Es gibt einige zusätzliche Faktoren, die Sie berücksichtigen sollten, bevor Sie die Funktionen der privaten API aktivieren:

- Durch die Einrichtung von VPC-Schnittstellenendpunkten für AWS AppSync aktivierte private DNS-Funktionen wird verhindert, dass Ressourcen in der VPC APIs mithilfe der generierten API-URL andere AWS AppSync öffentliche Daten aufrufen können. AWS AppSync Dies liegt daran, dass die Anfrage an die öffentliche API über den Schnittstellenendpunkt weitergeleitet wird, was öffentlich nicht zulässig ist. APIs Um APIs in diesem Szenario public aufzurufen, wird empfohlen, benutzerdefinierte Domainnamen auf public zu konfigurieren APIs, die dann von Ressourcen in der VPC zum Aufrufen der öffentlichen API verwendet werden können.
- APIs Ihr AWS AppSync Privatkonto ist nur in Ihrer VPC verfügbar. Der Abfrage-Editor der AWS AppSync Konsole kann Ihre API nur erreichen, wenn die Netzwerkkonfiguration Ihres Browsers den Datenverkehr zu Ihrer VPC weiterleiten kann (z. B. Verbindung über VPN oder über AWS Direct Connect).
- Mit einem VPC-Schnittstellenendpunkt für AWS AppSync können Sie auf jede private API in demselben AWS Konto und derselben Region zugreifen. Um den Zugriff auf Private weiter einzuschränken APIs, können Sie die folgenden Optionen in Betracht ziehen:
  - Stellen Sie sicher, dass nur die erforderlichen Administratoren VPC-Endpunktschnittstellen für AWS AppSync erstellen können.
  - Verwenden von benutzerdefinierten VPC-Endpunktrichtlinien, um einzuschränken, welche von Ressourcen in der VPC aufgerufen werden APIs können.
  - Für Ressourcen in der VPC empfehlen wir, die IAM-Autorisierung zum Aufrufen zu verwenden, AWS AppSync APIs indem Sie sicherstellen, dass den Ressourcen abgegrenzte Rollen zugewiesen werden. APIs
- Wenn Sie Richtlinien erstellen oder verwenden, die IAM-Prinzipale einschränken, müssen Sie die Methode auf oder setzen. `authorizationType` `AWS_IAM` `NONE`

Private erstellen AWS AppSync APIs

Die folgenden Schritte zeigen Ihnen, wie Sie Private APIs im AWS AppSync Service erstellen.

> **Warning**
>
> Sie können private API-Funktionen nur während der Erstellung der API aktivieren. Diese Einstellung kann für eine AWS AppSync API oder eine AWS AppSync private API nicht geändert werden, nachdem sie erstellt wurde.

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
   - Wählen Sie im Dashboard Create API (API erstellen) aus.
2. Wählen Sie „Eine API von Grund auf neu entwerfen“ und anschließend „Weiter“.
3. Wähle im Abschnitt Private API die Option Private API-Funktionen verwenden aus.
4. Konfigurieren Sie die restlichen Optionen, überprüfen Sie die Daten Ihrer API und wählen Sie dann Erstellen aus.

Bevor Sie Ihre AWS AppSync private API verwenden können, müssen Sie einen Schnittstellenendpunkt für AWS AppSync in Ihrer VPC konfigurieren. Beachten Sie, dass sich sowohl die Private API als auch die VPC im selben AWS Konto und in derselben Region befinden müssen.

Erstellen eines Schnittstellenendpunkts für AWS AppSync

Sie können einen Schnittstellenendpunkt für die AWS AppSync Verwendung entweder der Amazon VPC-Konsole oder der AWS Command Line Interface (AWS CLI) erstellen. Weitere Informationen finden Sie unter Erstellung eines Schnittstellenendpunkts im Benutzerhandbuch für Amazon VPC.

Console

1. Melden Sie sich auf der Seite Endpoints der Amazon VPC-Konsole an AWS Management Console und öffnen Sie sie.
2. Wählen Sie Endpunkt erstellen aus.
   a. Vergewissern Sie sich, dass im Feld Servicekategorie AWS Dienste ausgewählt sind.
   b. Wählen Sie in der Tabelle Dienste die Option`com.amazonaws.{region}.appsync-api`. Vergewissern Sie sich, dass der Wert in der Spalte Typ lautet`Interface`.
   c. Wählen Sie im Feld VPC eine VPC und ihre Subnetze aus.
   d. Um private DNS-Funktionen für den Schnittstellenendpunkt zu aktivieren, aktivieren Sie das Kontrollkästchen DNS-Name aktivieren.

e. Wählen Sie im Feld Sicherheitsgruppe eine oder mehrere Sicherheitsgruppen aus.

3. Wählen Sie Endpunkt erstellen aus.

CLI

Verwenden Sie den `create-vpc-endpoint`-Befehl und geben Sie die VPC-ID an, den VPC-Endpunkttyp (Schnittstelle), den Servicenamen, die Subnetze, die den Endpunkt verwenden sollen, sowie die Sicherheitsgruppen, die den Endpunktnetzwerkschnittstellen zugeordnet werden sollen. Zum Beispiel:

```
$ 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
```

Um die private DNS-Option zu verwenden, müssen Sie die `enableDnsSupportattributes` Werte `enableDnsHostnames` und für Ihre VPC festlegen. Weitere Informationen finden Sie unter Anzeigen und Aktualisieren der DNS-Unterstützung für Ihre VPC im Amazon-VPC-Benutzerhandbuch. Wenn Sie private DNS-Funktionen für den Schnittstellenendpunkt aktivieren, können Sie mithilfe der standardmäßigen öffentlichen DNS-Endpunkte Anfragen an Ihren AWS AppSync API-GraphQL- und Echtzeit-Endpunkt stellen. Verwenden Sie dazu das folgende Format:

```
https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql
```

Weitere Informationen zu Dienstendpunkten finden Sie unter Dienstendpunkte und Kontingente in der Allgemeinen Referenz.AWS

Weitere Informationen zu Serviceinteraktionen mit Schnittstellenendpunkten finden Sie unter Zugreifen auf einen Service über einen Schnittstellenendpunkt im Amazon VPC-Benutzerhandbuch.

Informationen zum Erstellen und Konfigurieren eines Endpunkts mithilfe AWS CloudFormation finden Sie in der VPCEndpoint Ressource AWS EC2:::: im AWS CloudFormation Benutzerhandbuch.

Fortschrittliche -Beispiele

Wenn Sie private DNS-Funktionen für den Schnittstellenendpunkt aktivieren, können Sie mithilfe der standardmäßigen öffentlichen DNS-Endpunkte Anfragen an Ihren AWS AppSync API-GraphQL- und Echtzeit-Endpunkt stellen. Verwenden Sie dazu das folgende Format:

```
https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql
```

Unter Verwendung der öffentlichen DNS-Hostnamen des VPC-Endpunkts der Schnittstelle hat die Basis-URL zum Aufrufen der API das folgende Format:

```
https://{vpc_endpoint_id}-{endpoint_dns_identifier}.appsync-api.
{region}.vpce.amazonaws.com/graphql
```

Sie können auch den AZ-spezifischen DNS-Hostnamen verwenden, wenn Sie einen Endpunkt in der AZ bereitgestellt haben:

```
https://{vpc_endpoint_id}-{endpoint_dns_identifier}-{az_id}.appsync-api.
{region}.vpce.amazonaws.com/graphql.
```

Wenn Sie den öffentlichen DNS-Namen des VPC-Endpunkts verwenden, muss der Hostname des AWS AppSync API-Endpunkts als `Host` oder als `x-appsync-domain` Header an die Anfrage übergeben werden. In diesen Beispielen wird eine verwendet`TodoAPI`, die im Leitfaden Launch a sample schema erstellt wurde:

```
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"}}}'
```

In den folgenden Beispielen verwenden wir die Todo-App, die im Leitfaden Launch a sample schema generiert wurde. Um die Todo-Beispiel-API zu testen, werden wir das Private DNS verwenden, um die API aufzurufen. Sie können jedes Befehlszeilentool Ihrer Wahl verwenden. In diesem Beispiel wird curl zum Senden von Abfragen und Mutationen und wscat zum Einrichten von Abonnements verwendet. Um unser Beispiel zu emulieren, ersetzen Sie die Werte in Klammern `{ }` in den folgenden Befehlen durch die entsprechenden Werte aus Ihrem Konto. AWS

Mutationsoperation testen — Anfrage **`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"}}}'
```

Mutationsoperation testen — **`createTodo`** Antwort

```
{
    "data": {
        "createTodo": {
            "id": "<todo-id>",
            "name": "My first GraphQL task",
            "where": "Day 1",
            "when": "Friday Night",
            "description": "Learn more about GraphQL"
        }
    }
}
```

Abfragevorgang testen — **`listTodos`** Anfrage

```
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"}}}'
```

Abfragevorgang testen — **`listTodos`** Anfrage

```
{
  "data": {
    "listTodos": {
      "items": [
        {
          "description": "Learn more about GraphQL",
          "id": "<todo-id>",
          "name": "My first GraphQL task",
          "when": "Friday night",
```

```
          "where": "Day 1"
        }
      ]
    }
  }
}
```

Testen des Abonnementvorgangs — Mutation abonnieren **`createTodo`**

Informationen zum Einrichten von GraphQL-Abonnements finden Sie unter Erstellen eines WebSocket Echtzeitclients. AWS AppSync Von einer EC2 Amazon-Instance in einer VPC aus können Sie Ihren AWS AppSync privaten API-Abonnementendpunkt mit wscat testen. Das folgende Beispiel verwendet eine `API KEY` für die Autorisierung.

```
$ 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"}
```

Verwenden Sie alternativ den Domainnamen des VPC-Endpunkts und stellen Sie sicher, dass Sie den Host-Header im `wscat` Befehl angeben, um den Websocket einzurichten:

```
$ 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"}
```

Führen Sie den folgenden Mutationscode aus:

```
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"}}}'
```

Danach wird ein Abonnement ausgelöst und die Nachrichtenbenachrichtigung wird wie folgt angezeigt:

```
< {"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"}}}}
```

Verwendung von IAM-Richtlinien zur Beschränkung der Erstellung öffentlicher APIs

AWS AppSync unterstützt `Condition`IAM-Anweisungen zur Verwendung mit Private. APIs Das `visibility` Feld kann in die IAM-Richtlinienanweisungen aufgenommen werden, um zu steuern, welche IAM-Rollen und Benutzer private und öffentliche IAM-Rollen erstellen können. `appsync:CreateGraphqlApi` APIs Dies gibt einem IAM-Administrator die Möglichkeit, eine IAM-Richtlinie zu definieren, die es einem Benutzer nur ermöglicht, eine private GraphQL-API zu erstellen. Ein Benutzer, der versucht, eine öffentliche API zu erstellen, erhält eine nicht autorisierte Nachricht.

Ein IAM-Administrator könnte beispielsweise die folgende IAM-Richtlinienerklärung erstellen, um die Erstellung von Private zu ermöglichen: APIs

```
{
    "Sid": "AllowPrivateAppSyncApis",
```

```
    "Effect": "Allow",
    "Action": "appsync:CreateGraphqlApi",
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "appsync:Visibility": "PRIVATE"
        }
    }
}
```

Ein IAM-Administrator könnte auch die folgende Dienststeuerungsrichtlinie hinzufügen, um zu verhindern, dass alle Benutzer in einer AWS Organisation etwas AWS AppSync APIs anderes als privat erstellen: APIs

```
{
    "Sid": "BlockNonPrivateAppSyncApis",
    "Effect": "Deny",
    "Action": "appsync:CreateGraphqlApi",
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringNotEquals": {
            "appsync:Visibility": "PRIVATE"
        }
    }
}
```

AWS AppSync GraphQL teilen APIs

AWS AppSync integriert sich in AWS Resource Access Manager (AWS RAM), um die gemeinsame Nutzung von Ressourcen zu ermöglichen. AWS RAM ist ein Dienst, der es Ihnen ermöglicht, Aufrufaktionen (Abfrage-, Mutations- und Abonnementvorgänge und Verbindungsanfragen mit Ihrem WebSocket Echtzeitendpunkt) auf AWS AppSync GraphQL APIs mit anderen AWS-Konten oder über zu teilen. AWS Organizations Mit können Sie Ressourcen AWS RAM, die Ihnen gehören, gemeinsam nutzen, indem Sie eine gemeinsame Nutzung erstellen. Eine Ressourcenfreigabe legt die freizugebenden Ressourcen und die Konsumenten fest, für die sie freigegeben werden sollen. Verbraucher können Folgendes einbeziehen.

- AWS-Konten Spezifisch innerhalb oder außerhalb seiner Organisation in AWS Organizations
- Eine Organisationseinheit innerhalb ihrer Organisation in AWS Organizations

- Eine ganze Organisation in AWS Organizations

Weitere Informationen zu AWS RAM finden Sie im AWS Resource Access Manager Benutzerhandbuch.

Themen

- Voraussetzungen für die gemeinsame Nutzung von AWS AppSync GraphQL APIs
- AWS AppSync GraphQL teilen APIs
- AWS AppSync GraphQL nicht mehr teilen APIs
- Kontoübergreifende Ereignisse

Voraussetzungen für die gemeinsame Nutzung von AWS AppSync GraphQL APIs

Für die gemeinsame Nutzung von AWS AppSync GraphQL APIs gelten die folgenden Voraussetzungen.

- Um eine AWS AppSync GraphQL-API gemeinsam zu nutzen, müssen Sie sie in Ihrer AWS-Konto besitzen. Das bedeutet, dass die AWS AppSync GraphQL-API in Ihrem Konto zugewiesen oder bereitgestellt werden muss.
- Um eine AWS AppSync GraphQL-API mit Ihrer Organisation oder einer Organisationseinheit in zu teilen AWS Organizations, müssen Sie das Teilen mit AWS Organizations aktivieren. Weitere Informationen finden Sie unter Ressourcenfreigabe für AWS Organizations aktivieren im AWS Resource Access Manager -Benutzerhandbuch.

AWS AppSync GraphQL teilen APIs

Um eine AWS AppSync GraphQL-API gemeinsam zu nutzen, erstellen Sie zunächst eine Ressourcenfreigabe mit AWS Resource Access Manager. Eine Ressourcenfreigabe gibt an, welche Ressourcen gemeinsam genutzt werden sollen, mit welchen Verbrauchern sie geteilt werden und welche Aktionen Principals ausführen können. Wenn Sie eine AWS AppSync GraphQL-API, die Sie besitzen, mit anderen teilen AWS-Konten, ermöglichen Sie diesen Konten, diese AWS AppSync API in Ihrem AWS-Konto aufzurufen.

Wenn Sie Teil einer Organisation sind und die gemeinsame Nutzung innerhalb Ihrer Organisation aktiviert ist, erhalten Verbraucher in Ihrer Organisation automatisch Zugriff auf die gemeinsam

genutzte Ressource. AWS Organizations Andernfalls erhalten Verbraucher eine Einladung zur Teilnahme am Resource Share und erhalten nach Annahme der Einladung Zugriff auf die gemeinsam genutzte Ressource.

Überlegungen zur gemeinsamen Nutzung

- Sie können nur AWS AppSync GraphQL teilen APIs, keine anderen API-Typen wie Event APIs.
- Sie können nur AWS AppSync GraphQL teilen, für APIs die `AWS_IAM` einer der in der API konfigurierten Autorisierungsmodi konfiguriert ist.

  Wenn `AWS_IAM` aus der Liste der Autorisierungsmodi für eine gemeinsam genutzte AppSync GraphQL-API entfernt wird, obwohl die Ressourcenfreigabe möglicherweise noch existiert, wird sie unwirksam.
- Sie können sowohl öffentliches als auch privates AWS AppSync GraphQL APIs teilen.
- Auf Private AWS AppSync GraphQL APIs kann immer über VPC-Endpunkte VPCs im Ursprung zugegriffen werden AWS-Konto, und alle Autorisierungsmodi werden unterstützt, nicht nur. `AWS_IAM`
- Bei Shared AWS AppSync GraphQL APIs werden Berechtigungen nur für die API-Ressource verwaltet und unterstützen keine detaillierten Berechtigungen für Feld und Typ sowie Feldressourcen. Wenn Sie eine API teilen, teilen Sie den API-ARN und den ARNs für alle ihre Typen und Felder.

Erstellen Sie mithilfe der AWS RAM Konsole eine Ressourcenfreigabe, die Ihnen gehört

Um eine AWS AppSync GraphQL-API gemeinsam zu nutzen, verwenden Sie das im AWS Resource Access Manager Benutzerhandbuch unter Creating a Resource Share beschriebene Verfahren, indem Sie RAM Name `AWSRAMPermissionAppSyncGraphQLApiInvokeAccess` der Berechtigung.

Erstellen und verwenden Sie eine vom Kunden verwaltete Berechtigung, um eine private AWS AppSync GraphQL-API über die AWS RAM Konsole zu teilen

Um eine private AWS AppSync GraphQL-API gemeinsam zu nutzen, erstellen Sie eine vom Kunden verwaltete Berechtigung. Gehen Sie dabei wie im AWS Resource Access Manager Benutzerhandbuch unter Erstellen und Verwenden von kundenverwalteten Berechtigungen beschrieben vor.

Beispielsweise möchte ein Besitzer von Konto A Principals in Konto B die Erlaubnis erteilen, auf eine private AWS AppSync GraphQL-API (A) für Aufrufe über VPCE-B (PrivateApieinen VPC-Endpunkt, der Konto B gehört) zuzugreifen. In diesem Fall muss der Eigentümer für Konto A wie folgt eine vom Kunden verwaltete Berechtigung erstellen. AWS RAM

```
{
    "Effect": "Allow",
    "Action": ["appsync:GraphQL"],
    "Condition": {
        "StringEqualsIgnoreCase": {
            "aws:SourceVpce": [
                "VPCE-B"
            ]
        }
    }
}
```

Gehen Sie davon aus, dass diese neue vom Kunden verwaltete AWS RAM Berechtigung benannt ist`private-api-A-access-via-vpce-b`.

Um den kontoübergreifenden Zugriff auf `PrivateApiA` via zu ermöglichen`VPCE-B`, kann der Kunde eine AWS RAM Ressourcenfreigabe mit den folgenden Parametern und der vom Kunden verwalteten Berechtigung aus dem vorherigen Beispiel erstellen.

- Art der Ressource: `appsync:Apis`
- Ressource: `arn:aws:appsync:us-west-2:A:apis/PrivateApiA`
- Berechtigung: `private-api-A-access-via-vpce-b` (Vom Kunden verwaltete Berechtigung)
- Schulleiter: `Account: B`

Erstellen Sie eine Ressourcenfreigabe, die Sie besitzen, mit dem AWS CLI

Um eine AWS AppSync GraphQL-API mithilfe von gemeinsam zu nutzen AWS CLI, verwenden Sie den `create-resource-share` Befehl with `arn:aws:ram::aws:permission/AWSRAMPermissionAppSyncApiInvokeAccess` als Wert für den `--permission-arns` Switch.

Eine vollständige Liste der verfügbaren Befehle für AWS RAM finden Sie in der AWS RAM CLI-Referenz.

AWS AppSync GraphQL nicht mehr teilen APIs

Um die gemeinsame Nutzung von AWS AppSync GraphQL, APIs das Sie besitzen, zu beenden, müssen Sie entweder die Ressourcenfreigabe löschen oder die Prinzipale aktualisieren, mit denen Sie die Ressource geteilt haben. Informationen zu der Aktion, die Sie ausführen möchten, finden Sie in der Dokumentation in den folgenden Abschnitten.

Um die gemeinsame Nutzung einer Ressource, deren Eigentümer Sie sind, über die AWS RAM Konsole zu beenden

Weitere Informationen finden Sie im AWS Resource Access Manager Benutzerhandbuch unter Aktualisieren einer gemeinsam genutzten Ressource.

Um die gemeinsame Nutzung einer Ressource, deren Eigentümer Sie sind, zu beenden, verwenden Sie AWS CLI

Verwenden Sie den disassociate-resource-share-Befehl.

Um eine Ressource zu löschen, die Ihnen gehört, verwenden Sie die AWS RAM Konsole

Weitere Informationen finden Sie im AWS Resource Access Manager Benutzerhandbuch unter Löschen einer Ressourcenfreigabe.

Um eine Ressourcenfreigabe zu löschen, die Sie besitzen, verwenden Sie AWS CLI

Verwenden Sie den delete-resource-share-Befehl.

Eine vollständige Liste der verfügbaren Befehle für AWS RAM finden Sie in der AWS RAM CLI-Referenz.

Kontoübergreifende Ereignisse

Sie können sich für die Protokollierung von AWS CloudTrail Datenereignissen entscheiden, um kontenübergreifende AWS AppSync GraphQL-API-Aktivitäten zu überwachen und zu prüfen. DataPlane Weitere Informationen finden Sie unter Protokollieren von Datenereignissen im Benutzerhandbuch für AWS CloudTrail .

Konfiguration von GraphQL-Laufkomplexität, Abfragetiefe und Introspektion mit AWS AppSync

AWS AppSync ermöglicht es Ihnen, Introspektionsfunktionen zu aktivieren oder zu deaktivieren und Grenzwerte für die Anzahl der verschachtelten Ebenen und Resolver in einer einzelnen Abfrage festzulegen.

Verwenden der Introspektionsfunktion

> **Tip**
>
> Weitere Informationen zur Selbstbeobachtung in GraphQL finden Sie in diesem Artikel auf der Website der GraphQL Foundation.

Standardmäßig ermöglicht Ihnen GraphQL die Introspektion, um das Schema selbst abzufragen, um seine Typen, Felder, Abfragen, Mutationen, Abonnements usw. zu ermitteln. Dies ist eine wichtige Funktion, um zu erfahren, wie die Daten von Ihrem GraphQL-Dienst geformt und verarbeitet werden. Beim Umgang mit Introspektion sind jedoch einige Dinge zu beachten. Möglicherweise haben Sie einen Anwendungsfall, der von einer Deaktivierung der Introspektion profitieren würde, z. B. einen Fall, in dem Feldnamen sensibel oder versteckt sein können oder das vollständige API-Schema für Verbraucher undokumentiert bleiben soll. In diesen Fällen könnte die Veröffentlichung von Schemadaten durch Introspektion dazu führen, dass bewusst vertrauliche Daten verloren gehen.

Um dies zu verhindern, können Sie die Introspektion deaktivieren. Dadurch wird verhindert, dass Unbefugte Introspektionsfelder in Ihrem Schema verwenden. Es ist jedoch wichtig zu beachten, dass Introspektion für Entwicklungsteams nützlich ist, um zu erfahren, wie Daten in ihrem Service verarbeitet werden. Intern kann es hilfreich sein, die Introspektion aktiviert zu lassen und sie gleichzeitig als zusätzliche Sicherheitsebene im Produktionscode zu deaktivieren. Eine andere Möglichkeit, dies zu handhaben, besteht darin, eine Autorisierungsmethode hinzuzufügen, die AWS AppSync auch Folgendes bietet: Weitere Informationen finden Sie unter -Autorisierung.

AWS AppSync ermöglicht es Ihnen, die Introspektion auf API-Ebene zu aktivieren oder zu deaktivieren. Gehen Sie wie folgt vor, um die Introspektion zu aktivieren oder zu deaktivieren:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
2. Wählen Sie auf der APIsSeite den Namen einer GraphQL-API aus.
3. Wählen Sie auf der Startseite Ihrer API im Navigationsbereich Einstellungen aus.

4. Wählen Sie in API-Konfigurationen die Option Bearbeiten aus.
5. Gehen Sie unter Introspektionsabfragen wie folgt vor:
   - Aktivieren oder deaktivieren Sie Introspektionsabfragen aktivieren.
6. Wählen Sie Save (Speichern) aus.

Wenn die Introspektion aktiviert ist (das Standardverhalten), funktioniert die Verwendung des Introspektionssystems normal. Die Abbildung unten zeigt beispielsweise ein `__schema` Feld, das alle verfügbaren Typen im Schema verarbeitet:

Wenn Sie diese Funktion deaktivieren, wird stattdessen ein Validierungsfehler in der Antwort angezeigt:

Konfiguration von Grenzwerten für die Abfragetiefe

Es gibt Zeiten, in denen Sie möglicherweise eine genauere Kontrolle darüber wünschen, wie die API während eines Vorgangs funktioniert. Ein solches Steuerelement besteht darin, die Anzahl der verschachtelten Ebenen, die eine Abfrage verarbeiten darf, zu begrenzen. Standardmäßig können Abfragen eine unbegrenzte Anzahl verschachtelter Ebenen verarbeiten. Die Beschränkung von Abfragen auf eine bestimmte Anzahl verschachtelter Ebenen hat potenzielle Auswirkungen auf die Leistung und Flexibilität Ihres Projekts. Führen Sie die folgende Abfrage aus:

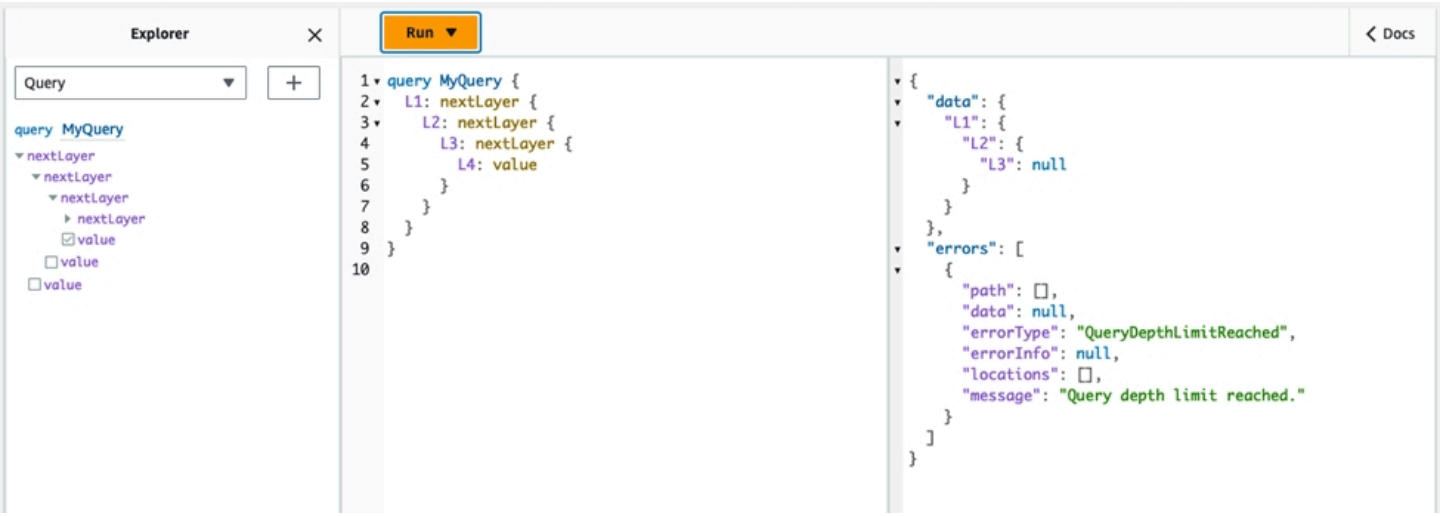
```
query MyQuery {
  L1: nextLayer {
    L2: nextLayer {
      L3: nextLayer {
        L4: value
      }
    }
  }
}
```

Ihr Projekt erfordert möglicherweise die Beschränkung von Abfragen auf `L1` oder `L2` für einen bestimmten Zweck. Standardmäßig `L4` würde die gesamte Abfrage von `L1` bis verarbeitet, ohne dass dies kontrolliert werden kann. Durch die Festlegung eines Grenzwerts könnten Sie verhindern, dass Abfragen auf Daten zugreifen, die über die angegebene Ebene hinausgehen.

Gehen Sie wie folgt vor, um eine Abfragetiefenbeschränkung hinzuzufügen:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.
2. Wählen Sie auf der APIsSeite den Namen einer GraphQL-API aus.
3. Wählen Sie auf der Startseite Ihrer API im Navigationsbereich Einstellungen aus.
4. Wählen Sie in API-Konfigurationen die Option Bearbeiten aus.
5. Gehen Sie unter Abfragetiefe wie folgt vor:
   a. Aktivieren oder deaktivieren Sie die Option Abfragetiefe aktivieren.
   b. Stellen Sie unter Maximale Tiefe die Tiefenbegrenzung ein. Dieser Wert kann zwischen 1 und liegen`75`.
6. Wählen Sie Save (Speichern) aus.

Wenn ein Limit festgelegt ist, führt das Überschreiten seiner Obergrenze zu einem `QueryDepthLimitReached` Fehler. Die Abbildung unten zeigt beispielsweise eine Abfrage mit einer Tiefenbeschränkung, bei der das Limit bis zur dritten (`L3`) und vierten (`L4`) Ebene überschritten 2 wird:



Beachten Sie, dass Felder im Schema immer noch als nullwertfähig oder nicht nullwertfähig markiert werden können. Wenn ein Feld, das keine NULL-Werte zulässt, einen `QueryDepthLimitReached` Fehler erhält, wird dieser Fehler auf das erste übergeordnete Feld übertragen.

Konfiguration der Grenzwerte für die Anzahl der Resolver

Sie können auch steuern, wie viele Resolver jede Abfrage verarbeiten kann. Wie bei der Abfragetiefe können Sie auch für diesen Betrag ein Limit festlegen. Nehmen wir die folgende Abfrage, die drei Resolver enthält:

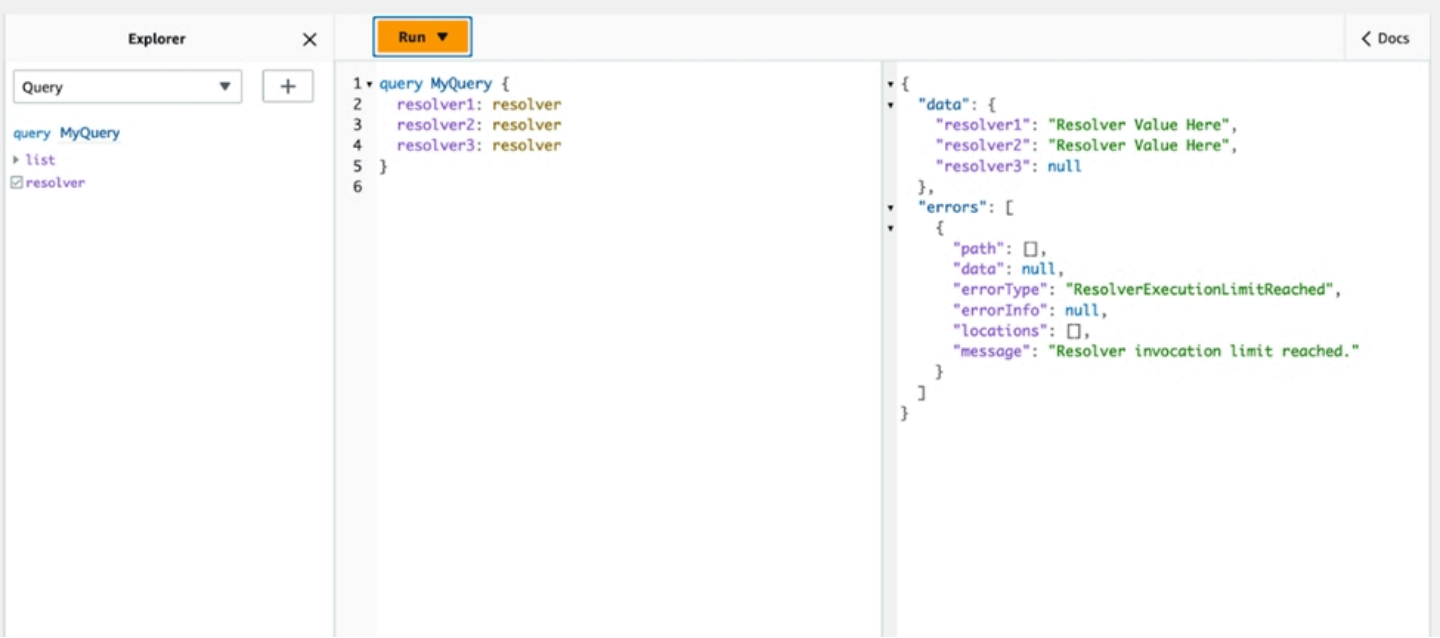
```
query MyQuery {
  resolver1: resolver
  resolver2: resolver
  resolver3: resolver
}
```

Standardmäßig kann jede Abfrage bis zu 10000 Resolver verarbeiten. Im obigen Beispiel `resolver3` werden`resolver1`,`resolver2`, und verarbeitet. In Ihrem Projekt ist es jedoch möglicherweise erforderlich, jede Abfrage auf die Verarbeitung von insgesamt einem oder zwei Resolvern zu beschränken. Indem Sie ein Limit festlegen, können Sie die Abfrage so einstellen, dass sie keine Resolver verarbeitet, die eine bestimmte Anzahl überschreiten, wie dies bei den Resolvern First (`resolver1`) oder Second (`resolver2`) der Fall ist.

Gehen Sie wie folgt vor, um ein Limit für die Anzahl der Resolver hinzuzufügen:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.
2. Wählen Sie auf der APIsSeite den Namen einer GraphQL-API aus.
3. Wählen Sie auf der Startseite Ihrer API im Navigationsbereich Einstellungen aus.
4. Wählen Sie in API-Konfigurationen die Option Bearbeiten aus.
5. Gehen Sie unter Auflösungslimit wie folgt vor:
   a. Aktivieren Sie die Option Anzahl der Resolver aktivieren.
   b. Legen Sie unter Maximale Anzahl an Resolvern die maximale Anzahl an Resolvern fest. Dieser Wert kann zwischen 1 und `10000` liegen.
6. Wählen Sie Save (Speichern) aus.

Wie bei der Abfragetiefenbeschränkung führt eine Überschreitung der konfigurierten Auflösungsgrenze dazu, dass die Abfrage bei weiteren Resolvern mit einem `ResolverExecutionLimitReached` Fehler endet. In der Abbildung unten versucht eine Abfrage mit einem Auflösungslimit von 2, drei Resolver zu verarbeiten. Aufgrund des Limits gibt der dritte Resolver einen Fehler aus und wird nicht ausgeführt.



Verwendung von Umgebungsvariablen in AWS AppSync

Sie können Umgebungsvariablen verwenden, um das Verhalten Ihrer AWS AppSync Resolver und Funktionen anzupassen, ohne Ihren Code zu aktualisieren. Umgebungsvariablen sind Zeichenkettenpaare, die in Ihrer API-Konfiguration gespeichert sind und Ihren Resolvern und Funktionen zur Verfügung gestellt werden, um sie zur Laufzeit zu nutzen. Sie sind besonders nützlich in Situationen, in denen Sie auf Konfigurationsdaten verweisen müssen, die nur bei der Ersteinrichtung verfügbar sind, aber während der Ausführung von Ihren Resolvern und Funktionen verwendet werden müssen. Umgebungsvariablen machen Konfigurationsdaten in Ihrem Code verfügbar, wodurch die Notwendigkeit reduziert wird, diese Werte fest zu codieren.

> **Note**
>
> Um die Datenbanksicherheit zu erhöhen, empfehlen wir, Secrets Manager oder AWS Systems Manager Parameter Store anstelle von Umgebungsvariablen zum Speichern von Anmeldeinformationen oder vertraulichen Informationen zu verwenden. Informationen zur Nutzung dieser Funktion finden Sie unter AWS Dienste mit AWS AppSync HTTP-Datenquellen aufrufen.

Umgebungsvariablen müssen verschiedenen Verhaltensweisen und Regeln folgen, um ordnungsgemäß zu funktionieren:

- Sowohl JavaScript Resolver/Funktionen als auch VTL-Vorlagen unterstützen Umgebungsvariablen.
- Umgebungsvariablen werden vor dem Funktionsaufruf nicht ausgewertet.
- Umgebungsvariablen unterstützen nur Zeichenkettenwerte.
- Jeder definierte Wert in einer Umgebungsvariablen wird als Zeichenkettenliteral betrachtet und nicht erweitert.
- Variablenauswertungen sollten idealerweise im Funktionscode durchgeführt werden.

Konfiguration von Umgebungsvariablen (Konsole)

Sie können Umgebungsvariablen für Ihre AWS AppSync GraphQL-API konfigurieren, indem Sie die Variable erstellen und ihr Schlüssel-Wert-Paar definieren. Ihre Resolver und Funktionen verwenden den Schlüsselnamen der Umgebungsvariablen, um den Wert zur Laufzeit abzurufen. So legen Sie Umgebungsvariablen in der AWS AppSync Konsole fest:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSyncKonsole.
2. Wählen Sie auf der APIsSeite den Namen einer GraphQL-API aus.
3. Wählen Sie auf der Startseite Ihrer API im Navigationsbereich Einstellungen aus.
4. Wählen Sie unter Umgebungsvariablen die Option Umgebungsvariable hinzufügen aus.
5. Wählen Sie Umgebungsvariablen hinzufügen aus.
6. Geben Sie einen Schlüssel und einen Wert ein.
7. Falls erforderlich, wiederholen Sie die Schritte 5 und 6, um weitere Schlüsselwerte hinzuzufügen. Wenn Sie einen Schlüsselwert entfernen müssen, wählen Sie die Option Entfernen und die Schlüssel, die entfernt werden sollen.
8. Wählen Sie Absenden aus.

> **Tip**
>
> Beim Erstellen von Schlüsseln und Werten müssen Sie einige Regeln beachten:
>
> - Schlüssel müssen mit einem Buchstaben beginnen.
> - Schlüssel müssen mindestens zwei Zeichen lang sein.
> - Schlüssel dürfen nur Buchstaben, Zahlen und den Unterstrich (_) enthalten.
> - Werte können bis zu 512 Zeichen lang sein.
> - Sie können bis zu 50 Schlüssel-Wert-Paare in einer GraphQL-API konfigurieren.

Konfiguration von Umgebungsvariablen (API)

Um eine Umgebungsvariable festzulegen APIs, können Sie verwendenPutGraphqlApiEnvironmentVariables. Der entsprechende CLI-Befehl lautetput-graphql-api-environment-variables.

Um eine Umgebungsvariable mit abzurufen APIs, können Sie verwendenGetGraphqlApiEnvironmentVariables. Der entsprechende CLI-Befehl lautetget-graphql-api-environment-variables.

Der Befehl muss die API-ID und eine Liste der Umgebungsvariablen enthalten:

```
aws appsync put-graphql-api-environment-variables \
```

```
  --api-id "<api-id>" \
  --environment-variables '{"key1":"value1","key2":"value2", …}'
```

Im folgenden Beispiel werden zwei Umgebungsvariablen in einer API mit der ID für die `abcdefghijklmnopqrstuvwxyz` Verwendung des `put-graphql-api-environment-variables` Befehls festgelegt:

```
aws appsync put-graphql-api-environment-variables \
  --api-id "abcdefghijklmnopqrstuvwxyz" \
  --environment-variables '{"USER_TABLE":"users_prod","DEBUG":"true"}'
```

Beachten Sie, dass beim Anwenden von Umgebungsvariablen mit dem `put-graphql-api-environment-variables` Befehl der Inhalt der Struktur der Umgebungsvariablen überschrieben wird. Das bedeutet, dass vorhandene Umgebungsvariablen verloren gehen. Um bestehende Umgebungsvariablen beizubehalten, wenn Sie neue hinzufügen, nehmen Sie alle vorhandenen Schlüssel-Wert-Paare zusammen mit den neuen in Ihre Anfrage auf. Wenn Sie anhand des obigen Beispiels etwas hinzufügen möchten`"EMPTY":""`, können Sie Folgendes tun:

```
aws appsync put-graphql-api-environment-variables \
  --api-id "abcdefghijklmnopqrstuvwxyz" \
  --environment-variables '{"USER_TABLE":"users_prod","DEBUG":"true", "EMPTY":""}'
```

Verwenden Sie den `get-graphql-api-environment-variables` folgenden Befehl, um die aktuelle Konfiguration abzurufen:

```
aws appsync get-graphql-api-environment-variables --api-id "<api-id>"
```

Unter Verwendung des obigen Beispiels könnten Sie den folgenden Befehl verwenden:

```
aws appsync get-graphql-api-environment-variables --api-id "abcdefghijklmnopqrstuvwxyz"
```

Das Ergebnis zeigt die Liste der Umgebungsvariablen zusammen mit ihren Schlüsselwerten:

```
{
    "environmentVariables": {
        "USER_TABLE": "users_prod",
        "DEBUG": "true",
        "EMPTY": ""
    }
```

```
}
```

Konfiguration von Umgebungsvariablen (CFN)

Sie können die folgende Vorlage verwenden, um Umgebungsvariablen zu erstellen:

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  GraphQLApiWithEnvVariables:
    Type: "AWS::AppSync::GraphQLApi"
    Properties:
      Name: "MyApiWithEnvVars"
      AuthenticationType: "AWS_IAM"
      EnvironmentVariables:
        EnvKey1: "non-empty"
        EnvKey2: ""
```

Umgebungsvariablen und zusammengeführt APIs

In Source definierte Umgebungsvariablen APIs sind auch in Ihrem Merged verfügbar APIs. Umgebungsvariablen in Merged APIs sind schreibgeschützt und können nicht aktualisiert werden. Beachten Sie, dass Ihre Umgebungsvariablenschlüssel in allen Quelldateien eindeutig sein müssen, APIs damit Ihre Zusammenführungen erfolgreich sind. Doppelte Schlüssel führen immer zu einem Fehler bei der Zusammenführung.

Abrufen von Umgebungsvariablen

Um Umgebungsvariablen in Ihrem Funktionscode abzurufen, rufen Sie den Wert aus dem `ctx.env` Objekt in Ihren Resolvern und Funktionen ab. Im Folgenden finden Sie einige Beispiele dafür in Aktion.

Publishing to Amazon SNS

> In diesem Beispiel sendet unser HTTP-Resolver eine Nachricht an ein Amazon SNS SNS-Thema. Der ARN des Themas ist erst bekannt, nachdem der Stack, der die GraphQL-API und das Thema definiert, bereitgestellt wurden.

```
/**
 * 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

In diesem Beispiel unterscheiden sich die Namen der DynamoDB-Tabelle, wenn die API für das Staging bereitgestellt wird oder sich bereits in der Produktion befindet. Der Resolver-Code muss nicht geändert werden. Die Werte der Umgebungsvariablen werden je nachdem, wo die API bereitgestellt wird, aktualisiert.

```
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
      },
    ],
  };
}
```

Konfiguration von Autorisierung und Authentifizierung zur Sicherung Ihres GraphQL APIs

AWS AppSync bietet die folgenden Autorisierungstypen zur Sicherung von GraphQL APIs: API-Schlüssel, Lambda, IAM, OpenID Connect und Cognito User Pools. Jede Option bietet eine andere Sicherheitsmethode:

1. API-Schlüsselautorisierung: Steuert die Drosselung für nicht authentifizierte Benutzer und bietet so eine APIs einfache Sicherheitsoption.
2. Lambda-Autorisierung: Aktiviert eine benutzerdefinierte Autorisierungslogik, die Funktionseingaben und -ausgaben detailliert erklärt.
3. IAM-Autorisierung: Nutzt den Signaturprozess AWS der Version 4, der eine differenzierte Zugriffskontrolle durch IAM-Richtlinien ermöglicht.
4. OpenID Connect-Autorisierung: Lässt sich in OIDC-konforme Dienste zur Benutzerauthentifizierung integrieren.
5. Cognito-Benutzerpools: Implementiert eine gruppenbasierte Zugriffskontrolle mithilfe der Benutzerverwaltungsfunktionen von Cognito.

Autorisierungstypen

Es gibt fünf Möglichkeiten, Anwendungen für die Interaktion mit Ihrer AWS AppSync GraphQL-API zu autorisieren. Sie geben an, welchen Autorisierungstyp Sie verwenden, indem Sie in Ihrem AWS AppSync API- oder CLI-Aufruf einen der folgenden Autorisierungstypwerte angeben:

- **`API_KEY`**

  Für die Verwendung von API-Schlüsseln.

- **`AWS_LAMBDA`**

  Für die Verwendung einer AWS Lambda Funktion.

- **`AWS_IAM`**

  Für die Verwendung von AWS Identity and Access Management (IAM -) Berechtigungen.

- **`OPENID_CONNECT`**

  Für die Verwendung Ihres OpenID Connect-Anbieters.

- **`AMAZON_COGNITO_USER_POOLS`**

  Für die Verwendung eines Amazon Cognito Cognito-Benutzerpools.

Diese grundlegenden Autorisierungstypen funktionieren für die meisten Entwickler. Für fortgeschrittenere Anwendungsfälle können Sie zusätzliche Autorisierungsmodi über die Konsole, die CLI und hinzufügen AWS CloudFormation. AWS AppSync Stellt für zusätzliche Autorisierungsmodi einen Autorisierungstyp bereit, der die oben aufgeführten Werte verwendet (d. `API_KEY` h.`AWS_LAMBDA`,`AWS_IAM`,`OPENID_CONNECT`, und`AMAZON_COGNITO_USER_POOLS`).

Wenn Sie `API_KEY``AWS_LAMBDA`, oder `AWS_IAM` als Haupt- oder Standardautorisierungstyp angeben, können Sie sie nicht erneut als einen der zusätzlichen Autorisierungsmodi angeben. Ebenso können Sie die zusätzlichen Autorisierungsmodi nicht duplizieren `API_KEY` `AWS_LAMBDA` oder `AWS_IAM` innerhalb der zusätzlichen Autorisierungsmodi verwenden. Sie können mehrere Amazon Cognito Cognito-Benutzerpools und OpenID Connect-Anbieter verwenden. Sie können jedoch keine doppelten Amazon Cognito Cognito-Benutzerpools oder OpenID Connect-Anbieter zwischen dem Standardautorisierungsmodus und einem der zusätzlichen Autorisierungsmodi verwenden. Sie können verschiedene Clients für Ihren Amazon Cognito Cognito-Benutzerpool oder OpenID Connect-Anbieter angeben, indem Sie den entsprechenden regulären Konfigurationsausdruck verwenden.

Wenn Sie Änderungen an Ihrer API-Konfiguration speichern, AWS AppSync beginnt die Übertragung der Änderungen. Solange Ihre Konfigurationsänderung nicht übernommen wurde, werden AWS AppSync weiterhin Ihre Inhalte aus der vorherigen Konfiguration bereitgestellt. Sobald Ihre Konfigurationsänderung übernommen wurde, wird AWS AppSync sofort mit der Bereitstellung Ihrer Inhalte auf der Grundlage der neuen Konfiguration begonnen. Während AWS AppSync Ihre Änderungen für eine API übertragen werden, können wir nicht anhand der vorherigen Konfiguration oder der neuen Konfiguration feststellen, ob die API Ihre Inhalte bereitstellt.

API_KEY Autorisierung

Für nicht authentifizierte ist eine strengere Drosselung APIs erforderlich als für authentifizierte. APIs Eine Möglichkeit zur Kontrolle der Ablehnung für nicht authentifizierte GraphQL-Endpunkte ist die Verwendung von API-Schlüsseln. Ein API-Schlüssel ist ein hartcodierter Wert in Ihrer Anwendung,

der vom AWS AppSync-Service beim Erstellen eines nicht authentifizierten GraphQL-Endpunktes generiert wird. Sie können API-Schlüssel von der Konsole, von der CLI oder von der AWS AppSync API-Referenz aus rotieren.

Console

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.
   a. Wählen Sie im APIs Dashboard Ihre GraphQL-API aus.
   b. Wählen Sie in der Seitenleiste Einstellungen.
2. Wählen Sie unter Standardautorisierungsmodus die Option API-Schlüssel aus.
3. Wählen Sie in der Tabelle mit den API-Schlüsseln die Option API-Schlüssel hinzufügen aus.

   In der Tabelle wird ein neuer API-Schlüssel generiert.

   - Um einen alten API-Schlüssel zu löschen, wählen Sie den API-Schlüssel in der Tabelle aus und wählen Sie dann Löschen.
4. Wählen Sie unten auf der Seite die Option Save aus.

CLI

1. Falls Sie dies noch nicht getan haben, konfigurieren Sie Ihren Zugriff auf die AWS CLI. Weitere Informationen finden Sie unter Grundlagen der Konfiguration.
2. Erstellen Sie ein GraphQL-API-Objekt, indem Sie den `update-graphql-api`Befehl ausführen.

   Sie müssen zwei Parameter für diesen speziellen Befehl eingeben:

   1. Die `api-id` Ihrer GraphQL-API.
   2. Das Neue `name` Ihrer API. Sie können dasselbe verwenden`name`.
   3. Das`authentication-type`, was sein wird`API_KEY`.

   > **Note**
   >
   > Es gibt andere Parameter wie diesen`Region`, die konfiguriert werden müssen, aber normalerweise werden sie standardmäßig auf Ihre CLI-Konfigurationswerte gesetzt.

Ein Beispielbefehl könnte wie folgt aussehen:

```
aws appsync update-graphql-api --api-id abcdefghijklmnopqrstuvwxyz --name
 TestAPI --authentication-type API_KEY
```

Eine Ausgabe wird in der CLI zurückgegeben. Hier ist ein Beispiel in 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"
    }
}
```

API-Schlüssel sind für bis zu 365 Tage konfigurierbar und Sie können diese ab dem Ablaufdatum für bis zu weitere 365 Tage verlängern. API-Schlüssel werden für Entwicklungszwecke oder Anwendungsfälle empfohlen, bei denen die Bereitstellung einer öffentliche API sicher ist.

Auf dem Client ist der API-Schlüssel mit dem Header `x-api-key` angegeben.

Wenn z. B. der Wert Ihres `API_KEY 'ABC123'` lautet, können Sie über `curl` eine GraphQL-Abfrage senden. Gehen Sie dazu wie folgt vor:

```
$ curl -XPOST -H "Content-Type:application/graphql" -H "x-api-key:ABC123" -d
 '{ "query": "query { movies { id } }" }' https://YOURAPPSYNCENDPOINT/graphql
```

AWS_LAMBDA Autorisierung

Sie können Ihre eigene API-Autorisierungslogik mithilfe einer AWS Lambda Funktion implementieren. Sie können eine Lambda-Funktion entweder für Ihren primären oder sekundären Autorisierer verwenden, aber es kann nur eine Lambda-Autorisierungsfunktion pro API geben. Bei der Verwendung von Lambda-Funktionen für die Autorisierung gilt Folgendes:

- Wenn für die API die Modi `AWS_LAMBDA` und `AWS_IAM` Autorisierung aktiviert sind, kann die SigV4-Signatur nicht als `AWS_LAMBDA` Autorisierungstoken verwendet werden.
- Wenn für die API die `OPENID_CONNECT` Autorisierungsmodi `AWS_LAMBDA` und oder der `AMAZON_COGNITO_USER_POOLS` Autorisierungsmodus aktiviert sind, kann das OIDC-Token nicht als Autorisierungstoken verwendet werden. `AWS_LAMBDA` Beachten Sie, dass es sich bei dem OIDC-Token um ein Bearer-Schema handeln kann.
- Eine Lambda-Funktion darf nicht mehr als 5 MB an Kontextdaten für Resolver zurückgeben.

Wenn Ihr Autorisierungstoken beispielsweise lautet `'ABC123'`, können Sie eine GraphQL-Abfrage über curl wie folgt senden:

```
$ curl -XPOST -H "Content-Type:application/graphql" -H "Authorization:ABC123" -d
 '{ "query":
        "query { movies { id } }" }' https://YOURAPPSYNCENDPOINT/graphql
```

Lambda-Funktionen werden vor jeder Abfrage oder Mutation aufgerufen. Der Rückgabewert kann basierend auf der API-ID und dem Authentifizierungstoken zwischengespeichert werden. Wenn eine Lambda-Autorisierungsantwort weniger als 1.048.576 Byte umfasst, wird die Antwort für nachfolgende Anfragen AWS AppSync zwischengespeichert. Wenn die Antwort des Lambda-Autorisierers gleich oder größer als 1.048.576 Byte ist, wird die Antwort AWS AppSync nicht zwischengespeichert und der Lambda-Authorizer wird für jede eingehende Anfrage aufgerufen. Um die Leistung zu optimieren und die Kosten für Lambda-Aufrufe zu minimieren, empfehlen wir Ihnen, Ihre Lambda-Autorisierungsantworten auf 1.048.576 Byte zu beschränken. Standardmäßig ist das Caching nicht aktiviert, aber es kann auf API-Ebene oder durch Setzen des Werts im Rückgabewert einer Funktion aktiviert werden. `ttlOverride`

Bei Bedarf kann ein regulärer Ausdruck angegeben werden, der Autorisierungstoken validiert, bevor die Funktion aufgerufen wird. Diese regulären Ausdrücke werden verwendet, um zu überprüfen, ob ein Autorisierungstoken das richtige Format hat, bevor Ihre Funktion aufgerufen wird. Jede Anfrage,

die ein Token verwendet, das nicht mit diesem regulären Ausdruck übereinstimmt, wird automatisch abgelehnt.

Lambda-Funktionen, die für die Autorisierung verwendet werden, erfordern, `appsync.amazonaws.com` dass eine Hauptrichtlinie auf sie angewendet wird, AWS AppSync damit sie aufgerufen werden können. Diese Aktion wird automatisch in der AWS AppSync Konsole ausgeführt. Die AWS AppSync Konsole entfernt die Richtlinie nicht. Weitere Informationen zum Anhängen von Richtlinien an Lambda-Funktionen finden Sie unter Ressourcenbasierte Richtlinien im Entwicklerhandbuch. AWS Lambda

Die von Ihnen angegebene Lambda-Funktion erhält ein Ereignis mit der folgenden Form:

```
{
    "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
    }
}
```

Das `event` Objekt enthält die Header, die in der Anfrage vom Anwendungsclient an gesendet wurden. AWS AppSync

Die Autorisierungsfunktion muss mindestens einen booleschen Wert zurückgeben`isAuthorized`, der angibt, ob die Anfrage autorisiert ist. AWS AppSync erkennt die folgenden Schlüssel, die von Lambda-Autorisierungsfunktionen zurückgegeben wurden:

> **Note**
>
> Der Wert für den `operationName` in der Operation `requestContext` for a WebSocket connect wird mit "`DeepDish:Connect`" AWS AppSync gesetzt.

Funktionsliste

`isAuthorized`(boolescher Wert, erforderlich)

Ein boolescher Wert, der angibt, ob der Wert in berechtigt `authorizationToken` ist, Aufrufe an die GraphQL-API zu tätigen.

Wenn dieser Wert wahr ist, wird die Ausführung der GraphQL-API fortgesetzt. Wenn dieser Wert falsch ist, `UnauthorizedException` wird an ausgelöst

`deniedFields`(Liste der Zeichenketten, optional)

Eine Liste, in die zwangsweise geändert wird`null`, auch wenn ein Wert von einem Resolver zurückgegeben wurde.

Jeder Artikel ist entweder ein vollständig qualifizierter Feld-ARN in der Form von `arn:aws:appsync:`*`us-east-1`*`:`*`111122223333`*`:apis/`*`GraphQLApiId`*`/types/`*`TypeName`*`/fields/`*`FieldName` oder eine Kurzform von*`TypeName`*`.`*`FieldName`. Das vollständige ARN-Formular sollte verwendet werden, wenn sich zwei einen Lambda-Funktionsautorisierer APIs teilen und es zu Unklarheiten zwischen gemeinsamen Typen und Feldern zwischen den beiden kommen kann. APIs

`resolverContext`(JSON-Objekt, optional)

Ein JSON-Objekt, das wie `$ctx.identity.resolverContext` in Resolver-Vorlagen sichtbar ist. Zum Beispiel, wenn die folgende Struktur von einem Resolver zurückgegeben wird:

```
{
  "isAuthorized":true
  "resolverContext": {
    "banana":"very yellow",
    "apple":"very green"
  }
}
```

Der Wert von `ctx.identity.resolverContext.apple` in Resolver-Vorlagen wird "“ `very green` sein. Das `resolverContext` Objekt unterstützt nur Schlüssel-Wert-Paare. Verschachtelte Schlüssel werden nicht unterstützt.

> **Warning**
>
> Die Gesamtgröße dieses JSON-Objekts darf 5 MB nicht überschreiten.

`ttlOverride`(Ganzzahl, optional)

Die Anzahl der Sekunden, für die die Antwort zwischengespeichert werden soll. Wenn kein Wert zurückgegeben wird, wird der Wert aus der API verwendet. Wenn dieser Wert 0 ist, wird die Antwort nicht zwischengespeichert.

Lambda-Autorisierer haben ein Timeout von 10 Sekunden. Wir empfehlen, Funktionen so zu entwerfen, dass sie so schnell wie möglich ausgeführt werden, um die Leistung Ihrer API zu skalieren.

Mehrere AWS AppSync APIs können sich eine einzige Authentifizierungs-Lambda-Funktion teilen. Die kontoübergreifende Verwendung von Autorisierern ist nicht zulässig.

Wenn Sie eine Autorisierungsfunktion von mehreren Personen gemeinsam nutzen APIs, beachten Sie, dass Kurzform-Feldnamen (*`typename.fieldname`*) versehentlich Felder verbergen können. Um ein Feld in eindeutig zu kennzeichnen`deniedFields`, können Sie einen eindeutigen Feld-ARN in der Form von angeben. `arn:aws:appsync:`*`region`*`:`*`accountId`*`:apis/`*`GraphQLApiId`*`/types/`*`typeName`*`/fields/`*`fieldName`*

Um eine Lambda-Funktion als Standard-Autorisierungsmodus hinzuzufügen in AWS AppSync:

Console

1. Melden Sie sich bei der AWS AppSync Konsole an und navigieren Sie zu der API, die Sie aktualisieren möchten.
2. Navigieren Sie zur Einstellungsseite für Ihre API.

   Ändern Sie die Autorisierung auf API-Ebene auf. AWS Lambda
3. Wählen Sie den AWS-Region und den Lambda-ARN aus, für den API-Aufrufe autorisiert werden sollen.

   > **Note**
   >
   > Die entsprechende Prinzipalrichtlinie wird automatisch hinzugefügt, sodass AWS AppSync Sie Ihre Lambda-Funktion aufrufen können.
4. Legen Sie optional die Antwort-TTL und den regulären Ausdruck für die Token-Validierung fest.

AWS CLI

1. Hängen Sie die folgende Richtlinie an die verwendete Lambda-Funktion an:

```
aws lambda add-permission --function-name "my-function" --statement-id "appsync"
 --principal appsync.amazonaws.com --action lambda:InvokeFunction --output text
```

> **Important**
>
> Wenn Sie möchten, dass die Richtlinie der Funktion an eine einzelne GraphQL-API gebunden ist, können Sie diesen Befehl ausführen:
>
> ```
> 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. Aktualisieren Sie Ihre AWS AppSync API, um die angegebene Lambda-Funktion ARN als Autorisierer zu verwenden:

```
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**
>
> Sie können auch andere Konfigurationsoptionen wie den regulären Token-Ausdruck einbeziehen.

Das folgende Beispiel beschreibt eine Lambda-Funktion, die die verschiedenen Authentifizierungs- und Fehlerzustände demonstriert, die eine Lambda-Funktion haben kann, wenn sie als AWS AppSync Autorisierungsmechanismus verwendet wird:

```
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 {}
```

Umgehung der Autorisierungsbeschränkungen für SigV4- und OIDC-Tokens

Die folgenden Methoden können verwendet werden, um das Problem zu umgehen, dass Sie Ihre SigV4-Signatur oder Ihr OIDC-Token nicht als Lambda-Autorisierungstoken verwenden können, wenn bestimmte Autorisierungsmodi aktiviert sind.

Wenn Sie die SigV4-Signatur als Lambda-Autorisierungstoken verwenden möchten, wenn die Autorisierungsmodi `AWS_IAM` und die `AWS_LAMBDA` Autorisierungsmodi für AWS AppSync die API aktiviert sind, gehen Sie wie folgt vor:

- Um ein neues Lambda-Autorisierungstoken zu erstellen, fügen Sie der SigV4-Signatur zufällige and/or Suffix-Präfixe hinzu.
- Um die ursprüngliche Sigv4-Signatur abzurufen, aktualisieren Sie Ihre Lambda-Funktion, indem Sie die zufälligen and/or Präfix-Suffixe aus dem Lambda-Autorisierungstoken entfernen. Verwenden Sie dann die ursprüngliche SigV4-Signatur zur Authentifizierung.

Wenn Sie das OIDC-Token als Lambda-Autorisierungstoken verwenden möchten, wenn der Autorisierungsmodus oder die `OPENID_CONNECT` Autorisierungsmodi `AMAZON_COGNITO_USER_POOLS` und die `AWS_LAMBDA` Autorisierungsmodi für AWS AppSync die API aktiviert sind, gehen Sie wie folgt vor:

- Um ein neues Lambda-Autorisierungstoken zu erstellen, fügen Sie dem OIDC-Token zufällige and/or Suffix-Präfixe hinzu. Das Lambda-Autorisierungstoken sollte kein Bearer-Schema-Präfix enthalten.
- Um das ursprüngliche OIDC-Token abzurufen, aktualisieren Sie Ihre Lambda-Funktion, indem Sie die zufälligen and/or Präfix-Suffixe aus dem Lambda-Autorisierungstoken entfernen. Verwenden Sie dann das ursprüngliche OIDC-Token zur Authentifizierung.

AWS_IAM Autorisierung

Dieser Autorisierungstyp erzwingt den AWS Signaturprozess der Signaturversion 4 auf der GraphQL-API. Sie können vordefinierte Zugriffsrichtlinien für Identity and Access Management (IAM) mit diesem Autorisierungstyp verknüpfen. Ihre Anwendung kann diese Zuordnung nutzen, indem sie einen Zugriffsschlüssel (der aus einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel besteht) oder kurzlebige, temporäre Anmeldeinformationen verwendet, die von Amazon Cognito Federated Identities bereitgestellt werden.

Wenn Sie eine Rolle möchten, die Zugriff darauf hat, alle Datenoperationen durchzuführen:

JSON

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

```
            "appsync:GraphQL"
         ],
         "Resource": [
            "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/*"
         ]
      }
   ]
}
```

Sie finden es auf `YourGraphQLApiId` der API-Hauptseite in der AppSync Konsole direkt unter dem Namen Ihrer API. Alternativ können Sie sie mit der CLI abrufen: `aws appsync list-graphql-apis`.

Wenn Sie den Zugriff auf nur bestimmte GraphQL-Operationen einschränken möchten, können Sie dies für die Stammfelder `Query`, `Mutation` und `Subscription` tun.

JSON

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

Angenommen, Sie haben das folgende Schema und Sie möchten den Zugriff zum Abrufen aller Beiträge beschränken:

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

type Query {
   posts:[Post!]!
}

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

Die entsprechende IAM-Richtlinie für eine Rolle (die Sie beispielsweise an einen Amazon Cognito Cognito-Identitätspool anhängen könnten) würde wie folgt aussehen:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
            "appsync:GraphQL"
            ],
            "Resource": [
                "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/
types/Query/fields/posts"
            ]
        }
    ]
}
```

OPENID_CONNECT Autorisierung

Dieser Autorisierungstyp erzwingt OpenID Connect (OIDC) -Token, die von einem OIDC-kompatiblen Dienst bereitgestellt werden. Ihre Anwendung kann Benutzer und Berechtigungen nutzen, die von Ihrem OIDC-Anbieter zur Kontrolle des Zugriffs definiert wurden.

Eine Aussteller-URL ist der einzige erforderliche Konfigurationswert, den Sie AWS AppSync bereitstellen, (z. B. `https://auth.example.com`). Diese URL muss über HTTPS adressierbar sein. AWS AppSync hängt `/.well-known/openid-configuration` an die Aussteller-URL an und lokalisiert die OpenID-Konfiguration `https://auth.example.com/.well-known/openid-configuration` gemäß der OpenID Connect Discovery-Spezifikation. Es wird erwartet, dass unter dieser URL ein RFC5785konformes JSON-Dokument abgerufen wird. Dieses JSON-Dokument muss einen `jwks_uri` Schlüssel enthalten, der auf das JWKS-Dokument (JSON Web Key Set) mit den Signaturschlüsseln verweist. AWS AppSync erfordert, dass das JWKS die JSON-Felder und enthält. `kty` `kid`

AWS AppSync unterstützt eine Vielzahl von Signaturalgorithmen.

Signaturalgorithmen
RS256
RS384
RS512
PS256
PS384
PS512
HS256
HS384
HS512
ES256

Signaturalgorithmen
ES384
ES512

Wir empfehlen die Verwendung der RSA-Algorithmen. Token, die vom Anbieter ausgestellt wurden, müssen die Uhrzeit, zu der das Token ausgestellt wurde (`iat`), und den Zeitpunkt, zu dem es authentifiziert wurde (`auth_time`), enthalten. Sie können TTL-Werte für die ausgestellte Zeit (`iatTTL`) und die Authentifizierungszeit (`authTTL`) in Ihrer OpenID Connect-Konfiguration zur zusätzlichen Validierung bereitstellen. Wenn Ihr Anbieter mehrere Anwendungen autorisiert, können Sie auch einen regulären Ausdruck (`clientId`) angeben, der von der Client-ID zur Autorisierung verwendet wird. Wenn das in Ihrer OpenID Connect-Konfiguration vorhanden `clientId` ist, AWS AppSync validiert es den Anspruch, indem es verlangt, dass der `clientId` entweder mit dem `aud` oder `azp` -Anspruch im Token übereinstimmt.

Um mehrere Clients zu validieren, IDs verwenden Sie den Pipeline-Operator („|“), der im regulären Ausdruck ein „oder“ ist. Wenn Ihre OIDC-Anwendung beispielsweise über vier Clients mit einem Client IDs wie 0A1S2D, 1F4G9H, 1J6L4B, 6 MG verfügt, würden Sie, um nur die ersten drei Clients zu validieren, 1F4G9H|1J6L4B|6 GS5 MG in das Feld Client-ID eingeben. IDs GS5

Wenn eine API mit mehreren Autorisierungstypen konfiguriert ist, AWS AppSync validiert sie den im JWT-Token vorhandenen Aussteller (ISS-Anspruch) anhand der Anforderungsheader, indem er mit der in der API-Konfiguration angegebenen Aussteller-URL verglichen wird. Wenn eine API jedoch nur mit OPENID_CONNECT Autorisierung konfiguriert ist, wird dieser Schritt zur Überprüfung der Aussteller-URL AWS AppSync übersprungen.

Autorisierung durch AMAZON_COGNITO_USER_POOLS

Dieser Autorisierungstyp erzwingt OIDC-Token, die von Amazon Cognito Cognito-Benutzerpools bereitgestellt werden. Ihre Anwendung kann die Benutzer und Gruppen sowohl in Ihren Benutzerpools als auch in Benutzerpools von einem anderen AWS Konto nutzen und diese mit GraphQL-Feldern verknüpfen, um den Zugriff zu kontrollieren.

Wenn Sie Amazon Cognito User Pools verwenden, können Sie Gruppen erstellen, denen Benutzer angehören. Diese Informationen sind in einem JWT-Token kodiert, an das Ihre Anwendung beim Senden von AWS AppSync GraphQL-Operationen in einem Autorisierungsheader sendet. Sie können GraphQL-Richtlinien für das Schema verwenden, um zu steuern, welche Gruppen welche

Resolver auf einem Feld aufrufen können, wodurch Ihren Kunden mehr kontrollierter Zugriff gewährt wird.

Angenommen, Sie haben das folgende GraphQL-Schema:

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

type Query {
   posts:[Post!]!
}

type Mutation {
   addPost(id:ID!, title:String!):Post!
}
...
```

Wenn Sie zwei Gruppen in Amazon Cognito Cognito-Benutzerpools haben — Blogger und Leser — und Sie die Anzahl der Leser einschränken möchten, sodass sie keine neuen Einträge hinzufügen können, sollte Ihr Schema wie folgt aussehen:

```
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"])
}
...
```

Beachten Sie, dass Sie die `@aws_auth` Direktive weglassen können, wenn Sie standardmäßig eine bestimmte grant-or-deny Zugriffsstrategie verwenden möchten. Sie können die grant-or-deny

Strategie in der Benutzerpool-Konfiguration angeben, wenn Sie Ihre GraphQL-API über die Konsole oder über den folgenden CLI-Befehl erstellen:

```
$ 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"}'
```

Verwenden zusätzlicher Autorisierungsmodi

Wenn Sie zusätzliche Autorisierungsmodi hinzufügen, können Sie die Autorisierungseinstellung direkt auf der AWS AppSync GraphQL-API-Ebene konfigurieren (d. h. das `authenticationType` Feld, das Sie direkt für das `GraphqlApi` Objekt konfigurieren können) und sie fungiert als Standard im Schema. Dies bedeutet, dass jeder Typ, der keine bestimmte Richtlinie hat, die Autorisierungseinstellung auf API-Ebene übergeben muss.

Auf Schemaebene können Sie zusätzliche Autorisierungsmodi mithilfe von Anweisungen für das Schema angeben. Sie können Autorisierungsmodi für einzelne Felder im Schema angeben. Beispielsweise würden Sie für die `API_KEY`-Autorisierung für Schemaobjekttypdefinitionen/-felder `@aws_api_key` verwenden. Die folgenden Anweisungen werden für Schemafelder und Objekttypdefinitionen unterstützt:

- `@aws_api_key` – Zur Angabe, dass das Feld `API_KEY`-autorisiert ist.
- `@aws_iam` – Zur Angabe, dass das Feld `AWS_IAM`-autorisiert ist.
- `@aws_oidc` – Zur Angabe, dass das Feld `OPENID_CONNECT`-autorisiert ist.
- `@aws_cognito_user_pools` – Zur Angabe, dass das Feld `AMAZON_COGNITO_USER_POOLS`-autorisiert ist.
- `@aws_lambda` – Zur Angabe, dass das Feld `AWS_LAMBDA`-autorisiert ist.

Sie können die Richtlinie `@aws_auth` nicht zusammen mit zusätzlichen Autorisierungsmodi verwenden. `@aws_auth` funktioniert nur im Kontext der `AMAZON_COGNITO_USER_POOLS`-Autorisierung ohne zusätzliche Autorisierungsmodi. Sie können jedoch die `@aws_cognito_user_pools`-Anweisung anstelle der `@aws_auth`-Richtlinie verwenden, indem Sie dieselben Argumente verwenden. Der Hauptunterschied zwischen den beiden besteht darin, dass Sie für jede Feld- und Objekttypdefinition `@aws_cognito_user_pools` angeben können.

Um zu verstehen, wie die zusätzlichen Autorisierungsmodi funktionieren und wie sie in einem Schema angegeben werden können, sehen wir uns das folgende Schema an:

```
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!
}
...
```

Gehen Sie für dieses Schema davon aus, dass `AWS_IAM` dies der Standardautorisierungstyp auf der AWS AppSync GraphQL-API ist. Dies bedeutet, dass Felder, die keine Richtlinie haben, mit `AWS_IAM` geschützt werden. Dies ist beispielsweise der Fall für das Feld `getPost` des Typs `Query`. Mit Schemarichtlinien können Sie mehr als einen Autorisierungsmodus verwenden. Sie können beispielsweise einen zusätzlichen Autorisierungsmodus in der AWS AppSync GraphQL-API `API_KEY` konfiguriert haben, und Sie können ein Feld mithilfe der `@aws_api_key` Direktive markieren (z. B. `getAllPosts` in diesem Beispiel). Richtlinien funktionieren auf Feldebene, sodass Sie `API_KEY` auch Zugriff auf den Typ `Post` gewähren müssen. Sie können dies entweder tun,

indem Sie jedes Feld des Typs `Post` mit einer Richtlinie markieren oder den Typ `Post` mit der Richtlinie `@aws_api_key` markieren.

Um den Zugriff auf Felder des Typs `Post` weiter einzuschränken, können Sie Anweisungen für einzelne Felder des Typs `Post` verwenden, wie im Folgenden gezeigt.

Sie können beispielsweise ein `restrictedContent`-Feld zum Typ `Post` hinzufügen und den Zugriff darauf mithilfe der Richtlinie `@aws_iam` einschränken. `AWS_IAM`-authentifizierte Anfragen könnten auf `restrictedContent` zugreifen, `API_KEY`-Anfragen wären dazu jedoch nicht fähig.

```
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
}
...
```

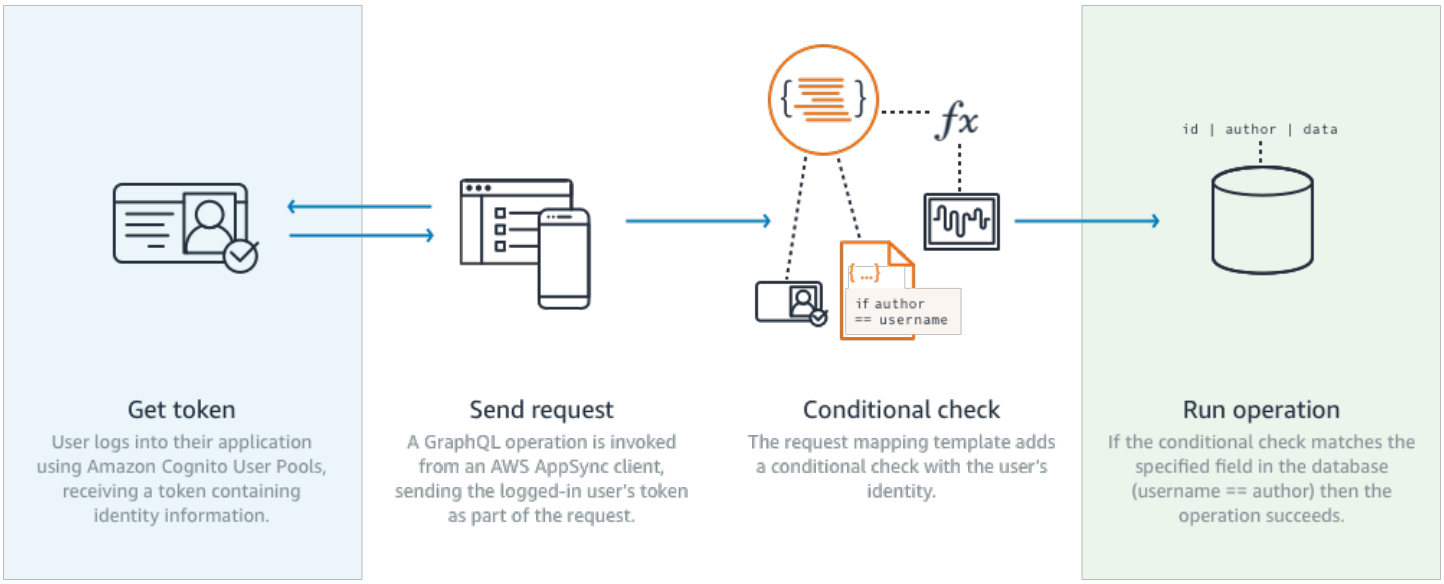
Differenzierte Zugriffskontrolle

Die vorstehenden Informationen zeigen, wie Sie den Zugriff auf bestimmte GraphQL-Felder beschränken oder gewähren. Wenn Sie Zugriffskontrollen für die Daten basierend auf bestimmten Bedingungen festlegen möchten (z. B. basierend auf dem Benutzer, der einen Aufruf ausführt, oder ob der Benutzer Eigentümer der Daten ist), können Sie dazu Zuweisungsvorlagen in Ihren Resolvern verwenden. Sie können auch eine komplexere Geschäftslogik durchführen, die wir in Filtern von Informationen beschreiben.

In diesem Abschnitt wird gezeigt, wie Sie mithilfe einer DynamoDB-Resolver-Mapping-Vorlage Zugriffskontrollen für Ihre Daten einrichten.

Bevor Sie fortfahren, sollten Sie, falls Sie mit Mapping-Vorlagen in nicht vertraut sind AWS AppSync, die Resolver-Mapping-Vorlagenreferenz und die Resolver-Mapping-Vorlagenreferenz für DynamoDB lesen.

Gehen Sie im folgenden Beispiel mit DynamoDB davon aus, dass Sie das vorherige Blogbeitragsschema verwenden und dass nur Benutzer, die einen Beitrag erstellt haben, ihn bearbeiten dürfen. Im Auswertungsverfahren würde der Benutzer dann z. B. mithilfe von Amazon Cognito-Benutzerpools Anmeldeinformationen in seiner Anwendung erhalten und anschließend diese Anmeldeinformationen als Teil einer GraphQL-Operation weiterleiten. Die Zuweisungsvorlage wird dann einen Wert aus den Anmeldeinformationen (z. B. den Benutzernamen) in einer bedingten Anweisung ersetzen, die anschließend mit einem Wert in Ihrer Datenbank verglichen wird.



Um diese Funktionalität zu erhalten, fügen Sie ein GraphQL-Feld von `editPost` folgendermaßen hinzu:

```
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!
}
...
```

Die Resolver-Zuweisungsvorlage für `editPost` (siehe Beispiel am Ende dieses Abschnitts) muss einen logischen Abgleich Ihres Datenspeichers durchführen, damit nur der Benutzer, der einen Beitrag erstellt hat, diesen auch bearbeiten kann. Da es sich um eine Bearbeitungsoperation handelt, entspricht sie einer `UpdateItem` in DynamoDB. Sie können vor der Durchführung der Aktion eine bedingte Prüfung mithilfe von Kontext, der zur Benutzeridentitätsvalidierung weitergeleitet wurde, vornehmen. Dieser wird in einem `Identity`-Objekt gespeichert, das die folgenden Werte hat:

```
{
   "accountId" : "12321434323",
   "cognitoIdentityPoolId" : "",
   "cognitoIdentityId" : "",
   "sourceIP" : "",
   "caller" : "ThisistheprincipalARN",
   "username" : "username",
   "userArn" : "Sameasabove"
}
```

Um dieses Objekt in einem `UpdateItem` DynamoDB-Aufruf zu verwenden, müssen Sie die Benutzeridentitätsinformationen zum Vergleich in der Tabelle speichern. Als Erstes muss Ihre `addPost`-Mutation den Ersteller speichern. Als Zweites muss Ihre `editPost`-Mutation vor dem Update die bedingte Prüfung ausführen.

Hier ist ein Beispiel für den Resolver-Code für`addPost`, der die Benutzeridentität als Spalte speichert: `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;
```

Beachten Sie, dass für das `Author`-Attribut die Daten aus dem `Identity`-Objekt der Anwendung übernommen werden.

Abschließend noch ein Beispiel für den Resolver-Code für`editPost`, der den Inhalt des Blogbeitrags nur aktualisiert, wenn die Anfrage von dem Benutzer kommt, der den Beitrag erstellt hat:

```
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;
```

In diesem Beispiel wird ein verwendet`PutItem`, das alle Werte überschreibt, und nicht ein`UpdateItem`, aber das gleiche Konzept gilt für den `condition` Anweisungsblock.

Informationen filtern

Es kann vorkommen, dass Sie die Antwort von Ihrer Datenquelle nicht steuern können, aber keine unnötigen Informationen an Clients in einem erfolgreichen Lese- oder Schreibvorgang der Datenquelle senden möchten. In diesen Fällen können Sie Informationen mithilfe einer Antwortzuweisungsvorlage filtern.

Nehmen wir beispielsweise an, Sie haben keinen geeigneten Index in der DynamoDB-Tabelle für Ihren Blogbeitrag (z. B. einen Index für). `Author` Sie könnten den folgenden Resolver verwenden:

```
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;
}
```

Der Request-Handler ruft das Element ab, auch wenn der Anrufer nicht der Autor ist, der den Beitrag erstellt hat. Um zu verhindern, dass alle Daten zurückgegeben werden, überprüft der Antworthandler, ob der Aufrufer mit dem Autor des Elements übereinstimmt. Wenn der Aufrufer nicht den Prüfkriterien entspricht, wird nur eine Null-Antwort zurückgegeben.

Datenquellenzugriff

AWS AppSync kommuniziert mit Datenquellen mithilfe von Identity and Access Management (IAM) -Rollen und Zugriffsrichtlinien. Wenn Sie eine bestehende Rolle verwenden, muss eine Vertrauensrichtlinie hinzugefügt werden, um die Rolle übernehmen AWS AppSync zu können. Die Vertrauensbeziehung sieht etwa wie folgt aus:

JSON

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

Es ist wichtig, den Geltungsbereich der Zugriffsrichtlinie für die Rolle so zu verringern, dass nur die Berechtigungen für die minimal erforderliche Ressourcengruppe verfügbar sind. Wenn Sie die AppSync Konsole verwenden, um eine Datenquelle und eine Rolle zu erstellen, erfolgt dies automatisch für Sie. Wenn Sie jedoch eine integrierte Beispielvorlage aus der IAM-Konsole verwenden, um eine Rolle außerhalb der AWS AppSync-Konsole zu erstellen, wird der Berechtigungsumfang nicht automatisch auf eine Ressource beschränkt. Sie sollten dies tun, bevor Sie die Anwendung tatsächlich in Betrieb nehmen.

Anwendungsfälle der Zugriffskontrolle zur Sicherung von Anfragen und Antworten

Im Abschnitt Sicherheit haben Sie die unterschiedlichen Autorisierungsmodi für den Schutz Ihrer API kennengelernt und eine Einführung in detaillierte Autorisierungsmechanismen zum Verständnis der Konzepte und des Ablaufs erhalten. Da Sie mithilfe von GraphQL Resolver Mapping-Vorlagen logische Operationen an Daten ausführen AWS AppSync können, können Sie Daten beim Lesen oder Schreiben auf sehr flexible Weise schützen, indem Sie eine Kombination aus Benutzeridentität, Bedingungen und Dateneinspeisung verwenden.

Wenn Sie mit der Bearbeitung von AWS AppSync Resolvern nicht vertraut sind, lesen Sie das Programmierhandbuch.

Übersicht

Die Gewährung des Zugriffs auf Daten in einem System erfolgt traditionell über eine Zugriffskontrollmatrix, bei der sich die gewährten Berechtigungen aus einer Zeile (Ressource) und einer Spalte (Benutzer/Rolle) überschneiden.

AWS AppSync verwendet Ressourcen in Ihrem eigenen Konto und überträgt Identitätsinformationen (Benutzer/Rolle) in die GraphQL-Anfrage und -Antwort als Kontextobjekt, das Sie im Resolver verwenden können. Das bedeutet, dass Berechtigungen basierend auf der Resolver-Logik entweder für Schreib- oder Lesevorgänge entsprechend gewährt werden können. Befindet sich diese Logik auf Ressourcenebene, sodass beispielsweise nur bestimmte benannte Benutzer oder Gruppen Zugriff read/write auf eine bestimmte Datenbankzeile haben, müssen diese „Autorisierungsmetadaten“ gespeichert werden. AWS AppSync speichert keine Daten, daher müssen Sie diese Autorisierungsmetadaten zusammen mit den Ressourcen speichern, damit die Berechtigungen berechnet werden können. Autorisierungsmetadaten stellen in der Regel ein Attribut (Spalte) in einer DynamoDB-Tabelle dar, wie z. B. ein Owner oder eine Liste von Benutzern/Gruppen. Es können zum Beispiel die Attribute Readers und Writers vorliegen.

Im Allgemeinen bedeutet dies, dass Sie beim Lesen eines einzelnen Elements aus einer Datenquelle eine bedingte `#if () ... #end`-Anweisung in der Antwortvorlage ausführen, nachdem der Resolver den Lesevorgang aus der Datenquelle beendet hat. Bei der Prüfung werden normalerweise Benutzer- oder Gruppenwerte in `$context.identity` zur Prüfung der Mitgliedschaft anhand der Autorisierungsmetadaten verwendet, die von einem Lesevorgang zurückgegeben werden. Für mehrere Datensätze, wie z B. aus einer Tabelle `Scan` oder `Query` zurückgegebene Listen, senden

Sie die Bedingungsprüfung mithilfe ähnlicher Benutzer- oder Gruppenwerte als Teil der Operation an die Datenquelle.

Entsprechend wenden Sie beim Schreiben von Daten eine bedingte Anweisung auf die Aktion an (z. B. `PutItem` oder `UpdateItem`), um zu sehen, ob der Benutzer oder die Gruppe, der bzw. die eine Mutation vornimmt, dazu berechtigt ist. Die Bedingung verwendet häufig einen Wert in `$context.identity` zum Vergleich mit den Autorisierungsmetadaten dieser Ressource. Für beide Anforderungs- und Antwortvorlagen können Sie benutzerdefinierte Header von Clients verwenden, um Validierungsprüfungen durchzuführen.

Lesen von Daten

Wie oben erläutert, müssen die Autorisierungsmetadaten zum Durchführen einer Prüfung mit einer Ressource gespeichert oder an die GraphQL-Anforderung (Identität, Header usw.) weitergegeben werden. Um dies zu veranschaulichen, nehmen wir an, Sie haben die unten angegebene DynamoDB-Tabelle vorliegen:

ID	Data	PeopleCanAccess	GroupsCanAccess	Owner
123	{my: data,...}	[Mary, Joe]	[Admins, Editors]	Nadia

Der Primärschlüssel ist `id` und die Daten, auf die zugegriffen werden soll, entsprechen `Data`. Die anderen Spalten sind Beispiele für Prüfungen, die Sie für die Autorisierung durchführen können. `Owner`würde eine `String` Weile dauern `PeopleCanAccess` und `GroupsCanAccess` wäre `String Sets` so, wie in der Referenz zur Resolver-Mapping-Vorlage für DynamoDB beschrieben.

Das Diagramm in der Übersicht über die Resolver-Zuweisungsvorlage zeigt, wie die Antwortvorlage nicht nur das Kontextobjekt, sondern auch die Ergebnisse aus der Datenquelle enthält. Für GraphQL-Abfragen einzelner Elemente können Sie anhand der Antwortvorlage prüfen, ob der Benutzer diese Ergebnisse sehen darf, oder eine Autorisierungsfehlermeldung zurückgeben. Dies wird manchmal auch als „Autorisierungsfilter“ bezeichnet. Bei GraphQL-Abfragen, die Listen zurückgeben, ist es bei Verwendung eines Scans oder einer Abfrage effizienter, die Prüfung für die Anforderungsvorlage durchzuführen und Daten nur zurückzugeben, wenn eine Autorisierungsbedingung erfüllt ist. Die Implementierung lautet wie folgt:

1. GetItem - Autorisierungsprüfung für einzelne Datensätze. Dieser Vorgang wird mit `#if() ... #end`-Anweisungen ausgeführt.

2. Scan-/Abfrageoperationen – Die Autorisierungsprüfung besteht aus einer `"filter": {"expression":...}`-Anweisung. Gängige Prüfungen erfolgen auf Gleichheit (`attribute = :input`) oder darauf, ob ein Wert in einer Liste (`contains(attribute, :input)`) enthalten ist.

Im zweiten Fall steht `attribute` in beiden Anweisungen für den Spaltennamen des Datensatzes in einer Tabelle, wie z. B. `Owner` in unserem Beispiel oben. Sie können diesen Wert mit einem #-Zeichen versehen und `"expressionNames":{...}` verwenden. Dies ist aber nicht obligatorisch. `:input` ist eine Referenz auf den Wert, den Sie mit dem Datenbankattribut vergleichen möchten, das Sie in `"expressionValues":{...}` definieren. Diese Beispiele finden Sie weiter unten.

Anwendungsfall: Besitzer kann lesen

Wenn Sie unter Verwendung der Tabelle oben nur Daten zurückgegeben möchten, wenn `Owner == Nadia` für einen einzelnen Lesevorgang (`GetItem`) gilt, sieht Ihre Vorlage wie folgt aus:

```
#if($context.result["Owner"] == $context.identity.username)
    $utils.toJson($context.result)
#else
    $utils.unauthorized()
#end
```

An dieser Stelle möchten wir auf einige Punkte hinweisen, die für die verbleibenden Abschnitte relevant sind. Bei der Prüfung wird zunächst der benutzerfreundliche Anmeldename verwendet`$context.identity.username`, wenn Amazon Cognito-Benutzerpools verwendet werden, und der Nutzeridentität, wenn IAM verwendet wird (einschließlich Amazon Cognito Federated Identities). Es gibt noch andere Werte, die für einen Besitzer gespeichert werden müssen, z. B. den eindeutigen Wert „Amazon Cognito Identity“, der nützlich ist, wenn Anmeldungen von mehreren Standorten aus zusammengeführt werden. Sie sollten sich die Optionen ansehen, die in der Resolver Mapping Template Context Reference verfügbar sind.

Zweitens: Die bedingte Prüfung, die mit `$util.unauthorized()` antwortet, ist zwar optional, wird jedoch als bewährte Methode beim Entwerfen Ihrer GraphQL-API empfohlen.

Anwendungsfall: Hardcode-spezifischer Zugriff

```
// 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
```

Anwendungsfall: Filtern einer Ergebnisliste

Im vorherigen Beispiel haben Sie eine Prüfung für `$context.result` direkt durchführen können, da ein einzelnes Element zurückgegeben wurde. Bei einigen Operationen wie z. B. einem Scan werden jedoch mehrere Elemente in `$context.result.items` zurückgegeben. In diesen Fällen müssen Sie den Autorisierungsfilter anwenden, sodass nur Ergebnisse erscheinen, die der Benutzer sehen darf. Angenommen, für das `Owner` Feld war dieses Mal die Amazon Cognito IdentityID für den Datensatz festgelegt. Sie könnten dann die folgende Antwortzuordnungsvorlage verwenden, um zu filtern, dass nur die Datensätze angezeigt werden, die dem Benutzer gehörten:

```
#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)
```

Anwendungsfall: mehrere Personen können lesen

Eine weitere gängige Autorisierungsoption besteht darin, einer Gruppe von Benutzern das Lesen der Daten zu erlauben. Im folgenden Beispiel gibt `"filter":{"expression":...}` nur die Werte aus einem Tabellenscan zurück, wenn der Benutzer, der die GraphQL-Abfrage ausführt, im Set für `PeopleCanAccess` aufgelistet wird.

```
{
    "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)
        }
    }
}
```

Anwendungsfall: Gruppe kann lesen

Ähnlich wie im letzten Anwendungsfall kann es vorkommen, dass nur Benutzer in einer oder mehreren Gruppen Leserechte für bestimmte Elemente in einer Datenbank besitzen. Die Verwendung der `"expression": "contains()"`-Operation ist zwar ähnlich, es muss in der Gruppenmitgliedschaft jedoch ein logisches ODER zwischen allen Gruppen, denen der Benutzer angehören kann, berücksichtigt werden. In diesem Fall erstellen wir eine `$expression`-Anweisung unten für jede Gruppe, in der sich der Benutzer befindet, und übergeben diese an den Filter:

```
#set($expression = "")
#set($expressionValues = {})
#foreach($group in $context.identity.claims.get("cognito:groups"))
    #set( $expression = "${expression} contains(groupsCanAccess, :var
$foreach.count )" )
    #set( $val = {})
    #set( $test = $val.put("S", $group))
```

```
    #set( $values = $expressionValues.put(":var$foreach.count", $val))
    #if ( $foreach.hasNext )
    #set( $expression = "${expression} OR" )
    #end
#end
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "limit": #if(${context.arguments.count}) $util.toJson($context.arguments.count)
 #else 20 #end,
    "nextToken": #if(${context.arguments.nextToken})
 $util.toJson($context.arguments.nextToken) #else null #end,
    "filter":{
        "expression": "$expression",
        "expressionValues": $utils.toJson($expressionValues)
    }
}
```

Daten schreiben

Das Schreiben von Daten für Mutationen wird immer in der Anforderungszuweisungsvorlage gesteuert. Im Fall von DynamoDB-Datenquellen besteht der Schlüssel in der Verwendung einer entsprechenden `"condition":{"expression"...}"` zur Durchführung der Validierung anhand der Autorisierungsmetadaten in dieser Tabelle. Unter Sicherheit befindet sich ein Beispiel, das zum Prüfen des Felds `Author` in einer Tabelle verwendet werden kann. Die Anwendungsfälle in diesem Abschnitt veranschaulichen weitere Fälle dieser Art.

Anwendungsfall: mehrere Besitzer

Aus dem bereits verwendeten Beispieltabellendiagramm wird die `PeopleCanAccess`-Liste vorausgesetzt.

```
{
    "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)
        }
    }
}
```

Anwendungsfall: Die Gruppe kann einen neuen Datensatz erstellen

```
#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)
    }
```

```
}
```

Anwendungsfall: Die Gruppe kann einen vorhandenen Datensatz aktualisieren

```
#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)
    }
}
```

Öffentliche und private Aufzeichnungen

Mit den bedingten Filtern können Sie Daten auch als privat, öffentlich oder mit einer anderen booleschen Prüfung kennzeichnen. Diese Vorgehensweise lässt sich mit einem Autorisierungsfilter innerhalb der Antwortvorlage kombinieren. Mit dieser Prüfung können Daten vorübergehend ausgeblendet oder aus der Anzeige entfernt werden, ohne dass die Gruppenmitgliedschaft kontrolliert werden muss.

Nehmen wir beispielsweise an, Sie haben ein Attribut für jedes Element in Ihrer DynamoDB-Tabelle namens `public` entweder mit dem Wert `yes` oder no hinzugefügt. Die folgende Antwortvorlage kann für einen `GetItem`-Aufruf verwendet werden, um nur Daten anzuzeigen, wenn sich der Benutzer in einer Gruppe befindet, die über Zugriff verfügt, UND wenn diese Daten als öffentlich gekennzeichnet sind:

```
#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
```

Der obige Code kann auch ein logisches ODER (`||`) verwenden, um Benutzern das Lesen zu erlauben, wenn sie über die Berechtigung für einen Datensatz verfügen oder dieser öffentlich ist:

```
#if($hasPermission || $context.result.public == 'yes')
    $utils.toJson($context.result)
#else
    $utils.unauthorized()
#end
```

Im Allgemeinen sind die Standardoperatoren `==`, `!=`, `&&` und `||` beim Ausführen der Autorisierungsprüfungen nützlich.

Daten in Echtzeit

Sie können eine differenzierte Zugriffskontrolle auf GraphQL-Abonnements zu dem Zeitpunkt anwenden, zu dem ein Client ein Abonnement abschließt. Verwenden Sie dazu die gleichen Techniken, wie weiter oben in dieser Dokumentation beschrieben. Sie fügen dem Abonnementfeld einen Resolver an. An diesem Punkt können Sie Daten aus einer Datenquelle abfragen und eine

bedingte Logik entweder in der Anforderungs- oder der Antwortzuweisungsvorlage ausführen. Sie können auch zusätzliche Daten an den Client zurückgeben, z. B. die ersten Ergebnisse aus einem Abonnement, solange die Datenstruktur der Struktur des zurückgegebenen Typs in Ihrem GraphQL-Abonnement entspricht.

Anwendungsfall: Der Benutzer kann nur bestimmte Konversationen abonnieren

Ein häufiger Anwendungsfall für Echtzeitdaten mit GraphQL-Abonnements ist die Erstellung einer Messaging-App oder einer Anwendung für private Chats. Beim Erstellen einer Chat-Anwendung, die von mehreren Benutzern verwendet wird, können Unterhaltungen zwischen zwei Personen oder zwischen mehreren Personen stattfinden. Diese können in „Räume“ gruppiert werden, die privat oder öffentlich sind. Daher soll nur ein Benutzer autorisiert werden, um eine Unterhaltung zu abonnieren (die zwischen zwei Personen oder in einer Gruppe stattfinden kann), für die ihnen Zugriff gewährt wurde. Zu Demonstrationszwecken wird im Beispiel unten ein einfacher Anwendungsfall von einem Benutzer gezeigt, der eine privaten Nachricht an einen anderen Benutzer sendet. Das Setup umfasst zwei Amazon DynamoDB-Tabellen:

- Nachrichtentabelle: (Primärschlüssel) `toUser`, (Sortierschlüssel) `id`
- Berechtigungstabelle: (Primärschlüssel) `username`

Die Nachrichtentabelle speichert die tatsächlichen Nachrichten, die über eine GraphQL-Mutation gesendet werden. Die Berechtigungstabelle wird vom GraphQL-Abonnement im Hinblick auf die Autorisierung zur Client-Verbindungszeit geprüft. Im folgenden Beispiel wird davon ausgegangen, dass Sie das folgende GraphQL-Schema verwenden:

```
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
}
```

Einige der Standardoperationen, wie z. B., werden im Folgenden nicht behandelt`createUserPermissions()`, um die Abonnement-Resolver zu veranschaulichen, sondern sind Standardimplementierungen von DynamoDB-Resolvern. Stattdessen konzentrieren wir uns auf Autorisierungsabläufe für Abonnements mit Resolvern. Zum Senden einer Nachricht von einem Benutzer an einen anderen fügen Sie dem `sendMessage()`-Feld einen Resolver an und wählen die Nachrichten-Tabelle als Datenquelle mit der folgenden Anforderungsvorlage aus:

```
{
```

```
    "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),
    }
}
```

In diesem Beispiel verwenden wir `$context.identity.username`. Dadurch werden Benutzerinformationen für AWS Identity and Access Management unsere Amazon Cognito Cognito-Benutzer zurückgegeben. Die Antwortvorlage ist ein einfacher Pass-Through von `$util.toJson($ctx.result)`. Speichern Sie die Daten und wechseln Sie wieder zur Seite "Schema". Fügen Sie anschließend einen Resolver für das `newMessage()`-Abonnement mit der Tabelle Berechtigungen als Datenquelle und der folgenden Anforderungszuweisungsvorlage an:

```
{
    "version": "2018-05-29",
    "operation": "GetItem",
    "key": {
        "username": $util.dynamodb.toDynamoDBJson($ctx.identity.username),
    },
}
```

Führen Sie mit der folgenden Antwortzuweisungsvorlage die Autorisierungsprüfungen anhand der Daten aus der Tabelle Berechtigungen durch:

```
#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
```

In diesem Fall nehmen Sie drei Autorisierungsprüfungen vor. Die erste stellt sicher, dass ein Ergebnis zurückgegeben wird. Die zweite stellt sicher, dass der Benutzer keine Nachrichten abonniert, die für eine andere Person gedacht sind. Die dritte Option stellt sicher, dass der Benutzer beliebige Felder abonnieren darf, indem ein DynamoDB-Attribut als `isAuthorizedForSubscriptions` gespeichert überprüft wird. `BOOL`

Um die Dinge zu testen, könnten Sie sich mit Amazon Cognito Cognito-Benutzerpools und einem Benutzer namens „Nadia“ bei der AWS AppSync Konsole anmelden und dann das folgende GraphQL-Abonnement ausführen:

```
subscription AuthorizedSubscription {
    newMessage(toUser: "Nadia") {
        id
        toUser
        fromUser
        content
    }
}
```

Wenn in der Tabelle Permissions (Berechtigungen) ein Datensatz für das `username`-Schlüsselattribut `Nadia` enthalten ist, wobei `isAuthorizedForSubscriptions` auf `true` gesetzt ist, sehen Sie eine erfolgreiche Antwort. Wenn Sie einen anderen `username` in der `newMessage()`-Abfrage oben angeben, wird ein Fehler zurückgegeben.

Verwenden Sie AWS WAF zum Schutz Ihrer AWS AppSync APIs

AWS WAF ist eine Firewall für Webanwendungen, die zum Schutz von Webanwendungen und APIs vor Angriffen beiträgt. Sie ermöglicht es Ihnen, eine Reihe von Regeln zu konfigurieren, die als Web-Zugriffskontrollliste (Web Access Control List, Web-ACL) bezeichnet werden und Webanfragen auf der Grundlage von anpassbaren Websicherheitsregeln und -bedingungen, die Sie definieren, zulassen, blockieren oder überwachen (zählen). Wenn Sie Ihre AWS AppSync API integrieren AWS WAF, erhalten Sie mehr Kontrolle und Einblick in den von Ihrer API akzeptierten HTTP-Verkehr. Weitere Informationen AWS WAF dazu finden Sie im AWS WAF Entwicklerhandbuch unter So AWS WAF funktioniert es.

Sie können AWS WAF es verwenden, um Ihre AppSync API vor gängigen Web-Exploits wie SQL-Injection und Cross-Site Scripting (XSS) -Angriffen zu schützen. Diese können sich auf die Verfügbarkeit und Leistung der API auswirken, die Sicherheit gefährden oder übermäßige Ressourcen verbrauchen. Sie können beispielsweise Regeln zum Zulassen oder Blockieren

von Anforderungen von angegebenen IP-Adressbereichen, Anforderungen von CIDR-Blocks, Anforderungen aus einem bestimmten Land oder einer bestimmten Region, Anforderungen mit bösartigem SQL-Code oder Anforderungen mit bösartigem Skript erstellen.

Sie können auch Regeln erstellen, die mit einer bestimmten Zeichenfolge oder einem regulären Ausdrucksmuster in HTTP-Headern, einer Methode, einer Abfragezeichenfolge, einer URI und dem Anforderungstext (nur für die ersten 8 KB) übereinstimmen. Außerdem können Sie Regeln zum Blockieren von Angriffen von bestimmten Benutzeragenten, bösartigen Bots und Content Scrapers erstellen. Beispielsweise können Sie mit durchsatzbasierten Regeln die Anzahl der zulässigen Webanforderungen angeben, die von jeder Client-IP in einem sich anschließenden, fortlaufend aktualisierten 5-Minuten-Zeitraum zugelassen werden.

Weitere Informationen zu den unterstützten Regeltypen und zusätzlichen AWS WAF Funktionen finden Sie im AWS WAF Entwicklerhandbuch und in der API-Referenz.AWS WAF

> **Important**
>
> AWS WAF ist Ihre erste Verteidigungslinie gegen Web-Exploits. Wenn auf einer API aktiviert AWS WAF ist, werden AWS WAF Regeln vor anderen Zugriffskontrollfunktionen wie API-Schlüsselautorisierung, IAM-Richtlinien, OIDC-Token und Amazon Cognito Cognito-Benutzerpools ausgewertet.

Integrieren Sie eine API mit AppSync AWS WAF

Sie können eine Appsync-API AWS WAF mithilfe des AWS Management Console, des AWS CLI AWS CloudFormation, oder eines anderen kompatiblen Clients integrieren.

Um eine AWS AppSync API zu integrieren mit AWS WAF

1. Erstellen Sie eine AWS WAF Web-ACL. Eine ausführliche Anleitung zur Verwendung der AWS WAF Konsole finden Sie unter Web-ACL erstellen.
2. Definieren Sie die Regeln für die Web-ACL. Eine oder mehrere Regeln werden bei der Erstellung der Web-ACL definiert. Informationen zur Strukturierung von Regeln finden Sie unter AWS WAF Regeln. Beispiele für nützliche Regeln, die Sie für Ihre AWS AppSync API definieren können, finden Sie unterRegeln für eine Web-ACL erstellen.
3. Ordnen Sie die Web-ACL einer AWS AppSync API zu. Sie können diesen Schritt in der AWS WAF Konsole oder in der AppSync Konsole ausführen.

- Um die Web-ACL einer AWS AppSync API in der AWS WAF Konsole zuzuordnen, folgen Sie den Anweisungen zum Zuordnen oder Aufheben der Zuordnung einer Web-ACL zu einer AWS Ressource im AWS WAF Entwicklerhandbuch.
- So verknüpfen Sie die Web-ACL mit einer AWS AppSync API in der Konsole AWS AppSync

  a. Melden Sie sich bei der an AWS Management Console und öffnen Sie die AppSync Konsole.

  b. Wählen Sie die API aus, die Sie einer Web-ACL zuordnen möchten.

  c. Wählen Sie im Navigationsbereich Settings (Einstellungen).

  d. Aktivieren Sie im Abschnitt Firewall für Webanwendungen die Option Aktivieren AWS WAF.

  e. Wählen Sie in der Dropdownliste Web-ACL den Namen der Web-ACL aus, die Sie Ihrer API zuordnen möchten.

  f. Wählen Sie Speichern, um die Web-ACL mit Ihrer API zu verknüpfen.

> **Note**
>
> Nachdem Sie eine Web-ACL in der AWS WAF Konsole erstellt haben, kann es einige Minuten dauern, bis die neue Web-ACL verfügbar ist. Wenn Sie im Firewall-Menü der Webanwendung keine neu erstellte Web-ACL sehen, warten Sie einige Minuten und wiederholen Sie die Schritte, um die Web-ACL Ihrer API zuzuordnen.

> **Note**
>
> AWS WAF Die Integration unterstützt das `Subscription registration message` Ereignis nur für Echtzeit-Endpunkte. AWS AppSync antwortet mit einer Fehlermeldung statt mit einer `start_ack` Meldung für alle, die `Subscription registration message` blockiert wurden AWS WAF.

Nachdem Sie einer AWS AppSync API eine Web-ACL zugeordnet haben, verwalten Sie die Web-ACL mithilfe der AWS WAF APIs. Sie müssen die Web-ACL nicht erneut mit Ihrer AWS AppSync API verknüpfen, es sei denn, Sie möchten die AWS AppSync API einer anderen Web-ACL zuordnen.

Regeln für eine Web-ACL erstellen

Regeln definieren, wie Webanfragen geprüft werden und was zu tun ist, wenn eine Webanfrage den Inspektionskriterien entspricht. Regeln existieren nicht für AWS WAF sich allein. Sie können auf eine Regel anhand des Namens in einer Regelgruppe oder in der Web-ACL zugreifen, in der sie definiert ist. Weitere Informationen finden Sie unter AWS WAF Regeln. Die folgenden Beispiele zeigen, wie Regeln definiert und verknüpft werden, die für den Schutz einer AppSync API nützlich sind.

Example Web-ACL-Regel zur Begrenzung der Größe des Anfragetextes

Im Folgenden finden Sie ein Beispiel für eine Regel, die die Größe des Hauptteils von Anfragen begrenzt. Dies würde beim Erstellen einer Web-ACL in der AWS WAF Konsole in den JSON-Editor für Regeln eingegeben werden.

```
{
    "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
        }
}
```

Nachdem Sie Ihre Web-ACL mithilfe der obigen Beispielregel erstellt haben, müssen Sie sie mit Ihrer AppSync API verknüpfen. Als Alternative zur Verwendung von können Sie diesen Schritt in der ausführen AWS Management Console, AWS CLI indem Sie den folgenden Befehl ausführen.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Es kann einige Minuten dauern, bis die Änderungen übernommen werden, aber nach der Ausführung dieses Befehls werden Anfragen, deren Hauptteil größer als 1024 Byte ist, von AWS AppSync abgewiesen.

> **Note**
>
> Nachdem Sie eine neue Web-ACL in der AWS WAF Konsole erstellt haben, kann es einige Minuten dauern, bis die Web-ACL für die Verknüpfung mit einer API verfügbar ist. Wenn Sie den CLI-Befehl ausführen und eine `WAFUnavailableEntityException` Fehlermeldung erhalten, warten Sie einige Minuten und versuchen Sie erneut, den Befehl auszuführen.

Example Web-ACL-Regel zur Beschränkung von Anfragen von einer einzelnen IP-Adresse

Im Folgenden finden Sie ein Beispiel für eine Regel, mit der eine AppSync API auf 100 Anfragen von einer einzigen IP-Adresse gedrosselt wird. Dies würde beim Erstellen einer Web-ACL mit einer ratenbasierten Regel in der Konsole in den JSON-Editor für Regeln eingegeben werden. AWS WAF

```
{
  "Name": "Throttle",
  "Priority": 0,
  "Action": {
    "Block": {}
  },
  "VisibilityConfig": {
    "SampledRequestsEnabled": true,
    "CloudWatchMetricsEnabled": true,
    "MetricName": "Throttle"
  },
  "Statement": {
    "RateBasedStatement": {
      "Limit": 100,
      "AggregateKeyType": "IP"
    }
```

```
  }
}
```

Nachdem Sie Ihre Web-ACL anhand der obigen Beispielregel erstellt haben, müssen Sie sie mit Ihrer AppSync API verknüpfen. Sie können diesen Schritt in der ausführen, AWS CLI indem Sie den folgenden Befehl ausführen.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Example Web-ACL-Regel zur Verhinderung von GraphQL __schema Introspection-Abfragen an eine API

Das Folgende ist ein Beispiel für eine Regel, die GraphQL __schema Introspection-Abfragen an eine API verhindert. Jeder HTTP-Hauptteil, der die Zeichenfolge „__schema“ enthält, wird blockiert. Dies würde beim Erstellen einer Web-ACL in der AWS WAF Konsole in den JSON-Editor für Regeln eingegeben werden.

```
{
  "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
        }
      ]
    }
```

```
  }
}
```

Nachdem Sie Ihre Web-ACL mithilfe der obigen Beispielregel erstellt haben, müssen Sie sie mit Ihrer AppSync API verknüpfen. Sie können diesen Schritt in der ausführen, AWS CLI indem Sie den folgenden Befehl ausführen.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Sicherheit in AWS AppSync

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von Rechenzentren und Netzwerkarchitekturen, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame AWS Verantwortung von Ihnen und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud selbst und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS Dienste in der AWS Cloud ausführt. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Externe Prüfer testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS . Weitere Informationen zu den Compliance-Programmen, die für gelten AWS AppSync, finden Sie unter AWS Services im Umfang nach Compliance-Programmen AWS .
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. Sie sind auch für andere Faktoren verantwortlich, etwa für die Vertraulichkeit Ihrer Daten, für die Anforderungen Ihres Unternehmens und für die geltenden Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Nutzung anwenden können AWS AppSync. In den folgenden Themen erfahren Sie, wie Sie die Konfiguration vornehmen AWS AppSync , um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie erfahren auch, wie Sie andere AWS Dienste nutzen können, die Sie bei der Überwachung und Sicherung Ihrer AWS AppSync Ressourcen unterstützen.

Themen

- Datenschutz in AWS AppSync
- Konformitätsüberprüfung für AWS AppSync
- Sicherheit der Infrastruktur in AWS AppSync
- Resilienz in AWS AppSync
- Identitäts- und Zugriffsmanagement für AWS AppSync
- Protokollieren von AWS AppSync API-Aufrufen mit AWS CloudTrail
- Bewährte Sicherheitsmethoden für AWS AppSync

Datenschutz in AWS AppSync

Das Modell der AWS gemeinsamen Verantwortung und geteilter Verantwortung gilt für den Datenschutz in AWS AppSync. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag AWS -Modell der geteilten Verantwortung und in der DSGVO im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Wird verwendet SSL/TLS , um mit AWS Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein AWS CloudTrail. Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter Arbeiten mit CloudTrail Pfaden im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-3.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit der Konsole, der AWS AppSync API oder auf andere AWS-Services Weise arbeiten oder diese verwenden. AWS CLI AWS SDKs Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet

werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Verschlüsselung in Bewegung

AWS AppSync verwendet, wie alle AWS Dienste, TLS1 1.2 und höher für die Kommunikation bei der Nutzung der AWS veröffentlichten APIs und SDKs.

Die Verwendung AWS AppSync mit anderen AWS Diensten wie Amazon DynamoDB gewährleistet die Verschlüsselung bei der Übertragung: Alle AWS Dienste verwenden TLS 1.2 und höher, um miteinander zu kommunizieren, sofern nicht anders angegeben. Für Resolver, die Amazon EC2 oder verwenden CloudFront, liegt es in Ihrer Verantwortung, zu überprüfen, ob TLS (HTTPS) konfiguriert und sicher ist. Informationen zur Konfiguration von HTTPS in Amazon EC2 finden Sie unter Konfiguration SSL/TLS auf Amazon Linux 2 im EC2 Amazon-Benutzerhandbuch. Informationen zur Konfiguration von HTTPS auf CloudFront finden Sie unter HTTPS CloudFront in Amazon im CloudFront Benutzerhandbuch.

Konformitätsüberprüfung für AWS AppSync

Externe Prüfer bewerten die Sicherheit und Einhaltung von Vorschriften im AWS AppSync Rahmen mehrerer AWS Compliance-Programme. AWS AppSync entspricht den Programmen SOC, PCI, HIPAA/HIPAA BAA, IRAP, C5, ENS High, OSPAR und HITRUST CSF.

Informationen darüber, ob ein in den Geltungsbereich bestimmter Compliance-Programme AWS-Service fällt, finden Sie unter Umfang nach Compliance-Programm unter AWS-Services Umfang nach Compliance-Programm. Wählen Sie aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. AWS stellt die folgenden Ressourcen zur Verfügung, die Sie bei der Einhaltung der Vorschriften unterstützen:

- Compliance und Governance im Bereich Sicherheit – In diesen Anleitungen für die Lösungsimplementierung werden Überlegungen zur Architektur behandelt. Außerdem werden Schritte für die Bereitstellung von Sicherheits- und Compliance-Features beschrieben.
- Referenz für berechtigte HIPAA-Services – Listet berechtigte HIPAA-Services auf. Nicht alle AWS-Services sind HIPAA-fähig.
- AWS Compliance-Ressourcen — Diese Sammlung von Arbeitsmappen und Leitfäden gilt möglicherweise für Ihre Branche und Ihren Standort.
- AWS Leitfäden zur Einhaltung von Vorschriften für Kunden — Verstehen Sie das Modell der gemeinsamen Verantwortung aus dem Blickwinkel der Einhaltung von Vorschriften. In den Leitfäden werden die bewährten Verfahren zur Sicherung zusammengefasst AWS-Services und die Leitlinien den Sicherheitskontrollen in verschiedenen Frameworks (einschließlich des National Institute of Standards and Technology (NIST), des Payment Card Industry Security Standards Council (PCI) und der International Organization for Standardization (ISO)) zugeordnet.
- Evaluierung von Ressourcen anhand von Regeln im AWS Config Entwicklerhandbuch — Der AWS Config Service bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- AWS Security Hub— Auf diese AWS-Service Weise erhalten Sie einen umfassenden Überblick über Ihren internen Sicherheitsstatus. AWS Security Hub verwendet Sicherheitskontrollen, um Ihre AWS -Ressourcen zu bewerten und Ihre Einhaltung von Sicherheitsstandards und bewährten Methoden zu überprüfen. Die Liste der unterstützten Services und Kontrollen finden Sie in der Security-Hub-Steuerelementreferenz.
- Amazon GuardDuty — Dies AWS-Service erkennt potenzielle Bedrohungen für Ihre Workloads AWS-Konten, Container und Daten, indem es Ihre Umgebung auf verdächtige und böswillige Aktivitäten überwacht. GuardDuty kann Ihnen helfen, verschiedene Compliance-Anforderungen wie PCI DSS zu erfüllen, indem es die in bestimmten Compliance-Frameworks vorgeschriebenen Anforderungen zur Erkennung von Eindringlingen erfüllt.
- AWS Audit Manager— Auf diese AWS-Service Weise können Sie Ihre AWS Nutzung kontinuierlich überprüfen, um das Risikomanagement und die Einhaltung von Vorschriften und Industriestandards zu vereinfachen.

Sicherheit der Infrastruktur in AWS AppSync

Als verwalteter Dienst AWS AppSync ist er durch AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter AWS Cloud-Sicherheit. Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung

der bewährten Methoden für die Infrastruktursicherheit finden Sie unter Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe für den Zugriff AWS AppSync über das Netzwerk. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit AWS Security Token Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Resilienz in AWS AppSync

Die AWS globale Infrastruktur basiert auf AWS Regionen und Availability Zones. AWS Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Zusätzlich zur AWS globalen Infrastruktur AWS AppSync können die meisten Ressourcen mithilfe von AWS CloudFormation Vorlagen definiert werden. Ein Beispiel für die Verwendung von AWS CloudFormation Vorlagen zur Deklaration von AWS AppSync Ressourcen finden Sie unter Praktische Anwendungsfälle für AWS AppSync Pipeline-Resolver im AWS Blog und im AWS CloudFormation Benutzerhandbuch.

Identitäts- und Zugriffsmanagement für AWS AppSync

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu verwenden. AWS AppSync IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

- Zielgruppe
- Authentifizierung mit Identitäten
- Verwalten des Zugriffs mit Richtlinien
- Wie AWS AppSync funktioniert mit IAM
- Identitätsbasierte Richtlinien für AWS AppSync
- Fehlerbehebung bei AWS AppSync Identität und Zugriff

Zielgruppe

Die Art und Weise, wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von der Arbeit ab, in der Sie tätig sind. AWS AppSync

Dienstbenutzer — Wenn Sie den AWS AppSync Dienst für Ihre Arbeit verwenden, stellt Ihnen Ihr Administrator die erforderlichen Anmeldeinformationen und Berechtigungen zur Verfügung. Wenn Sie für Ihre Arbeit mehr AWS AppSync Funktionen verwenden, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Funktionsweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anfordern müssen. Unter Fehlerbehebung bei AWS AppSync Identität und Zugriff finden Sie nützliche Informationen für den Fall, dass Sie keinen Zugriff auf eine Feature in AWS AppSync haben.

Serviceadministrator — Wenn Sie in Ihrem Unternehmen für die AWS AppSync Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollen Zugriff auf AWS AppSync. Es ist Ihre Aufgabe, zu bestimmen, auf welche AWS AppSync Funktionen und Ressourcen Ihre Servicebenutzer zugreifen sollen. Anschließend müssen Sie Anforderungen an Ihren IAM-Administrator senden, um die Berechtigungen der Servicebenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachzuvollziehen. Weitere Informationen darüber, wie Ihr Unternehmen IAM nutzen kann AWS AppSync, finden Sie unterWie AWS AppSync funktioniert mit IAM.

IAM-Administrator: Wenn Sie als IAM-Administrator fungieren, sollten Sie Einzelheiten dazu kennen, wie Sie Richtlinien zur Verwaltung des Zugriffs auf AWS AppSync verfassen können. Beispiele für

AWS AppSync identitätsbasierte Richtlinien, die Sie in IAM verwenden können, finden Sie unter. Identitätsbasierte Richtlinien für AWS AppSync

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen als IAM-Benutzer authentifiziert (angemeldet AWS) sein oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich AWS als föderierte Identität anmelden, indem Sie Anmeldeinformationen verwenden, die über eine Identitätsquelle bereitgestellt wurden. AWS IAM Identity Center (IAM Identity Center) -Benutzer, die Single Sign-On-Authentifizierung Ihres Unternehmens und Ihre Google- oder Facebook-Anmeldeinformationen sind Beispiele für föderierte Identitäten. Wenn Sie sich als Verbundidentität anmelden, hat der Administrator vorher mithilfe von IAM-Rollen einen Identitätsverbund eingerichtet. Wenn Sie über den Verbund darauf zugreifen AWS , übernehmen Sie indirekt eine Rolle.

Je nachdem, welcher Benutzertyp Sie sind, können Sie sich beim AWS Management Console oder beim AWS Zugangsportal anmelden. Weitere Informationen zur Anmeldung finden Sie AWS unter So melden Sie sich bei Ihrem an AWS-Konto im AWS-Anmeldung Benutzerhandbuch.

Wenn Sie AWS programmgesteuert darauf zugreifen, AWS stellt es ein Software Development Kit (SDK) und eine Befehlszeilenschnittstelle (CLI) bereit, mit denen Sie Ihre Anfragen mithilfe Ihrer Anmeldeinformationen kryptografisch signieren können. Wenn Sie keine AWS Tools verwenden, müssen Sie Anfragen selbst signieren. Weitere Informationen zur Verwendung der empfohlenen Methode für die Selbstsignierung von Anforderungen finden Sie unter AWS Signature Version 4 für API-Anforderungen im IAM-Benutzerhandbuch.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen bereitstellen. AWS Empfiehlt beispielsweise, die Multi-Faktor-Authentifizierung (MFA) zu verwenden, um die Sicherheit Ihres Kontos zu erhöhen. Weitere Informationen finden Sie unter Multi-Faktor-Authentifizierung im AWS IAM Identity Center - Benutzerhandbuch und AWS Multi-Faktor-Authentifizierung (MFA) in IAM im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle AWS-Services Ressourcen im Konto hat. Diese Identität wird als AWS-Konto Root-Benutzer bezeichnet. Sie können darauf zugreifen, indem Sie sich mit der E-Mail-Adresse und

dem Passwort anmelden, mit denen Sie das Konto erstellt haben. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Schützen Sie Ihre Root-Benutzer-Anmeldeinformationen. Verwenden Sie diese nur, um die Aufgaben auszuführen, die nur der Root-Benutzer ausführen kann. Eine vollständige Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Aufgaben, die Root-Benutzer-Anmeldeinformationen erfordern im IAM-Benutzerhandbuch.

Verbundidentität

Als bewährte Methode sollten menschliche Benutzer, einschließlich Benutzer, die Administratorzugriff benötigen, für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen den Verbund mit einem Identitätsanbieter verwenden.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensbenutzerverzeichnis, einem Web-Identitätsanbieter AWS Directory Service, dem Identity Center-Verzeichnis oder einem beliebigen Benutzer, der mithilfe AWS-Services von Anmeldeinformationen zugreift, die über eine Identitätsquelle bereitgestellt wurden. Wenn föderierte Identitäten darauf zugreifen AWS-Konten, übernehmen sie Rollen, und die Rollen stellen temporäre Anmeldeinformationen bereit.

Für die zentrale Zugriffsverwaltung empfehlen wir Ihnen, AWS IAM Identity Center zu verwenden. Sie können Benutzer und Gruppen in IAM Identity Center erstellen, oder Sie können eine Verbindung zu einer Gruppe von Benutzern und Gruppen in Ihrer eigenen Identitätsquelle herstellen und diese synchronisieren, um sie in all Ihren AWS-Konten Anwendungen zu verwenden. Informationen zu IAM Identity Center finden Sie unter Was ist IAM Identity Center? im AWS IAM Identity Center - Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität innerhalb Ihres Unternehmens AWS-Konto , die über spezifische Berechtigungen für eine einzelne Person oder Anwendung verfügt. Wenn möglich, empfehlen wir, temporäre Anmeldeinformationen zu verwenden, anstatt IAM-Benutzer zu erstellen, die langfristige Anmeldeinformationen wie Passwörter und Zugriffsschlüssel haben. Bei speziellen Anwendungsfällen, die langfristige Anmeldeinformationen mit IAM-Benutzern erfordern, empfehlen wir jedoch, die Zugriffsschlüssel zu rotieren. Weitere Informationen finden Sie unter Regelmäßiges Rotieren von Zugriffsschlüsseln für Anwendungsfälle, die langfristige Anmeldeinformationen erfordern im IAM-Benutzerhandbuch.

Eine IAM-Gruppe ist eine Identität, die eine Sammlung von IAM-Benutzern angibt. Sie können sich nicht als Gruppe anmelden. Mithilfe von Gruppen können Sie Berechtigungen für mehrere Benutzer

gleichzeitig angeben. Gruppen vereinfachen die Verwaltung von Berechtigungen, wenn es zahlreiche Benutzer gibt. Sie könnten beispielsweise eine Gruppe benennen IAMAdminsund dieser Gruppe Berechtigungen zur Verwaltung von IAM-Ressourcen erteilen.

Benutzer unterscheiden sich von Rollen. Ein Benutzer ist einer einzigen Person oder Anwendung eindeutig zugeordnet. Eine Rolle kann von allen Personen angenommen werden, die sie benötigen. Benutzer besitzen dauerhafte Anmeldeinformationen. Rollen stellen temporäre Anmeldeinformationen bereit. Weitere Informationen finden Sie unter Anwendungsfälle für IAM-Benutzer im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität innerhalb von Ihnen AWS-Konto , die über bestimmte Berechtigungen verfügt. Sie ist einem IAM-Benutzer vergleichbar, jedoch nicht mit einer bestimmten Person verknüpft. Um vorübergehend eine IAM-Rolle in der zu übernehmen AWS Management Console, können Sie von einer Benutzer- zu einer IAM-Rolle (Konsole) wechseln. Sie können eine Rolle übernehmen, indem Sie eine AWS CLI oder AWS API-Operation aufrufen oder eine benutzerdefinierte URL verwenden. Weitere Informationen zu Methoden für die Verwendung von Rollen finden Sie unter Methoden für die Übernahme einer Rolle im IAM-Benutzerhandbuch.

IAM-Rollen mit temporären Anmeldeinformationen sind in folgenden Situationen hilfreich:

- Verbundbenutzerzugriff – Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie unter Erstellen von Rollen für externe Identitätsanbieter (Verbund) im IAM-Benutzerhandbuch. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Wenn Sie steuern möchten, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in IAM. Informationen zu Berechtigungssätzen finden Sie unter Berechtigungssätze im AWS IAM Identity Center -Benutzerhandbuch.
- Temporäre IAM-Benutzerberechtigungen – Ein IAM-Benutzer oder eine -Rolle kann eine IAM-Rolle übernehmen, um vorübergehend andere Berechtigungen für eine bestimmte Aufgabe zu erhalten.
- Kontoübergreifender Zugriff – Sie können eine IAM-Rolle verwenden, um einem vertrauenswürdigen Prinzipal in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. Bei einigen können Sie AWS-Services jedoch eine Richtlinie direkt an eine Ressource anhängen (anstatt eine Rolle als Proxy zu verwenden). Informationen zu den Unterschieden

zwischen Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

- Serviceübergreifender Zugriff — Einige AWS-Services verwenden Funktionen in anderen AWS-Services. Wenn Sie beispielsweise einen Service aufrufen, ist es üblich, dass dieser Service Anwendungen in Amazon ausführt EC2 oder Objekte in Amazon S3 speichert. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
  - Forward Access Sessions (FAS) — Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle verwenden, um Aktionen auszuführen AWS, gelten Sie als Principal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service initiieren. FAS verwendet die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, in Kombination mit der Anfrage, Anfragen an AWS-Service nachgelagerte Dienste zu stellen. FAS-Anfragen werden nur gestellt, wenn ein Dienst eine Anfrage erhält, für deren Abschluss Interaktionen mit anderen AWS-Services oder Ressourcen erforderlich sind. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter Zugriffssitzungen weiterleiten.
  - Servicerolle – Eine Servicerolle ist eine IAM-Rolle, die ein Service übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.
  - Dienstbezogene Rolle — Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Servicebezogene Rollen erscheinen in Ihrem Dienst AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.
- Auf Amazon ausgeführte Anwendungen EC2 — Sie können eine IAM-Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2 Instance ausgeführt werden und AWS API-Anfragen stellen AWS CLI . Dies ist dem Speichern von Zugriffsschlüsseln innerhalb der EC2 Instance vorzuziehen. Um einer EC2 Instanz eine AWS Rolle zuzuweisen und sie allen ihren Anwendungen zur Verfügung zu stellen, erstellen Sie ein Instanzprofil, das an die Instanz angehängt ist. Ein Instanzprofil enthält die Rolle und ermöglicht Programmen, die auf der EC2 Instanz ausgeführt werden, temporäre Anmeldeinformationen abzurufen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Verwenden einer IAM-Rolle, um Berechtigungen für Anwendungen zu gewähren, die auf EC2 Amazon-Instances ausgeführt werden.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie ist ein Objekt, AWS das, wenn es einer Identität oder Ressource zugeordnet ist, deren Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn ein Prinzipal (Benutzer, Root-Benutzer oder Rollensitzung) eine Anfrage stellt. Die Berechtigungen in den Richtlinien legen fest, ob eine Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu Struktur und Inhalten von JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen. Der Administrator kann dann die IAM-Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen annehmen.

IAM-Richtlinien definieren Berechtigungen für eine Aktion unabhängig von der Methode, die Sie zur Ausführung der Aktion verwenden. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Rolleninformationen von der AWS Management Console AWS CLI, der oder der AWS API abrufen.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können weiter als Inline-Richtlinien oder verwaltete Richtlinien kategorisiert werden. Inline-Richtlinien sind direkt in einen einzelnen Benutzer, eine einzelne Gruppe oder eine einzelne Rolle eingebettet. Verwaltete Richtlinien sind eigenständige Richtlinien, die Sie mehreren Benutzern, Gruppen und Rollen in Ihrem System zuordnen können AWS-Konto.

Zu den verwalteten Richtlinien gehören AWS verwaltete Richtlinien und vom Kunden verwaltete Richtlinien. Informationen dazu, wie Sie zwischen einer verwalteten Richtlinie und einer Inline-Richtlinie wählen, finden Sie unter Auswählen zwischen verwalteten und eingebundenen Richtlinien im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffskontrolllisten () ACLs

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Dienste, die Unterstützung ACLs bieten. AWS WAF Weitere Informationen finden Sie unter Übersicht über ACLs die Zugriffskontrollliste (ACL) im Amazon Simple Storage Service Developer Guide.

Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger verbreitete Richtlinientypen. Diese Richtlinientypen können die maximalen Berechtigungen festlegen, die Ihnen von den häufiger verwendeten Richtlinientypen erteilt werden können.

- Berechtigungsgrenzen – Eine Berechtigungsgrenze ist ein erweitertes Feature, mit der Sie die maximalen Berechtigungen festlegen können, die eine identitätsbasierte Richtlinie einer IAM-Entität (IAM-Benutzer oder -Rolle) erteilen kann. Sie können eine Berechtigungsgrenze für eine Entität festlegen. Die daraus resultierenden Berechtigungen sind der Schnittpunkt der

identitätsbasierten Richtlinien einer Entität und ihrer Berechtigungsgrenzen. Ressourcenbasierte Richtlinien, die den Benutzer oder die Rolle im Feld `Principal` angeben, werden nicht durch Berechtigungsgrenzen eingeschränkt. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen über Berechtigungsgrenzen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im IAM-Benutzerhandbuch.

- Dienststeuerungsrichtlinien (SCPs) — SCPs sind JSON-Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OU) in festlegen. AWS Organizations AWS Organizations ist ein Dienst zur Gruppierung und zentralen Verwaltung mehrerer Objekte AWS-Konten , die Ihrem Unternehmen gehören. Wenn Sie alle Funktionen in einer Organisation aktivieren, können Sie Richtlinien zur Servicesteuerung (SCPs) auf einige oder alle Ihre Konten anwenden. Das SCP schränkt die Berechtigungen für Entitäten in Mitgliedskonten ein, einschließlich der einzelnen Root-Benutzer des AWS-Kontos Entitäten. Weitere Informationen zu Organizations und SCPs finden Sie unter Richtlinien zur Servicesteuerung im AWS Organizations Benutzerhandbuch.
- Ressourcenkontrollrichtlinien (RCPs) — RCPs sind JSON-Richtlinien, mit denen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten festlegen können, ohne die IAM-Richtlinien aktualisieren zu müssen, die jeder Ressource zugeordnet sind, deren Eigentümer Sie sind. Das RCP schränkt die Berechtigungen für Ressourcen in Mitgliedskonten ein und kann sich auf die effektiven Berechtigungen für Identitäten auswirken, einschließlich der Root-Benutzer des AWS-Kontos, unabhängig davon, ob sie zu Ihrer Organisation gehören. Weitere Informationen zu Organizations RCPs, einschließlich einer Liste AWS-Services dieser Support-Leistungen RCPs, finden Sie unter Resource Control Policies (RCPs) im AWS Organizations Benutzerhandbuch.
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die Sie als Parameter übergeben, wenn Sie eine temporäre Sitzung für eine Rolle oder einen verbundenen Benutzer programmgesteuert erstellen. Die resultierenden Sitzungsberechtigungen sind eine Schnittmenge der auf der Identität des Benutzers oder der Rolle basierenden Richtlinien und der Sitzungsrichtlinien. Berechtigungen können auch aus einer ressourcenbasierten Richtlinie stammen. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn mehrere auf eine Anforderung mehrere Richtlinientypen angewendet werden können, sind die entsprechenden Berechtigungen komplizierter. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie im IAM-Benutzerhandbuch unter Bewertungslogik für Richtlinien.

Wie AWS AppSync funktioniert mit IAM

Bevor Sie IAM zur Verwaltung des Zugriffs auf verwenden, sollten Sie sich darüber informieren AWS AppSync, mit welchen IAM-Funktionen Sie arbeiten können. AWS AppSync

IAM-Funktionen, die Sie mit verwenden können AWS AppSync

IAM-Feature	AWS AppSync Unterstützung
Identitätsbasierte Richtlinien	Ja
Ressourcenbasierte Richtlinien	Nein
Richtlinienaktionen	Ja
Richtlinienressourcen	Ja
Bedingungsschlüssel für die Richtlinie	Nein
ACLs	Nein
ABAC (Tags in Richtlinien)	Teilweise
Temporäre Anmeldeinformationen	Ja
Forward Access Sessions (FAS)	Teilweise
Servicerollen	Nein
Serviceverknüpfte Rollen	Teilweise

Einen allgemeinen Überblick darüber, wie AWS AppSync und andere AWS Dienste mit den meisten IAM-Funktionen funktionieren, finden Sie im IAM-Benutzerhandbuch unter AWS Dienste, die mit IAM funktionieren.

Identitätsbasierte Richtlinien für AWS AppSync

Unterstützt Richtlinien auf Identitätsbasis: Ja

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen zugelassen oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. Sie können den Prinzipal nicht in einer identitätsbasierten Richtlinie angeben, da er für den Benutzer oder die Rolle gilt, dem er zugeordnet ist. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Beispiele für identitätsbasierte Richtlinien für AWS AppSync

Beispiele für AWS AppSync identitätsbasierte Richtlinien finden Sie unter. Identitätsbasierte Richtlinien für AWS AppSync

Ressourcenbasierte Richtlinien finden Sie in AWS AppSync

Unterstützt ressourcenbasierte Richtlinien: Nein

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Um kontoübergreifenden Zugriff zu ermöglichen, können Sie ein gesamtes Konto oder IAM-Entitäten in einem anderen Konto als Prinzipal in einer ressourcenbasierten Richtlinie angeben. Durch das Hinzufügen eines kontoübergreifenden Auftraggebers zu einer ressourcenbasierten Richtlinie ist nur die halbe Vertrauensbeziehung eingerichtet. Wenn sich der Prinzipal und die Ressource unterscheiden AWS-Konten, muss ein IAM-Administrator des vertrauenswürdigen Kontos auch der Prinzipalentität (Benutzer oder Rolle) die Berechtigung zum Zugriff auf die Ressource erteilen. Sie erteilen Berechtigungen, indem Sie der juristischen Stelle eine identitätsbasierte Richtlinie anfügen.

Wenn jedoch eine ressourcenbasierte Richtlinie Zugriff auf einen Prinzipal in demselben Konto gewährt, ist keine zusätzliche identitätsbasierte Richtlinie erforderlich. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Richtlinienaktionen für AWS AppSync

Unterstützt Richtlinienaktionen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Richtlinienaktionen haben normalerweise denselben Namen wie der zugehörige AWS API-Vorgang. Es gibt einige Ausnahmen, z. B. Aktionen, die nur mit Genehmigung durchgeführt werden können und für die es keinen passenden API-Vorgang gibt. Es gibt auch einige Operationen, die mehrere Aktionen in einer Richtlinie erfordern. Diese zusätzlichen Aktionen werden als abhängige Aktionen bezeichnet.

Schließen Sie Aktionen in eine Richtlinie ein, um Berechtigungen zur Durchführung der zugeordneten Operation zu erteilen.

Eine Liste der AWS AppSync Aktionen finden Sie unter Aktionen definiert von AWS AppSync in der Serviceautorisierungsreferenz.

Bei Richtlinienaktionen wird vor der Aktion das folgende Präfix AWS AppSync verwendet:

```
appsync
```

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie mit Kommata:

```
"Action": [
      "appsync:action1",
      "appsync:action2"
         ]
```

Beispiele für AWS AppSync identitätsbasierte Richtlinien finden Sie unter. Identitätsbasierte Richtlinien für AWS AppSync

Politische Ressourcen für AWS AppSync

Unterstützt Richtlinienressourcen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Anweisungen müssen entweder ein – `Resource`oder ein `NotResource`-Element enthalten. Als bewährte Methode geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. Sie können dies für Aktionen tun, die einen bestimmten Ressourcentyp unterstützen, der als Berechtigungen auf Ressourcenebene bezeichnet wird.

Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, z. B. Auflistungsoperationen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

Eine Liste der AWS AppSync Ressourcentypen und ihrer ARNs Eigenschaften finden Sie unter Resources defined by AWS AppSync in der Service Authorization Reference. Informationen zu den Aktionen, mit denen Sie den ARN einzelner Ressourcen angeben können, finden Sie unter Von AWS definierte AktionenAppSync.

Beispiele für AWS AppSync identitätsbasierte Richtlinien finden Sie unter. Identitätsbasierte Richtlinien für AWS AppSync

Bedingungsschlüssel für Richtlinien für AWS AppSync

Unterstützt dienstspezifische Richtlinien-Bedingungsschlüssel: Nein

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das Element `Condition` (oder `Condition` block) ermöglicht Ihnen die Angabe der Bedingungen, unter denen eine Anweisung wirksam ist. Das Element `Condition` ist optional. Sie können bedingte Ausdrücke erstellen, die Bedingungsoperatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt.

Wenn Sie mehrere `Condition`-Elemente in einer Anweisung oder mehrere Schlüssel in einem einzelnen `Condition`-Element angeben, wertet AWS diese mittels einer logischen `AND`-Operation aus. Wenn Sie mehrere Werte für einen einzelnen Bedingungsschlüssel angeben, AWS wertet die Bedingung mithilfe einer logischen `OR` Operation aus. Alle Bedingungen müssen erfüllt werden, bevor die Berechtigungen der Anweisung gewährt werden.

Sie können auch Platzhaltervariablen verwenden, wenn Sie Bedingungen angeben. Beispielsweise können Sie einem IAM-Benutzer die Berechtigung für den Zugriff auf eine Ressource nur dann gewähren, wenn sie mit dessen IAM-Benutzernamen gekennzeichnet ist. Weitere Informationen finden Sie unter IAM-Richtlinienelemente: Variablen und Tags im IAM-Benutzerhandbuch.

AWS unterstützt globale Bedingungsschlüssel und dienstspezifische Bedingungsschlüssel. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Eine Liste der AWS AppSync Bedingungsschlüssel finden Sie unter Bedingungsschlüssel für AWS AppSync in der Service Authorization Reference. Informationen zu den Aktionen und Ressourcen, mit denen Sie einen Bedingungsschlüssel verwenden können, finden Sie unter Aktionen definiert von AWS AppSync.

Beispiele für AWS AppSync identitätsbasierte Richtlinien finden Sie unter. Identitätsbasierte Richtlinien für AWS AppSync

Zugriffskontrolllisten () ACLs in AWS AppSync

Unterstützt ACLs: Nein

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Attributbasierte Zugriffskontrolle (ABAC) mit AWS AppSync

Unterstützt ABAC (Tags in Richtlinien): Teilweise

Die attributbasierte Zugriffskontrolle (ABAC) ist eine Autorisierungsstrategie, bei der Berechtigungen basierend auf Attributen definiert werden. In werden AWS diese Attribute als Tags bezeichnet. Sie können Tags an IAM-Entitäten (Benutzer oder Rollen) und an viele AWS Ressourcen anhängen. Das Markieren von Entitäten und Ressourcen ist der erste Schritt von ABAC. Anschließend entwerfen

Sie ABAC-Richtlinien, um Operationen zuzulassen, wenn das Tag des Prinzipals mit dem Tag der Ressource übereinstimmt, auf die sie zugreifen möchten.

ABAC ist in Umgebungen hilfreich, die schnell wachsen, und unterstützt Sie in Situationen, in denen die Richtlinienverwaltung mühsam wird.

Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungselement einer Richtlinie Tag-Informationen an, indem Sie die Schlüssel `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, oder Bedingung `aws:TagKeys` verwenden.

Wenn ein Service alle drei Bedingungsschlüssel für jeden Ressourcentyp unterstützt, lautet der Wert für den Service Ja. Wenn ein Service alle drei Bedingungsschlüssel für nur einige Ressourcentypen unterstützt, lautet der Wert Teilweise.

Weitere Informationen zu ABAC finden Sie unter Definieren von Berechtigungen mit ABAC-Autorisierung im IAM-Benutzerhandbuch. Um ein Tutorial mit Schritten zur Einstellung von ABAC anzuzeigen, siehe Attributbasierte Zugriffskontrolle (ABAC) verwenden im IAM-Benutzerhandbuch.

Verwenden temporärer Anmeldeinformationen mit AWS AppSync

Unterstützt temporäre Anmeldeinformationen: Ja

Einige funktionieren AWS-Services nicht, wenn Sie sich mit temporären Anmeldeinformationen anmelden. Weitere Informationen, einschließlich Informationen, die mit temporären Anmeldeinformationen AWS-Services funktionieren AWS-Services , finden Sie im IAM-Benutzerhandbuch unter Diese Option funktioniert mit IAM.

Sie verwenden temporäre Anmeldeinformationen, wenn Sie sich mit einer anderen AWS Management Console Methode als einem Benutzernamen und einem Passwort anmelden. Wenn Sie beispielsweise AWS über den Single Sign-On-Link (SSO) Ihres Unternehmens darauf zugreifen, werden bei diesem Vorgang automatisch temporäre Anmeldeinformationen erstellt. Sie erstellen auch automatisch temporäre Anmeldeinformationen, wenn Sie sich als Benutzer bei der Konsole anmelden und dann die Rollen wechseln. Weitere Informationen zum Wechseln von Rollen finden Sie unter Wechseln von einer Benutzerrolle zu einer IAM-Rolle (Konsole) im IAM-Benutzerhandbuch.

Mithilfe der AWS API AWS CLI oder können Sie temporäre Anmeldeinformationen manuell erstellen. Sie können diese temporären Anmeldeinformationen dann für den Zugriff verwenden AWS. AWS empfiehlt, temporäre Anmeldeinformationen dynamisch zu generieren, anstatt langfristige Zugriffsschlüssel zu verwenden. Weitere Informationen finden Sie unter Temporäre Sicherheitsanmeldeinformationen in IAM.

Zugriffssitzungen weiterleiten für AWS AppSync

Unterstützt Forward Access Sessions (FAS): Teilweise

Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle verwenden, um Aktionen auszuführen AWS, gelten Sie als Principal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service initiieren. FAS verwendet die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, kombiniert mit der Anforderung, Anfragen an nachgelagerte Dienste AWS-Service zu stellen. FAS-Anfragen werden nur gestellt, wenn ein Dienst eine Anfrage erhält, für deren Abschluss Interaktionen mit anderen AWS-Services oder Ressourcen erforderlich sind. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter Zugriffssitzungen weiterleiten.

Servicerollen für AWS AppSync

Unterstützt Servicerollen: Nein

Eine Servicerolle ist eine IAM-Rolle, die ein Service annimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.

> **Warning**
>
> Durch das Ändern der Berechtigungen für eine Servicerolle kann die AWS AppSync Funktionalität beeinträchtigt werden. Bearbeiten Sie Servicerollen nur, AWS AppSync wenn Sie dazu eine Anleitung erhalten.

Dienstbezogene Rollen für AWS AppSync

Unterstützt serviceverknüpfte Rollen: Ja

Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Dienstbezogene Rollen werden in Ihrem Dienst angezeigt AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

Einzelheiten zum Erstellen oder Verwalten von dienstbezogenen Rollen finden Sie im AWS IAM-Benutzerhandbuch unter Dienste, die mit IAM funktionieren. Suchen Sie in der Tabelle nach einem

Service mit einem `Yes` in der Spalte Service-linked role (Serviceverknüpfte Rolle). Wählen Sie den Link Yes (Ja) aus, um die Dokumentation für die serviceverknüpfte Rolle für diesen Service anzuzeigen.

Identitätsbasierte Richtlinien für AWS AppSync

Benutzer und Rollen haben standardmäßig nicht die Berechtigung, AWS AppSync-Ressourcen zu erstellen oder zu ändern. Sie können auch keine Aufgaben mithilfe der AWS Management Console, AWS Command Line Interface (AWS CLI) oder API ausführen. AWS Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen. Der Administrator kann dann die IAM-Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen annehmen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter Erstellen von IAM-Richtlinien (Konsole) im IAM-Benutzerhandbuch.

Einzelheiten zu Aktionen und Ressourcentypen, die von definiert wurden AWS AppSync, einschließlich des Formats von ARNs für jeden der Ressourcentypen, finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für AWS AppSync in der Service Authorization Reference.

Informationen zu den bewährten Methoden für die Erstellung und Konfiguration identitätsbasierter IAM-Richtlinien finden Sie unter. the section called “Bewährte Methoden für IAM-Richtlinien”

Eine Liste der identitätsbasierten IAM-Richtlinien für finden Sie unter. AWS AppSync AWS verwaltete Richtlinien für AWS AppSync

Themen

- Verwenden der AWS AppSync-Konsole
- Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer
- Zugreifen auf einen Amazon-S3-Bucket
- AWS AppSync widgetsAuf Stichwörtern basierende Anzeige
- AWS verwaltete Richtlinien für AWS AppSync

Verwenden der AWS AppSync-Konsole

Um auf die AWS AppSync Konsole zugreifen zu können, benötigen Sie ein Mindestmaß an Berechtigungen. Diese Berechtigungen müssen es Ihnen ermöglichen, Details zu den AWS AppSync

Ressourcen in Ihrem aufzulisten und anzuzeigen AWS-Konto. Wenn Sie eine identitätsbasierte Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie.

Sie müssen Benutzern, die nur die API AWS CLI oder die AWS API aufrufen, keine Mindestberechtigungen für die Konsole gewähren. Stattdessen sollten Sie nur Zugriff auf die Aktionen zulassen, die der API-Operation entsprechen, die die Benutzer ausführen möchten.

Um sicherzustellen, dass IAM-Benutzer und -Rollen die AWS AppSync Konsole weiterhin verwenden können, fügen Sie den Entitäten auch die AWS AppSync `ConsoleAccess` oder die `ReadOnly` AWS verwaltete Richtlinie hinzu. Weitere Informationen finden Sie unter Hinzufügen von Berechtigungen zu einem Benutzer im IAM-Benutzerhandbuch.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der AWS CLI API oder. 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": "*"
        }
    ]
}
```

Zugreifen auf einen Amazon-S3-Bucket

In diesem Beispiel möchten Sie einem IAM-Benutzer in Ihrem AWS Konto Zugriff auf einen Ihrer Amazon S3 S3-Buckets gewähren. `examplebucket` Sie möchten dem Benutzer außerdem Berechtigungen zum Hinzufügen, Aktualisieren und Löschen von Objekten gewähren.

Zusätzlich zum Erteilen der Berechtigungen `s3:PutObject`, `s3:GetObject` und `s3:DeleteObject` für den Benutzer, gewährt die Richtlinie die Berechtigungen `s3:ListAllMyBuckets`, `s3:GetBucketLocation` und `s3:ListBucket`. Dies sind die zusätzlichen Berechtigungen, die von der Konsole benötigt werden. Außerdem sind die Aktionen `s3:PutObjectAcl` und `s3:GetObjectAcl` erforderlich, um Objekte in der Konsole kopieren, ausschneiden und einfügen zu können. Ein Beispiel für eine exemplarische Vorgehensweise, bei der Benutzern Berechtigungen erteilt und diese mithilfe der Konsole getestet werden, finden Sie unter Eine exemplarische Vorgehensweise: Verwenden von Benutzerrichtlinien zur Steuerung des Zugriffs auf Ihren Bucket.

JSON

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

AWS AppSync *widgets*Auf Stichwörtern basierende Anzeige

Sie können in Ihrer identitätsbasierten Richtlinie Bedingungen für die Steuerung des Zugriffs auf AWS AppSync-Ressourcen auf der Basis von Tags verwenden. Dieses Beispiel zeigt, wie Sie eine Richtlinie erstellen könnten, die die Anzeige einer *widget* gestattet. Die Berechtigung wird jedoch nur gewährt, wenn der Wert des *widget*-Tags `0wner` der Name des Benutzers ist. Diese Richtlinie gewährt auch die Berechtigungen, die für die Ausführung dieser Aktion auf der Konsole erforderlich sind.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListWidgetsInConsole",
            "Effect": "Allow",
            "Action": ":ListWidgets",
```

```
            "Resource": "*"
        },
        {
            "Sid": "ViewWidgetIfOwner",
            "Effect": "Allow",
            "Action": ":GetWidget",
            "Resource": "arn:aws::*:*:widget/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

Sie können diese Richtlinie den IAM-Benutzern in Ihrem Konto anfügen. Wenn ein Benutzer mit dem Namen `richard-roe` versucht, eine AWS AppSync-*`widget`* anzuzeigen, muss die *`widget`* das Tag `Owner=richard-roe` oder `owner=richard-roe` besitzen. Andernfalls wird der Zugriff abgelehnt. Der Tag-Schlüssel `Owner` der Bedingung stimmt sowohl mit `Owner` als auch mit `owner` überein, da die Namen von Bedingungsschlüsseln nicht zwischen Groß- und Kleinschreibung unterscheiden. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

AWS verwaltete Richtlinien für AWS AppSync

Um Benutzern, Gruppen und Rollen Berechtigungen hinzuzufügen, ist es einfacher, AWS verwaltete Richtlinien zu verwenden, als Richtlinien selbst zu schreiben. Es erfordert Zeit und Fachwissen, um von Kunden verwaltete IAM-Richtlinien zu erstellen, die Ihrem Team nur die benötigten Berechtigungen bieten. Um schnell loszulegen, können Sie unsere AWS verwalteten Richtlinien verwenden. Diese Richtlinien decken allgemeine Anwendungsfälle ab und sind in Ihrem AWS-Konto verfügbar. Weitere Informationen zu AWS verwalteten Richtlinien finden Sie im IAM-Benutzerhandbuch unter AWS Verwaltete Richtlinien.

AWS Dienste verwalten und aktualisieren AWS verwaltete Richtlinien. Sie können die Berechtigungen in AWS verwalteten Richtlinien nicht ändern. Dienste fügen einer AWS verwalteten Richtlinie gelegentlich zusätzliche Berechtigungen hinzu, um neue Funktionen zu unterstützen. Diese Art von Update betrifft alle Identitäten (Benutzer, Gruppen und Rollen), an welche die Richtlinie angehängt ist. Es ist sehr wahrscheinlich, dass Dienste eine AWS verwaltete Richtlinie

aktualisieren, wenn eine neue Funktion eingeführt wird oder wenn neue Operationen verfügbar werden. Dienste entfernen keine Berechtigungen aus einer AWS verwalteten Richtlinie, sodass durch Richtlinienaktualisierungen Ihre bestehenden Berechtigungen nicht beeinträchtigt werden.

AWS Unterstützt außerdem verwaltete Richtlinien für Jobfunktionen, die sich über mehrere Dienste erstrecken. Die ReadOnlyAccess AWS verwaltete Richtlinie bietet beispielsweise schreibgeschützten Zugriff auf alle AWS Dienste und Ressourcen. Wenn ein Dienst eine neue Funktion startet, werden nur Leseberechtigungen für neue Operationen und Ressourcen AWS hinzugefügt. Eine Liste und Beschreibungen der Richtlinien für Auftragsfunktionen finden Sie in Verwaltete AWS -Richtlinien für Auftragsfunktionen im IAM-Leitfaden.

AWS verwaltete Richtlinie: AWSApp SyncInvokeFullAccess

Verwenden Sie die `AWSAppSyncInvokeFullAccess` AWS verwaltete Richtlinie, um Ihren Administratoren den Zugriff auf den AWS AppSync Service über die Konsole oder unabhängig zu ermöglichen.

Sie können die `AWSAppSyncInvokeFullAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `AWS AppSync`— Ermöglicht vollen Administratorzugriff auf alle Ressourcen in AWS AppSync

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "appsync:GraphQL",
                "appsync:GetGraphqlApi",
                "appsync:ListGraphqlApis",
                "appsync:ListApiKeys"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

AWS verwaltete Richtlinie: AWSApp SyncSchemaAuthor

Verwenden Sie die `AWSAppSyncSchemaAuthor` AWS verwaltete Richtlinie, um IAM-Benutzern Zugriff zu gewähren, um ihre GraphQL-Schemas zu erstellen, zu aktualisieren und abzufragen. Informationen darüber, was Benutzer mit diesen Berechtigungen tun können, finden Sie unter. [GraphQL APIs entwerfen mit AWS AppSync]

Sie können die `AWSAppSyncSchemaAuthor`-Richtlinie an Ihre IAM-Identitäten anfügen.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `AWS AppSync`— Erlaubt die folgenden Aktionen:
  - GraphQL-Schemas erstellen
  - Ermöglicht die Erstellung, Änderung und Löschung von GraphQL-Typen, Resolvern und Funktionen
  - Evaluierung der Logik von Anfrage- und Antwortvorlagen
  - Evaluieren von Code anhand einer Laufzeit und eines Kontextes
  - Senden von GraphQL-Abfragen an GraphQL APIs
  - GraphQL-Daten abrufen

JSON

```
{
    "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 verwaltete Richtlinie: AWSApp SyncPushToCloudWatchLogs

AWS AppSync verwendet Amazon CloudWatch, um die Leistung Ihrer Anwendung zu überwachen, indem es Protokolle generiert, die Sie zur Fehlerbehebung und Optimierung Ihrer GraphQL-

Anfragen verwenden können. Weitere Informationen finden Sie unter Verwendung CloudWatch zur Überwachung und Protokollierung von GraphQL-API-Daten.

Verwenden Sie die `AWSAppSyncPushToCloudWatchLogs` AWS verwaltete Richtlinie, um das AWS AppSync Senden von Protokollen an das Konto eines IAM-Benutzers CloudWatch zu ermöglichen.

Sie können die `AWSAppSyncPushToCloudWatchLogs`-Richtlinie an Ihre IAM-Identitäten anfügen.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `CloudWatch Logs`— Ermöglicht AWS AppSync das Erstellen von Protokollgruppen und Streams mit bestimmten Namen. AWS AppSyncüberträgt Protokollereignisse in den angegebenen Protokollstream.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS verwaltete Richtlinie: AWSApp SyncAdministrator

Verwenden Sie die `AWSAppSyncAdministrator` AWS verwaltete Richtlinie, um Ihren Administratoren Zugriff auf alle Optionen AWS AppSync mit Ausnahme der AWS Konsole zu gewähren.

Sie können eine Verbindung `AWSAppSyncAdministrator` zu Ihren IAM-Entitäten herstellen. AWS AppSync ordnet diese Richtlinie auch einer Servicerolle zu, sodass sie Aktionen in Ihrem Namen ausführen kann.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `AWS AppSync`— Ermöglicht vollen Administratorzugriff auf alle Ressourcen in AWS AppSync
- `IAM`— Erlaubt die folgenden Aktionen:
  - Erstellen von dienstbezogenen Rollen, AWS AppSync damit Sie Ressourcen in anderen Diensten in Ihrem Namen analysieren können
  - Löschen von dienstbezogenen Rollen
  - Weitergabe von dienstbezogenen Rollen an andere AWS Dienste, um die Rolle später zu übernehmen und Aktionen in Ihrem Namen auszuführen

JSON

```
{
    "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 verwaltete Richtlinie: AWSApp SyncServiceRolePolicy

Verwenden Sie die `AWSAppSyncServiceRolePolicy` AWS verwaltete Richtlinie, um den Zugriff auf AWS Dienste und Ressourcen zu ermöglichen, die AWS AppSync diese verwenden oder verwalten.

Sie können `AWSAppSyncServiceRolePolicy` nicht an Ihre IAM-Entitäten anhängen. Diese Richtlinie ist mit einer dienstbezogenen Rolle verknüpft, mit der AWS AppSync Sie Aktionen in Ihrem Namen ausführen können. Weitere Informationen finden Sie unter Dienstbezogene Rollen für AWS AppSync.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `X-Ray`— AWS AppSync verwendet AWS X-Ray , um Daten über Anfragen zu sammeln, die im Rahmen Ihrer Anwendung gestellt wurden. Weitere Informationen finden Sie unter Wird verwendet AWS X-Ray , um Anfragen zu verfolgen in AWS AppSync.

  Diese Richtlinie ermöglicht die folgenden Aktionen:

  - Abrufen von Stichprobenregeln und deren Ergebnissen
  - Trace-Daten an den X-Ray-Daemon senden

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingTargets",
                "xray:GetSamplingRules",
                "xray:GetSamplingStatisticSummaries"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS AppSync Aktualisierungen der AWS verwalteten Richtlinien

Hier finden Sie Informationen zu Aktualisierungen AWS verwalteter Richtlinien, die AWS AppSync seit Beginn der Nachverfolgung dieser Änderungen durch diesen Dienst vorgenommen wurden.

Abonnieren Sie den RSS-Feed auf der Seite AWS AppSync Dokumentenverlauf, um automatische Benachrichtigungen über Änderungen an dieser Seite zu erhalten.

Änderung	Beschreibung	Datum
AWSAppSyncSchemaAuthor – Aktualisierung auf eine bestehende Richtlinie	Es wurde eine `EvaluateCode` Richtlinienaktion hinzugefügt, die es Benutzern ermöglicht, Code anhand einer Laufzeit und eines Kontextes auszuwerten.	07. Februar 2023
AWSAppSyncSchemaAuthor – Aktualisierung auf eine bestehende Richtlinie	Es wurden Richtlinienaktionen hinzugefügt, um die Funktionen zum Auflisten, Abrufen, Erstellen, Aktualisieren und Löschen für eine API zu ermöglichen. Es wurde eine `EvaluateMappingTemplate` Richtlinienaktion hinzugefügt, mit der Benutzer die Logik der Vorlagen für die Zuordnung von Anfragen und Antworten auf Resolver auswerten können. Es wurden Richtlinienaktionen hinzugefügt, um das Markieren von Ressourcen zu ermöglichen.	25. August 2022
AWS AppSync hat begonnen, Änderungen zu verfolgen	AWS AppSync hat begonnen, Änderungen für die AWS verwalteten Richtlinien zu verfolgen.	25. August 2022

Fehlerbehebung bei AWS AppSync Identität und Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit AWS AppSync und IAM auftreten können.

Ich bin nicht autorisiert, eine Aktion in AWS AppSync auszuführen.

Wenn Ihnen AWS Management Console mitgeteilt wird, dass Sie nicht berechtigt sind, eine Aktion auszuführen, müssen Sie sich an Ihren Administrator wenden, um Unterstützung zu erhalten. Ihr Administrator ist die Person, die Ihnen Ihren Benutzernamen und Ihr Passwort bereitgestellt hat.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer `mateojackson` versucht, die Konsole zu verwenden, um Details zu einer fiktiven *`my-example-widget`* Ressource anzuzeigen, er aber nicht über die fiktiven `appsync:`*`GetWidget`* Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 appsync:GetWidget on resource: my-example-widget
```

In diesem Fall bittet Mateo seinen Administrator um die Aktualisierung seiner Richtlinien, um unter Verwendung der Aktion *`my-example-widget`* auf die Ressource `appsync:`*`GetWidget`* zugreifen zu können.

Ich bin nicht berechtigt, IAM auszuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an AWS AppSync übergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in AWS AppSync auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen odzur Verfügung gestellt.

Ich möchte meine Zugriffsschlüssel anzeigen

Nachdem Sie Ihre IAM-Benutzerzugriffsschlüssel erstellt haben, können Sie Ihre Zugriffsschlüssel-ID jederzeit anzeigen. Sie können Ihren geheimen Zugriffsschlüssel jedoch nicht erneut anzeigen. Wenn Sie den geheimen Zugriffsschlüssel verlieren, müssen Sie ein neues Zugriffsschlüsselpaar erstellen.

Zugriffsschlüssel bestehen aus zwei Teilen: einer Zugriffsschlüssel-ID (z. B. `AKIAIOSFODNN7EXAMPLE`) und einem geheimen Zugriffsschlüssel (z. B. `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Ähnlich wie bei Benutzernamen und Passwörtern müssen Sie die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel zusammen verwenden, um Ihre Anforderungen zu authentifizieren. Verwalten Sie Ihre Zugriffsschlüssel so sicher wie Ihren Benutzernamen und Ihr Passwort.

> **Important**
>
> Geben Sie Ihre Zugriffsschlüssel nicht an Dritte weiter, auch nicht für die Suche nach Ihrer kanonischen Benutzer-ID. Auf diese Weise können Sie jemandem dauerhaften Zugriff auf Ihre gewähren AWS-Konto.

Während der Erstellung eines Zugriffsschlüsselpaars werden Sie aufgefordert, die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel an einem sicheren Speicherort zu speichern. Der geheime Zugriffsschlüssel ist nur zu dem Zeitpunkt verfügbar, an dem Sie ihn erstellen. Wenn Sie Ihren geheimen Zugriffsschlüssel verlieren, müssen Sie Ihrem IAM-Benutzer neue Zugriffsschlüssel hinzufügen. Sie können maximal zwei Zugriffsschlüssel besitzen. Wenn Sie bereits zwei Zugriffschlüssel besitzen, müssen Sie ein Schlüsselpaar löschen, bevor Sie ein neues erstellen. Anweisungen hierfür finden Sie unter Verwalten von Zugriffsschlüsseln im IAM-Benutzerhandbuch.

Ich bin Administrator und möchte anderen den Zugriff ermöglichen AWS AppSync

Um anderen den Zugriff zu ermöglichen AWS AppSync, müssen Sie den Personen oder Anwendungen, die Zugriff benötigen, die entsprechenden Berechtigungen erteilen. Wenn Sie Personen und Anwendungen verwalten, weisen Sie Benutzern oder Gruppen Berechtigungssätze

zu, um deren Zugriffsebene zu definieren. AWS IAM Identity Center Mit Berechtigungssätzen werden automatisch IAM-Richtlinien erstellt und den IAM-Rollen zugewiesen, die der Person oder Anwendung zugeordnet sind. Weitere Informationen finden Sie im AWS IAM Identity Center Benutzerhandbuch unter Berechtigungssätze.

Wenn Sie IAM Identity Center nicht verwenden, müssen Sie IAM-Entitäten (Benutzer oder Rollen) für die Personen oder Anwendungen erstellen, die Zugriff benötigen. Anschließend müssen Sie der Entität eine Richtlinie anfügen, die dieser die korrekten Berechtigungen in AWS AppSync gewährt. Nachdem die Berechtigungen erteilt wurden, stellen Sie dem Benutzer oder Anwendungsentwickler die Anmeldeinformationen zur Verfügung. Sie werden diese Anmeldeinformationen für den Zugriff verwenden AWS. Weitere Informationen zum Erstellen von IAM-Benutzern, -Gruppen, -Richtlinien und -Berechtigungen finden Sie im IAM-Benutzerhandbuch unter IAM-Identitäten sowie Richtlinien und Berechtigungen in IAM.

Ich möchte Personen außerhalb meines Kontos den Zugriff auf meine AWS Ressourcen ermöglichen AWS AppSync

Sie können eine Rolle erstellen, die Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation für den Zugriff auf Ihre Ressourcen verwenden können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen darüber, ob diese Funktionen AWS AppSync unterstützt werden, finden Sie unter. Wie AWS AppSync funktioniert mit IAM
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs auf einen IAM-Benutzer in einem anderen AWS-Konto , den Sie besitzen.
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Protokollieren von AWS AppSync API-Aufrufen mit AWS CloudTrail

AWS AppSync ist in einen Dienst integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in ausgeführt wurden AWS AppSync. CloudTrail erfasst API-Aufrufe AWS AppSync als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der AWS AppSync Konsole und Code-Aufrufe der AWS AppSync API-Operationen. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail Ereignissen an einen Amazon S3 S3-Bucket aktivieren, einschließlich Ereignissen für AWS AppSync. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse trotzdem in der CloudTrail Konsole im Ereignisverlauf anzeigen. Anhand der von gesammelten Informationen können Sie die Anfrage ermitteln CloudTrail, an die die Anfrage gestellt wurde AWS AppSync, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Informationen.

Weitere Informationen CloudTrail dazu finden Sie im AWS CloudTrail Benutzerhandbuch.

AWS AppSync Informationen in CloudTrail

CloudTrail ist in Ihrem AWS Konto aktiviert, wenn Sie das Konto erstellen. Wenn eine Aktivität in stattfindet AWS AppSync, wird diese Aktivität zusammen mit anderen CloudTrail AWS Serviceereignissen im Ereignisverlauf in einem Ereignis aufgezeichnet. Sie können aktuelle Ereignisse in Ihrem AWS Konto ansehen, suchen und herunterladen. Weitere Informationen finden Sie unter Ereignisse mit dem CloudTrail Ereignisverlauf anzeigen.

Für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem AWS Konto, einschließlich der Ereignisse für AWS AppSync, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Wenn Sie einen Trail in der Konsole erstellen, gilt der Trail standardmäßig für alle AWS Regionen. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3 S3-Bucket. Darüber hinaus können Sie andere AWS Dienste konfigurieren, um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie hier:

- Übersicht zum Erstellen eines Trails
- CloudTrail unterstützte Dienste und Integrationen
- Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten

AWS AppSync unterstützt die Protokollierung von Aufrufen über die AWS AppSync API. Derzeit werden sowohl Anrufe an Ihren APIs Server als auch Anrufe an Resolver nicht protokolliert AWS AppSync . CloudTrail

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anfrage mit Root- oder AWS Identity and Access Management (IAM-) Benutzeranmeldedaten gestellt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie unter CloudTrail -Element userIdentity.

AWS AppSync Logdateieinträge verstehen

Ein Trail ist eine Konfiguration, die die Übertragung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket ermöglicht. CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält Informationen über die angeforderte Aktion, Datum und Uhrzeit der Aktion, Anforderungsparameter usw. CloudTrail Protokolldateien sind kein geordneter Stack-Trace der öffentlichen API-Aufrufe, sodass sie nicht in einer bestimmten Reihenfolge angezeigt werden.

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die über die AWS AppSync Konsole ausgeführte `GetGraphqlApi` Aktion demonstriert:

```
{
    "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"
}
```

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `CreateApikey` Aktion demonstriert, die über 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"
}
```

Bewährte Sicherheitsmethoden für AWS AppSync

Bei der Sicherung AWS AppSync geht es um mehr, als nur ein paar Hebel einzuschalten oder die Protokollierung einzurichten. In den folgenden Abschnitten werden bewährte Sicherheitsmethoden behandelt, die je nachdem, wie Sie den Service nutzen, variieren.

Machen Sie sich mit Authentifizierungsmethoden vertraut

AWS AppSync bietet mehrere Möglichkeiten, Ihre Benutzer bei Ihrem APIs GraphQL zu authentifizieren. Jede Methode hat Kompromisse in Bezug auf Sicherheit, Überprüfbarkeit und Benutzerfreundlichkeit.

Die folgenden gängigen Authentifizierungsmethoden sind verfügbar:

- Amazon Cognito Cognito-Benutzerpools ermöglichen es Ihrer GraphQL-API, Benutzerattribute für eine detaillierte Zugriffskontrolle und Filterung zu verwenden.

- API-Token haben eine begrenzte Lebensdauer und eignen sich für automatisierte Systeme wie Continuous Integration-Systeme und die Integration mit externen Systemen. APIs
- AWS Identity and Access Management (IAM) eignet sich für interne Anwendungen, die in Ihrem AWS-Konto verwaltet werden.
- OpenID Connect ermöglicht es Ihnen, den Zugriff mit dem OpenID Connect-Protokoll zu kontrollieren und zu föderieren.

Weitere Informationen zur Authentifizierung und Autorisierung in AWS AppSync finden Sie unter. Konfiguration von Autorisierung und Authentifizierung zur Sicherung Ihres GraphQL APIs

Verstehen Sie, wie sich Änderungen an der API-Konfiguration ausbreiten

Wenn Sie Änderungen an Ihrer API-Konfiguration speichern, AWS AppSync beginnt die Übertragung der Änderungen. Solange Ihre Konfigurationsänderung nicht übernommen wurde, werden AWS AppSync weiterhin Ihre Inhalte aus der vorherigen Konfiguration bereitgestellt. Sobald Ihre Konfigurationsänderung übernommen wurde, wird AWS AppSync sofort mit der Bereitstellung Ihrer Inhalte auf der Grundlage der neuen Konfiguration begonnen. Während AWS AppSync Ihre Änderungen für eine API übertragen werden, können wir nicht anhand der vorherigen Konfiguration oder der neuen Konfiguration feststellen, ob die API Ihre Inhalte bereitstellt.

Verwenden Sie TLS für HTTP-Resolver

Achten Sie bei der Verwendung von HTTP-Resolvern darauf, nach Möglichkeit TLS-gesicherte Verbindungen (HTTPS) zu verwenden. Eine vollständige Liste der vertrauenswürdigen TLS-Zertifikate finden Sie unter AWS AppSync . Von vier HTTPS-Endpunkten anerkannte AWS AppSync Zertifizierungsstellen (CA)

Verwenden Sie Rollen mit den geringstmöglichen Berechtigungen

Wenn Sie Resolver wie den DynamoDB-Resolver verwenden, verwenden Sie Rollen, die die restriktivste Sicht auf Ihre Ressourcen bieten, z. B. Ihre Amazon DynamoDB-Tabellen.

Bewährte Methoden für IAM-Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand AWS AppSync Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Befolgen Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Anleitungen und Empfehlungen:

- Beginnen Sie mit AWS verwalteten Richtlinien und wechseln Sie zu Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um Ihren Benutzern und Workloads zunächst Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter AWS -verwaltete Richtlinien oder AWS -verwaltete Richtlinien für Auftrags-Funktionen im IAM-Benutzerhandbuch.
- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter Richtlinien und Berechtigungen in IAM im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen bestimmten Zweck verwendet werden AWS-Service, z. AWS CloudFormation B. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.
- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter Sicherer API-Zugriff mit MFA im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter Bewährte Methoden für die Sicherheit in IAM im IAM-Benutzerhandbuch.

AWS AppSync Resolver-Referenz () JavaScript

Die folgenden Abschnitte enthalten die `APPSYNC_JS` Laufzeit- und JavaScript Resolver-Referenz:

- JavaScriptÜbersicht über Resolver — Erfahren Sie mehr über die Funktionsweise von Resolvern in. AWS AppSync
- Objektreferenz für den Resolver-Kontext — Erfahren Sie mehr über das Kontextobjekt und seine Verwendung in Resolvern.
- JavaScript Laufzeitfunktionen für Resolver und Funktionen — Erfahren Sie mehr über unterstützte Runtime-Funktionen und die Verwendung von Hilfsprogrammen zur Codevereinfachung.
- JavaScriptResolver-Funktionsreferenz für DynamoDB — Erfahren Sie mehr darüber, wie Resolver mit DynamoDB interagieren.
- JavaScriptReferenz zur Resolver-Funktion für OpenSearch — Erfahren Sie mehr über die Struktur von Anfragen und Antworten auf Resolver sowie über Interaktionen mit dem Service. OpenSearch
- JavaScript Resolver-Funktionsreferenz für Lambda — Erfahren Sie mehr über die Resolver-Anfrage- und Antwortstruktur und Interaktionen mit Lambda.
- JavaScriptResolver-Funktionsreferenz für die EventBridge Datenquelle — Erfahren Sie mehr über die Resolver-Anfrage- und Antwortstruktur und Interaktionen mit. EventBridge
- JavaScript Referenz zur Resolverfunktion für die Datenquelle None — Erfahren Sie mehr über die Anforderungs- und Antwortstruktur des Resolvers sowie über Interaktionen mit NONE-Datenquellen.
- JavaScript Resolver-Funktionsreferenz für HTTP — Erfahren Sie mehr über die Resolver-Anforderungs- und Antwortstruktur und Interaktionen mit HTTP-Endpunkten.
- JavaScript Resolver-Funktionsreferenz für Amazon RDS — Erfahren Sie mehr über die Resolver-Struktur und Interaktionen mit RDS.
- JavaScript Resolver-Funktionsreferenz für Amazon Bedrock — Erfahren Sie mehr über die Resolver-Struktur und Interaktionen mit Amazon Bedrock.

AWS AppSync JavaScript Übersicht über die Resolver

AWS AppSync ermöglicht es Ihnen, auf GraphQL-Anfragen zu antworten, indem Sie Operationen an Ihren Datenquellen ausführen. Für jedes GraphQL-Feld, für das Sie eine Abfrage, Mutation oder ein Abonnement ausführen möchten, muss ein Resolver angehängt werden.

Resolver sind die Verbindungen zwischen GraphQL und einer Datenquelle. Sie erklären, AWS AppSync wie Sie eine eingehende GraphQL-Anfrage in Anweisungen für Ihre Backend-Datenquelle übersetzen und wie Sie die Antwort von dieser Datenquelle zurück in eine GraphQL-Antwort übersetzen. Mit AWS AppSync können Sie Ihre Resolver in der () -Umgebung schreiben JavaScript und ausführen. AWS AppSync `APPSYNC_JS`

AWS AppSync ermöglicht es Ihnen, Unit-Resolver oder Pipeline-Resolver zu schreiben, die aus mehreren AWS AppSync Funktionen in einer Pipeline bestehen.

Unterstützte Runtime-Funktionen

Die AWS AppSync JavaScript Runtime bietet eine Teilmenge von JavaScript Bibliotheken, Dienstprogrammen und Funktionen. Eine vollständige Liste der Features und Funktionen, die von der Runtime unterstützt werden, finden Sie unter `APPSYNC_JS` JavaScript Runtime-Features für Resolver und Funktionen.

Resolver für Einheiten

Ein Unit-Resolver besteht aus Code, der einen Anforderungs- und Antworthandler definiert, die für eine Datenquelle ausgeführt werden. Der Anforderungshandler verwendet ein Kontextobjekt als Argument und gibt die Anforderungsnutzdaten zurück, die zum Aufrufen Ihrer Datenquelle verwendet wurden. Der Antworthandler erhält von der Datenquelle eine Nutzlast mit dem Ergebnis der ausgeführten Anfrage zurück. Der Response-Handler wandelt die Nutzlast in eine GraphQL-Antwort um, um das GraphQL-Feld aufzulösen. Im folgenden Beispiel ruft ein Resolver ein Element aus einer DynamoDB-Datenquelle ab:

```
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;
```

Aufbau eines Pipeline-Resolvers JavaScript

Ein Pipeline-Resolver besteht aus Code, der einen Anfrage- und Antworthandler sowie eine Liste von Funktionen definiert. Jede Funktion hat einen Anforderungs - und Antworthandler, den sie für eine

Datenquelle ausführt. Da ein Pipeline-Resolver Läufe an eine Liste von Funktionen delegiert, ist er daher mit keiner Datenquelle verknüpft. Unit-Resolver und -Funktionen sind Primitive, die Operation auf Datenquellen auszuführen.

Anforderungshandler für Pipeline-Resolver

Der Anforderungshandler eines Pipeline-Resolvers (der vorherige Schritt) ermöglicht es Ihnen, einige Vorbereitungslogik durchzuführen, bevor Sie die definierten Funktionen ausführen.

Funktionsliste

Die Liste der Funktionen eines Pipeline-Resolvers wird nacheinander ausgeführt. Das Evaluierungsergebnis des Pipeline-Resolver-Request-Handlers wird der ersten Funktion als zur Verfügung gestellt. `ctx.prev.result` Jedes Ergebnis der Funktionsauswertung steht der nächsten Funktion als `ctx.prev.result` zur Verfügung.

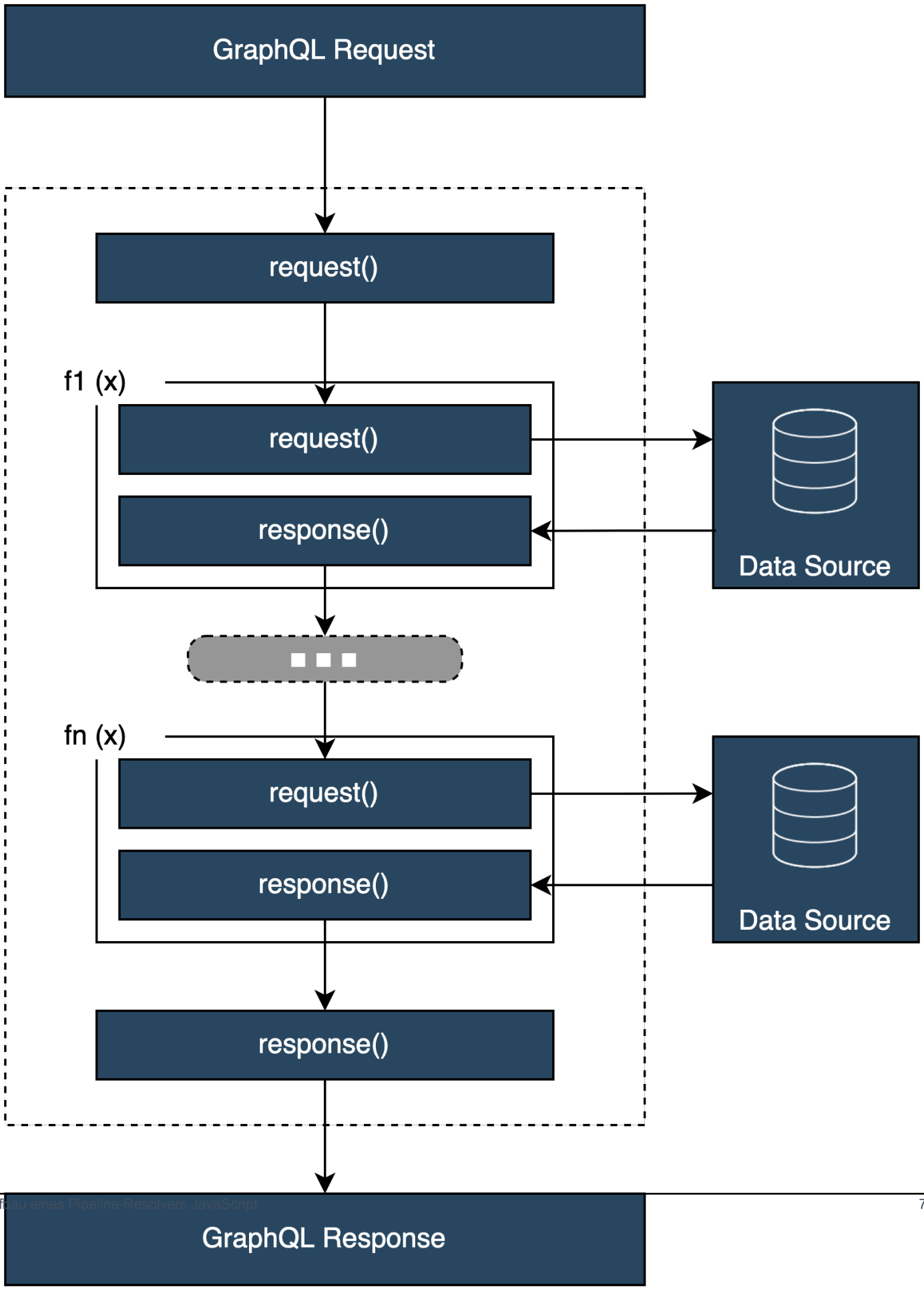
Antworthandler für den Pipeline-Resolver

Der Response-Handler eines Pipeline-Resolvers ermöglicht es Ihnen, einige letzte Logik von der Ausgabe der letzten Funktion bis zum erwarteten GraphQL-Feldtyp auszuführen. Die Ausgabe der letzten Funktion in der Funktionsliste ist im Response-Handler des Pipeline-Resolvers als oder verfügbar. `ctx.prev.result` `ctx.result`

Ablauf der Ausführung

Bei einem Pipeline-Resolver, der aus zwei Funktionen besteht, stellt die folgende Liste den Ausführungsablauf dar, wenn der Resolver aufgerufen wird:

1. Anforderungshandler für Pipeline-Resolver
2. Funktion 1: Funktionsanforderungshandler
3. Funktion 1: Aufruf der Datenquelle
4. Funktion 1: Funktionsantwort-Handler
5. Funktion 2: Funktionsanforderungshandler
6. Funktion 2: Aufruf der Datenquelle
7. Funktion 2: Funktionsantwort-Handler
8. Antworthandler für den Pipeline-Resolver


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

Nützliche integrierte `APPSYNC_JS` Runtime-Dienstprogramme

Die folgenden Dienstprogramme unterstützen die Arbeit mit Pipeline-Resolvern.

ctx.stash

Der Stash ist ein Objekt, das in jedem Resolver und jedem Funktionsanforderungs- und Antworthandler zur Verfügung gestellt wird. Dieselbe Stash-Instanz durchläuft einen einzigen Resolver-Lauf. Das bedeutet, dass Sie den Stash verwenden können, um beliebige Daten zwischen Anfrage- und Antworthandlern und zwischen Funktionen in einem Pipeline-Resolver zu übergeben. Sie können den Stash wie ein normales Objekt testen. JavaScript

ctx.prev.result

`ctx.prev.result` ist das Ergebnis der vorherigen Operation, die in der Pipeline ausgeführt wurde. Wenn es sich bei der vorherigen Operation um den Anforderungshandler für den Pipeline-Resolver handelte, `ctx.prev.result` wird er der ersten Funktion in der Kette zur Verfügung gestellt. Wenn die vorherige Operation die erste Funktion betraf, steht `ctx.prev.result` für die Ausgabe der ersten Funktion und wird der zweiten Funktion in der Pipeline zur Verfügung gestellt. Wenn die vorherige Operation die letzte Funktion war, stellt sie die Ausgabe der letzten Funktion `ctx.prev.result` dar und wird dem Response-Handler des Pipeline-Resolvers zur Verfügung gestellt.

util.error

Das Dienstprogramm `util.error` ist nützlich, um ein Feldfehler auszulösen. Die Verwendung `util.error` innerhalb eines Funktionsanforderungs- oder Antworthandlers löst sofort einen Feldfehler aus, der verhindert, dass nachfolgende Funktionen ausgeführt werden. Weitere Informationen und andere `util.error` Signaturen finden Sie unter JavaScriptLaufzeitfunktionen für Resolver und Funktionen.

util.AppendError

`util.appendError`ist ähnlich wie`util.error()`, mit dem großen Unterschied, dass es die Auswertung des Handlers nicht unterbricht. Stattdessen signalisiert es, dass ein Fehler mit dem Feld aufgetreten ist, ermöglicht aber die Auswertung des Handlers und damit die Rückgabe von Daten. Die Verwendung von `util.appendError` in einer Funktion hat keine Auswirkungen auf die Ausführung des Pipeline-Ablaufs. Weitere Informationen und andere `util.error` Signaturen finden Sie in den JavaScript Laufzeitfunktionen für Resolver und Funktionen.

Runtime.EarlyReturn

Mit dieser `runtime.earlyReturn` Funktion können Sie von jeder Anforderungsfunktion vorzeitig zurückkehren. Wenn Sie den Request-Handler `runtime.earlyReturn` innerhalb eines Resolvers verwenden, wird er vom Resolver zurückgegeben. Wenn Sie ihn von einem AWS AppSync Funktionsanforderungshandler aus aufrufen, kehren Sie von der Funktion zurück und setzen die Ausführung entweder zur nächsten Funktion in der Pipeline oder zum Resolver-Response-Handler fort.

Pipeline-Resolver schreiben

Ein Pipeline-Resolver hat außerdem einen Request- und einen Response-Handler, der die Ausführung der Funktionen in der Pipeline umgibt: Sein Request-Handler wird vor der Anfrage der ersten Funktion ausgeführt, und sein Response-Handler wird nach der Antwort der letzten Funktion ausgeführt. Der Resolver-Anforderungshandler kann Daten so einrichten, dass sie von Funktionen in der Pipeline verwendet werden. Der Resolver-Response-Handler ist für die Rückgabe von Daten verantwortlich, die dem GraphQL-Feldausgabetyp zugeordnet sind. Im folgenden Beispiel definiert ein Resolver-Request-Handler`allowedGroups`; die zurückgegebenen Daten sollten zu einer dieser Gruppen gehören. Dieser Wert kann von den Funktionen des Resolvers verwendet werden, um Daten anzufordern. Der Response-Handler des Resolvers führt eine letzte Prüfung durch und filtert das Ergebnis, um sicherzustellen, dass nur Elemente zurückgegeben werden, die zu den zulässigen Gruppen gehören.

```
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;
}
```

Funktionen schreiben AWS AppSync

AWS AppSync Funktionen ermöglichen es Ihnen, allgemeine Logik zu schreiben, die Sie für mehrere Resolver in Ihrem Schema wiederverwenden können. Sie können beispielsweise eine AWS AppSync Funktion aufrufen, die für `QUERY_ITEMS` die Abfrage von Elementen aus einer Amazon DynamoDB DynamoDB-Datenquelle zuständig ist. Für Resolver, mit denen Sie Elemente abfragen möchten, fügen Sie die Funktion einfach zur Pipeline des Resolvers hinzu und geben Sie den zu verwendenden Abfrageindex an. Die Logik muss nicht erneut implementiert werden.

Zusätzliche Themen

Topics

- Beispiel für einen Pipeline-Resolver mit Amazon DynamoDB
- Konfiguration von Dienstprogrammen für die Laufzeit `APPSYNC_JS`
- Bündelung TypeScript und Quellzuordnungen für die Runtime `APPSYNC_JS`
- Testen Sie Ihren Resolver und Ihre Funktionshandler
- Migration von VTL zu JavaScript
- Wahl zwischen direktem Datenquellenzugriff und Proxying über eine Lambda-Datenquelle

Beispiel für einen Pipeline-Resolver mit Amazon DynamoDB

Angenommen, Sie möchten einen Pipeline-Resolver an ein Feld mit dem Namen anhängen`getPost(id:ID!)`, das einen `Post` Typ aus einer Amazon DynamoDB DynamoDB-Datenquelle mit der folgenden GraphQL-Abfrage zurückgibt:

```
getPost(id:1){
    id
    title
    content
```

```
}
```

Hängen Sie zunächst einen einfachen Resolver mit dem folgenden Code an`Query.getPost`. Dies ist ein Beispiel für einfachen Resolver-Code. Im Request-Handler ist keine Logik definiert, und der Response-Handler gibt einfach das Ergebnis der letzten Funktion zurück.

```
/**
 * 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
}
```

Definieren Sie als Nächstes eine Funktion`GET_ITEM`, die ein Postitem aus Ihrer Datenquelle abruft:

```
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
}
```

Wenn während der Anfrage ein Fehler auftritt, hängt der Antworthandler der Funktion einen Fehler an, der in der GraphQL-Antwort an den aufrufenden Client zurückgegeben wird. Fügen Sie die `GET_ITEM` Funktion zu Ihrer Liste der Resolver-Funktionen hinzu. Wenn Sie die Abfrage ausführen, verwendet der Request-Handler der `GET_ITEM` Funktion die vom DynamoDB-Modul bereitgestellten AWS AppSync Utils, um eine `DynamoDBGetItem` Anfrage mit dem `id` als Schlüssel zu erstellen. `ddb.get({ key: { id } })`generiert die entsprechende Operation: `GetItem`

```
{
    "operation" : "GetItem",
    "key" : {
        "id" : { "S" : "1" }
    }
}
```

AWS AppSync verwendet die Anfrage, um die Daten von Amazon DynamoDB abzurufen. Sobald die Daten zurückgegeben wurden, werden sie vom Response-Handler der `GET_ITEM` Funktion verarbeitet, der nach Fehlern sucht und dann das Ergebnis zurückgibt.

```
{
  "result" : {
    "id": 1,
    "title": "hello world",
    "content": "<long story>"
  }
}
```

Schließlich gibt der Response-Handler des Resolvers das Ergebnis direkt zurück.

Mit Fehlern arbeiten

Wenn in Ihrer Funktion während einer Anfrage ein Fehler auftritt, wird der Fehler in Ihrem Funktionsantwort-Handler unter verfügbar gemacht`ctx.error`. Sie können den Fehler mit dem Hilfsprogramm an Ihre GraphQL-Antwort anhängen. `util.appendError` Sie können den Fehler mithilfe des Stash für andere Funktionen in der Pipeline verfügbar machen. Sehen Sie sich das folgende Beispiel an:

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

Dienstprogramme für die **APPSYNC_JS** Laufzeit konfigurieren

AWS AppSync stellt zwei Bibliotheken zur Verfügung, die bei der Entwicklung von Resolvern mit der APPSYNC_JS Runtime helfen:

- `@aws-appsync/eslint-plugin`- Erkennt und behebt Probleme während der Entwicklung schnell.
- `@aws-appsync/utils`- Ermöglicht Typvalidierung und Autovervollständigung in Code-Editoren.

Konfiguration des Eslint-Plugins

ESLintist ein Tool, das Ihren Code statisch analysiert, um Probleme schnell zu finden. Sie können es ESLint als Teil Ihrer kontinuierlichen Integrationspipeline ausführen. `@aws-appsync/eslint-plugin`ist ein ESLint Plugin, das bei der Nutzung der Laufzeit ungültige Syntax in Ihrem Code

erkennt. APPSYNC_JS Das Plugin ermöglicht es Ihnen, während der Entwicklung schnell Feedback zu Ihrem Code zu erhalten, ohne Ihre Änderungen in die Cloud übertragen zu müssen.

`@aws-appsync/eslint-plugin`bietet zwei Regelsätze, die Sie während der Entwicklung verwenden können.

„plugin: @aws -appsync/base“ konfiguriert ein Basisregelwerk, das Sie in Ihrem Projekt nutzen können:

Regel	Beschreibung
nicht asynchron	Asynchrone Prozesse und Versprechen werden nicht unterstützt.
kein Warten	Asynchrone Prozesse und Versprechen werden nicht unterstützt.
keine Klassen	Klassen werden nicht unterstützt.
nein für	`for`wird nicht unterstützt (außer für `for-in` und`for-of`, die unterstützt werden)
nicht weitermachen	`continue` wird nicht unterstützt.
keine Generatoren	Generatoren werden nicht unterstützt.
kein Ertrag	`yield` wird nicht unterstützt.
keine Etiketten	Labels werden nicht unterstützt.
nein, das	`this`Schlüsselwort wird nicht unterstützt.
versuchen Sie es nicht	Die Try/Catch-Struktur wird nicht unterstützt.
keine Weile	While-Loops werden nicht unterstützt.
no-disallowed-unary-operators	`++`,`--`, und ~ unäre Operatoren sind nicht zulässig.
no-disallowed-binary-operators	Der `instanceof` Operator ist nicht erlaubt.

Regel	Beschreibung
kein Versprechen	Asynchrone Prozesse und Versprechen werden nicht unterstützt.

„plugin: @aws -appsync/recommended“ bietet einige zusätzliche Regeln, erfordert aber auch, dass Sie Ihrem Projekt Konfigurationen hinzufügen TypeScript .

Regel	Beschreibung
keine Rekursion	Rekursive Funktionsaufrufen sind nicht erlaubt.
no-disallowed-methods	Einige Methoden sind nicht erlaubt. Eine vollständige Liste der unterstützten integrierten Funktionen finden Sie in der Referenz.
no-function-passing	Das Übergeben von Funktionen als Funktions argumente an Funktionen ist nicht zulässig.
no-function-reassign	Funktionen können nicht neu zugewiesen werden.
no-function-return	Funktionen können nicht der Rückgabewert von Funktionen sein.

Um das Plugin zu Ihrem Projekt hinzuzufügen, folgen Sie den Installations- und Nutzungsschritten unter Erste Schritte mit ESLint. Installieren Sie dann das Plugin in Ihrem Projekt mit Ihrem Projektpaketmanager (z. B. npm, yarn oder pnpm):

```
$ npm install @aws-appsync/eslint-plugin
```

Fügen Sie in Ihrer `.eslintrc.{js,yml,json}` Datei der Eigenschaft „plugin: @aws -appsync/base“ oder „plugin: @aws -appsync/recommended“ hinzu. `extends` Das folgende Snippet ist eine grundlegende Beispielkonfiguration für: `.eslintrc` JavaScript

```
{
  "extends": ["plugin:@aws-appsync/base"]
```

```
}
```

Um den Regelsatz „plugin: @aws -appsync/recommended“ zu verwenden, installieren Sie die erforderliche Abhängigkeit:

```
$ npm install -D @typescript-eslint/parser
```

Erstellen Sie dann eine Datei: `.eslintrc.js`

```
{
  "parser": "@typescript-eslint/parser",
  "parserOptions": {
    "ecmaVersion": 2018,
    "project": "./tsconfig.json"
  },
  "extends": ["plugin:@aws-appsync/recommended"]
}
```

Bündelung TypeScript, und Quellzuordnungen für die Laufzeit **APPSYNC_JS**

TypeScript verbessert die AWS AppSync Entwicklung durch Typsicherheit und frühzeitige Fehlererkennung. Sie können TypeScript Code lokal schreiben und ihn transpilieren, JavaScript bevor Sie ihn mit der APPSYNC_JS Runtime verwenden. Der Prozess beginnt mit der Installation TypeScript und Konfiguration von tsconfig.json für die Umgebung. APPSYNC_JS Anschließend können Sie Bündelungstools wie esbuild verwenden, um den Code zu kompilieren und zu bündeln. Die Amplify-CLI generiert Typen aus dem GraphQL-Schema, sodass Sie diese Typen im Resolver-Code verwenden können.

Sie können benutzerdefinierte und externe Bibliotheken in Ihrem Resolver und Funktionscode nutzen, sofern diese den Anforderungen entsprechen. APPSYNC_JS Bündelungstools kombinieren Code in einer einzigen Datei zur Verwendung in. AWS AppSync Quellzuordnungen können hinzugefügt werden, um das Debuggen zu erleichtern.

Nutzen Sie Bibliotheken und bündeln Sie Ihren Code

In Ihrem Resolver und Funktionscode können Sie sowohl benutzerdefinierte als auch externe Bibliotheken nutzen, sofern sie den Anforderungen entsprechen. APPSYNC_JS Dadurch ist es möglich, vorhandenen Code in Ihrer Anwendung wiederzuverwenden. Um Bibliotheken zu

verwenden, die durch mehrere Dateien definiert sind, müssen Sie ein Bündelungstool wie Esbuild verwenden, um Ihren Code in einer einzigen Datei zu kombinieren, die dann auf Ihrem AWS AppSync Resolver oder Ihrer Funktion gespeichert werden kann.

Beachten Sie beim Bündeln Ihres Codes Folgendes:

- `APPSYNC_JS`unterstützt nur ECMAScript Module (ESM).
- `@aws-appsync/*`Module sind in Ihren Code integriert `APPSYNC_JS` und sollten nicht mit diesem gebündelt werden.
- Die `APPSYNC_JS` Laufzeitumgebung ähnelt NodeJS darin, dass Code nicht in einer Browserumgebung ausgeführt wird.
- Sie können eine optionale Quellzuordnung hinzufügen. Schließen Sie den Quellinhalt jedoch nicht ein.

  Weitere Informationen zu Quellzuordnungen finden Sie unter Quellzuordnungen verwenden.

Um beispielsweise Ihren Resolver-Code unter zu bündeln`src/appsync/getPost.resolver.js`, können Sie den folgenden esbuild-CLI-Befehl verwenden:

```
$ 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
```

Erstellen Sie Ihren Code und arbeiten Sie mit TypeScript

TypeScriptist eine von Microsoft entwickelte Programmiersprache, die alle Funktionen zusammen mit dem TypeScript Tippsystem bietet. JavaScript Sie können TypeScript damit typsicheren Code schreiben und Fehler und Bugs bei der Erstellung abfangen, bevor Sie Ihren Code unter speichern. AWS AppSync Das `@aws-appsync/utils` Paket ist vollständig typisiert.

Die `APPSYNC_JS` Runtime unterstützt nicht TypeScript direkt. Sie müssen Ihren TypeScript Code zuerst in Code transpilieren JavaScript , den die `APPSYNC_JS` Runtime unterstützt, bevor Sie Ihren

Code darin AWS AppSync speichern. Sie können TypeScript damit Ihren Code in Ihrer lokalen integrierten Entwicklungsumgebung (IDE) schreiben. Beachten Sie jedoch, dass Sie in der AWS AppSync Konsole keinen TypeScript Code erstellen können.

Stellen Sie zunächst sicher, dass Sie die TypeScriptInstallation in Ihrem Projekt installiert haben. Konfigurieren Sie dann Ihre Einstellungen für die TypeScript Transkompilierung so, dass sie mit der `APPSYNC_JS` Laufzeitumgebung funktionieren. TSConfig Hier ist ein Beispiel für eine `tsconfig.json` Basisdatei, die Sie verwenden können:

```
// tsconfig.json
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
   "noEmit": true,
   "moduleResolution": "node",
  }
}
```

Sie können dann ein Bündelungstool wie esbuild verwenden, um Ihren Code zu kompilieren und zu bündeln. Bei einem Projekt, in dem sich Ihr AWS AppSync Code befindet, können Sie beispielsweise den folgenden Befehl verwenden`src/appsync`, um Ihren Code zu kompilieren und zu bündeln:

```
$ esbuild --bundle \
--sourcemap=inline \
--sources-content=false \
--target=esnext \
--platform=node \
--format=esm \
--external:@aws-appsync/utils \
--outdir=out/appsync \
 src/appsync/**/*.ts
```

Verwenden von Amplify Codegen

Sie können die Amplify CLI verwenden, um die Typen für Ihr Schema zu generieren. Führen Sie in dem Verzeichnis, in dem sich Ihre `schema.graphql` Datei befindet, den folgenden Befehl aus und überprüfen Sie die Eingabeaufforderungen zur Konfiguration Ihres Codegens:

```
$  npx @aws-amplify/cli codegen add
```

Um Ihr Codegen unter bestimmten Umständen neu zu generieren (z. B. wenn Ihr Schema aktualisiert wird), führen Sie den folgenden Befehl aus:

```
$ npx @aws-amplify/cli codegen
```

Sie können dann die generierten Typen in Ihrem Resolver-Code verwenden. Zum Beispiel anhand des folgenden Schemas:

```
type Todo {
  id: ID!
  title: String!
  description: String
}

type Mutation {
  createTodo(title: String!, description: String): Todo
}

type Query {
  listTodos: Todo
}
```

Sie könnten die generierten Typen in der folgenden AWS AppSync Beispielfunktion verwenden:

```
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
}
```

Verwendung von Generika in TypeScript

Sie können Generika mit mehreren der bereitgestellten Typen verwenden. Das folgende Snippet ist beispielsweise ein Typ: Todo

```
export type Todo = {
  __typename: "Todo",
  id: string,
  title: string,
  description?: string | null,
};
```

Sie können einen Resolver für ein Abonnement schreiben, das von verwendet. `Todo` In Ihrer IDE führen Sie Typdefinitionen und Hinweise zur automatischen Vervollständigung dazu, das `toSubscriptionFilter` Transform-Hilfsprogramm richtig zu verwenden:

```
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
}
```

Deine Bündel einbinden

Du kannst deine Bundles automatisch fesseln, indem du das Plugin importierst. `esbuild-plugin-eslint` Sie können es dann aktivieren, indem Sie einen `plugins` Wert angeben, der die Eslint-Funktionen aktiviert. Unten finden Sie ein Snippet, das die JavaScript Esbuild-API in einer Datei namens verwendet: `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 })],
})
```

Quellzuordnungen verwenden

Sie können eine Inline-Quellkarte (`sourcemap`) mit Ihrem JavaScript Code bereitstellen. Quellzuordnungen sind nützlich, wenn Sie bündeln JavaScript oder TypeScript programmieren und Verweise auf Ihre Eingabequelldateien in Ihren Protokollen und JavaScript Laufzeitfehlermeldungen sehen möchten.

Sie `sourcemap` müssen am Ende Ihres Codes erscheinen. Es wird durch eine einzelne Kommentarzeile definiert, die dem folgenden Format folgt:

```
//# sourceMappingURL=data:application/json; base64,<base64 encoded string>
```

Ein Beispiel:

```
//# sourceMappingURL=data:application/
json; base64,ewogICJ2ZXJzaW9uIjogMywKICAic291cmNlcyI6IFsibGliLmpzIiwgImNvZGUuanMiXSwKICAibWFwcGl
```

Quellzuordnungen können mit esbuild erstellt werden. Das folgende Beispiel zeigt Ihnen, wie Sie die JavaScript Esbuild-API verwenden, um beim Erstellen und Bündeln von Code eine Inline-Quellenzuweisung einzubeziehen:

```
/* 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 })],
})
```

Insbesondere geben die `sourcesContent` Optionen `sourcemap` und an, dass am Ende jedes Builds eine Quellzuordnung in der Zeile hinzugefügt werden soll, den Quellinhalt jedoch nicht enthalten sollte. Als Konvention empfehlen wir, keine Quellinhalte in Ihre aufzunehmen`sourcemap`. Sie können dies in Esbuild deaktivieren, indem Sie `sources-content` auf `false` setzen.

Um zu veranschaulichen, wie Quellzuordnungen funktionieren, sehen Sie sich das folgende Beispiel an, in dem ein Resolver-Code auf Hilfsfunktionen aus einer Hilfsbibliothek verweist. Der Code enthält Protokollanweisungen im Resolver-Code und in der Hilfsbibliothek:

. /src/default.resolver.ts (dein Resolver)

```
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 (eine Hilfsdatei)

```
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..')
}
```

Wenn Sie die Resolver-Datei erstellen und bündeln, enthält Ihr Resolver-Code eine Inline-Quellzuweisung. Wenn Ihr Resolver ausgeführt wird, erscheinen die folgenden Einträge in den Protokollen: CloudWatch

```
INFO - ../src/default.resolver.ts:5:2: "start >"
INFO - ../src/helper.ts:3:2: "[logger]" "<hello world>" "<42>" "<true>"
INFO - ../src/default.resolver.ts:7:2: "< end"
{"logType":"BeforeRequestFunctionEvaluation","path":["logstuff"],"fieldName":"logstuff","resolverArn":"arn:aws:
INFO - ../src/helper.ts:8:2: "i just say hello.."
{"logType":"AfterResponseFunctionEvaluation","path":["logstuff"],"fieldName":"logstuff","resolverArn":"arn:aws:
```

Wenn Sie sich die Einträge im CloudWatch Protokoll ansehen, werden Sie feststellen, dass die Funktionalität der beiden Dateien gebündelt wurde und gleichzeitig ausgeführt wird. Der ursprüngliche Dateiname jeder Datei spiegelt sich auch deutlich in den Protokollen wider.

Testen Sie Ihren Resolver und Ihre Funktionshandler in AWS AppSync

Sie können den `EvaluateCode` API-Befehl verwenden, um Ihren Resolver und Funktionshandler aus der Ferne mit gefälschten Daten zu testen, bevor Sie Ihren Code in einem Resolver oder einer Funktion speichern. Um mit dem Befehl zu beginnen, stellen Sie sicher, dass Sie die `appsync:evaluatecode` entsprechende Berechtigung zu Ihrer Richtlinie hinzugefügt haben. Zum Beispiel:

JSON

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

```
        }
    ]
}
```

Sie können den Befehl nutzen, indem Sie die AWS CLI oder verwenden AWS SDKs. Um beispielsweise Ihren Code mit der CLI zu testen, zeigen Sie einfach auf Ihre Datei, geben Sie einen Kontext an und geben Sie den Handler an, den Sie auswerten möchten:

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

Die Antwort `evaluationResult` enthält eine, die die von Ihrem Handler zurückgegebene Nutzlast enthält. Es enthält auch ein `logs` Objekt, das die Liste der Protokolle enthält, die von Ihrem Handler während der Auswertung generiert wurden. Auf diese Weise können Sie Ihre Codeausführung einfach debuggen und Informationen zu Ihrer Evaluierung abrufen, um die Fehlerbehebung zu erleichtern. Zum Beispiel:

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

Das Evaluationsergebnis kann als JSON analysiert werden, was Folgendes ergibt:

```
{
  "operation": "PutItem",
  "key": {
    "id": {
      "S": "record-id"
    }
  },
  "attributeValues": {
```

```
    "owner": {
      "S": "John doe"
    },
    "expectedVersion": {
      "N": 2
    },
    "authorId": {
      "S": "Sammy Davis"
    }
  }
}
```

Mit dem SDK können Sie auf einfache Weise Tests aus Ihrer Testsuite integrieren, um das Verhalten Ihres Codes zu überprüfen. Unser Beispiel hier verwendet das Jest Testing Framework, aber jede Testsuite funktioniert. Der folgende Ausschnitt zeigt einen hypothetischen Validierungslauf. Beachten Sie, dass wir davon ausgehen, dass es sich bei der Bewertungsantwort um ein gültiges JSON handelt. Daher verwenden wir diese Methode, `JSON.parse` um JSON aus der Zeichenkettenantwort abzurufen:

```
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)
})
```

Dies führt zu dem folgenden Ergebnis:

```
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
```

Migration von VTL zu in JavaScript AWS AppSync

AWS AppSync ermöglicht es Ihnen, Ihre Geschäftslogik für Ihre Resolver und Funktionen mit VTL oder zu schreiben. JavaScript In beiden Sprachen schreiben Sie eine Logik, die den AWS AppSync Service anweist, wie er mit Ihren Datenquellen interagieren soll. Mit VTL schreiben Sie Zuordnungsvorlagen, die eine gültige JSON-kodierte Zeichenfolge ergeben müssen. Mit schreiben Sie JavaScript Anforderungs- und Antworthandler, die Objekte zurückgeben. Sie geben keine JSON-kodierte Zeichenfolge zurück.

Verwenden Sie zum Beispiel die folgende VTL-Zuordnungsvorlage, um ein Amazon DynamoDB DynamoDB-Element abzurufen:

```
{
    "operation": "GetItem",
    "key": {
        "id": $util.dynamodb.toDynamoDBJson($ctx.args.id),
    }
}
```

Das Hilfsprogramm `$util.dynamodb.toDynamoDBJson` gibt eine JSON-kodierte Zeichenfolge zurück. Wenn auf gesetzt `$ctx.args.id` ist<id>, ergibt die Vorlage eine gültige JSON-kodierte Zeichenfolge:

```
{
    "operation": "GetItem",
    "key": {
        "id": {"S": "<id>"},
    }
}
```

Wenn Sie mit arbeiten JavaScript, müssen Sie keine rohen JSON-codierten Zeichenketten in Ihrem Code ausdrucken, und die Verwendung eines Hilfsprogramms wie ist nicht erforderlich. `toDynamoDBJson` Ein äquivalentes Beispiel für die obige Mapping-Vorlage ist:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'GetItem',
    key: {id: util.dynamodb.toDynamoDB(ctx.args.id)}
  };
}
```

Eine Alternative ist die Verwendung `util.dynamodb.toMapValues` des empfohlenen Ansatzes zur Behandlung eines Werteobjekts:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'GetItem',
    key: util.dynamodb.toMapValues({ id: ctx.args.id }),
  };
}
```

Das ergibt:

```
{
  "operation": "GetItem",
  "key": {
    "id": {
      "S": "<id>"
    }
  }
}
```

Note

Wir empfehlen, das DynamoDB-Modul mit DynamoDB-Datenquellen zu verwenden:

```
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
 ddb.get({ key: { id: ctx.args.id } })
}
```

Nehmen Sie als weiteres Beispiel die folgende Zuordnungsvorlage, um ein Element in eine Amazon DynamoDB DynamoDB-Datenquelle einzufügen:

```
{
    "operation" : "PutItem",
    "key" : {
        "id": $util.dynamodb.toDynamoDBJson($util.autoId()),
    },
    "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
}
```

Bei der Auswertung muss diese Zeichenfolge für die Zuordnungsvorlage eine gültige JSON-kodierte Zeichenfolge ergeben. Bei Verwendung gibt Ihr Code JavaScript das Anforderungsobjekt direkt zurück:

```
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),
  };
}
```

was wie folgt ausgewertet wird:

```
{
  "operation": "PutItem",
  "key": {
    "id": { "S": "2bff3f05-ff8c-4ed8-92b4-767e29fc4e63" }
  },
  "attributeValues": {
    "firstname": { "S": "Shaggy" },
    "age": { "N": 4 }
  }
}
```

> **Note**
>
> Wir empfehlen, das DynamoDB-Modul mit DynamoDB-Datenquellen zu verwenden:

```
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 })
}
```

Wahl zwischen direktem Datenquellenzugriff und Proxying über eine Lambda-Datenquelle

Mit AWS AppSync und der `APPSYNC_JS` Runtime können Sie Ihren eigenen Code schreiben, der Ihre benutzerdefinierte Geschäftslogik implementiert, indem Sie AWS AppSync Funktionen für den Zugriff auf Ihre Datenquellen verwenden. Dies erleichtert Ihnen die direkte Interaktion mit Datenquellen wie Amazon DynamoDB, Aurora Serverless, OpenSearch Service, HTTP und anderen AWS Diensten APIs, ohne zusätzliche Rechendienste oder Infrastruktur bereitstellen zu müssen. AWS AppSync macht es auch einfach, mit einer AWS Lambda Funktion zu interagieren, indem eine Lambda-Datenquelle konfiguriert wird. Mit Lambda-Datenquellen können Sie komplexe Geschäftslogik ausführen und dabei alle Funktionen nutzen AWS Lambda, um eine GraphQL-Anfrage zu lösen. In den meisten Fällen bietet eine AWS AppSync Funktion, die direkt mit der Zieldatenquelle verbunden ist, alle Funktionen, die Sie benötigen. In Situationen, in denen Sie komplexe Geschäftslogik implementieren müssen, die von der `APPSYNC_JS` Laufzeit nicht unterstützt wird, können Sie eine Lambda-Datenquelle als Proxy für die Interaktion mit Ihrer Zieldatenquelle verwenden.

	Direkte Datenquellenintegration	Lambda-Datenquelle als Proxy
Anwendungsfall	AWS AppSync Funktionen interagieren direkt mit API-Datenquellen.	AWS AppSync Funktionen rufen Lambdas auf, die mit API-Datenquellen interagieren.
Laufzeit	*`APPSYNC_JS`* (JavaScript)	Jede unterstützte Lambda-Laufzeit

Maximale Größe des Codes	32.000 Zeichen pro Funktion AWS AppSync	50 MB (gezippt, für direkten Upload) pro Lambda
Externe Module	Eingeschränkt — nur von APPSYNC_JS unterstützte Funktionen	Ja
Rufen Sie einen beliebigen Dienst an AWS	Ja, es wird eine AWS AppSync HTTP-Datenquelle verwendet	Ja — SDK verwenden AWS
Zugriff auf den Anforderungsheader	Ja	Ja
Netzwerkzugriff	Nein	Ja
Zugriff auf das Dateisystem	Nein	Ja
Protokollierung und Metriken	Ja	Ja
Entwickeln und testen Sie komplett innerhalb AppSync	Ja	Nein
Kaltstart	Nein	Nein — Mit bereitgestellter Parallelität
Auto Scaling	Ja — transparent von AWS AppSync	Ja — Wie in Lambda konfiguriert
Preisgestaltung	Keine zusätzlichen Kosten	Wird für die Nutzung von Lambda berechnet

AWS AppSync Funktionen, die sich direkt in ihre Zieldatenquelle integrieren lassen, eignen sich ideal für Anwendungsfälle wie die folgenden:

- Interaktion mit Amazon DynamoDB, Aurora Serverless und Service OpenSearch
- Interaktion mit HTTP APIs und Weitergabe eingehender Header
- Interaktion mit AWS Diensten, die HTTP-Datenquellen verwenden (mit AWS AppSync automatischem Signieren von Anfragen mit der angegebenen Datenquellenrolle)

- Implementierung der Zugriffskontrolle vor dem Zugriff auf Datenquellen
- Implementierung der Filterung der abgerufenen Daten vor der Erfüllung einer Anfrage
- Implementierung einer einfachen Orchestrierung mit sequentieller Ausführung von AWS AppSync Funktionen in einer Resolver-Pipeline
- Steuerung von Caching- und Abonnementverbindungen bei Abfragen und Mutationen.

AWS AppSync Funktionen, die eine Lambda-Datenquelle als Proxy verwenden, eignen sich ideal für Anwendungsfälle wie die folgenden:

- Verwenden Sie eine andere Sprache als JavaScript Velocity Template Language (VTL)
- Anpassung und Steuerung der CPU oder des Speichers zur Leistungsoptimierung
- Importieren von Bibliotheken von Drittanbietern oder Anfordern nicht unterstützter Funktionen in `APPSYNC_JS`
- Mehrere Netzwerkanfragen stellen und Dateisystemzugriff and/or erhalten, um eine Anfrage zu erfüllen
- Batchverarbeitung von Anfragen mithilfe der Batching-Konfiguration.

AWS AppSync JavaScript Resolver-Kontext-Objektreferenz

AWS AppSync definiert eine Reihe von Variablen und Funktionen für die Arbeit mit Anfrage- und Antworthandlern. Dies erleichtert logische Operationen an Daten mit GraphQL. Dieses Dokument beschreibt diese Funktionen und enthält Beispiele.

Zugreifen auf den **`context`**

Das `context` Argument eines Request- und Response-Handlers ist ein Objekt, das alle Kontextinformationen für Ihren Resolver-Aufruf enthält. Sie hat die folgende Struktur:

```
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**
>
> Sie werden häufig feststellen, dass das `context` Objekt als bezeichnet wird. `ctx`

Jedes Feld im `context` Objekt ist wie folgt definiert:

`context`-Felder

`arguments`

Eine Zuordnung, die alle GraphQL-Argumente für dieses Feld enthält.

`identity`

Ein Objekt, das Informationen über den Aufrufer enthält. Weitere Informationen zur Struktur dieses Felds finden Sie unter Identity.

`source`

Eine Zuordnung, die die Auflösung des übergeordneten Feldes enthält.

`stash`

Der Stash ist ein Objekt, das in jedem Resolver und Funktionshandler verfügbar gemacht wird. Das gleiche Stash-Objekt durchläuft einen einzigen Resolver-Lauf. Das bedeutet, dass Sie den Stash verwenden können, um beliebige Daten zwischen Anfrage- und Antworthandlern und zwischen Funktionen in einem Pipeline-Resolver zu übergeben.

> **Note**
>
> Sie können nicht den gesamten Stash löschen oder ersetzen, aber Sie können Eigenschaften des Stashs hinzufügen, aktualisieren, löschen und lesen.

Sie können dem Stash Elemente hinzufügen, indem Sie eines der folgenden Codebeispiele ändern:

```
//Example 1
ctx.stash.newItem = { key: "something" }

//Example 2
Object.assign(ctx.stash, {key1: value1, key2: value})
```

Sie können Artikel aus dem Stash entfernen, indem Sie den folgenden Code ändern:

```
delete ctx.stash.key
```

`result`

Ein Container für die Ergebnisse dieses Resolvers. Dieses Feld ist nur für Antworthandler verfügbar.

Zum Beispiel, wenn Sie das `author` Feld der folgenden Abfrage auflösen:

```
query {
    getPost(id: 1234) {
        postId
        title
        content
        author {
            id
            name
        }
    }
}
```

Dann ist die vollständige `context` Variable verfügbar, wenn ein Antworthandler ausgewertet wird:

```
{
  "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`

Das Ergebnis einer beliebigen vorherigen Operation, die in einem Pipeline-Resolver ausgeführt wurde.

Wenn der vorherige Vorgang der Anforderungshandler des Pipeline-Resolvers war, `ctx.prev.result` stellt er dieses Auswertungsergebnis dar und wird der ersten Funktion in der Pipeline zur Verfügung gestellt.

Wenn die vorherige Operation die erste Funktion war, `ctx.prev.result` stellt sie das Auswertungsergebnis des Antworthandlers der ersten Funktion dar und wird der zweiten Funktion in der Pipeline zur Verfügung gestellt.

Wenn die vorherige Operation die letzte Funktion war, `ctx.prev.result` stellt sie das Auswertungsergebnis der letzten Funktion dar und wird dem Response-Handler des Pipeline-Resolvers zur Verfügung gestellt.

`info`

Ein Objekt, das Informationen über die GraphQL-Anforderung enthält. Die Struktur dieses Feldes finden Sie unter Info.

Identität

Der Abschnitt `identity` enthält Informationen über den Aufrufer. Die Form dieses Abschnitts hängt vom Autorisierungstyp Ihrer AWS AppSync API ab.

Weitere Informationen zu AWS AppSync Sicherheitsoptionen finden Sie unter Autorisierung und Authentifizierung.

`API_KEY`-Autorisierung

Das `identity` Feld ist nicht gefüllt.

`AWS_LAMBDA`-Autorisierung

Das `identity` hat die folgende Form:

```
type AppSyncIdentityLambda = {
  resolverContext: any;
};
```

Der `identity` enthält den `resolverContext` Schlüssel, der denselben `resolverContext` Inhalt enthält, der von der Lambda-Funktion zurückgegeben wurde, die die Anfrage autorisiert.

`AWS_IAM`-Autorisierung

Der `identity` hat die folgende Form:

```
type AppSyncIdentityIAM = {
  accountId: string;
  cognitoIdentityPoolId: string;
  cognitoIdentityId: string;
  sourceIp: string[];
  username: string;
  userArn: string;
  cognitoIdentityAuthType: string;
  cognitoIdentityAuthProvider: string;
};
```

`AMAZON_COGNITO_USER_POOLS`-Autorisierung

Der `identity` hat die folgende Form:

```
type AppSyncIdentityCognito = {
  sourceIp: string[];
  username: string;
  groups: string[] | null;
  sub: string;
  issuer: string;
```

```
  claims: any;
  defaultAuthStrategy: string;
};
```

Jedes Feld ist wie folgt definiert:

`accountId`

Die AWS Konto-ID des Anrufers.

`claims`

Die Ansprüche, die der Benutzer hat.

`cognitoIdentityAuthType`

Entweder authentifiziert oder nicht authentifiziert, basierend auf dem Identitätstyp.

`cognitoIdentityAuthProvider`

Eine durch Kommas getrennte Liste mit Informationen zu externen Identitätsanbietern, anhand derer die Anmeldeinformationen abgerufen wurden, mit denen die Anfrage signiert wurde.

`cognitoIdentityId`

Die Amazon Cognito Cognito-Identitäts-ID des Anrufers.

`cognitoIdentityPoolId`

Die dem Anrufer zugeordnete Amazon Cognito Cognito-Identitätspool-ID.

`defaultAuthStrategy`

Die Standardautorisierungsstrategie für diesen Aufrufer (`ALLOW` oder `DENY`).

`issuer`

Der Aussteller des Tokens.

`sourceIp`

Die Quell-IP-Adresse des Anrufers, der empfängt. AWS AppSync Wenn die Anfrage den `x-forwarded-for` Header nicht enthält, enthält der Quell-IP-Wert nur eine einzige IP-Adresse aus der TCP-Verbindung. Wenn die Anforderung einen `x-forwarded-for`-Header enthält, verfügt die Quell-IP zusätzlich zur IP-Adresse aus der TCP-Verbindung über eine Liste mit IP-Adressen aus dem `x-forwarded-for`-Header.

`sub`

Die UUID des authentifizierten Benutzers.

`user`

Der IAM-Benutzer.

`userArn`

Der Amazon-Ressourcenname (ARN) des IAM-Benutzers.

`username`

Der Benutzername des authentifizierten Benutzers. Bei einer `AMAZON_COGNITO_USER_POOLS`-Autorisierung ist der Wert des Benutzernamens der Wert des Attributs cognito:username.
Im Fall einer `AWS_IAM` Autorisierung entspricht der Wert von username dem Wert des AWS Benutzerprinzipals. Wenn Sie die IAM-Autorisierung mit Anmeldeinformationen verwenden, die aus Amazon Cognito Cognito-Identitätspools stammen, empfehlen wir Ihnen, diese zu verwenden. `cognitoIdentityId`

Auf Header von Anfragen zugreifen

AWS AppSync unterstützt die Übergabe benutzerdefinierter Header von Clients und den Zugriff auf sie in Ihren GraphQL-Resolvern mithilfe von. `ctx.request.headers` Sie können die Header-Werte dann für Aktionen wie das Einfügen von Daten in eine Datenquelle oder für Autorisierungsprüfungen verwenden. Sie können einzelne oder mehrere Anforderungsheader `$curl` mithilfe eines API-Schlüssels von der Befehlszeile aus verwenden, wie in den folgenden Beispielen gezeigt:

Beispiel für einen einzelnen Header

Angenommen, Sie legen wie folgt einen `custom`-Header mit dem Wert `nadia` fest:

```
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
```

Darauf könnte dann mit `ctx.request.headers.custom` zugegriffen werden. Es könnte beispielsweise im folgenden Code für DynamoDB enthalten sein:

```
"custom": util.dynamodb.toDynamoDB(ctx.request.headers.custom)
```

Beispiel für mehrere Header

Sie können auch mehrere Header in einer einzigen Anfrage übergeben und im Resolver-Handler auf diese zugreifen. Zum Beispiel, wenn der `custom` Header mit zwei Werten festgelegt ist:

```
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
```

Sie können dann darauf als Array zugreifen, z. B. `ctx.request.headers.custom[1]`.

> **Note**
>
> AWS AppSync macht den Cookie-Header in nicht verfügbar`ctx.request.headers`.

Greifen Sie auf den angeforderten benutzerdefinierten Domainnamen zu

AWS AppSync unterstützt die Konfiguration einer benutzerdefinierten Domain, mit der Sie auf Ihre GraphQL- und Echtzeit-Endpunkte für Ihre zugreifen können. APIs Wenn Sie eine Anfrage mit einem benutzerdefinierten Domainnamen stellen, können Sie den Domainnamen mithilfe von abrufen. `ctx.request.domainName`

Bei Verwendung des standardmäßigen GraphQL-Endpunktdomänennamens lautet der Wert. `null`

Informationen

Der Abschnitt `info` enthält Informationen über die GraphQL-Anforderung. Dieser Abschnitt hat die folgende Form:

```
type Info = {
  fieldName: string;
  parentTypeName: string;
  variables: any;
  selectionSetList: string[];
  selectionSetGraphQL: string;
};
```

Jedes Feld ist wie folgt definiert:

`fieldName`

Der Name des Feldes, das derzeit aufgelöst wird.

`parentTypeName`

Der Name des übergeordneten Typs für das Feld, das derzeit aufgelöst wird.

`variables`

Eine Zuordnung, die alle Variablen enthält, die an die GraphQL-Anforderung übergeben werden.

`selectionSetList`

Eine Listendarstellung der Felder im GraphQL-Auswahlsatz. Felder mit Aliasnamen werden nur durch den Aliasnamen referenziert, nicht durch den Feldnamen. Im folgenden Beispiel ist dies detailliert dargestellt.

`selectionSetGraphQL`

Eine Zeichenfolgendarstellung des Auswahlsatzes, formatiert als GraphQL-Schemadefinitionssprache (SDL). Fragmente werden zwar nicht mit dem Auswahlsatz zusammengeführt, Inline-Fragmente bleiben jedoch erhalten, wie im folgenden Beispiel gezeigt.

> **Note**
>
> `JSON.stringify`schließt `selectionSetGraphQL` und nicht `selectionSetList` in die Serialisierung der Zeichenfolge ein. Sie müssen direkt auf diese Eigenschaften verweisen.

Angenommen, Sie lösen das `getPost`-Feld der folgenden Abfrage auf:

```
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
  }
}
```

Dann könnte die vollständige `ctx.info` Variable, die bei der Verarbeitung eines Handlers verfügbar ist, wie folgt lauten:

```
{
  "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`macht nur Felder verfügbar, die zum aktuellen Typ gehören. Wenn es sich bei dem aktuellen Typ um eine Schnittstelle oder Union handelt, werden nur ausgewählte Felder angezeigt, die zur Schnittstelle gehören. Nehmen wir zum Beispiel das folgende Schema:

```
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
}
```

Und die folgende Abfrage:

```
query {
    node(id: "post1") {
        id
        ... on Post {
            title
        }

        ... on Blog {
            title
        }
    }
}
```

`ctx.info.selectionSetList`Beim Aufrufen mit der `Query.node` Feldauflösung `id` wird nur angezeigt:

```
"selectionSetList": [
    "id"
]
```

AWS AppSync JavaScript Laufzeitfunktionen für Resolver und Funktionen

Die `APPSYNC_JS` Laufzeitumgebung bietet ähnliche Funktionen wie ECMAScript (ES) Version 6.0. Sie unterstützt einen Teil ihrer Funktionen und bietet einige zusätzliche Methoden (Dienstprogramme), die nicht Teil der ES-Spezifikationen sind. In den folgenden Themen sind alle unterstützten Sprachfunktionen aufgeführt:

- Unterstützte Laufzeitfunktionen — Erfahren Sie mehr über unterstützte Kernfunktionen, primitive Objekte, integrierte Objekte und Funktionen usw.
- Integrierte Hilfsprogramme — Die Variable util enthält allgemeine Hilfsmethoden, die Ihnen bei der Arbeit mit Daten helfen. Sofern nicht anders angegeben, verwenden alle Dienstprogramme den UTF-8-Zeichensatz.
- Integrierte Module — Erfahren Sie mehr darüber, wie integrierte Module beim Schreiben von JavaScript Resolvern und Funktionen helfen können.
- Runtime-Utilities — Die Runtime-Bibliothek stellt Dienstprogramme zur Verfügung, mit denen Sie die Laufzeiteigenschaften Ihrer Resolver und Funktionen steuern oder ändern können.
- Zeithelfer in util.time — Die Variable util.time enthält Datetime-Methoden, mit deren Hilfe Zeitstempel generiert, zwischen Datetime-Formaten konvertiert und Datetime-Zeichenketten analysiert werden können. Darauf basiert die Syntax für Datetime-Formate, auf die Sie sich für weitere Dokumentationen beziehen können. DateTimeFormatter
- DynamoDB-Helfer in util.dynamodb — util.dynamodb enthält Hilfsmethoden, die das Schreiben und Lesen von Daten in Amazon DynamoDB erleichtern, wie z. B. automatische Typzuweisung und Formatierung.
- HTTP-Helfer in util.http — Das Hilfsprogramm util.http bietet Hilfsmethoden, mit denen Sie HTTP-Anforderungsparameter verwalten und Antwortheader hinzufügen können.
- Transformationshelfer in util.transform - util.transform enthält Hilfsmethoden, die es einfacher machen, komplexe Operationen mit Datenquellen durchzuführen.

- Zeichenkettenhelfer in util.str - util.str enthält Methoden, die bei gängigen String-Operationen helfen.
- Erweiterungen — Erweiterungen enthalten eine Reihe von Methoden, mit denen Sie zusätzliche Aktionen in Ihren Resolvern ausführen können.
- XML-Helfer in util.xml - util.xml enthält Methoden, die bei der Konvertierung von XML-Zeichenketten helfen.

> **Note**
>
> Derzeit gilt diese Referenz nur für Runtime-Version 1.0.0.

Unterstützte Runtime-Funktionen

In den folgenden Abschnitten werden die unterstützten Funktionen der APPSYNC_JS-Laufzeit beschrieben.

Kern-Features

Die folgenden Kernfunktionen werden unterstützt.

Types

Die folgenden Typen werden unterstützt:

- Ziffern
- Zeichenfolgen
- Boolesche Werte
- objects
- Arrays
- Funktionen

Operators

Operatoren werden unterstützt, darunter:

- mathematische Standardoperatoren (`+-/,%,*,` usw.)

- Koaleszenzoperator () auf Null setzen `??`
- Optionale Verkettung () `?.`
- bitweise Operatoren
- `void`und Operatoren `typeof`
- Spread-Operatoren (`...`)

Die folgenden Operatoren werden nicht unterstützt:

- unäre Operatoren (`++`-`--`, und`~`)
- `in`-Operator

> **Note**
>
> Verwenden Sie den `Object.hasOwn` Operator, um zu überprüfen, ob sich die angegebene Eigenschaft im angegebenen Objekt befindet.

Statements

Die folgenden Anweisungen werden unterstützt:

- `const`
- `let`
- `var`
- `break`
- `else`
- `for-in`
- `for-of`
- `if`
- `return`
- `switch`
- Spread-Syntax

Folgendes wird nicht unterstützt:

- `catch`
- `continue`
- `do-while`
- `finally`
- `for(initialization; condition; afterthought)`

  > **Note**
  >
  > Die Ausnahmen sind `for-in` und `for-of` Ausdrücke, die unterstützt werden.

- `throw`
- `try`
- `while`
- beschriftete Aussagen

Literals

Die folgenden ES 6-Vorlagenliterale werden unterstützt:

- Mehrzeilige Zeichenketten
- Interpolation von Ausdrücken
- Verschachtelung von Vorlagen

Functions

Die folgende Funktionssyntax wird unterstützt:

- Funktionsdeklarationen werden unterstützt.
- ES 6-Pfeilfunktionen werden unterstützt.
- Die ES 6-Restparameter-Syntax wird unterstützt.

Strict mode

Funktionen arbeiten standardmäßig im strikten Modus, sodass Sie Ihrem Funktionscode keine `use_strict`-Anweisung hinzufügen müssen. Dies können nicht geändert werden.

Primitive Objekte

Die folgenden primitiven Objekte von ES und ihre Funktionen werden unterstützt.

Object

Die folgenden Objekte werden unterstützt:

- `Object.assign()`
- `Object.entries()`
- `Object.hasOwn()`
- `Object.keys()`
- `Object.values()`
- `delete`

String

Die folgenden Zeichenketten werden unterstützt:

- `String.prototype.length()`
- `String.prototype.charAt()`
- `String.prototype.concat()`
- `String.prototype.endsWith()`
- `String.prototype.indexOf()`
- `String.prototype.lastIndexOf()`
- `String.raw()`
- `String.prototype.replace()`

> **Note**
>
> Reguläre Ausdrücke werden nicht unterstützt.
> Konstrukte regulärer Ausdrücke im Java-Stil werden jedoch im angegebenen Parameter unterstützt. Weitere Informationen finden Sie unter Muster.

- `String.prototype.replaceAll()`

> **Note**
>
> Reguläre Ausdrücke werden nicht unterstützt.
> Konstrukte regulärer Ausdrücke im Java-Stil werden jedoch im angegebenen Parameter unterstützt. Weitere Informationen finden Sie unter Muster.

- `String.prototype.slice()`
- `String.prototype.split()`
- `String.prototype.startsWith()`
- `String.prototype.toLowerCase()`
- `String.prototype.toUpperCase()`
- `String.prototype.trim()`
- `String.prototype.trimEnd()`
- `String.prototype.trimStart()`

Number

Die folgenden Zahlen werden unterstützt:

- `Number.isFinite`
- `Number.isNaN`

Integrierte Objekte und Funktionen

Die folgenden Funktionen und Objekte werden unterstützt.

Math

Die folgenden mathematischen Funktionen werden unterstützt:

- `Math.random()`
- `Math.min()`
- `Math.max()`
- `Math.round()`
- `Math.floor()`

- `Math.ceil()`

Array

Die folgenden Array-Methoden werden unterstützt:

- `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()`unterstützt keine Argumente.

- `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`

Console

Das Konsolenobjekt ist für das Debuggen verfügbar. Während der Live-Abfrageausführung werden log/error Konsolenanweisungen an Amazon CloudWatch Logs gesendet (sofern die Protokollierung aktiviert ist). Während der Codeauswertung mit `evaluateCode` werden Protokollanweisungen in der Befehlsantwort zurückgegeben.

- `console.error()`
- `console.log()`

Function

- Die `call` Methoden `applybind`, und werden nicht unterstützt.
- Funktionskonstruktoren werden nicht unterstützt.
- Die Übergabe einer Funktion als Argument wird nicht unterstützt.
- Rekursive Funktionsaufrufen werden nicht unterstützt.

JSON

Die folgenden JSON-Methoden werden unterstützt:

- `JSON.parse()`

> **Note**
>
> Gibt eine leere Zeichenfolge zurück, wenn es sich bei der analysierten Zeichenfolge nicht um eine gültige JSON-Zeichenfolge handelt.

- `JSON.stringify()`

Promises

Asynchrone Prozesse werden nicht unterstützt, und Promises werden nicht unterstützt.

> **Note**
>
> Der Netzwerk- und Dateisystemzugriff wird in der `APPSYNC_JS` Runtime in AWS AppSync nicht unterstützt. AWS AppSync verarbeitet alle I/O-Operationen, die auf den vom AWS AppSync Resolver oder der AWS AppSync Funktion gestellten Anforderungen basieren.

Globale

Die folgenden globalen Konstanten werden unterstützt:

- `NaN`
- `Infinity`
- `undefined`
- `util`
- `extensions`
- `runtime`

Fehlertypen

Das Auslösen von Fehlern mit `throw` wird nicht unterstützt. Sie können einen Fehler zurückgeben, indem Sie die `util.error()` Funktion verwenden. Sie können einen Fehler in Ihre GraphQL-Antwort aufnehmen, indem Sie die `util.appendError` Funktion verwenden.

Weitere Informationen finden Sie unter Error utils.

Integrierte Dienstprogramme

Die `util` Variable enthält allgemeine Hilfsmethoden, die Ihnen bei der Arbeit mit Daten helfen. Sofern nicht anders angegeben, verwenden alle Dienstprogramme den UTF-8-Zeichensatz.

Werkzeuge zum Kodieren

Liste der Kodierungswerkzeuge

`util.urlEncode(String)`

Gibt die Eingabezeichenfolge als eine `application/x-www-form-urlencoded`-kodierte Zeichenfolge zurück.

`util.urlDecode(String)`

Dekodiert eine `application/x-www-form-urlencoded`-kodierte Zeichenfolge zurück in ihre nicht kodierte Form.

`util.base64Encode(string) : string`

Verschlüsselt die Eingabe in eine base64-kodierte Zeichenfolge.

`util.base64Decode(string) : string`

Decodiert die Daten einer base64-verschlüsselten Zeichenfolge.

Dienstprogramme zur ID-Generierung

Liste der Tools zur ID-Generierung

`util.autoId()`

Gibt eine zufällig generierte 128-Bit-UUID zurück.

`util.autoUlid()`

Gibt eine zufällig generierte 128-Bit-ULID (Universally Unique Lexicographically Sortable Identifier) zurück.

`util.autoKsuid()`

Gibt eine zufällig generierte 128-Bit-KSUID (K-Sortable Unique Identifier) Base62 zurück, die als Zeichenfolge mit einer Länge von 27 codiert ist.

Fehler utils

Fehler-Utils-Liste

`util.error(String, String?, Object?, Object?)`

Gibt einen benutzerdefinierte Fehler aus. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Zusätzlich können ein `errorType` Feld, ein `data` Feld und ein `errorInfo` Feld angegeben werden. Der `data`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt.

> **Note**
>
> `data`wird auf der Grundlage des Abfrageauswahlsatzes gefiltert. Der `errorInfo`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt. `errorInfo`wird nicht auf der Grundlage des Abfrageauswahlsatzes gefiltert.

`util.appendError(String, String?, Object?, Object?)`

Fügt einen benutzerdefinierten Fehler an. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Zusätzlich können ein `errorType` Feld, ein `data` Feld und ein `errorInfo` Feld angegeben werden. Im Gegensatz zu `util.error(String, String?, Object?, Object?)` wird die Vorlagenbewertung nicht unterbrochen, sodass die Daten an den Aufrufer zurückgegeben werden können. Der `data`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt.

> **Note**
>
> `data`wird auf der Grundlage des Abfrageauswahlsatzes gefiltert. Der `errorInfo`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt. `errorInfo`wird nicht auf der Grundlage des Abfrageauswahlsatzes gefiltert.

Tools für den Typ- und Musterabgleich

Liste der Tools für den Typ- und Musterabgleich

`util.matches(String, String) : Boolean`

Gibt "true" zurück, wenn das angegebene Muster im ersten Argument den bereitgestellten Daten im zweiten Argument entspricht. Das Muster muss ein regulärer Ausdruck sein, wie z. B. `util.matches("a*b", "aaaaab")`. Die Funktionalität basiert auf Pattern, worauf Sie zur weiteren Dokumentation verweisen können.

`util.authType()`

Gibt eine Zeichenfolge zurück, die den Multi-Auth-Typ beschreibt, der von einer Anfrage verwendet wird, und gibt entweder „IAM-Autorisierung“, „Benutzerpool-Autorisierung“, „Open ID Connect-Autorisierung“ oder „API-Schlüsselautorisierung“ zurück.

Gibt den Wert zurück, Verhalten, utils

Liste der Verhaltenswerkzeuge für den Rückgabewert

`util.escapeJavaScript(String)`

Gibt die Eingabezeichenfolge als JavaScript Escape-Zeichenfolge zurück.

Tools zur Autorisierung von Resolver

Liste der Resolver-Autorisierungswerkzeuge

`util.unauthorized()`

Gibt `Unauthorized` für das Feld aus, das aufgelöst wird. Verwenden Sie dies in Vorlagen für die Zuordnung von Anfragen oder Antworten, um zu bestimmen, ob der Anrufer das Feld auflösen darf.

Integrierten Module

Module sind Teil der `APPSYNC_JS` Runtime und stellen Hilfsprogramme zur Verfügung, mit denen JavaScript Resolver und Funktionen geschrieben werden können. Beispiele und Beispiele finden Sie im aws-appsync-resolver-samples GitHub Repository.

Funktionen des DynamoDB-Moduls

DynamoDB-Modulfunktionen bieten eine verbesserte Erfahrung bei der Interaktion mit DynamoDB-Datenquellen. Sie können mithilfe der Funktionen Anfragen an Ihre DynamoDB-Datenquellen stellen, ohne eine Typzuordnung hinzuzufügen.

Module werden wie folgt importiert: `@aws-appsync/utils/dynamodb`

```
// Modules are imported using @aws-appsync/utils/dynamodb
import * as ddb from '@aws-appsync/utils/dynamodb';
```

Funktionen

Funktionsliste

`get<T>(payload: GetInput): DynamoDBGetItemRequest`

> **Tip**
>
> Informationen the section called “Eingaben” zu finden Sie unter GetInput.

Generiert ein `DynamoDBGetItemRequest` Objekt, um eine GetItemAnfrage an DynamoDB zu stellen.

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

export function request(ctx) {
 return get({ key: { id: ctx.args.id } });
}
```

`put<T>(payload): DynamoDBPutItemRequest`

Generiert ein `DynamoDBPutItemRequest` Objekt, um eine PutItemAnfrage an DynamoDB zu stellen.

```
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`

Generiert ein `DynamoDBDeleteItemRequest` Objekt, um eine DeleteItemAnfrage an DynamoDB zu stellen.

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
 return ddb.remove({ key: { id: ctx.args.id } });
}
```

`scan<T>(payload): DynamoDBScanRequest`

Generiert eine`DynamoDBScanRequest`, um eine Scan-Anfrage an DynamoDB zu stellen.

```
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`

Generiert ein `DynamoDBSyncRequest` Objekt, um eine Sync-Anfrage zu stellen. Die Anfrage empfängt nur die Daten, die seit der letzten Abfrage geändert wurden (Delta-Updates). Anfragen können nur an versionierte DynamoDB-Datenquellen gestellt werden.

```
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`

Generiert ein `DynamoDBUpdateItemRequest` Objekt, um eine UpdateItemAnfrage an DynamoDB zu stellen.

Operationen

Mithilfe von Operationshelfern können Sie bei Aktualisierungen bestimmte Aktionen für Teile Ihrer Daten ausführen. Importieren Sie zunächst `operations` aus`@aws-appsync/utils/dynamodb`:

```
// Modules are imported using operations
import {operations} from '@aws-appsync/utils/dynamodb';
```

Liste der Operationen

`add<T>(payload)`

Eine Hilfsfunktion, die beim Aktualisieren von DynamoDB ein neues Attributelement hinzufügt.

Beispiel

So fügen Sie einem vorhandenen DynamoDB-Element mithilfe des ID-Werts eine Adresse (Straße, Ort und Postleitzahl) hinzu:

```
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)`

Eine Hilfsfunktion, die eine Nutzlast an die bestehende Liste in DynamoDB anhängt.

Beispiel

So fügen Sie während eines Updates einen neu hinzugefügten Freund IDs (`newFriendIds`) an eine bestehende Freundesliste (`friendsIds`) an:

```
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?)`

Eine Hilfsfunktion, die den vorhandenen Attributwert im Element verringert, wenn DynamoDB aktualisiert wird.

Beispiel

Um den Zähler eines Freundes () `friendsCount` um 10 zu verringern:

```
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?)`

Eine Hilfsfunktion, die den vorhandenen Attributwert im Element erhöht, wenn DynamoDB aktualisiert wird.

Beispiel

Um den Zähler eines Freundes () `friendsCount` um 10 zu erhöhen:

```
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)`

Eine Hilfsfunktion, die der vorhandenen Liste in DynamoDB vorangestellt wird.

Beispiel

Um einen neu hinzugefügten Freund IDs (`newFriendIds`) während eines Updates einer bestehenden Freundesliste (`friendsIds`) voranzustellen:

```
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)`

Eine Hilfsfunktion, die ein vorhandenes Attribut ersetzt, wenn ein Element in DynamoDB aktualisiert wird. Dies ist nützlich, wenn Sie das gesamte Objekt oder Unterobjekt im Attribut aktualisieren möchten und nicht nur die Schlüssel in der Nutzlast.

Beispiel

Um eine Adresse (Straße, Ort und Postleitzahl) in einem Objekt zu ersetzen: `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)`

Eine Hilfsfunktion, die ein Element in einer Liste ersetzt.

Beispiel

Im Rahmen von update (`newFriendIds`) wurden in diesem Beispiel die `updateListItem` ID-Werte des zweiten Elements (index:1, neue ID:102) und des dritten Elements (index:2, neue ID:112) in einer Liste (`friendsIds`) aktualisiert.

```
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 });
}
```

Eingaben

Liste der Eingaben

`Type GetInput<T>`

```
GetInput<T>: {
    consistentRead?: boolean;
    key: DynamoDBKey<T>;
}
```

Deklaration eingeben

- `consistentRead?: boolean` (optional)

  Ein optionaler boolescher Wert, um anzugeben, ob Sie mit DynamoDB einen stark konsistenten Lesevorgang durchführen möchten.

- `key: DynamoDBKey<T>` (Erforderlich)

  Ein erforderlicher Parameter, der den Schlüssel des Elements in DynamoDB angibt. DynamoDB-Elemente können einen einzelnen Hashschlüssel oder Hash- und Sortierschlüssel haben.

`Type PutInput<T>`

```
PutInput<T>: {
    _version?: number;
    condition?: DynamoDBFilterObject<T> | null;
    customPartitionKey?: string;
    item: Partial<T>;
    key: DynamoDBKey<T>;
    populateIndexFields?: boolean;
}
```

Typ Deklaration

- `_version?: number` (optional)
- `condition?: DynamoDBFilterObject<T> | null` (optional)

  Wenn Sie ein Objekt in eine DynamoDB-Tabelle einfügen, können Sie optional einen bedingten Ausdruck angeben, der steuert, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Zustand des Objekts, das sich bereits in DynamoDB befand, bevor der Vorgang ausgeführt wird.

- `customPartitionKey?: string` (optional)

  Wenn diese Option aktiviert ist, ändert dieser Zeichenfolgenwert das Format der `ds_sk` und `ds_pk` -Datensätze, die von der Delta-Sync-Tabelle verwendet werden, wenn die Versionierung aktiviert wurde. Wenn diese Option aktiviert ist, ist auch die Verarbeitung des `populateIndexFields` Eintrags aktiviert.

- `item: Partial<T>` (Erforderlich)

  Die restlichen Attribute des Elements sollen in DynamoDB platziert werden.

- `key: DynamoDBKey<T>` (Erforderlich)

  Ein erforderlicher Parameter, der den Schlüssel des Elements in DynamoDB angibt, für das der Put ausgeführt wird. DynamoDB-Elemente können einen einzelnen Hashschlüssel oder Hash- und Sortierschlüssel haben.

- `populateIndexFields?: boolean` (optional)

  Ein boolescher Wert, der, wenn er zusammen mit dem aktiviert wird`customPartitionKey`, neue Einträge für jeden Datensatz in der Delta-Synchronisierungstabelle erstellt, insbesondere in den Spalten und. `gsi_ds_pk` `gsi_ds_sk` Weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch.

`Type QueryInput<T>`

```
QueryInput<T>: ScanInput<T> & {
    query: DynamoDBKeyCondition<Required<T>>;
}
```

Geben Sie Erklärung ein

- `query: DynamoDBKeyCondition<Required<T>>` (Erforderlich)

  Gibt eine Schlüsselbedingung an, die die abzufragenden Elemente beschreibt. Für einen bestimmten Index sollte die Bedingung für einen Partitionsschlüssel eine Gleichheit und der Sortierschlüssel ein Vergleich oder ein `beginsWith` (wenn es sich um eine Zeichenfolge handelt) sein. Für Partitions- und Sortierschlüssel werden nur Zahlen- und Zeichenfolgentypen unterstützt.

  Beispiel

  Nehmen Sie den folgenden `User` Typ:

  ```
  type User = {
    id: string;
    name: string;
    age: number;
    isVerified: boolean;
    friendsIds: string[]
  }
  ```

  Die Abfrage kann nur die folgenden Felder enthalten: `id`name, und`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;
}
```

Geben Sie Erklärung ein

- `_version?: number` (optional)
- `condition?: DynamoDBFilterObject<T>` (optional)

  Wenn Sie ein Objekt in DynamoDB entfernen, können Sie optional einen bedingten Ausdruck angeben, der steuert, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Zustand des Objekts, das sich bereits in DynamoDB befand, bevor der Vorgang ausgeführt wird.

  Beispiel

  Das folgende Beispiel ist ein `DeleteItem` Ausdruck, der eine Bedingung enthält, die es ermöglicht, dass der Vorgang nur dann erfolgreich ist, wenn der Eigentümer des Dokuments mit dem Benutzer übereinstimmt, der die Anfrage gestellt hat.

  ```
  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` (optional)

  Wenn diese Option aktiviert ist, ändert der `customPartitionKey` Wert das Format der `ds_sk` und `ds_pk` -Datensätze, die von der Delta-Synchronisierungstabelle verwendet werden, wenn die Versionsverwaltung aktiviert wurde. Wenn diese Option aktiviert ist, ist auch die Verarbeitung des `populateIndexFields` Eintrags aktiviert.

- `key: DynamoDBKey<T>` (Erforderlich)

  Ein erforderlicher Parameter, der den Schlüssel des Elements in DynamoDB angibt, das entfernt wird. DynamoDB-Elemente können einen einzelnen Hashschlüssel oder Hash- und Sortierschlüssel haben.

  Beispiel

  Wenn a `User` nur den Hash-Schlüssel mit einem Benutzer hat`id`, würde der Schlüssel wie folgt aussehen:

  ```
  type User = {
   id: number
   name: string
   age: number
   isVerified: boolean
  }
  const key: DynamoDBKey<User> = {
   id: 1,
  }
  ```

  Wenn der Tabellenbenutzer einen Hash-Schlüssel (`id`) und einen Sortierschlüssel (`name`) hat, dann würde der Schlüssel so aussehen:

  ```
  type User = {
   id: number
   name: string
   age: number
   isVerified: boolean
   friendsIds: string[]
  }

  const key: DynamoDBKey<User> = {
  ```

```
 id: 1,
 name: 'XXXXXXXXXX',
}
```

- `populateIndexFields?: boolean` (optional)

  Ein boolescher Wert, der, wenn er zusammen mit dem aktiviert wird`customPartitionKey`, neue Einträge für jeden Datensatz in der Delta-Sync-Tabelle erstellt, insbesondere in den Spalten `gsi_ds_pk` und`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;
}
```

Deklaration eingeben

- `consistentRead?: boolean | null` (optional)

  Ein optionaler boolescher Wert zur Angabe konsistenter Lesevorgänge bei der Abfrage von DynamoDB. Der Standardwert ist `false`.

- `filter?: DynamoDBFilterObject<T> | null` (optional)

  Ein optionaler Filter, der auf die Ergebnisse angewendet wird, nachdem sie aus der Tabelle abgerufen wurden.

- `index?: string | null` (optional)

  Ein optionaler Name des zu scannenden Indexes.

- `limit?: number | null` (optional)

  Eine optionale maximale Anzahl von Ergebnissen, die zurückgegeben werden sollen.

- `nextToken?: string | null` (optional)

Ein optionales Paginierungstoken, um eine vorherige Abfrage fortzusetzen. Dieses wäre von einer vorherigen Abfrage erhalten worden.

- `scanIndexForward?: boolean | null` (optional)

  Ein optionaler boolescher Wert, der angibt, ob die Abfrage in aufsteigender oder absteigender Reihenfolge ausgeführt wird. Standardmäßig ist dieser Wert auf `true` festgelegt.

- `segment?: number` (optional)
- `select?: DynamoDBSelectAttributes` (optional)

  Attribute, die von DynamoDB zurückgegeben werden sollen. Standardmäßig gibt der AWS AppSync DynamoDB-Resolver nur Attribute zurück, die in den Index projiziert werden. Die unterstützten Werte sind:

  - `ALL_ATTRIBUTES`

    Gibt alle Elementattribute aus der angegebenen Tabelle oder dem angegebenen Index zurück. Wenn Sie einen lokalen sekundären Index abfragen, ruft DynamoDB für jedes übereinstimmende Element im Index das gesamte Element aus der übergeordneten Tabelle ab. Wenn der Index konfiguriert ist, um alle Elementattribute zu projizieren, können Sie alle Daten aus dem lokalen sekundären Index erhalten und das Abrufen ist nicht erforderlich.

  - `ALL_PROJECTED_ATTRIBUTES`

    Gibt alle Attribute zurück, die in den Index projiziert wurden. Wenn der Index konfiguriert ist, um alle Attribute zu projizieren, entspricht dieser Rückgabewert der Angabe von `ALL_ATTRIBUTES`.

  - `SPECIFIC_ATTRIBUTES`

    Gibt nur die in aufgeführten Attribute zurück`ProjectionExpression`. Dieser Rückgabewert entspricht der Angabe `ProjectionExpression` ohne Angabe eines Werts für`AttributesToGet`.

- `totalSegments?: number` (optional)

Type DynamoDBSyncInput<T>

```
DynamoDBSyncInput<T>: {
    basePartitionKey?: string;
    deltaIndexName?: string;
    filter?: DynamoDBFilterObject<T> | null;
    lastSync?: number;
```

```
    limit?: number | null;
    nextToken?: string | null;
}
```

Typ Deklaration

- `basePartitionKey?: string` (optional)

  Der Partitionsschlüssel der Basistabelle, der bei der Ausführung eines Sync-Vorgangs verwendet werden soll. Dieses Feld ermöglicht die Ausführung eines Synchronisierungsvorgangs, wenn die Tabelle einen benutzerdefinierten Partitionsschlüssel verwendet.

- `deltaIndexName?: string` (optional)

  Der Index, der für den Sync-Vorgang verwendet wird. Dieser Index ist erforderlich, um einen Sync-Vorgang für die gesamte Delta-Store-Tabelle zu aktivieren, wenn die Tabelle einen benutzerdefinierten Partitionsschlüssel verwendet. Der Synchronisierungsvorgang wird auf der GSI (erstellt am `gsi_ds_pk` und`gsi_ds_sk`) ausgeführt.

- `filter?: DynamoDBFilterObject<T> | null` (optional)

  Ein optionaler Filter, der auf die Ergebnisse angewendet wird, nachdem sie aus der Tabelle abgerufen wurden.

- `lastSync?: number` (optional)

  Der Moment in Epochen-Millisekunden, zu dem der letzte erfolgreiche Synchronisierungsvorgang gestartet wurde. Wenn angegeben, werden nur Elemente zurückgegeben, die sich nach `lastSync` geändert haben. Dieses Feld sollte erst gefüllt werden, nachdem alle Seiten eines ersten Synchronisierungsvorgangs abgerufen wurden. Wenn es weggelassen wird, werden Ergebnisse aus der Basistabelle zurückgegeben. Andernfalls werden Ergebnisse aus der Delta-Tabelle zurückgegeben.

- `limit?: number | null` (optional)

  Eine optionale maximale Anzahl von Elementen, die gleichzeitig ausgewertet werden können. Wenn nicht angegeben, wird das Standardlimit auf `100` Elemente festgelegt. Der maximale Wert für dieses Feld ist `1000` Elemente.

- `nextToken?: string | null` (optional)

Type DynamoDBUpdateInput<T>

```
DynamoDBUpdateInput<T>: {
    _version?: number;
    condition?: DynamoDBFilterObject<T>;
    customPartitionKey?: string;
    key: DynamoDBKey<T>;
    populateIndexFields?: boolean;
    update: DynamoDBUpdateObject<T>;
}
```

Geben Sie Erklärung ein

- `_version?: number` (optional)
- `condition?: DynamoDBFilterObject<T>` (optional)

  Wenn Sie ein Objekt in DynamoDB aktualisieren, können Sie optional einen bedingten Ausdruck angeben, der steuert, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Zustand des Objekts, das sich bereits in DynamoDB befand, bevor der Vorgang ausgeführt wird.

- `customPartitionKey?: string` (optional)

  Wenn diese Option aktiviert ist, ändert der `customPartitionKey` Wert das Format der `ds_pk` AND-Datensätze, die von der `ds_sk` Delta-Synchronisierungstabelle verwendet werden, wenn die Versionierung aktiviert wurde. Wenn diese Option aktiviert ist, ist auch die Verarbeitung des `populateIndexFields` Eintrags aktiviert.

- `key: DynamoDBKey<T>` (Erforderlich)

  Ein erforderlicher Parameter, der den Schlüssel des Elements in DynamoDB angibt, das aktualisiert wird. DynamoDB-Elemente können einen einzelnen Hashschlüssel oder Hash- und Sortierschlüssel haben.

- `populateIndexFields?: boolean` (optional)

  Ein boolescher Wert, der, wenn er zusammen mit dem aktiviert wird`customPartitionKey`, neue Einträge für jeden Datensatz in der Delta-Synchronisierungstabelle erstellt, insbesondere in den Spalten und. `gsi_ds_pk` `gsi_ds_sk`

- `update: DynamoDBUpdateObject<T>`

Ein Objekt, das die zu aktualisierenden Attribute zusammen mit den neuen Werten für sie angibt. Das Aktualisierungsobjekt kann mitadd,,remove,replace,increment,decrement, appendprepend, verwendet werdenupdateListItem.

Funktionen des Amazon RDS-Moduls

Die Funktionen des Amazon RDS-Moduls bieten eine verbesserte Benutzererfahrung bei der Interaktion mit Datenbanken, die mit der Amazon RDS-Daten-API konfiguriert sind. Das Modul wird importiert mit@aws-appsync/utils/rds:

```
import * as rds from '@aws-appsync/utils/rds';
```

Funktionen können auch einzeln importiert werden. Zum Beispiel verwendet der folgende Importsql:

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

Funktionen

Sie können die Hilfsprogramme des AWS AppSync RDS-Moduls verwenden, um mit Ihrer Datenbank zu interagieren.

Select

Das select Hilfsprogramm erstellt eine SELECT Anweisung, um Ihre relationale Datenbank abzufragen.

Grundlegende Verwendung

In ihrer Grundform können Sie die Tabelle angeben, die Sie abfragen möchten:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

    // Generates statement:
    // "SELECT * FROM "persons"
    return createPgStatement(select({table: 'persons'}));
}
```

Beachten Sie, dass Sie das Schema auch in Ihrem Tabellen-Identifier angeben können:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

    // Generates statement:
    // SELECT * FROM "private"."persons"
    return createPgStatement(select({table: 'private.persons'}));
}
```

Spalten angeben

Sie können Spalten mit der `columns` Eigenschaft angeben. Wenn dieser Wert nicht auf einen Wert gesetzt ist, wird standardmäßig wie folgt vorgegeben*:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name']
    }));
}
```

Sie können auch die Tabelle einer Spalte angeben:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "persons"."name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'persons.name']
    }));
}
```

Grenzwerte und Offsets

Sie können `limit` und auf `offset` die Abfrage anwenden:

```
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
    }));
}
```

Sortiert nach

Sie können Ihre Ergebnisse nach der `orderBy` Eigenschaft sortieren. Geben Sie eine Reihe von Objekten an, die die Spalte und eine optionale `dir` Eigenschaft angeben:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name" FROM "persons"
    // ORDER BY "name", "id" DESC
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        orderBy: [{column: 'name'}, {column: 'id', dir: 'DESC'}]
    }));
}
```

Filter

Sie können Filter erstellen, indem Sie das Objekt mit der speziellen Bedingung verwenden:

```
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'}}
    }));
}
```

Sie können Filter auch kombinieren:

```
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}}
    }));
}
```

Sie können auch OR Aussagen erstellen:

```
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 } }
        ]}
    }));
}
```

Sie können eine Bedingung auch negieren mit`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 } }
        ]}
    }));
}
```

Sie können auch die folgenden Operatoren verwenden, um Werte zu vergleichen:

Operator	Beschreibung	Mögliche Wertetypen
eq	Gleich	Zahl, Zeichenfolge, boolescher Wert
Eins	Ungleich	Zahl, Zeichenfolge, boolescher Wert
le	Kleiner als oder gleich	Zahl, Zeichenfolge
lt	kleiner als	Zahl, Zeichenfolge
ge	Größer als oder gleich	Zahl, Zeichenfolge
gt	größer als	Zahl, Zeichenfolge
enthält	Wie	Zeichenfolge
Nicht enthält	Nicht wie	Zeichenfolge
Beginnt mit	Beginnt mit einem Präfix	Zeichenfolge
zwischen	Zwischen zwei Werten	Zahl, Zeichenfolge

Das Attribut ist vorhanden	Das Attribut ist nicht Null	Zahl, Zeichenfolge, boolesche r Wert
size	prüft die Länge des Elements	Zeichenfolge

Einfügen

Das `insert` Hilfsprogramm bietet eine einfache Möglichkeit, mit dem `INSERT` Vorgang einzelne Zeilenelemente in Ihre Datenbank einzufügen.

Einfügungen einzelner Elemente

Um ein Element einzufügen, geben Sie die Tabelle an und übergeben Sie dann Ihr Werteobjekt. Die Objektschlüssel sind Ihren Tabellenspalten zugeordnet. Spaltennamen werden automatisch maskiert, und Werte werden mithilfe der Variablenzuordnung an die Datenbank gesendet:

```
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)
}
```

MySQL-Anwendungsfall

Sie können ein `insert` gefolgt von einem kombinieren`select`, um Ihre eingefügte Zeile abzurufen:

```
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)
}
```

Anwendungsfall Postgres

Mit Postgres können Sie Daten aus der Zeile abrufen `returning`, die Sie eingefügt haben. Es akzeptiert * oder ein Array von Spaltennamen:

```
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)
}
```

Aktualisierung

Das `update` Tool ermöglicht es Ihnen, bestehende Zeilen zu aktualisieren. Sie können das Bedingungsobjekt verwenden, um Änderungen an den angegebenen Spalten in allen Zeilen vorzunehmen, die die Bedingung erfüllen. Nehmen wir zum Beispiel an, wir haben ein Schema, das es uns ermöglicht, diese Mutation vorzunehmen. Wir wollen das `name` von `Person` mit dem `id` Wert von aktualisieren, `3` aber nur, wenn wir sie (`known_since`) seit dem Jahr kennen`2000`:

```
mutation Update {
    updatePerson(
        input: {id: 3, name: "Jon"},
        condition: {known_since: {ge: "2000"}}
    ) {
    id
    name
  }
}
```

Unser Update Resolver sieht so aus:

```
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)
}
```

Wir können unserer Bedingung ein Häkchen hinzufügen, um sicherzustellen, dass nur die Zeile aktualisiert 3 wird, deren Primärschlüssel `id` gleich ist. In ähnlicher Weise können Sie Postgres verwenden`inserts`, `returning` um die geänderten Daten zurückzugeben.

Remove

Mit dem `remove` Hilfsprogramm können Sie vorhandene Zeilen löschen. Sie können das Bedingungsobjekt für alle Zeilen verwenden, die die Bedingung erfüllen. Beachten Sie, dass `delete` es sich um ein reserviertes Schlüsselwort in handelt JavaScript. `remove`sollte stattdessen verwendet werden:

```
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)
}
```

Umwandlung

In einigen Fällen benötigen Sie möglicherweise genauere Angaben zum richtigen Objekttyp, den Sie in Ihrer Anweisung verwenden möchten. Sie können die bereitgestellten Typhinweise verwenden, um den Typ Ihrer Parameter anzugeben. AWS AppSync unterstützt dieselben Typhinweise wie die Daten-API. Sie können Ihre Parameter mithilfe der `typeHint` Funktionen des AWS AppSync `rds` Moduls umwandeln.

Das folgende Beispiel ermöglicht es Ihnen, ein Array als Wert zu senden, der als JSON-Objekt umgewandelt wird. Wir verwenden den `->` Operator, um das Element `index` 2 im JSON-Array abzurufen:

```
import { sql, createPgStatement, toJsonObject, typeHint } from '@aws-appsync/utils/
rds';

export function request(ctx) {
```

```
    const arr = ctx.args.list_of_ids
    const statement = sql`select ${typeHint.JSON(arr)}->2 as value`
    return createPgStatement(statement)
}

export function response(ctx) {
    return toJsonObject(ctx.result)[0][0].value
}
```

Casting ist auch nützlich bei der Handhabung und beim Vergleichen von `DATETIME`, und`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)
}
```

Hier ist ein weiteres Beispiel, das zeigt, wie Sie das aktuelle Datum und die aktuelle Uhrzeit senden können:

```
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)}`)
}
```

Verfügbare Typhinweise

- `typeHint.DATE`- Der entsprechende Parameter wird als Objekt des `DATE` Typs an die Datenbank gesendet. Das akzeptierte Format ist `YYYY-MM-DD`.
- `typeHint.DECIMAL`- Der entsprechende Parameter wird als Objekt des `DECIMAL` Typs an die Datenbank gesendet.
- `typeHint.JSON`- Der entsprechende Parameter wird als Objekt des `JSON` Typs an die Datenbank gesendet.

- `typeHint.TIME`- Der entsprechende Zeichenkettenparameterwert wird als Objekt des `TIME` Typs an die Datenbank gesendet. Das akzeptierte Format ist `HH:MM:SS[.FFF]`.
- `typeHint.TIMESTAMP`- Der entsprechende Zeichenkettenparameterwert wird als Objekt des `TIMESTAMP` Typs an die Datenbank gesendet. Das akzeptierte Format ist `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `typeHint.UUID`- Der entsprechende Zeichenkettenparameterwert wird als Objekt des `UUID` Typs an die Datenbank gesendet.

Laufzeit-Dienstprogramme

Die `runtime` Bibliothek stellt Dienstprogramme zur Verfügung, mit denen Sie die Laufzeiteigenschaften Ihrer Resolver und Funktionen steuern oder ändern können.

Liste der Runtime-Utils

`runtime.earlyReturn(obj?: unknown, returnOptions?: {skipTo: 'END' | 'NEXT'}): never`

Wenn Sie diese Funktion aufrufen, wird die Ausführung des aktuellen Handlers, der AWS AppSync Funktion oder des Resolvers (Unit- oder Pipeline-Resolver) je nach aktuellem Kontext angehalten. Das angegebene Objekt wird als Ergebnis zurückgegeben.

- Beim Aufruf in einem AWS AppSync Funktionsanforderungshandler werden die Datenquelle und der Antworthandler übersprungen, und der nächste Funktionsanforderungshandler (oder der Antworthandler des Pipeline-Resolvers, falls dies die letzte AWS AppSync Funktion war) wird aufgerufen.
- Beim Aufruf in einem AWS AppSync Pipeline-Resolver-Anforderungshandler wird die Pipeline-Ausführung übersprungen und der Pipeline-Resolver-Response-Handler wird sofort aufgerufen.
- Wenn angegeben und der Wert auf „END“ `skipTo` gesetzt `returnOptions` ist, wird die Pipeline-Ausführung übersprungen und der Response-Handler für den Pipeline-Resolver wird sofort aufgerufen.
- Wenn angegeben und auf „NEXT“ `skipTo` gesetzt `returnOptions` ist, wird die Funktionsausführung übersprungen und der nächste Pipeline-Handler aufgerufen.

Beispiel

```
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
}
```

Zeithelfer in util.time

Die `util.time`-Variable enthält "datetime"-Methoden, um beim Generieren von Zeitstempeln, Konvertieren zwischen "datetime"-Formaten und Parsen von "datetime"-Zeichenfolgen zu helfen. Die Syntax für Datetime-Formate basiert auf dieser Syntax DateTimeFormatter, auf die Sie für weitere Dokumentation verweisen können.

Liste der Zeitwerkzeuge

`util.time.nowISO8601()`

Gibt eine String-Repräsentation von UTC im ISO8601-Format zurück.

`util.time.nowEpochSeconds()`

Gibt die Anzahl der Sekunden aus der Epoche von 1970-01-01T00:00:00Z bis jetzt zurück.

`util.time.nowEpochMilliSeconds()`

Gibt die Anzahl der Millisekunden aus der Epoche von 1970-01-01T00:00:00Z bis jetzt zurück.

`util.time.nowFormatted(String)`

Gibt eine Zeichenfolge des aktuellen Zeitstempels in UTC unter Verwendung des angegebenen Formats aus einem Zeichenfolge-Eingabetyp zurück.

`util.time.nowFormatted(String, String)`

Gibt eine Zeichenfolge des aktuellen Zeitstempels aus einer Zeitzone unter Verwendung des angegebenen Formats und der Zeitzone aus Zeichenfolge-Eingabetypen zurück.

`util.time.parseFormattedToEpochMilliSeconds(String, String)`

Analysiert einen als String übergebenen Zeitstempel zusammen mit einem Format, das eine Zeitzone enthält, und gibt dann den Zeitstempel als Millisekunden seit Epoche zurück.

`util.time.parseFormattedToEpochMilliSeconds(String, String, String)`

Analysiert einen als String übergebenen Zeitstempel zusammen mit einem Format und einer Zeitzone und gibt dann den Zeitstempel in Millisekunden seit der Epoche zurück.

`util.time.parseISO8601ToEpochMilliSeconds(String)`

Analysiert einen als String übergebenen Zeitstempel ISO86 01 und gibt dann den Zeitstempel in Millisekunden seit der Epoche zurück.

`util.time.epochMilliSecondsToSeconds(long)`

Konvertiert einen Epoche-Millisekunden-Zeitstempel in einen Epoche-Sekunden-Zeitstempel.

`util.time.epochMilliSecondsToISO8601(long)`

Konvertiert einen Epochen-Millisekunden-Zeitstempel in einen 01-Zeitstempel. ISO86

`util.time.epochMilliSecondsToFormatted(long, String)`

Konvertiert einen Epochen-Millisekunden-Zeitstempel, der als long übergeben wurde, in einen Zeitstempel, der gemäß dem angegebenen Format in UTC formatiert ist.

`util.time.epochMilliSecondsToFormatted(long, String, String)`

Konvertiert einen Epochen-Millisekunden-Zeitstempel, der als Long übergeben wurde, in einen Zeitstempel, der gemäß dem angegebenen Format in der angegebenen Zeitzone formatiert ist.

DynamoDB-Helfer in util.dynamodb

`util.dynamodb`enthält Hilfsmethoden, die das Schreiben und Lesen von Daten in Amazon DynamoDB erleichtern, z. B. automatische Typzuweisung und Formatierung.

Zu DynamoDB

Liste der ToDynamoDB-Dienstprogramme

`util.dynamodb.toDynamoDB(Object)`

Allgemeines Objektkonvertierungstool für DynamoDB, das Eingabeobjekte in die entsprechende DynamoDB-Darstellung konvertiert. Es ist vorbestimmt, wie einige Typen dargestellt werden: z. B.

werden Listen ("L") anstelle von Sätzen ("SS", "NS", "BS") verwendet. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

Beispiel für eine Zeichenfolge

```
Input:      util.dynamodb.toDynamoDB("foo")
Output:     { "S" : "foo" }
```

Beispiel für eine Zahl

```
Input:      util.dynamodb.toDynamoDB(12345)
Output:     { "N" : 12345 }
```

Boolesches Beispiel

```
Input:      util.dynamodb.toDynamoDB(true)
Output:     { "BOOL" : true }
```

Beispiel auflisten

```
Input:      util.dynamodb.toDynamoDB([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

Beispiel für eine Karte

```
Input:      util.dynamodb.toDynamoDB({ "foo": "bar", "baz" : 1234, "beep":
 [ "boop"] })
Output:     {
               "M" : {
                   "foo"  : { "S" : "bar" },
```

```
                "baz"  : { "N" : 1234 },
                "beep" : {
                    "L" : [
                        { "S" : "boop" }
                    ]
                }
            }
        }
```

ToString-Dienstprogramme

Liste der ToString-Dienstprogramme

`util.dynamodb.toString(String)`

Konvertiert eine Eingabezeichenfolge in das DynamoDB-Zeichenkettenformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      util.dynamodb.toString("foo")
Output:     { "S" : "foo" }
```

`util.dynamodb.toStringSet(List<String>)`

Konvertiert eine Liste mit Strings in das DynamoDB-Zeichenkettensatzformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      util.dynamodb.toStringSet([ "foo", "bar", "baz" ])
Output:     { "SS" : [ "foo", "bar", "baz" ] }
```

ToNumber-Dienstprogramme

Liste der ToNumber-Dienstprogramme

`util.dynamodb.toNumber(Number)`

Konvertiert eine Zahl in das DynamoDB-Zahlenformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      util.dynamodb.toNumber(12345)
```

```
Output:     { "N" : 12345 }
```

`util.dynamodb.toNumberSet(List<Number>)`

Konvertiert eine Liste von Zahlen in das DynamoDB-Nummernsatzformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      util.dynamodb.toNumberSet([ 1, 23, 4.56 ])
Output:     { "NS" : [ 1, 23, 4.56 ] }
```

ToBinary-Dienstprogramme

Liste der ToBinary-Dienstprogramme

`util.dynamodb.toBinary(String)`

Konvertiert als Base64-Zeichenfolge kodierte Binärdaten in das DynamoDB-Binärformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      util.dynamodb.toBinary("foo")
Output:     { "B" : "foo" }
```

`util.dynamodb.toBinarySet(List<String>)`

Konvertiert eine Liste von Binärdaten, die als Base64-Zeichenfolgen kodiert sind, in das DynamoDB-Binärsatzformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      util.dynamodb.toBinarySet([ "foo", "bar", "baz" ])
Output:     { "BS" : [ "foo", "bar", "baz" ] }
```

ToBoolesche Hilfsprogramme

Liste der toBooleschen Dienstprogramme

`util.dynamodb.toBoolean(Boolean)`

Konvertiert einen booleschen Wert in das entsprechende boolesche DynamoDB-Format. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      util.dynamodb.toBoolean(true)
Output:     { "BOOL" : true }
```

ToNull Utils

ToNull-Utils-Liste

`util.dynamodb.toNull()`

Gibt eine Null im DynamoDB-Null-Format zurück. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      util.dynamodb.toNull()
Output:     { "NULL" : null }
```

ToList-Dienstprogramme

Liste der ToList-Dienstprogramme

`util.dynamodb.toList(List)`

Konvertiert eine Liste von Objekten in das DynamoDB-Listenformat. Jedes Element in der Liste wird auch in das entsprechende DynamoDB-Format konvertiert. Es ist vorbestimmt, wie einige verschachtelte Objekte dargestellt werden: z. B. werden Listen ("L") anstelle von Sätzen ("SS", "NS", "BS") verwendet. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      util.dynamodb.toList([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

ToMap-Dienstprogramme

Liste der ToMap-Dienstprogramme

`util.dynamodb.toMap(Map)`

Konvertiert eine Map in das DynamoDB-Kartenformat. Jeder Wert in der Map wird ebenfalls in das entsprechende DynamoDB-Format konvertiert. Es ist vorbestimmt, wie einige verschachtelte Objekte dargestellt werden: z. B. werden Listen ("L") anstelle von Sätzen ("SS", "NS", "BS") verwendet. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
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)`

Erstellt eine Kopie der Map, in der jeder Wert in das entsprechende DynamoDB-Format konvertiert wurde. Es ist vorbestimmt, wie einige verschachtelte Objekte dargestellt werden: z. B. werden Listen ("L") anstelle von Sätzen ("SS", "NS", "BS") verwendet.

```
Input:      util.dynamodb.toMapValues({ "foo": "bar", "baz" : 1234, "beep":
 [ "boop"] })
Output:     {
               "foo"  : { "S" : "bar" },
               "baz"  : { "N" : 1234 },
               "beep" : {
                   "L" : [
                       { "S" : "boop" }
                   ]
               }
           }
```

> **Note**
>
> Dies unterscheidet sich `util.dynamodb.toMap(Map)` geringfügig davon, dass nur der Inhalt des DynamoDB-Attributwerts zurückgegeben wird, nicht jedoch der gesamte Attributwert selbst. Die folgenden Aussagen sind beispielsweise identisch:
>
> ```
> util.dynamodb.toMapValues(<map>)
> util.dynamodb.toMap(<map>)("M")
> ```

S3Object-Dienstprogramme

Liste der S3Object-Dienstprogramme

`util.dynamodb.toS3Object(String key, String bucket, String region)`

Konvertiert den Schlüssel, den Bucket und die Region in die DynamoDB S3-Objektdarstellung. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
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)`

Konvertiert den Schlüssel, den Bucket, die Region und die optionale Version in die DynamoDB S3-Objektdarstellung. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      util.dynamodb.toS3Object("foo", "bar", "baz", "beep")
Output:     { "S" : "{ \"s3\" : { \"key\" : \"foo\", \"bucket\" : \"bar\", \"region
\" : \"baz\", \"version\" = \"beep\" } }" }
```

`util.dynamodb.fromS3ObjectJson(String)`

Akzeptiert den Zeichenkettenwert eines DynamoDB S3-Objekts und gibt eine Map zurück, die den Schlüssel, den Bucket, die Region und die optionale Version enthält.

```
Input:      util.dynamodb.fromS3ObjectJson({ "S" : "{ \"s3\" : { \"key\" : \"foo\",
 \"bucket\" : \"bar\", \"region\" : \"baz\", \"version\" = \"beep\" } }" })
```

```
Output:     { "key" : "foo", "bucket" : "bar", "region" : "baz", "version" :
 "beep" }
```

HTTP-Helfer in util.http

Das `util.http` Hilfsprogramm stellt Hilfsmethoden bereit, mit denen Sie HTTP-Anforderungsparameter verwalten und Antwortheader hinzufügen können.

Liste der util.http-Dienstprogramme

`util.http.copyHeaders(headers)`

Kopiert die Header aus der Map, mit Ausnahme der folgenden eingeschränkten HTTP-Header:

- transfer-encoding
- connection
- host
- expect
- keep-alive
- upgrade
- proxy-authenticate
- proxy-authorization
- te
- content-length

`util.http.addResponseHeader(String, Object)`

Fügt einen einzelnen benutzerdefinierten Header mit dem Namen (`String`) und dem Wert (`Object`) der Antwort hinzu. Die folgenden Einschränkungen gelten:

- Zusätzlich zu der Liste der eingeschränkten Header für `copyHeaders(headers)` dürfen Header-Namen keinem der folgenden Werte entsprechen:
  - Access-Control-Allow-Credentials
  - Access-Control-Allow-Origin
  - Access-Control-Expose-Headers
  - Access-Control-Max-Age
  - Access-Control-Allow-Methods

  - Access-Control-Allow-Headers
  - Vary
  - Content-Type
- Headernamen dürfen nicht mit den eingeschränkten Präfixen `x-amzn-` oder beginnen. `x-amz-`
- Die Größe der benutzerdefinierten Antwort-Header darf 4 KB nicht überschreiten. Dies schließt Header-Namen und -Werte ein.
- Sie sollten jeden Antwortheader einmal pro GraphQL-Operation definieren. Wenn Sie jedoch mehrmals einen benutzerdefinierten Header mit demselben Namen definieren, erscheint die neueste Definition in der Antwort. Alle Header werden unabhängig von der Benennung auf die Größenbeschränkung für Header angerechnet.
- Header mit einem leeren oder eingeschränkten Namen (`String`) oder einem Nullwert (`Object`) werden ignoriert und führen zu einem `ResponseHeaderError` Fehler, der zur Ausgabe des `errors` Vorgangs hinzugefügt wird.

```
export function request(ctx) {
  util.http.addResponseHeader('itemsCount', 7)
  util.http.addResponseHeader('render', ctx.args.render)
  return {}
}
```

`util.http.addResponseHeaders(Map)`

Fügt der Antwort mehrere Antwort-Header aus der angegebenen Zuordnung von Namen (`String`) und Werten hinzu. (`Object`) Dieselben Einschränkungen, die für die `addResponseHeader(String, Object)` Methode aufgeführt sind, gelten auch für diese Methode.

```
export function request(ctx) {
  const headers = {
    headerInt: 12,
    headerString: 'stringValue',
    headerObject: {
      field1: 7,
      field2: 'string'
    }
  }
  util.http.addResponseHeaders(headers)
  return {}
```

```
}
```

Transformationshelfer in util.transform

`util.transform`enthält Hilfsmethoden, die es einfacher machen, komplexe Operationen an Datenquellen durchzuführen.

Liste der Hilfsprogramme für Transformationshelfer

`util.transform.toDynamoDBFilterExpression(filterObject: DynamoDBFilterObject) : string`

Konvertiert eine Eingabezeichenfolge in einen Filterausdruck zur Verwendung mit DynamoDB. Wir empfehlen die Verwendung `toDynamoDBFilterExpression` mit integrierten Modulfunktionen.

`util.transform.toElasticsearchQueryDSL(object: OpenSearchQueryObject) : string`

Konvertiert die angegebene Eingabe in den entsprechenden OpenSearch Query-DSL-Ausdruck und gibt sie als JSON-Zeichenfolge zurück.

Beispieleingabe:

```
util.transform.toElasticsearchQueryDSL({
    "upvotes":{
        "ne":15,
        "range":[
            10,
            20
        ]
    },
    "title":{
        "eq":"hihihi",
        "wildcard":"h*i"
    }
  })
```

Beispielausgabe:

```
{
    "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**
>
> Es wird angenommen, dass der Standardoperator AND ist.

`util.transform.toSubscriptionFilter(objFilter, ignoredFields?, rules?): SubscriptionFilter`

Konvertiert ein `Map` Eingabeobjekt in ein `SubscriptionFilter` Ausdrucksobjekt. Die `util.transform.toSubscriptionFilter` Methode wird als Eingabe für die `extensions.setSubscriptionFilter()` Erweiterung verwendet. Weitere Informationen finden Sie unter Erweiterungen.

> **Note**
>
> Die Parameter und die Rückgabeanweisung sind unten aufgeführt:
> Parameter
>
> - `objFilter`: `SubscriptionFilterObject`
>
> Ein `Map` Eingabeobjekt, das in das `SubscriptionFilter` Ausdrucksobjekt konvertiert wurde.
> - `ignoredFields`: `SubscriptionFilterExcludeKeysType` (optional)
>
> Eine `List` der Feldnamen im ersten Objekt, die ignoriert werden.
> - `rules`: `SubscriptionFilterRuleObject` (fakultativ)
>
> Ein `Map` Eingabeobjekt mit strengen Regeln, das bei der Konstruktion des `SubscriptionFilter` Ausdrucksobjekts berücksichtigt wird. Diese strengen Regeln werden in das `SubscriptionFilter` Ausdrucksobjekt aufgenommen, sodass mindestens eine der Regeln erfüllt ist, um den Abonnementfilter zu bestehen.
>
> Antwort
> Gibt eine `SubscriptionFilter` zurück.

`util.transform.toSubscriptionFilter(Map, List)`

Konvertiert ein `Map` Eingabeobjekt in ein `SubscriptionFilter` Ausdrucksobjekt. Die `util.transform.toSubscriptionFilter` Methode wird als Eingabe für die

`extensions.setSubscriptionFilter()` Erweiterung verwendet. Weitere Informationen finden Sie unter Erweiterungen.

Das erste Argument ist das `Map` Eingabeobjekt, das in das `SubscriptionFilter` Ausdrucksobjekt konvertiert wurde. Das zweite Argument besteht `List` aus Feldnamen, die im ersten `Map` Eingabeobjekt bei der Konstruktion des `SubscriptionFilter` Ausdrucksobjekts ignoriert werden.

`util.transform.toSubscriptionFilter(Map, List, Map)`

Konvertiert ein `Map` Eingabeobjekt in ein `SubscriptionFilter` Ausdrucksobjekt. Die `util.transform.toSubscriptionFilter` Methode wird als Eingabe für die `extensions.setSubscriptionFilter()` Erweiterung verwendet. Weitere Informationen finden Sie unter Erweiterungen.

`util.transform.toDynamoDBConditionExpression(conditionObject)`

Erstellt einen DynamoDB-Bedingungsausdruck.

Argumente für den Abonnementfilter

In der folgenden Tabelle wird erklärt, wie die Argumente der folgenden Dienstprogramme definiert sind:

- `Util.transform.toSubscriptionFilter(objFilter, ignoredFields?, rules?): SubscriptionFilter`

Argument 1: Map

Argument 1 ist ein `Map` Objekt mit den folgenden Schlüsselwerten:

- Feldnamen
- „und“
- „oder“

Bei Feldnamen als Schlüssel haben die Bedingungen für die Einträge dieser Felder die Form von`"operator" : "value"`.

Das folgende Beispiel zeigt, wie Einträge hinzugefügt werden können: `Map`

```
"field_name" : {
                    "operator1" : value
               }

## We can have multiple conditions for the same field_name:

"field_name" : {
                    "operator1" : value
                    "operator2" : value
                    .
                    .
                    .
               }
```

Wenn ein Feld zwei oder mehr Bedingungen enthält, wird davon ausgegangen, dass alle diese Bedingungen die OR-Operation verwenden.

Die Eingabe `Map` kann auch „und“ und „oder“ als Schlüssel enthalten, was bedeutet, dass alle darin enthaltenen Einträge je nach Schlüssel mit DER UND- oder OR-Logik verknüpft werden sollten. Die Schlüsselwerte „und“ und „oder“ erwarten eine Reihe von Bedingungen.

```
"and" : [

            {
                "field_name1" : {
                    "operator1" : value
                }
             },

             {
                "field_name2" : {
                    "operator1" : value
                }
             },
             .
             .
        ].
```

Beachten Sie, dass Sie „und“ und „oder“ verschachteln können. Das heißt, Sie können „und“ /"oder“ innerhalb eines anderen „und“ /"oder“ -Blocks verschachtelt haben. Dies funktioniert jedoch nicht für einfache Felder.

```
"and" : [

            {
                "field_name1" : {
                    "operator" : value
                }
             },

             {
                "or" : [
                            {
                                "field_name2" : {
                                    "operator" : value
                                }
                            },

                            {
                                "field_name3" : {
                                    "operator" : value
                                }
                            }

                        ].
```

Das folgende Beispiel zeigt eine Eingabe von Argument 1 mit`util.transform.toSubscriptionFilter(Map) : Map`.

Eingabe (en)

Argument 1: Karte:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
```

```
      "downvotes": {
        "gt": 2000
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Ausgabe

Das Ergebnis ist ein `Map` Objekt:

```
{
  "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

Argument 2 enthält eine Reihe `List` von Feldnamen, die bei der Erstellung des `SubscriptionFilter` Ausdrucksobjekts nicht in der Eingabe `Map` (Argument 1) berücksichtigt werden sollten. Das `List` kann auch leer sein.

Das folgende Beispiel zeigt die Eingaben von Argument 1 und Argument 2 mit`util.transform.toSubscriptionFilter(Map, List) : Map`.

Eingabe (en)

Argument 1: Karte:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Argument 2: Liste:

```
["percentageUp", "author"]
```

Ausgabe

Das Ergebnis ist ein `Map` Objekt:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    }
  ]
}
```

Argument 3: Map

Argument 3 ist ein `Map` Objekt, das Feldnamen als Schlüsselwerte hat (darf nicht „und“ oder „oder“ enthalten). Bei Feldnamen als Schlüssel handelt es sich bei den Bedingungen für diese Felder um Einträge in der Form von`"operator" : "value"`. Im Gegensatz zu Argument 1 kann Argument 3 nicht mehrere Bedingungen in demselben Schlüssel haben. Darüber hinaus hat Argument 3 keine „Und“ - oder „Oder“ -Klausel, sodass auch keine Verschachtelung erforderlich ist.

Argument 3 stellt eine Liste strenger Regeln dar, die dem `SubscriptionFilter` Ausdrucksobjekt hinzugefügt werden, sodass mindestens eine dieser Bedingungen erfüllt ist, um den Filter zu bestehen.

```
{
  "fieldname1": {
    "operator": value
```

```
  },
  "fieldname2": {
    "operator": value
  }
}
.
.
.
```

Das folgende Beispiel zeigt die Eingaben von Argument 1, Argument 2 und Argument 3 mithilfe von`util.transform.toSubscriptionFilter(Map, List, Map) : Map`.

Eingabe (en)

Argument 1: Karte:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "lt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
```

```
    }
  ]
}
```

Argument 2: Liste:

```
["percentageUp", "author"]
```

Argument 3: Karte:

```
{
  "upvotes": {
    "gte": 250
  },
  "author": {
    "eq": "Person1"
  }
}
```

Ausgabe

Das Ergebnis ist ein `Map` Objekt:

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

Zeichenketten-Helfer in util.str

`util.str`enthält Methoden, die bei gängigen String-Operationen helfen.

util.str Utils-Liste

`util.str.normalize(String, String)`

Normalisiert eine Zeichenfolge mit einer der vier Unicode-Normalisierungsformen: NFC, NFD, NFKC oder NFKD. Das erste Argument ist die zu normalisierende Zeichenfolge. Das zweite Argument ist entweder „nfc“, „nfd“, „nfkc“ oder „nfkd“ und gibt den Normalisierungstyp an, der für den Normalisierungsprozess verwendet werden soll.

Erweiterungen

`extensions`enthält eine Reihe von Methoden, mit denen Sie zusätzliche Aktionen in Ihren Resolvern ausführen können.

Erweiterungen zwischenspeichern

`extensions.evictFromApiCache(typeName: string, fieldName: string, keyValuePair: Record<string, string>) : Object`

Löscht ein Element aus dem serverseitigen AWS AppSync Cache. Das erste Argument ist der Typname. Das zweite Argument ist der Feldname. Das dritte Argument ist ein Objekt, das Schlüssel-Wert-Paar-Elemente enthält, die den Zwischenspeicher-Schlüsselwert angeben. Sie müssen die Elemente im Objekt in derselben Reihenfolge platzieren wie die Caching-Schlüssel in den zwischengespeicherten Resolvern. `cachingKey` Weitere Informationen zum Zwischenspeichern finden Sie unter Caching-Verhalten.

Beispiel 1:

In diesem Beispiel werden die Elemente entfernt, die für einen aufgerufenen Resolver zwischengespeichert wurden, für den ein aufgerufener `Query.allClasses` Caching-Schlüssel verwendet wurde. `context.arguments.semester` Wenn die Mutation aufgerufen wird und der Resolver ausgeführt wird und ein Eintrag erfolgreich gelöscht wurde, enthält die Antwort einen `apiCacheEntriesDeleted` Wert im Erweiterungsobjekt, der angibt, wie viele Einträge gelöscht wurden.

```
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**
>
> Diese Funktion funktioniert nur für Mutationen, nicht für Abfragen.

Abonnement-Erweiterungen

`extensions.setSubscriptionFilter(filterJsonObject)`

Definiert erweiterte Abonnementfilter. Jedes Abonnementbenachrichtigungsereignis wird anhand der bereitgestellten Abonnementfilter bewertet und sendet Benachrichtigungen an Kunden, wenn alle Filter als erfüllt gelten`true`. Das Argument ist `filterJsonObject` (Weitere Informationen zu diesem Argument finden Sie weiter unten im filterJsonObject Abschnitt Argument:.). Siehe Verbesserte Abonnementfilterung.

> **Note**
>
> Sie können diese Erweiterungsfunktion nur im Antworthandler eines Abonnement-Resolvers verwenden. Außerdem empfehlen wir die Verwendung, `util.transform.toSubscriptionFilter` um Ihren Filter zu erstellen.

`extensions.setSubscriptionInvalidationFilter(filterJsonObject)`

Definiert Filter für die Invalidierung von Abonnements. Abonnementfilter werden anhand der Payload für die Ungültigerklärung bewertet und machen dann ein bestimmtes Abonnement ungültig, wenn die Filter Folgendes ergeben. `true` Das Argument ist `filterJsonObject` (Weitere Informationen zu diesem Argument finden Sie weiter unten im filterJsonObject Abschnitt Argument:.). Siehe Verbesserte Abonnementfilterung.

> **Note**
>
> Sie können diese Erweiterungsfunktion nur im Antworthandler eines Abonnement-Resolvers verwenden. Außerdem empfehlen wir die Verwendung, `util.transform.toSubscriptionFilter` um Ihren Filter zu erstellen.

`extensions.invalidateSubscriptions(invalidationJsonObject)`

Wird verwendet, um die Invalidierung eines Abonnements aufgrund einer Mutation einzuleiten. Das Argument lautet `invalidationJsonObject` (Weitere Informationen zu diesem Argument finden Sie weiter unten im invalidationJsonObject Abschnitt Argument:.).

> **Note**
>
> Diese Erweiterung kann nur in den Response-Mapping-Vorlagen der Mutationsresolver verwendet werden.
> Sie können in einer einzelnen Anfrage nur maximal fünf eindeutige `extensions.invalidateSubscriptions()` Methodenaufrufen verwenden. Wenn Sie dieses Limit überschreiten, erhalten Sie einen GraphQL-Fehler.

Argument: filterJsonObject

Das JSON-Objekt definiert entweder Abonnement- oder Invalidierungsfilter. Es ist eine Reihe von Filtern in einem`filterGroup`. Jeder Filter ist eine Sammlung einzelner Filter.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                }
           ]

        },
        {
```

```
            "filters" : [
                 {
                     "fieldName" : "group",
                     "operator" : "in",
                     "value" : ["Admin", "Developer"]
                 }
            ]

        }
    ]
}
```

Jeder Filter hat drei Attribute:

- `fieldName`— Das GraphQL-Schemafeld.
- `operator`— Der Operatortyp.
- `value`— Die Werte, die mit dem `fieldName` Wert der Abonnementbenachrichtigung verglichen werden sollen.

Im Folgenden finden Sie ein Beispiel für die Zuweisung dieser Attribute:

```
{
 "fieldName" : "severity",
 "operator" : "le",
 "value" : context.result.severity
}
```

Argument: invalidationJsonObject

Das `invalidationJsonObject` definiert Folgendes:

- `subscriptionField`— Das GraphQL-Schemaabonnement, das ungültig werden soll. Ein einzelnes Abonnement, das als Zeichenfolge in der definiert ist`subscriptionField`, wird für ungültig erklärt.
- `payload`— Eine Liste mit Schlüssel-Wert-Paaren, die als Eingabe für die Ungültigerklärung von Abonnements verwendet wird, wenn der Invalidierungsfilter A anhand ihrer Werte auswertet. `true`

  Im folgenden Beispiel werden abonnierte und verbundene Clients, die das Abonnement verwenden, ungültig gemacht, wenn der im `onUserDelete` Abonnement-Resolver definierte Invalidierungsfilter das Ergebnis anhand des Werts auswertet. `true` `payload`

```
export const request = (ctx) => ({ payload: null });

export function response(ctx) {
 extensions.invalidateSubscriptions({
  subscriptionField: 'onUserDelete',
  payload: { group: 'Developer', type: 'Full-Time' },
 });
 return ctx.result;
}
```

XML-Hilfsprogramme in util.xml

`util.xml`enthält Methoden, die bei der Konvertierung von XML-Zeichenketten helfen.

Liste der Dienstprogramme von util.xml

`util.xml.toMap(String) : Object`

Konvertiert eine XML-Zeichenfolge in ein Wörterbuch.

Beispiel 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"
      }
    }
```

```
}
```

Beispiel 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`

Konvertiert eine XML-Zeichenfolge in eine JSON-Zeichenfolge. Dies ist ähnlich`toMap`, außer dass die Ausgabe eine Zeichenfolge ist. Dies ist nützlich, wenn Sie direkt umwandeln und die XML-Antwort aus einem HTTP-Objekt an JSON zurückgeben. Sie können einen optionalen booleschen Parameter festlegen, um zu bestimmen, ob Sie den JSON als Zeichenfolge kodieren möchten.

AWS AppSync JavaScript Resolver-Funktionsreferenz für DynamoDB

Die AWS AppSync DynamoDB-Funktion ermöglicht es Ihnen, GraphQL zum Speichern und Abrufen von Daten in vorhandenen Amazon DynamoDB-Tabellen in Ihrem Konto zu verwenden, indem Sie eine eingehende GraphQL-Anfrage einem DynamoDB-Aufruf zuordnen und dann die DynamoDB-Antwort wieder GraphQL zuordnen. In diesem Abschnitt werden die Anforderungs- und Antworthandler für unterstützte DynamoDB-Operationen beschrieben:

- GetItem- Mit der GetItem Anfrage können Sie die DynamoDB-Funktion anweisen, eine GetItem Anfrage an DynamoDB zu stellen, und Sie können den Schlüssel des Elements in DynamoDB angeben und angeben, ob ein konsistenter Lesevorgang verwendet werden soll oder nicht.
- PutItem - Mit dem PutItem Anforderungszuordnungsdokument können Sie die DynamoDB-Funktion anweisen, eine PutItem Anfrage an DynamoDB zu stellen, und Sie können den Schlüssel des Elements in DynamoDB, den vollständigen Inhalt des Elements (bestehend aus Schlüssel und AttributeValues) und Bedingungen für den Erfolg des Vorgangs angeben.
- UpdateItem - Mit der UpdateItem Anforderung können Sie die DynamoDB-Funktion anweisen, eine UpdateItem Anfrage an DynamoDB zu stellen, und Sie können den Schlüssel des Elements in DynamoDB, einen Aktualisierungsausdruck, der beschreibt, wie das Element in DynamoDB aktualisiert wird, und die Bedingungen für den Erfolg des Vorgangs angeben.
- DeleteItem - Mit der DeleteItem Anfrage können Sie die DynamoDB-Funktion anweisen, eine DeleteItem Anfrage an DynamoDB zu stellen, und Sie können den Schlüssel des Elements in DynamoDB und die Bedingungen für den Erfolg des Vorgangs angeben.
- Abfrage — Mit dem Query-Anforderungsobjekt können Sie den DynamoDB-Resolver anweisen, eine Abfrageanforderung an DynamoDB zu stellen, und Sie können den Schlüsselausdruck, den zu verwendenden Index, zusätzliche Filter, die Anzahl der zurückzugebenden Elemente, die Verwendung konsistenter Lesevorgänge, die Abfragerichtung (vorwärts oder rückwärts) und Paginierungstoken angeben.
- Scan — Mit der Scananforderung können Sie die DynamoDB-Funktion anweisen, eine Scananforderung an DynamoDB zu stellen, und Sie können einen Filter angeben, um Ergebnisse auszuschließen, welcher Index verwendet werden soll, wie viele Elemente zurückgegeben werden sollen, ob konsistente Lesevorgänge, Paginierungstoken und parallel Scans verwendet werden sollen.
- Sync — Mit dem Sync-Anforderungsobjekt können Sie alle Ergebnisse aus einer DynamoDB-Tabelle abrufen und anschließend nur die Daten empfangen, die seit Ihrer letzten Abfrage

geändert wurden (die Delta-Updates). Synchronisierungsanforderungen können nur an versionierte DynamoDB-Datenquellen gestellt werden. Sie können einen Filter angeben, um Ergebnisse, die Anzahl der zurückzugebenden Elemente, die Paginierungstoken und den Zeitpunkt, an dem Ihr letzter Synchronisierungsvorgang gestartet wurde, auszuschließen.

- BatchGetItem - Mit dem BatchGetItem Anforderungsobjekt können Sie die DynamoDB-Funktion anweisen, eine BatchGetItem Anforderung an DynamoDB zu stellen, um mehrere Elemente abzurufen, möglicherweise über mehrere Tabellen hinweg. Für dieses Anforderungsobjekt müssen Sie die Tabellennamen angeben, aus denen die Elemente abgerufen werden sollen, und die Schlüssel der Elemente, die aus jeder Tabelle abgerufen werden sollen.
- BatchDeleteItem - Mit dem BatchDeleteItem Anforderungsobjekt können Sie die DynamoDB-Funktion anweisen, eine BatchWriteItem Anforderung an DynamoDB zu stellen, um mehrere Elemente zu löschen, möglicherweise über mehrere Tabellen hinweg. Für dieses Anforderungsobjekt müssen Sie die Tabellennamen angeben, aus denen die Elemente gelöscht werden sollen, und die Schlüssel der Elemente, die aus jeder Tabelle gelöscht werden sollen.
- BatchPutItem - Mit dem BatchPutItem Anforderungsobjekt können Sie die DynamoDB-Funktion anweisen, eine BatchWriteItem Anforderung an DynamoDB zu stellen, um mehrere Elemente zu platzieren, möglicherweise in mehreren Tabellen. Für dieses Anforderungsobjekt müssen Sie die Tabellennamen angeben, in die die Elemente eingefügt werden sollen, und die vollständigen Elemente, die in jede Tabelle eingefügt werden sollen.
- TransactGetItems - Mit dem TransactGetItems Anforderungsobjekt können Sie die DynamoDB-Funktion anweisen, eine TransactGetItems Anforderung an DynamoDB zu stellen, um mehrere Elemente abzurufen, möglicherweise über mehrere Tabellen hinweg. Für dieses Anforderungsobjekt müssen Sie den Tabellennamen jedes Anforderungselements angeben, aus dem das Element abgerufen werden soll, und den Schlüssel jedes Anforderungselements, das aus jeder Tabelle abgerufen werden soll.
- TransactWriteItems - Mit dem TransactWriteItems Anforderungsobjekt können Sie die DynamoDB-Funktion anweisen, eine TransactWriteItems Anforderung an DynamoDB zu stellen, um mehrere Elemente zu schreiben, möglicherweise in mehrere Tabellen. Für dieses Anforderungsobjekt müssen Sie den Zieltabellennamen jedes Anforderungselements, die Operation jedes auszuführenden Anforderungselements und den Schlüssel jedes zu schreibenden Anforderungselements angeben.
- Typsystem (Anforderungszuordnung) — Erfahren Sie mehr darüber, wie DynamoDB-Typisierung in Anfragen integriert AWS AppSync wird.
- Typsystem (Antwortzuordnung) — Erfahren Sie mehr darüber, wie DynamoDB-Typen in einer Antwortnutzlast automatisch in GraphQL oder JSON konvertiert werden.

- Filter — Erfahren Sie mehr über Filter für Abfrage- und Scanvorgänge.
- Bedingungsausdrücke — Erfahren Sie mehr über Bedingungsausdrücke für PutItem UpdateItem, und DeleteItem Operationen.
- Ausdrücke für Transaktionsbedingungen — Erfahren Sie mehr über Bedingungsausdrücke für TransactWriteItems Operationen.
- Projektionen — Erfahren Sie mehr darüber, wie Sie Attribute bei Lesevorgängen angeben.

GetItem

Mit der `GetItem` Anfrage können Sie die AWS AppSync DynamoDB-Funktion anweisen, eine `GetItem` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Der Schlüssel des Elements in DynamoDB
- Ob ein Consistent-Lesevorgang verwendet wird oder nicht

Die `GetItem` Anfrage hat die folgende Struktur:

```
type DynamoDBGetItem = {
  operation: 'GetItem';
  key: { [key: string]: any };
  consistentRead?: ConsistentRead;
  projection?: {
    expression: string;
    expressionNames?: { [key: string]: string };
  };
};
```

Die Felder sind wie folgt definiert:

GetItem Felder

GetItem Liste der Felder

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `GetItem`-DynamoDB-Operation durchzuführen, muss diese auf `GetItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der Schlüssel des Elements in DynamoDB. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`consistentRead`

Ob ein stark konsistenter Lesevorgang mit DynamoDB durchgeführt werden soll oder nicht. Dieser Schritt ist optional und standardmäßig auf `false` gesetzt.

`projection`

Eine Projektion, die verwendet wird, um die Attribute anzugeben, die von der DynamoDB-Operation zurückgegeben werden sollen. Weitere Informationen zu Projektionen finden Sie unter Projektionen. Dies ist ein optionales Feld.

Das von DynamoDB zurückgegebene Element wird automatisch in primitive GraphQL- und JSON-Typen konvertiert und ist im Kontext result () verfügbar. `context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu JavaScript Resolvern finden Sie unter JavaScript Übersicht über Resolver.

Beispiel

Das folgende Beispiel ist ein Funktionsanforderungshandler für eine GraphQL-Abfrage`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
  }
}
```

Weitere Informationen zur DynamoDB `GetItem`-API finden Sie in der DynamoDB API-Dokumentation.

PutItem

Mit dem `PutItem` Anforderungszuordnungsdokument können Sie die AWS AppSync DynamoDB-Funktion anweisen, eine `PutItem` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Der Schlüssel des Elements in DynamoDB
- Der vollständige Inhalt des Elements (bestehend aus `key` und `attributeValues`)
- Bedingungen, damit die Operation erfolgreich ausgeführt werden kann

Die `PutItem` Anfrage hat die folgende Struktur:

```
type DynamoDBPutItemRequest = {
  operation: 'PutItem';
  key: { [key: string]: any };
  attributeValues: { [key: string]: any};
  condition?: ConditionCheckExpression;
  customPartitionKey?: string;
  populateIndexFields?: boolean;
  _version?: number;
};
```

Die Felder sind wie folgt definiert:

PutItem Felder

PutItem Liste der Felder

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `PutItem`-DynamoDB-Operation durchzuführen, muss diese auf `PutItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der Schlüssel des Elements in DynamoDB. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`attributeValues`

Der Rest der Attribute des Elements, die in DynamoDB gespeichert werden sollen. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dies ist ein optionales Feld.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, überschreibt die `PutItem`-Anforderung alle vorhandenen Einträge für dieses Element. Weitere Informationen zu Bedingungen finden Sie unter Bedingungsausdrücke. Dieser Wert ist optional.

`_version`

Ein numerischer Wert, der die neueste bekannte Version eines Elements darstellt. Dieser Wert ist optional. Dieses Feld wird für die Konflikterkennung verwendet und nur für versionsgesteuerte Datenquellen unterstützt.

`customPartitionKey`

Wenn diese Option aktiviert ist, ändert dieser Zeichenfolgenwert das Format der `ds_sk` und `ds_pk` -Datensätze, die von der Delta-Synchronisierungstabelle verwendet werden, wenn die Versionierung aktiviert wurde (weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch). Wenn diese Option aktiviert ist, ist auch die Verarbeitung des `populateIndexFields` Eintrags aktiviert. Dies ist ein optionales Feld.

`populateIndexFields`

Ein boolescher Wert, der, wenn er zusammen mit dem aktiviert wird`customPartitionKey`, neue Einträge für jeden Datensatz in der Delta-Synchronisierungstabelle erstellt, insbesondere in den Spalten `gsi_ds_pk` und`gsi_ds_sk`. Weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch. Dies ist ein optionales Feld.

Das in DynamoDB geschriebene Element wird automatisch in primitive GraphQL- und JSON-Typen konvertiert und ist im Kontext result () verfügbar. `context.result`

Das in DynamoDB geschriebene Element wird automatisch in primitive GraphQL- und JSON-Typen konvertiert und ist im Kontext result () verfügbar. `context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu JavaScript Resolvern finden Sie unter JavaScript Übersicht über Resolver.

Beispiel 1

Das folgende Beispiel ist ein Funktionsanforderungshandler für eine GraphQL-Mutation`updateThing(foo: String!, bar: String!, name: String!, version: Int!)`.

Wenn kein Element mit dem angegebenen Schlüssel vorhanden ist, wird es erstellt. Wenn ein Element mit dem angegebenen Schlüssel bereits vorhanden ist, wird es überschrieben.

```
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),
  };
}
```

Beispiel 2

Das folgende Beispiel ist ein Funktionsanforderungshandler für eine GraphQL-Mutation`updateThing(foo: String!, bar: String!, name: String!, expectedVersion: Int!)`.

In diesem Beispiel wird überprüft, ob für das Element, das sich derzeit in DynamoDB befindet, das `version` Feld auf gesetzt ist. `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,
  };
}
```

Weitere Informationen zur DynamoDB `PutItem`-API finden Sie in der DynamoDB API-Dokumentation.

UpdateItem

Mit der `UpdateItem` Anfrage können Sie die AWS AppSync DynamoDB-Funktion anweisen, eine `UpdateItem` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Der Schlüssel des Elements in DynamoDB
- Ein Aktualisierungsausdruck, der beschreibt, wie das Element in DynamoDB aktualisiert wird
- Bedingungen, damit die Operation erfolgreich ausgeführt werden kann

Die `UpdateItem` Anfrage hat die folgende Struktur:

```
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;
};
```

Die Felder sind wie folgt definiert:

UpdateItem Felder

UpdateItem Liste der Felder

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `UpdateItem`-DynamoDB-Operation durchzuführen, muss diese auf `UpdateItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der Schlüssel des Elements in DynamoDB. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Hinweise zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`update`

`update`In diesem Abschnitt können Sie einen Aktualisierungsausdruck angeben, der beschreibt, wie das Element in DynamoDB aktualisiert wird. Weitere Informationen zum Schreiben von Aktualisierungsausdrücken finden Sie in der DynamoDB-Dokumentation UpdateExpressions . Dieser Abschnitt ist erforderlich.

Der `update`-Abschnitt enthält drei Komponenten:

`expression`

Den Aktualisierungsausdruck. Dieser Wert ist erforderlich.

`expressionNames`

Die Ersetzungen für Platzhalter der Namen von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter`expression`, der in der verwendet wird, und der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Namen von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

`expressionValues`

Die Ersetzungen für Platzhalter der Werte von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Wertplatzhalter, der im `expression` verwendet wird, und der Wert muss ein typisierter Wert sein. Weitere Informationen zur Angabe eines

„typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser muss angegeben werden. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Werte von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, aktualisiert die `UpdateItem`-Anforderung jeden vorhandenen Eintrag unabhängig vom aktuellen Status. Weitere Informationen zu Bedingungen finden Sie unter Bedingungsausdrücke. Dieser Wert ist optional.

`_version`

Ein numerischer Wert, der die neueste bekannte Version eines Elements darstellt. Dieser Wert ist optional. Dieses Feld wird für die Konflikterkennung verwendet und nur für versionsgesteuerte Datenquellen unterstützt.

`customPartitionKey`

Wenn diese Option aktiviert ist, ändert dieser Zeichenfolgenwert das Format der `ds_sk` und `ds_pk` -Datensätze, die von der Delta-Synchronisierungstabelle verwendet werden, wenn die Versionierung aktiviert wurde (weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch). Wenn diese Option aktiviert ist, ist auch die Verarbeitung des `populateIndexFields` Eintrags aktiviert. Dies ist ein optionales Feld.

`populateIndexFields`

Ein boolescher Wert, der, wenn er zusammen mit dem aktiviert wird`customPartitionKey`, neue Einträge für jeden Datensatz in der Delta-Synchronisierungstabelle erstellt, insbesondere in den Spalten `gsi_ds_pk` und`gsi_ds_sk`. Weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch. Dies ist ein optionales Feld.

Das in DynamoDB aktualisierte Element wird automatisch in primitive GraphQL- und JSON-Typen konvertiert und ist im Kontext result () verfügbar. `context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu JavaScript Resolvern finden Sie unter JavaScript Übersicht über Resolver.

Beispiel 1

Das folgende Beispiel ist ein Funktionsanforderungshandler für die GraphQL-Mutation`upvote(id: ID!)`.

In diesem Beispiel werden die `version` Felder `upvotes` und eines Elements in DynamoDB um 1 erhöht.

```
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 } },
    },
  };
}
```

Beispiel 2

Das folgende Beispiel ist ein Funktionsanforderungshandler für eine GraphQL-Mutation`updateItem(id: ID!, title: String, author: String, expectedVersion: Int!)`.

Hierbei handelt es sich um ein komplexes Beispiel, das die Argumente prüft und den Aktualisierungsausdruck dynamisch generiert, der nur die Argumente enthält, die vom Client bereitgestellt wurden. Beispiel: Wenn `title` und `author` nicht angegeben werden, werden sie nicht aktualisiert. Wenn ein Argument angegeben ist, sein Wert jedoch angegeben ist`null`, wird dieses Feld aus dem Objekt in DynamoDB gelöscht. Schließlich hat der Vorgang eine Bedingung, die überprüft, ob für das Element, das sich derzeit in DynamoDB befindet, das `version` Feld wie folgt gesetzt ist: `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),
    },
  };
}
```

Weitere Informationen zur DynamoDB `UpdateItem`-API finden Sie in der DynamoDB API-Dokumentation.

DeleteItem

Mit der `DeleteItem` Anfrage können Sie die AWS AppSync DynamoDB-Funktion anweisen, eine `DeleteItem` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Der Schlüssel des Elements in DynamoDB
- Bedingungen, damit die Operation erfolgreich ausgeführt werden kann

Die `DeleteItem` Anfrage hat die folgende Struktur:

```
type DynamoDBDeleteItemRequest = {
  operation: 'DeleteItem';
  key: { [key: string]: any };
  condition?: ConditionCheckExpression;
  customPartitionKey?: string;
  populateIndexFields?: boolean;
  _version?: number;
};
```

Die Felder sind wie folgt definiert:

DeleteItem Felder

DeleteItem Liste der Felder

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `DeleteItem`-DynamoDB-Operation durchzuführen, muss diese auf `DeleteItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der Schlüssel des Elements in DynamoDB. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Hinweise zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, löscht die `DeleteItem`-Anforderung ein Element unabhängig vom aktuellen Status. Weitere Informationen zu Bedingungen finden Sie unter Bedingungsausdrücke. Dieser Wert ist optional.

`_version`

Ein numerischer Wert, der die neueste bekannte Version eines Elements darstellt. Dieser Wert ist optional. Dieses Feld wird für die Konflikterkennung verwendet und nur für versionsgesteuerte Datenquellen unterstützt.

`customPartitionKey`

Wenn diese Option aktiviert ist, ändert dieser Zeichenfolgenwert das Format der `ds_sk` und `ds_pk` -Datensätze, die von der Delta-Synchronisierungstabelle verwendet werden, wenn die Versionierung aktiviert wurde (weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch). Wenn diese Option aktiviert ist, ist auch die Verarbeitung des `populateIndexFields` Eintrags aktiviert. Dies ist ein optionales Feld.

`populateIndexFields`

Ein boolescher Wert, der, wenn er zusammen mit dem aktiviert wird`customPartitionKey`, neue Einträge für jeden Datensatz in der Delta-Synchronisierungstabelle erstellt, insbesondere in den Spalten `gsi_ds_pk` und`gsi_ds_sk`. Weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch. Dies ist ein optionales Feld.

Das aus DynamoDB gelöschte Element wird automatisch in primitive GraphQL- und JSON-Typen konvertiert und ist im Kontext result () verfügbar. `context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu JavaScript Resolvern finden Sie unter JavaScript Übersicht über Resolver.

Beispiel 1

Das folgende Beispiel ist ein Funktionsanforderungshandler für eine GraphQL-Mutation`deleteItem(id: ID!)`. Wenn ein Element mit dieser ID vorhanden ist, wird es gelöscht.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'DeleteItem',
    key: util.dynamodb.toMapValues({ id: ctx.args.id }),
  };
}
```

Beispiel 2

Das folgende Beispiel ist ein Funktionsanforderungshandler für eine GraphQL-Mutation`deleteItem(id: ID!, expectedVersion: Int!)`. Wenn ein Element mit dieser ID vorhanden ist, wird es nur dann gelöscht, wenn das `version`-Feld auf `expectedVersion` gesetzt ist:

```
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),
  };
}
```

Weitere Informationen zur DynamoDB `DeleteItem`-API finden Sie in der DynamoDB API-Dokumentation.

Abfrage

Mit dem `Query` Anforderungsobjekt können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `Query` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Schlüsselausdruck
- Welcher Index verwendet werden soll
- Jeder zusätzliche Filter
- Wie viele Elemente zurückgegeben werden sollen
- Ob Consistent-Lesevorgänge verwendet werden sollen
- Abfragerichtung (vorwärts oder rückwärts)
- Paginierungs-Token

Das `Query` Anforderungsobjekt hat die folgende Struktur:

```
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 };
  };
};
```

Die Felder sind wie folgt definiert:

Felder abfragen

Liste der Abfragefelder

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `Query`-DynamoDB-Operation durchzuführen, muss diese auf `Query` gesetzt sein. Dieser Wert ist erforderlich.

`query`

`query`In diesem Abschnitt können Sie einen Schlüsselbedingungsausdruck angeben, der beschreibt, welche Elemente aus DynamoDB abgerufen werden sollen. Weitere Informationen zum Schreiben von Ausdrücken für Schlüsselbedingungen finden Sie in der DynamoDB-Dokumentation KeyConditions . Dieser Abschnitt muss angegeben werden.

`expression`

Der Abfrageausdruck. Dieses Feld muss angegeben werden.

`expressionNames`

Die Ersetzungen für Platzhalter der Namen von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter`expression`, der in der verwendet wird, und der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Namen von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

`expressionValues`

Die Ersetzungen für Platzhalter der Werte von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Wertplatzhalter, der im `expression` verwendet wird, und der Wert muss ein typisierter Wert sein. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Werte von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

`filter`

Ein zusätzlicher Filter, der für das Filtern der Ergebnisse von DynamoDB verwendet werden kann, bevor sie zurückgegeben werden. Weitere Informationen zu Filtern finden Sie unter Filter. Dies ist ein optionales Feld.

`index`

Der Name des abzufragenden Indexes. Der DynamoDB-Abfragevorgang ermöglicht es Ihnen, zusätzlich zum Primärschlüsselindex auch lokale sekundäre Indizes und globale sekundäre Indizes nach einem Hashschlüssel zu durchsuchen. Falls angegeben, weist dies DynamoDB an, den angegebenen Index abzufragen. Wenn nicht angegeben, wird der Primärschlüsselindex abgefragt.

`nextToken`

Das Paginierungs-Token für die Fortsetzung einer früheren Abfrage. Dieses wäre von einer vorherigen Abfrage erhalten worden. Dies ist ein optionales Feld.

`limit`

Die maximale Anzahl der auszuwertenden Elemente (nicht notwendigerweise die Anzahl der übereinstimmenden Elemente). Dies ist ein optionales Feld.

`scanIndexForward`

Ein boolescher Wert, der angibt, ob vorwärts oder rückwärts abgefragt werden soll. Dieses Feld ist optional und standardmäßig auf `true` gesetzt.

`consistentRead`

Ein boolescher Wert, der angibt, ob bei der Abfrage von DynamoDB konsistente Lesevorgänge verwendet werden sollen. Dieses Feld ist optional und standardmäßig auf `false` gesetzt.

`select`

Standardmäßig gibt der AWS AppSync DynamoDB-Resolver nur Attribute zurück, die in den Index projiziert werden. Wenn weitere Attribute erforderlich sind, kann dieses Feld festgelegt werden. Dies ist ein optionales Feld. Die unterstützten Werte sind:

`ALL_ATTRIBUTES`

Gibt alle Elementattribute aus der angegebenen Tabelle oder dem Index zurück. Wenn Sie einen lokalen sekundären Index abfragen, ruft DynamoDB für jedes übereinstimmende Element im Index das gesamte Element aus der übergeordneten Tabelle ab. Wenn der Index

konfiguriert ist, um alle Elementattribute zu projizieren, können Sie alle Daten aus dem lokalen sekundären Index erhalten und das Abrufen ist nicht erforderlich.

`ALL_PROJECTED_ATTRIBUTES`

Nur bei der Abfrage eines Indexes erlaubt. Ruft alle Attribute ab, die in den Index projiziert wurden. Wenn der Index konfiguriert ist, um alle Attribute zu projizieren, entspricht dieser Rückgabewert der Angabe von `ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Gibt nur die in den s aufgeführten Attribute zurück. `projection expression` Dieser Rückgabewert entspricht der Angabe `projection` von 's' `expression` ohne Angabe eines Werts für`Select`.

`projection`

Eine Projektion, die verwendet wird, um die Attribute anzugeben, die von der DynamoDB-Operation zurückgegeben werden sollen. Weitere Informationen zu Projektionen finden Sie unter Projektionen. Dies ist ein optionales Feld.

Die Ergebnisse von DynamoDB werden automatisch in primitive GraphQL- und JSON-Typen konvertiert und sind im Kontext result () verfügbar. `context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu JavaScript Resolvern finden Sie unter JavaScript Übersicht über Resolver.

Die Ergebnisse weisen die folgende Struktur auf:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Die Felder sind wie folgt definiert:

`items`

Eine Liste mit den von der DynamoDB-Abfrage zurückgegebenen Elementen.

`nextToken`

Wenn es mehr Ergebnisse geben könnte, enthält `nextToken` ein Paginierungs-Token, das in einer anderen Anfrage verwendet werden kann. Beachten Sie, dass das von DynamoDB zurückgegebene Paginierungstoken AWS AppSync verschlüsselt und verschleiert wird. Damit sickern Daten aus Ihren Tabellen nicht versehentlich an den Aufrufer durch. Beachten Sie außerdem, dass diese Paginierungstoken nicht für verschiedene Funktionen oder Resolver verwendet werden können.

`scannedCount`

Die Anzahl der Elemente, die mit dem Abfragebedingungsausdruck übereinstimmten, bevor ein Filterausdruck (falls vorhanden) angewendet wurde.

Beispiel

Das folgende Beispiel ist ein Funktionsanforderungshandler für eine GraphQL-Abfrage`getPosts(owner: ID!)`.

In diesem Beispiel wird ein globaler sekundärer Index in einer Tabelle abgefragt, um alle Beiträge im Besitz der angegebenen ID zurückzugeben.

```
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',
  };
}
```

Weitere Informationen zur DynamoDB `Query`-API finden Sie in der DynamoDB API-Dokumentation.

Scan

Mit der `Scan` Anfrage können Sie die AWS AppSync DynamoDB-Funktion anweisen, eine `Scan` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Ein Filter zum Ausschließen von Ergebnissen
- Welcher Index verwendet werden soll
- Wie viele Elemente zurückgegeben werden sollen
- Ob Consistent-Lesevorgänge verwendet werden sollen
- Paginierungs-Token
- Parallele Scans

Das `Scan` Anforderungsobjekt hat die folgende Struktur:

```
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 };
  };
};
```

Die Felder sind wie folgt definiert:

Felder scannen

Liste der Felder scannen

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `Scan`-DynamoDB-Operation durchzuführen, muss diese auf `Scan` gesetzt sein. Dieser Wert ist erforderlich.

`filter`

Ein Filter, mit dem die Ergebnisse aus DynamoDB gefiltert werden können, bevor sie zurückgegeben werden. Weitere Informationen zu Filtern finden Sie unter Filter. Dies ist ein optionales Feld.

`index`

Der Name des abzufragenden Indexes. Der DynamoDB-Abfragevorgang ermöglicht es Ihnen, zusätzlich zum Primärschlüsselindex auch lokale sekundäre Indizes und globale sekundäre Indizes nach einem Hashschlüssel zu durchsuchen. Falls angegeben, weist dies DynamoDB an, den angegebenen Index abzufragen. Wenn nicht angegeben, wird der Primärschlüsselindex abgefragt.

`limit`

Die maximale Anzahl der zu einem bestimmten Zeitpunkt auszuwertenden Elemente. Dies ist ein optionales Feld.

`consistentRead`

Ein boolescher Wert, der angibt, ob bei der Abfrage von DynamoDB konsistente Lesevorgänge verwendet werden sollen. Dieses Feld ist optional und standardmäßig auf `false` gesetzt.

`nextToken`

Das Paginierungs-Token für die Fortsetzung einer früheren Abfrage. Dieses wäre von einer vorherigen Abfrage erhalten worden. Dies ist ein optionales Feld.

`select`

Standardmäßig gibt die AWS AppSync DynamoDB-Funktion nur die Attribute zurück, die in den Index projiziert werden. Wenn weitere Attribute erforderlich sind, kann dieses Feld festgelegt werden. Dies ist ein optionales Feld. Die unterstützten Werte sind:

`ALL_ATTRIBUTES`

Gibt alle Elementattribute aus der angegebenen Tabelle oder dem Index zurück. Wenn Sie einen lokalen sekundären Index abfragen, ruft DynamoDB für jedes übereinstimmende Element im Index das gesamte Element aus der übergeordneten Tabelle ab. Wenn der Index konfiguriert ist, um alle Elementattribute zu projizieren, können Sie alle Daten aus dem lokalen sekundären Index erhalten und das Abrufen ist nicht erforderlich.

`ALL_PROJECTED_ATTRIBUTES`

Nur bei der Abfrage eines Indexes erlaubt. Ruft alle Attribute ab, die in den Index projiziert wurden. Wenn der Index konfiguriert ist, um alle Attribute zu projizieren, entspricht dieser Rückgabewert der Angabe von `ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Gibt nur die in den s aufgeführten Attribute zurück. `projection expression` Dieser Rückgabewert entspricht der Angabe `projection` von 's' `expression` ohne Angabe eines Werts für`Select`.

`totalSegments`

Die Anzahl der Segmente zum Partitionieren der Tabelle, wenn ein paralleler Scan durchgeführt wird. Dieses Feld ist optional, muss aber angegeben werden, wenn `segment` angegeben ist.

`segment`

Das Tabellensegment in dieser Operation beim Durchführen eines parallelen Scans. Dieses Feld ist optional, muss aber angegeben werden, wenn `totalSegments` angegeben ist.

`projection`

Eine Projektion, die verwendet wird, um die Attribute anzugeben, die von der DynamoDB-Operation zurückgegeben werden sollen. Weitere Informationen zu Projektionen finden Sie unter Projektionen. Dies ist ein optionales Feld.

Die vom DynamoDB-Scan zurückgegebenen Ergebnisse werden automatisch in primitive GraphQL- und JSON-Typen konvertiert und sind im Kontext result () verfügbar. `context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu JavaScript Resolvern finden Sie unter JavaScript Übersicht über Resolver.

Die Ergebnisse weisen die folgende Struktur auf:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Die Felder sind wie folgt definiert:

`items`

Eine Liste mit den vom DynamoDB-Scan zurückgegebenen Elementen.

`nextToken`

Falls es mehr Ergebnisse gibt, `nextToken` enthält sie ein Paginierungstoken, das Sie in einer anderen Anfrage verwenden können. AWS AppSync verschlüsselt und verschleiert das von DynamoDB zurückgegebene Paginierungstoken. Damit sickern Daten aus Ihren Tabellen nicht versehentlich an den Aufrufer durch. Außerdem können diese Paginierungstoken nicht für verschiedene Funktionen oder Resolver verwendet werden.

`scannedCount`

Die Anzahl der Elemente, die von DynamoDB abgerufen wurden, bevor ein Filterausdruck (falls vorhanden) angewendet wurde.

Beispiel 1

Das folgende Beispiel ist ein Funktionsanforderungshandler für die GraphQL-Abfrage:`allPosts`.

In diesem Beispiel werden alle Einträge in der Tabelle zurückgegeben.

```
export function request(ctx) {
  return { operation: 'Scan' };
}
```

Beispiel 2

Das folgende Beispiel ist ein Funktionsanforderungshandler für die GraphQL-Abfrage:`postsMatching(title: String!)`.

In diesem Beispiel werden alle Einträge in der Tabelle zurückgegeben, bei denen der Titel mit dem `title`-Argument beginnt.

```
export function request(ctx) {
  const { title } = ctx.args;
  const filter = { filter: { beginsWith: title } };
  return {
    operation: 'Scan',
    filter: JSON.parse(util.transform.toDynamoDBFilterExpression(filter)),
  };
}
```

Weitere Informationen zur DynamoDB `Scan`-API finden Sie in der DynamoDB API-Dokumentation.

Synchronisierung

Mit dem `Sync` Anforderungsobjekt können Sie alle Ergebnisse aus einer DynamoDB-Tabelle abrufen und anschließend nur die Daten empfangen, die seit Ihrer letzten Abfrage geändert wurden (die Delta-Updates). `Sync`Anfragen können nur an versionierte DynamoDB-Datenquellen gestellt werden. Sie können folgende Formen angeben:

- Ein Filter zum Ausschließen von Ergebnissen
- Wie viele Elemente zurückgegeben werden sollen
- Paginierungstoken
- Wann Ihre letzte `Sync`-Operation gestartet wurde

Das `Sync` Anforderungsobjekt hat die folgende Struktur:

```
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 };
  };
```

```
};
```

Die Felder sind wie folgt definiert:

Felder synchronisieren

Liste der Felder synchronisieren

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `Sync`--Operation durchzuführen, muss diese auf `Sync` gesetzt sein. Dieser Wert ist erforderlich.

`filter`

Ein Filter, mit dem die Ergebnisse aus DynamoDB gefiltert werden können, bevor sie zurückgegeben werden. Weitere Informationen zu Filtern finden Sie unter Filter. Dies ist ein optionales Feld.

`limit`

Die maximale Anzahl der zu einem bestimmten Zeitpunkt auszuwertenden Elemente. Dies ist ein optionales Feld. Wenn nicht angegeben, wird das Standardlimit auf `100` Elemente festgelegt. Der maximale Wert für dieses Feld ist `1000` Elemente.

`nextToken`

Das Paginierungs-Token für die Fortsetzung einer früheren Abfrage. Dieses wäre von einer vorherigen Abfrage erhalten worden. Dies ist ein optionales Feld.

`lastSync`

Der Moment, in Epochenmillisekunden, an dem die letzte erfolgreiche `Sync`-Operation gestartet wurde. Wenn angegeben, werden nur Elemente zurückgegeben, die sich nach `lastSync` geändert haben. Dieses Feld ist optional und sollte nur ausgefüllt werden, nachdem alle Seiten von einem ersten `Sync`-Vorgang abgerufen wurden. Wenn nicht angegeben, werden Ergebnisse aus der Basis-Tabelle zurückgegeben, andernfalls werden Ergebnisse aus der Delta-Tabelle zurückgegeben.

`basePartitionKey`

Der Partitionsschlüssel der Basistabelle, der bei der Ausführung eines `Sync` Vorgangs verwendet wird. In diesem Feld kann eine `Sync` Operation ausgeführt werden, wenn die Tabelle einen benutzerdefinierten Partitionsschlüssel verwendet. Dies ist ein optionales Feld.

`deltaIndexName`

Der für die `Sync` Operation verwendete Index. Dieser Index ist erforderlich, um einen `Sync` Vorgang für die gesamte Delta-Store-Tabelle zu aktivieren, wenn die Tabelle einen benutzerdefinierten Partitionsschlüssel verwendet. Die `Sync` Operation wird auf der GSI (erstellt am `gsi_ds_pk` und`gsi_ds_sk`) ausgeführt. Dies ist ein optionales Feld.

Die von der DynamoDB-Synchronisierung zurückgegebenen Ergebnisse werden automatisch in primitive GraphQL- und JSON-Typen konvertiert und sind im Kontext result () verfügbar. `context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu JavaScript Resolvern finden Sie unter JavaScript Übersicht über Resolver.

Die Ergebnisse weisen die folgende Struktur auf:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10,
    startedAt = 1550000000000
}
```

Die Felder sind wie folgt definiert:

`items`

Eine Liste mit den Elementen, die von der Synchronisierung zurückgegeben wurden.

`nextToken`

Falls es möglicherweise mehr Ergebnisse gibt, `nextToken` enthält es ein Paginierungstoken, das Sie in einer anderen Anfrage verwenden können. AWS AppSync verschlüsselt und verschleiert das von DynamoDB zurückgegebene Paginierungstoken. Damit sickern Daten aus Ihren Tabellen nicht versehentlich an den Aufrufer durch. Außerdem können diese Paginierungstoken nicht für verschiedene Funktionen oder Resolver verwendet werden.

`scannedCount`

Die Anzahl der Elemente, die von DynamoDB abgerufen wurden, bevor ein Filterausdruck (falls vorhanden) angewendet wurde.

`startedAt`

Der Moment, in Epochenmillisekunden, an dem der Synchronisierungsvorgang gestartet wurde, den Sie lokal speichern und in einer anderen Anforderung als `lastSync`-Argument verwenden können. Wenn ein Paginierungstoken in der Anforderung enthalten war, entspricht dieser Wert dem Wert, der von der Anforderung für die erste Ergebnisseite zurückgegeben wird.

Beispiel

Das folgende Beispiel ist ein Funktionsanforderungshandler für die GraphQL-Abfrage:`syncPosts(nextToken: String, lastSync: AWSTimestamp)`.

Wenn in diesem Beispiel `lastSync` weggelassen wird, werden alle Einträge in der Basistabelle zurückgegeben. Wenn `lastSync` angegeben wird, werden nur die Einträge in der Delta-Synchronisationstabelle zurückgegeben, die sich seit `lastSync` geändert haben.

```
export function request(ctx) {
  const { nextToken, lastSync } = ctx.args;
  return { operation: 'Sync', limit: 100, nextToken, lastSync };
}
```

BatchGetItem

Mit dem `BatchGetItem` Anforderungsobjekt können Sie die AWS AppSync DynamoDB-Funktion anweisen, eine `BatchGetItem` Anforderung an DynamoDB zu stellen, um mehrere Elemente abzurufen, möglicherweise über mehrere Tabellen hinweg. Für dieses Anforderungsobjekt müssen Sie Folgendes angeben:

- Die Namen der Tabellen, aus denen die Elemente abgerufen werden sollen.
- Die Schlüssel der Elemente, die aus den einzelnen Tabellen abgerufen werden sollen.

Es gelten die DynamoDB `BatchGetItem`-Grenzwerte, und es kann kein Bedingungsausdruck bereitgestellt werden.

Das `BatchGetItem` Anforderungsobjekt hat die folgende Struktur:

```
type DynamoDBBatchGetItemRequest = {
  operation: 'BatchGetItem';
  tables: {
    [tableName: string]: {
      keys: { [key: string]: any }[];
      consistentRead?: boolean;
      projection?: {
        expression: string;
        expressionNames?: { [key: string]: string };
      };
    };
  };
};
```

Die Felder sind wie folgt definiert:

BatchGetItem Felder

BatchGetItem Liste der Felder

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `BatchGetItem`-DynamoDB-Operation durchzuführen, muss diese auf `BatchGetItem` gesetzt sein. Dieser Wert ist erforderlich.

`tables`

Die DynamoDB-Tabellen, aus denen die Elemente abgerufen werden sollen. Der Wert ist eine Zuweisung, in der Tabellennamen als Schlüssel der Zuweisung angegeben werden. Mindestens eine Tabelle muss angegeben werden. Dieser `tables`-Wert ist erforderlich.

`keys`

Liste der DynamoDB-Schlüssel, die den Primärschlüssel der abzurufenden Elemente darstellen. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung).

`consistentRead`

Ob bei der Ausführung eines GetItemVorgangs ein konsistenter Lesevorgang verwendet werden soll. Dieser Wert ist optional und ist standardmäßig false.

`projection`

Eine Projektion, die verwendet wird, um die Attribute anzugeben, die von der DynamoDB-Operation zurückgegeben werden sollen. Weitere Informationen zu Projektionen finden Sie unter Projektionen. Dies ist ein optionales Feld.

Beachten Sie Folgendes:

- Wenn ein Element nicht aus der Tabelle abgerufen wurde, enthält der Datenblock für diese Tabelle ein null-Element.
- Die Aufrufergebnisse werden nach Tabelle sortiert, basierend auf der Reihenfolge, in der sie innerhalb des Anforderungsobjekts bereitgestellt wurden.
- Jeder `Get` Befehl innerhalb von a `BatchGetItem` ist atomar, ein Batch kann jedoch teilweise verarbeitet werden. Wenn ein Stapel aufgrund eines Fehlers teilweise verarbeitet wird, werden die nicht verarbeiteten Schlüssel als Teil des Aufrufergebnisses im Block unprocessedKeys zurückgegeben.
- `BatchGetItem` ist auf 100 Schlüssel beschränkt.

Für das folgende Beispiel für einen Funktionsanforderungshandler:

```
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 })],
    },
  };
}
```

Das in `ctx.result` verfügbare Aufrufergebnis sieht wie folgt aus:

```
{
   "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": []
   }
}
```

`ctx.error` enthält die Einzelheiten zu dem Fehler. Die Schlüssel data, unprocessedKeys und jeder Tabellenschlüssel, der im Ergebnis im Funktionsanforderungsobjekt bereitgestellt wurde, sind garantiert im Aufrufergebnis vorhanden. Elemente, die gelöscht wurden, befinden sich im Block data. Elemente, die nicht verarbeitet wurden, werden im Datenblock mit null markiert und im Block unprocessedKeys platziert.

BatchDeleteItem

Mit dem `BatchDeleteItem` Anforderungsobjekt können Sie die AWS AppSync DynamoDB-Funktion anweisen, eine `BatchWriteItem` Anforderung an DynamoDB zu stellen, um mehrere Elemente zu löschen, möglicherweise über mehrere Tabellen hinweg. Für dieses Anforderungsobjekt müssen Sie Folgendes angeben:

- Die Namen der Tabellen, aus denen die Elemente gelöscht werden sollen.
- Die Schlüssel der Elemente, die aus den einzelnen Tabellen gelöscht werden sollen.

Es gelten die DynamoDB `BatchWriteItem`-Grenzwerte, und es kann kein Bedingungsausdruck bereitgestellt werden.

Das `BatchDeleteItem` Anforderungsobjekt hat die folgende Struktur:

```
type DynamoDBBatchDeleteItemRequest = {
  operation: 'BatchDeleteItem';
  tables: {
    [tableName: string]: { [key: string]: any }[];
  };
};
```

Die Felder sind wie folgt definiert:

BatchDeleteItem Felder

BatchDeleteItem Liste der Felder

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `BatchDeleteItem`-DynamoDB-Operation durchzuführen, muss diese auf `BatchDeleteItem` gesetzt sein. Dieser Wert ist erforderlich.

`tables`

Die DynamoDB-Tabellen, aus denen die Elemente gelöscht werden sollen. Jede Tabelle ist eine Liste von DynamoDB-Schlüsseln, die den Primärschlüssel der zu löschenden Elemente darstellen. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Mindestens eine Tabelle muss angegeben werden. Der `tables` Wert ist erforderlich.

Beachten Sie Folgendes:

- Im Gegensatz zum `DeleteItem`-Vorgang wird nicht das vollständig gelöschte Element in der Antwort zurückgegeben. Nur der übergebene Schlüssel wird zurückgegeben.
- Wenn ein Element nicht aus der Tabelle gelöscht wurde, enthält der Datenblock für diese Tabelle ein null-Element.
- Die Aufrufergebnisse werden nach Tabelle sortiert, basierend auf der Reihenfolge, in der sie innerhalb des Anforderungsobjekts bereitgestellt wurden.
- Jeder `Delete` Befehl innerhalb von a `BatchDeleteItem` ist atomar. Ein Stapel kann jedoch teilweise verarbeitet werden. Wenn ein Stapel aufgrund eines Fehlers teilweise verarbeitet wird, werden die nicht verarbeiteten Schlüssel als Teil des Aufrufergebnisses im Block unprocessedKeys zurückgegeben.

- `BatchDeleteItem` ist auf 25 Schlüssel beschränkt.
- Dieser Vorgang wird nicht unterstützt, wenn er zusammen mit der Konflikterkennung verwendet wird. Wenn Sie beide gleichzeitig verwenden, kann dies zu einem Fehler führen.

Für das folgende Beispiel für einen Funktionsanforderungshandler:

```
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 })],
    },
  };
}
```

Das in `ctx.result` verfügbare Aufrufergebnis sieht wie folgt aus:

```
{
   "data": {
     "authors": [null],
     "posts": [
        // Was deleted
        {
          "authorId": "a1",
          "postId": "p2"
        }
     ]
   },
   "unprocessedKeys": {
     "authors": [
        // This key was not processed due to an error
        {
          "authorId": "a1"
        }
      ],
     "posts": []
   }
```

```
}
```

`ctx.error` enthält die Einzelheiten zu dem Fehler. Die Schlüssel data, unprocessedKeys und jeder Tabellenschlüssel, der im Funktionsanforderungsobjekt bereitgestellt wurde, sind garantiert im Aufrufergebnis vorhanden. Elemente, die gelöscht wurden, befinden sich im Block data. Elemente, die nicht verarbeitet wurden, werden im Datenblock mit null markiert und im Block unprocessedKeys platziert.

BatchPutItem

Mit dem `BatchPutItem` Anforderungsobjekt können Sie die AWS AppSync DynamoDB-Funktion anweisen, eine `BatchWriteItem` Anforderung an DynamoDB zu stellen, um mehrere Elemente zu platzieren, möglicherweise in mehreren Tabellen. Für dieses Anforderungsobjekt müssen Sie Folgendes angeben:

- Die Namen der Tabellen, in denen die Elemente abgelegt werden sollen.
- Die vollständigen Elemente, die in jeder Tabelle abgelegt werden sollen

Es gelten die DynamoDB `BatchWriteItem`-Grenzwerte, und es kann kein Bedingungsausdruck bereitgestellt werden.

Das `BatchPutItem` Anforderungsobjekt hat die folgende Struktur:

```
type DynamoDBBatchPutItemRequest = {
  operation: 'BatchPutItem';
  tables: {
    [tableName: string]: { [key: string]: any}[];
  };
};
```

Die Felder sind wie folgt definiert:

BatchPutItem Felder

BatchPutItem Liste der Felder

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `BatchPutItem`-DynamoDB-Operation durchzuführen, muss diese auf `BatchPutItem` gesetzt sein. Dieser Wert ist erforderlich.

`tables`

Die DynamoDB-Tabellen, in die die Elemente eingefügt werden sollen. Jeder Tabelleneintrag stellt eine Liste von DynamoDB-Elementen dar, die für diese spezifische Tabelle eingefügt werden sollen. Mindestens eine Tabelle muss angegeben werden. Dieser Wert ist erforderlich.

Beachten Sie Folgendes:

- Bei Erfolg werden die vollständig eingefügten Elemente in der Antwort zurückgegeben.
- Wenn ein Element nicht in die Tabelle eingefügt wurde, wird im Datenblock für diese Tabelle ein null-Element angezeigt.
- Die eingefügten Elemente werden nach Tabelle sortiert, basierend auf der Reihenfolge, in der sie im Anforderungsobjekt bereitgestellt wurden.
- Jeder `Put` Befehl innerhalb von a `BatchPutItem` ist atomar, ein Batch kann jedoch teilweise verarbeitet werden. Wenn ein Stapel aufgrund eines Fehlers teilweise verarbeitet wird, werden die nicht verarbeiteten Schlüssel als Teil des Aufrufergebnisses im Block unprocessedKeys zurückgegeben.
- `BatchPutItem` ist auf 25 Elemente beschränkt.
- Dieser Vorgang wird nicht unterstützt, wenn er zusammen mit der Konflikterkennung verwendet wird. Wenn Sie beide gleichzeitig verwenden, kann dies zu einem Fehler führen.

Für das folgende Beispiel für einen Funktionsanforderungshandler:

```
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 })],
    },
  };
}
```

Das in `ctx.result` verfügbare Aufrufergebnis sieht wie folgt aus:

```
{
   "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": []
   }
}
```

`ctx.error` enthält die Einzelheiten zu dem Fehler. Die Schlüssel data, unprocessedItems und jeder Tabellenschlüssel, der im Anforderungsobjekt bereitgestellt wurde, sind garantiert im Aufrufergebnis vorhanden. Elemente, die eingefügt wurden, befinden sich im Block data. Elemente, die nicht verarbeitet wurden, werden im Datenblock mit null markiert und im Block unprocessedItems platziert.

TransactGetItems

Mit dem `TransactGetItems` Anforderungsobjekt können Sie die AWS AppSync DynamoDB-Funktion anweisen, eine `TransactGetItems` Anforderung an DynamoDB zu stellen, um mehrere Elemente abzurufen, möglicherweise über mehrere Tabellen hinweg. Für dieses Anforderungsobjekt müssen Sie Folgendes angeben:

- Der Tabellenname jedes Anforderungselements, von dem das Element abgerufen werden soll
- Der Schlüssel jedes Anforderungselements, das aus jeder Tabelle abgerufen werden soll

Es gelten die DynamoDB `TransactGetItems`-Grenzwerte, und es kann kein Bedingungsausdruck bereitgestellt werden.

Das `TransactGetItems` Anforderungsobjekt hat die folgende Struktur:

```
type DynamoDBTransactGetItemsRequest = {
  operation: 'TransactGetItems';
  transactItems: { table: string; key: { [key: string]: any }; projection?:
 { expression: string; expressionNames?: { [key: string]: string }; }[];
  };
};
```

Die Felder sind wie folgt definiert:

TransactGetItems Felder

TransactGetItems Liste der Felder

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `TransactGetItems`-DynamoDB-Operation durchzuführen, muss diese auf `TransactGetItems` gesetzt sein. Dieser Wert ist erforderlich.

`transactItems`

Die einzuschließenden Anforderungselemente. Der Wert ist ein Array von Anforderungselementen. Es muss mindestens ein Anforderungselement angegeben werden. Dieser `transactItems`-Wert ist erforderlich.

`table`

Die DynamoDB-Tabelle, aus der das Element abgerufen werden soll. Der Wert ist eine Zeichenfolge des Tabellennamens. Dieser `table`-Wert ist erforderlich.

`key`

Der DynamoDB-Schlüssel, der den Primärschlüssel des abzurufenden Elements darstellt. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung).

`projection`

Eine Projektion, die verwendet wird, um die Attribute anzugeben, die von der DynamoDB-Operation zurückgegeben werden sollen. Weitere Informationen zu Projektionen finden Sie unter Projektionen. Dies ist ein optionales Feld.

Beachten Sie Folgendes:

- Wenn eine Transaktion erfolgreich ist, entspricht die Reihenfolge der abgerufenen Elemente im `items`-Block der Reihenfolge der Anforderungselemente.
- Transaktionen werden in irgendeiner all-or-nothing Weise ausgeführt. Wenn ein Anforderungselement einen Fehler verursacht, wird die gesamte Transaktion nicht ausgeführt, und Fehlerdetails werden zurückgegeben.
- Ein Anforderungselement, das nicht abgerufen werden kann, ist kein Fehler. Stattdessen erscheint ein Null-Element im Elemente-Block an der entsprechenden Position.
- Wenn der Fehler einer Transaktion lautet TransactionCanceledException, wird der `cancellationReasons` Block aufgefüllt. Die Reihenfolge der Stornierungsgründe im `cancellationReasons`-Block ist die gleiche wie die Reihenfolge der Anforderungselemente.
- `TransactGetItems`ist auf 100 Anfrageelemente begrenzt.

Für das folgende Beispiel für einen Funktionsanforderungshandler:

```
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 }),
      },
    ],
  };
```

```
}
```

Wenn die Transaktion erfolgreich ist und nur das erste angeforderte Element abgerufen wird, ist das Aufrufergebnis in `ctx.result` wie folgt verfügbar:

```
{
    "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
}
```

Wenn die Transaktion TransactionCanceledExceptionaufgrund des ersten Anforderungselements fehlschlägt, `ctx.result` ist das Aufrufergebnis wie folgt verfügbar:

```
{
    "items": null,
    "cancellationReasons": [
       {
           "type":"Sample error type",
           "message":"Sample error message"
       },
       {
           "type":"None",
           "message":"None"
       }
    ]
}
```

`ctx.error` enthält die Einzelheiten zu dem Fehler. Die Schlüssel-Elemente und Stornierungsgründe sind garantiert in `ctx.result` vorhanden.

TransactWriteItems

Mit dem `TransactWriteItems` Anforderungsobjekt können Sie die AWS AppSync DynamoDB-Funktion anweisen, eine `TransactWriteItems` Anforderung an DynamoDB zu stellen, um mehrere Elemente zu schreiben, möglicherweise in mehrere Tabellen. Für dieses Anforderungsobjekt müssen Sie Folgendes angeben:

- Der Name der Zieltabelle jedes Anforderungselements
- Der Vorgang jedes auszuführenden Anforderungselements. Es gibt vier Arten von Operationen, die unterstützt werden: PutItemUpdateItem, DeleteItem, und ConditionCheck
- Der Schlüssel jedes zu schreibenden Anforderungselements

Es gelten die DynamoDB `TransactWriteItems`-Grenzwerte.

Das `TransactWriteItems` Anforderungsobjekt hat die folgende Struktur:

```
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;
};
```

TransactWriteItems Felder

TransactWriteItems Liste der Felder

Die Felder sind wie folgt definiert:

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `TransactWriteItems`-DynamoDB-Operation durchzuführen, muss diese auf `TransactWriteItems` gesetzt sein. Dieser Wert ist erforderlich.

`transactItems`

Die einzuschließenden Anforderungselemente. Der Wert ist ein Array von Anforderungselementen. Es muss mindestens ein Anforderungselement angegeben werden. Dieser `transactItems`-Wert ist erforderlich.

Für `PutItem` sind die Felder wie folgt definiert:

`table`

Die DynamoDB-Zieltabelle. Der Wert ist eine Zeichenfolge des Tabellennamens. Dieser `table`-Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `PutItem`-DynamoDB-Operation durchzuführen, muss diese auf `PutItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der DynamoDB-Schlüssel, der den Primärschlüssel des einzufügenden Elements darstellt. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`attributeValues`

Der Rest der Attribute des Elements, die in DynamoDB gespeichert werden sollen. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dies ist ein optionales Feld.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, überschreibt die `PutItem`-Anforderung alle vorhandenen Einträge für dieses Element. Sie können angeben, ob das vorhandene Element zurückgerufen werden soll, wenn die Bedingungsprüfung fehlschlägt. Weitere Informationen zu Transaktionsbedingungen finden Sie unter Ausdrücke für Transaktionsbedingungen. Dieser Wert ist optional.

Für `UpdateItem` sind die Felder wie folgt definiert:

`table`

Die zu aktualisierende DynamoDB-Tabelle. Der Wert ist eine Zeichenfolge des Tabellennamens. Dieser `table`-Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `UpdateItem`-DynamoDB-Operation durchzuführen, muss diese auf `UpdateItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der DynamoDB-Schlüssel, der den Primärschlüssel des zu aktualisierenden Elements darstellt. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen

Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`update`

`update`In diesem Abschnitt können Sie einen Aktualisierungsausdruck angeben, der beschreibt, wie das Element in DynamoDB aktualisiert wird. Weitere Informationen zum Schreiben von Aktualisierungsausdrücken finden Sie in der DynamoDB-Dokumentation UpdateExpressions . Dieser Abschnitt ist erforderlich.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, aktualisiert die `UpdateItem`-Anforderung jeden vorhandenen Eintrag unabhängig vom aktuellen Status. Sie können angeben, ob das vorhandene Element zurückgerufen werden soll, wenn die Bedingungsprüfung fehlschlägt. Weitere Informationen zu Transaktionsbedingungen finden Sie unter Ausdrücke für Transaktionsbedingungen. Dieser Wert ist optional.

Für `DeleteItem` sind die Felder wie folgt definiert:

`table`

Die DynamoDB-Tabelle, in der das Element gelöscht werden soll. Der Wert ist eine Zeichenfolge des Tabellennamens. Dieser `table`-Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `DeleteItem`-DynamoDB-Operation durchzuführen, muss diese auf `DeleteItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der DynamoDB-Schlüssel, der den Primärschlüssel des zu löschenden Elements darstellt. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, löscht die `DeleteItem`-Anforderung ein Element unabhängig vom aktuellen Status. Sie können angeben, ob das vorhandene Element zurückgerufen werden soll, wenn die Bedingungsprüfung fehlschlägt. Weitere Informationen zu Transaktionsbedingungen finden Sie unter Ausdrücke für Transaktionsbedingungen. Dieser Wert ist optional.

Für `ConditionCheck` sind die Felder wie folgt definiert:

`table`

Die DynamoDB-Tabelle, in der der Zustand überprüft werden soll. Der Wert ist eine Zeichenfolge des Tabellennamens. Dieser `table`-Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `ConditionCheck`-DynamoDB-Operation durchzuführen, muss diese auf `ConditionCheck` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der DynamoDB-Schlüssel, der den Primärschlüssel des Artikels zur Zustandsprüfung darstellt. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Sie können angeben, ob das vorhandene Element zurückgerufen werden soll, wenn die Bedingungsprüfung fehlschlägt. Weitere Informationen zu Transaktionsbedingungen finden Sie unter Ausdrücke für Transaktionsbedingungen. Dieser Wert ist erforderlich.

Beachten Sie Folgendes:

- Nur Schlüssel von Anforderungselementen werden in der Antwort zurückgegeben, wenn diese erfolgreich ist. Die Reihenfolge der Schlüssel entspricht der Reihenfolge der Anforderungselemente.
- Transaktionen werden auf irgendeine Weise ausgeführt. all-or-nothing Wenn ein Anforderungselement einen Fehler verursacht, wird die gesamte Transaktion nicht ausgeführt, und Fehlerdetails werden zurückgegeben.
- Es können keine zwei Anforderungselemente auf dasselbe Element ausgerichtet werden. Andernfalls führen sie zu TransactionCanceledExceptionFehlern.
- Wenn der Fehler einer Transaktion vorliegt TransactionCanceledException, wird der `cancellationReasons` Block aufgefüllt. Wenn die Bedingungsprüfung eines Anforderungselements fehlschlägt und Sie nicht `returnValuesOnConditionCheckFailure` als `false` angegeben haben, wird das in der Tabelle vorhandene Element `item` an der entsprechenden Position des `cancellationReasons`-Blocks abgerufen und gespeichert.
- `TransactWriteItems`ist auf 100 Anfrageelemente begrenzt.
- Dieser Vorgang wird nicht unterstützt, wenn er zusammen mit der Konflikterkennung verwendet wird. Wenn Sie beide gleichzeitig verwenden, kann dies zu einem Fehler führen.

Für das folgende Beispiel für einen Funktionsanforderungshandler:

```
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 }),
        },
      },
    ],
  };
}
```

Wenn die Transaktion erfolgreich ist, ist das Aufrufergebnis in `ctx.result` wie folgt verfügbar:

```
{
    "keys": [
       // Key of the PutItem request
       {
           "post_id": "p1",
       },
       // Key of the UpdateItem request
       {
           "author_id": "a1"
       }
    ],
    "cancellationReasons": null
}
```

Wenn die Transaktion aufgrund eines Fehlers bei der Zustandsprüfung der `PutItem` Anforderung fehlschlägt, `ctx.result` ist das Aufrufergebnis wie folgt verfügbar:

```
{
    "keys": null,
    "cancellationReasons": [
       {
           "item": {
               "post_id": "p1",
               "post_title": "Actual old title",
               "post_description": "Old description"
           },
           "type": "ConditionCheckFailed",
           "message": "The condition check failed."
       },
       {
           "type": "None",
```

```
            "message": "None"
        }
    ]
}
```

`ctx.error` enthält die Einzelheiten zu dem Fehler. Die Schlüssel keys und cancellationReasons sind garantiert in `ctx.result` vorhanden.

Geben Sie System ein (Zuordnung der Anfrage)

Wenn Sie die AWS AppSync DynamoDB-Funktion zum Aufrufen Ihrer DynamoDB-Tabellen verwenden, AWS AppSync müssen Sie den Typ der einzelnen Werte kennen, die in diesem Aufruf verwendet werden sollen. Dies liegt daran, dass DynamoDB mehr Typprimitive unterstützt als GraphQL oder JSON (z. B. Sets und Binärdaten). AWS AppSync benötigt einige Hinweise bei der Übersetzung zwischen GraphQL und DynamoDB, andernfalls müsste es einige Annahmen darüber treffen, wie Daten in Ihrer Tabelle strukturiert sind.

Weitere Informationen zu DynamoDB-Datentypen finden Sie in der Dokumentation zu DynamoDB-Datentypdeskriptoren und Datentypen.

Ein DynamoDB-Wert wird durch ein JSON-Objekt dargestellt, das ein einzelnes Schlüssel-Wert-Paar enthält. Der Schlüssel gibt den DynamoDB-Typ an, und der Wert gibt den Wert selbst an. Im folgenden Beispiel gibt der Schlüssel `S` an, dass der Wert eine Zeichenfolge und der Wert `identifier` selbst ein Zeichenfolgewert ist.

```
{ "S" : "identifier" }
```

Beachten Sie, dass das JSON-Objekt nicht mehr als ein Schlüssel-Wert-Paar haben kann. Wenn mehr als ein Schlüssel-Wert-Paar angegeben ist, wird das Anforderungsobjekt nicht analysiert.

Ein DynamoDB-Wert wird überall in einem Anforderungsobjekt verwendet, wo Sie einen Wert angeben müssen. Zu den Stellen, an denen Sie dies durchführen müssen, gehören: die Abschnitte `key` und `attributeValue` und der Bereich `expressionValues` von Ausdrucksabschnitten. Im folgenden Beispiel `identifier` wird der DynamoDB-Zeichenkettenwert dem `id` Feld in einem `key` Abschnitt (möglicherweise in einem `GetItem` Anforderungsobjekt) zugewiesen.

```
"key" : {
   "id" : { "S" : "identifier" }
```

```
}
```

Unterstützte Typen

AWS AppSync unterstützt die folgenden DynamoDB-Skalar-, Dokument- und Satztypen:

Zeichenfolgetyp **S**

Ein einzelner Zeichenfolgewert. Ein DynamoDB-Zeichenkettenwert wird wie folgt bezeichnet:

```
{ "S" : "some string" }
```

Eine Verwendungsbeispiel ist:

```
"key" : {
   "id" : { "S" : "some string" }
}
```

Zeichenfolgesatztyp **SS**

Ein Satz von Zeichenfolgewerten. Ein DynamoDB String Set-Wert wird wie folgt bezeichnet:

```
{ "SS" : [ "first value", "second value", ... ] }
```

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "phoneNumbers" : { "SS" : [ "+1 555 123 4567", "+1 555 234 5678" ] }
}
```

Zahlentyp **N**

Ein einzelner numerischer Wert. Ein DynamoDB-Zahlenwert wird wie folgt bezeichnet:

```
{ "N" : 1234 }
```

Eine Verwendungsbeispiel ist:

```
"expressionValues" : {
```

```
   ":expectedVersion" : { "N" : 1 }
}
```

Zahlensatztyp **NS**

Ein Satz von Zahlenwerten. Ein DynamoDB Number Set-Wert wird wie folgt bezeichnet:

```
{ "NS" : [ 1, 2.3, 4 ... ] }
```

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "sensorReadings" : { "NS" : [ 67.8, 12.2, 70 ] }
}
```

Binärtyp **B**

Ein Binärwert. Ein DynamoDB-Binärwert wird wie folgt bezeichnet:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

Beachten Sie, dass es sich bei dem Wert tatsächlich um eine Zeichenfolge handelt, wobei die Zeichenfolge die Base64-kodierte Darstellung der Binärdaten ist. AWS AppSync dekodiert diese Zeichenfolge wieder in ihren Binärwert, bevor sie an DynamoDB gesendet wird. AWS AppSync verwendet das Base64-Dekodierungsschema, wie es in RFC 2045 definiert ist: Jedes Zeichen, das nicht im Base64-Alphabet vorkommt, wird ignoriert.

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "binaryMessage" : { "B" : "SGVsbG8sIFdvcmxkIQo=" }
}
```

Binärsatztyp **BS**

Ein Satz von Binärwerten. Ein DynamoDB Binary Set-Wert wird wie folgt bezeichnet:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

Beachten Sie, dass es sich bei dem Wert tatsächlich um eine Zeichenfolge handelt, wobei die Zeichenfolge die Base64-kodierte Darstellung der Binärdaten ist. AWS AppSync dekodiert diese Zeichenfolge wieder in ihren Binärwert, bevor sie an DynamoDB gesendet wird. AWS AppSync verwendet das Base64-Dekodierungsschema, wie es in RFC 2045 definiert ist: Jedes Zeichen, das nicht im Base64-Alphabet vorkommt, wird ignoriert.

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "binaryMessages" : { "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ] }
}
```

Boolescher Typ **BOOL**

Ein Boolescher Wert Ein boolescher DynamoDB-Wert wird wie folgt bezeichnet:

```
{ "BOOL" : true }
```

Beachten Sie, dass nur `true` und `false` gültige Werte sind.

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "orderComplete" : { "BOOL" : false }
}
```

Listentyp **L**

Eine Liste aller anderen unterstützten DynamoDB-Werte. Ein DynamoDB-Listenwert wird wie folgt gekennzeichnet:

```
{ "L" : [ ... ] }
```

Beachten Sie, dass es sich bei dem Wert um einen zusammengesetzten Wert handelt, wobei die Liste null oder mehr aller unterstützten DynamoDB-Werte (einschließlich anderer Listen) enthalten kann. Die Liste kann auch eine Mischung aus verschiedenen Typen enthalten.

Eine Verwendungsbeispiel ist:

```
{ "L" : [
```

```
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

Zuordnungstyp **M**

Stellt eine ungeordnete Sammlung von Schlüssel-Wert-Paaren anderer unterstützter DynamoDB-Werte dar. Ein DynamoDB-Zuordnungswert wird wie folgt bezeichnet:

```
{ "M" : { ... } }
```

Beachten Sie, dass eine Zuordnung null oder mehrere Schlüssel-Wert-Paare enthalten kann. Der Schlüssel muss eine Zeichenfolge sein, und der Wert kann ein beliebiger unterstützter DynamoDB-Wert sein (einschließlich anderer Maps). Die Zuordnung kann auch eine Mischung aus verschiedenen Typen enthalten.

Eine Verwendungsbeispiel ist:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

Null-Typ **NULL**

Ein Null-Wert. Ein DynamoDB-Null-Wert wird wie folgt bezeichnet:

```
{ "NULL" : null }
```

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "phoneNumbers" : { "NULL" : null }
}
```

Weitere Informationen zu den einzelnen Typen finden Sie in der DynamoDB-Dokumentation.

Geben Sie System ein (Antwortzuordnung)

Wenn Sie eine Antwort von DynamoDB erhalten, konvertiert sie AWS AppSync automatisch in primitive GraphQL- und JSON-Typen. Jedes Attribut in DynamoDB wird dekodiert und im Kontext des Antworthandlers zurückgegeben.

Wenn DynamoDB beispielsweise Folgendes zurückgibt:

```
{
    "id" : { "S" : "1234" },
    "name" : { "S" : "Nadia" },
    "age" : { "N" : 25 }
}
```

Wenn das Ergebnis von Ihrem Pipeline-Resolver zurückgegeben wird, AWS AppSync konvertiert es in GraphQL- und JSON-Typen wie folgt:

```
{
    "id" : "1234",
    "name" : "Nadia",
    "age" : 25
}
```

In diesem Abschnitt wird erklärt, wie die folgenden DynamoDB-Skalar-, Dokument- und Satztypen AWS AppSync konvertiert werden:

Zeichenfolgetyp **S**

Ein einzelner Zeichenfolgewert. Ein DynamoDB-Zeichenkettenwert wird als Zeichenfolge zurückgegeben.

Beispiel: DynamoDB hat den folgenden DynamoDB-Zeichenkettenwert zurückgegeben:

```
{ "S" : "some string" }
```

AWS AppSync konvertiert ihn in eine Zeichenfolge:

```
"some string"
```

Zeichenfolgesatztyp **SS**

Ein Satz von Zeichenfolgewerten. Ein DynamoDB String Set-Wert wird als eine Liste von Zeichenketten zurückgegeben.

Beispiel: DynamoDB hat den folgenden DynamoDB String Set-Wert zurückgegeben:

```
{ "SS" : [ "first value", "second value", ... ] }
```

AWS AppSync konvertiert ihn in eine Liste von Zeichenketten:

```
[ "+1 555 123 4567", "+1 555 234 5678" ]
```

Zahlentyp **N**

Ein einzelner numerischer Wert. Ein DynamoDB-Zahlenwert wird als Zahl zurückgegeben.

Beispiel: DynamoDB hat den folgenden DynamoDB-Zahlenwert zurückgegeben:

```
{ "N" : 1234 }
```

AWS AppSync wandelt ihn in eine Zahl um:

```
1234
```

Zahlensatztyp **NS**

Ein Satz von Zahlenwerten. Ein DynamoDB Number Set-Wert wird als eine Liste von Zahlen zurückgegeben.

Beispiel: DynamoDB hat den folgenden DynamoDB Number Set-Wert zurückgegeben:

```
{ "NS" : [ 67.8, 12.2, 70 ] }
```

AWS AppSync wandelt ihn in eine Liste von Zahlen um:

```
[ 67.8, 12.2, 70 ]
```

Binärtyp **B**

Ein Binärwert. Ein DynamoDB-Binärwert wird als Zeichenfolge zurückgegeben, die die Base64-Darstellung dieses Werts enthält.

Wenn DynamoDB beispielsweise den folgenden DynamoDB-Binärwert zurückgegeben hat:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

AWS AppSync konvertiert ihn in eine Zeichenfolge, die die Base64-Darstellung des Werts enthält:

```
"SGVsbG8sIFdvcmxkIQo="
```

Beachten Sie, dass die Binärdaten im Base64-Codierungsschema codiert werden, wie in RFC 4648 und RFC 2045 angegeben.

Binärsatztyp **BS**

Ein Satz von Binärwerten. Ein DynamoDB Binary Set-Wert wird als eine Liste von Zeichenketten zurückgegeben, die die Base64-Darstellung der Werte enthalten.

Beispiel: DynamoDB hat den folgenden DynamoDB Binary Set-Wert zurückgegeben:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

AWS AppSync konvertiert ihn in eine Liste von Zeichenketten, die die Base64-Darstellung der Werte enthalten:

```
[ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ]
```

Beachten Sie, dass die Binärdaten im Base64-Codierungsschema codiert werden, wie in RFC 4648 und RFC 2045 angegeben.

Boolescher Typ **BOOL**

Ein Boolescher Wert Ein boolescher DynamoDB-Wert wird als boolescher Wert zurückgegeben.

Wenn DynamoDB beispielsweise den folgenden booleschen DynamoDB-Wert zurückgegeben hat:

```
{ "BOOL" : true }
```

AWS AppSync wandelt ihn in einen booleschen Wert um:

```
true
```

Listentyp **L**

Eine Liste aller anderen unterstützten DynamoDB-Werte. Ein DynamoDB-Listenwert wird als Werteliste zurückgegeben, wobei jeder innere Wert ebenfalls konvertiert wird.

Wenn DynamoDB beispielsweise den folgenden DynamoDB-Listenwert zurückgegeben hat:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

AWS AppSync konvertiert ihn in eine Liste von konvertierten Werten:

```
[ "A string value", 1, [ "Another string value", "Even more string values!" ] ]
```

Zuordnungstyp **M**

A key/value collection of any other supported DynamoDB value. A DynamoDB Map value is returned as a JSON object, where each key/value wird ebenfalls konvertiert.

Wenn DynamoDB beispielsweise den folgenden DynamoDB-Zuordnungswert zurückgegeben hat:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

AWS AppSync konvertiert ihn in ein JSON-Objekt:

```
{
   "someString" : "A string value",
   "someNumber" : 1,
   "stringSet"  : [ "Another string value", "Even more string values!" ]
}
```

Null-Typ **`NULL`**

Ein Null-Wert.

Beispiel: DynamoDB hat den folgenden DynamoDB-Null-Wert zurückgegeben:

```
{ "NULL" : null }
```

AWS AppSync wandelt ihn in einen Nullwert um:

```
null
```

Filter

Wenn Sie Objekte in DynamoDB mithilfe der `Scan` Operationen `Query` und abfragen, können Sie optional a angeben, `filter` das die Ergebnisse auswertet und nur die gewünschten Werte zurückgibt.

Die Filtereigenschaft einer `Query` `Scan` OR-Anforderung hat die folgende Struktur:

```
type DynamoDBExpression = {
  expression: string;
  expressionNames?: { [key: string]: string};
  expressionValues?: { [key: string]: any};
};
```

Die Felder sind wie folgt definiert:

`expression`

Der Abfrageausdruck. Weitere Informationen zum Schreiben von Filterausdrücken finden Sie in der Dokumentation zu DynamoDB QueryFilter und DynamoDB ScanFilter. Dieses Feld muss angegeben werden.

`expressionNames`

Die Ersetzungen für Platzhalter der Namen von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter, der in der `expression` verwendet wird. Der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Namen von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

`expressionValues`

Die Ersetzungen für Platzhalter der Werte von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Wertplatzhalter, der im `expression` verwendet wird, und der Wert muss ein typisierter Wert sein. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser muss angegeben werden. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Werte von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

Beispiel

Das folgende Beispiel ist ein Filterabschnitt für eine Anfrage, in dem aus DynamoDB abgerufene Einträge nur zurückgegeben werden, wenn der Titel mit dem `title` Argument beginnt.

Hier verwenden wir die`util.transform.toDynamoDBFilterExpression`, um automatisch einen Filter aus einem Objekt zu erstellen:

```
const filter = util.transform.toDynamoDBFilterExpression({
  title: { beginsWith: 'far away' },
});

const request = {};
request.filter = JSON.parse(filter);
```

Dadurch wird der folgende Filter generiert:

```
{
  "filter": {
    "expression": "(begins_with(#title,:title_beginsWith))",
    "expressionNames": { "#title": "title" },
    "expressionValues": {
      ":title_beginsWith": { "S": "far away" }
    }
```

```
  }
}
```

Bedingungsausdrücke

Wenn Sie Objekte in DynamoDB mithilfe der Operationen,, und `DeleteItem` DynamoDB mutieren `PutItemUpdateItem`, können Sie optional einen Bedingungsausdruck angeben, der steuert, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Zustand des Objekts, das sich bereits in DynamoDB befand, bevor der Vorgang ausgeführt wird.

Die AWS AppSync DynamoDB-Funktion ermöglicht die Angabe eines Bedingungsausdrucks in `PutItemUpdateItem`, und `DeleteItem` Request-Objekten sowie eine Strategie, die befolgt werden kann, wenn die Bedingung fehlschlägt und das Objekt nicht aktualisiert wurde.

Beispiel 1

Das folgende `PutItem` Anforderungsobjekt hat keinen Bedingungsausdruck. Dadurch wird ein Element in DynamoDB platziert, auch wenn ein Element mit demselben Schlüssel bereits vorhanden ist, wodurch das vorhandene Element überschrieben wird.

```
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),
  };
}
```

Beispiel 2

Das folgende `PutItem` Objekt hat einen Bedingungsausdruck, der nur dann ermöglicht, dass der Vorgang erfolgreich ist, wenn ein Element mit demselben Schlüssel nicht in DynamoDB vorhanden ist.

```
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)" }
  };
}
```

Wenn die Zustandsprüfung fehlschlägt, gibt die AWS AppSync DynamoDB-Funktion standardmäßig einen Fehler für die Mutation aus.

Die AWS AppSync DynamoDB-Funktion bietet jedoch einige zusätzliche Funktionen, die Entwicklern helfen sollen, einige häufig auftretende Randfälle zu bewältigen:

- Wenn AWS AppSync DynamoDB-Funktionen feststellen können, dass der aktuelle Wert in DynamoDB dem gewünschten Ergebnis entspricht, wird der Vorgang so behandelt, als ob er trotzdem erfolgreich gewesen wäre.
- Anstatt einen Fehler zurückzugeben, können Sie die Funktion so konfigurieren, dass sie eine benutzerdefinierte Lambda-Funktion aufruft, um zu entscheiden, wie die AWS AppSync DynamoDB-Funktion den Fehler behandeln soll.

Diese werden im Abschnitt Behandlung eines Fehlers bei der Zustandsprüfung ausführlicher beschrieben.

Weitere Informationen zu DynamoDB-Bedingungsausdrücken finden Sie in der ConditionExpressions DynamoDB-Dokumentation.

Angabe einer Bedingung

Die Objekte `PutItem`, `UpdateItem`, und `DeleteItem` request ermöglichen alle die Angabe eines optionalen `condition` Abschnitts. Wenn nicht angegeben, wird keine Bedingungsprüfung ausgeführt. Wenn angegeben, muss die Bedingung wahr sein, damit die Operation erfolgreich ausgeführt werden kann.

Ein `condition`-Abschnitt weist die folgende Struktur auf:

```
type ConditionCheckExpression = {
  expression: string;
  expressionNames?: { [key: string]: string};
  expressionValues?: { [key: string]: any};
  equalsIgnore?: string[];
  consistentRead?: boolean;
```

```
  conditionalCheckFailedHandler?: {
    strategy: 'Custom' | 'Reject';
    lambdaArn?: string;
  };
};
```

Die folgenden Felder geben die Bedingung an:

`expression`

Der Aktualisierungsausdruck selbst. Weitere Informationen zum Schreiben von Bedingungsausdrücken finden Sie in der DynamoDB-Dokumentation ConditionExpressions . Dieses Feld muss angegeben werden.

`expressionNames`

Die Ersetzungen für Platzhalter für Ausdrucksattributnamen in der Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter, der im Ausdruck verwendet wird, und der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Namen von Ausdrucksattributen gefüllt sein, die im Ausdruck verwendet werden.

`expressionValues`

Die Ersetzungen für Platzhalter der Werte von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Wertplatzhalter, der im Ausdruck verwendet wird, und der Wert muss ein typisierter Wert sein. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser muss angegeben werden. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Werte von Ausdrucksattributen gefüllt sein, die im Ausdruck verwendet werden.

Die verbleibenden Felder teilen der AWS AppSync DynamoDB-Funktion mit, wie sie mit einem Fehler bei der Zustandsprüfung umgehen soll:

`equalsIgnore`

Wenn eine Zustandsprüfung bei der Verwendung des `PutItem` Vorgangs fehlschlägt, vergleicht die AWS AppSync DynamoDB-Funktion das Element, das sich derzeit in DynamoDB befindet, mit dem Element, das sie zu schreiben versucht hat. Wenn sie identisch sind, wird die Operation behandelt, als wäre sie trotzdem erfolgreich ausgeführt worden. Sie können das `equalsIgnore` Feld verwenden, um eine Liste von Attributen anzugeben, die bei diesem Vergleich ignoriert AWS

AppSync werden sollen. Wenn der einzige Unterschied beispielsweise ein `version` Attribut war, wird der Vorgang so behandelt, als ob er erfolgreich war. Dies ist ein optionales Feld.

`consistentRead`

Wenn eine Zustandsprüfung fehlschlägt, AWS AppSync wird der aktuelle Wert des Elements mithilfe eines stark konsistenten Lesevorgangs von DynamoDB abgerufen. Sie können dieses Feld verwenden, um die AWS AppSync DynamoDB-Funktion anzuweisen, stattdessen einen eventuell konsistenten Lesevorgang zu verwenden. Dieses Feld ist optional und standardmäßig auf `true` gesetzt.

`conditionalCheckFailedHandler`

In diesem Abschnitt können Sie angeben, wie die AWS AppSync DynamoDB-Funktion einen Fehler bei der Zustandsprüfung behandelt, nachdem der aktuelle Wert in DynamoDB mit dem erwarteten Ergebnis verglichen wurde. Dieser Abschnitt ist optional. Wenn nicht angegeben, wird standardmäßig eine Strategie von `Reject` verwendet.

`strategy`

Die Strategie, die die AWS AppSync DynamoDB-Funktion verfolgt, nachdem sie den aktuellen Wert in DynamoDB mit dem erwarteten Ergebnis verglichen hat. Dieses Feld ist erforderlich und besitzt die folgenden möglichen Werte:

`Reject`

Die Mutation schlägt fehl und der GraphQL-Antwort wird ein Fehler hinzugefügt.

`Custom`

Die AWS AppSync DynamoDB-Funktion ruft eine benutzerdefinierte Lambda-Funktion auf, um zu entscheiden, wie mit dem Fehler bei der Zustandsprüfung umgegangen werden soll. Wenn die `strategy` auf `Custom` gesetzt ist, muss das `lambdaArn`-Feld den ARN der aufzurufenden Lambda-Funktion enthalten.

`lambdaArn`

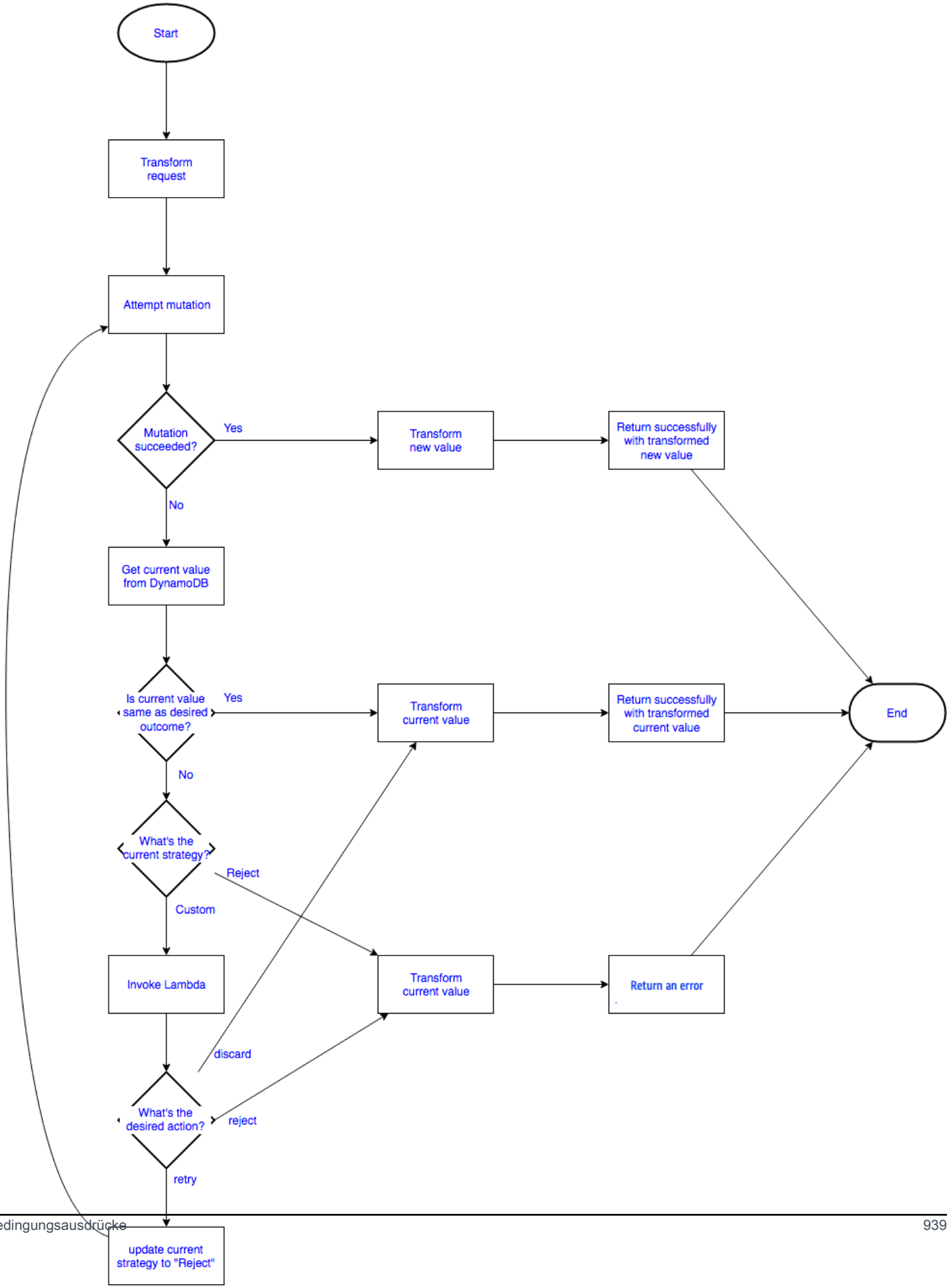
Der ARN der aufzurufenden Lambda-Funktion, der bestimmt, wie die AWS AppSync DynamoDB-Funktion den Fehler bei der Zustandsprüfung behandeln soll. Dieses Feld muss nur angegeben werden, wenn `strategy` auf `Custom` festgelegt ist. Weitere Informationen zur Verwendung dieser Funktion finden Sie unter Behandlung eines Fehlers bei der Zustandsprüfung.

Behandlung eines Fehlers bei der Zustandsprüfung

Wenn eine Zustandsprüfung fehlschlägt, kann die AWS AppSync DynamoDB-Funktion den Fehler für die Mutation und den aktuellen Wert des Objekts mithilfe des `util.appendError` Dienstprogramms weitergeben. Die AWS AppSync DynamoDB-Funktion bietet jedoch einige zusätzliche Funktionen, die Entwicklern helfen sollen, einige häufig auftretende Randfälle zu bewältigen:

- Wenn AWS AppSync DynamoDB-Funktionen feststellen können, dass der aktuelle Wert in DynamoDB dem gewünschten Ergebnis entspricht, wird der Vorgang so behandelt, als ob er trotzdem erfolgreich gewesen wäre.
- Anstatt einen Fehler zurückzugeben, können Sie die Funktion so konfigurieren, dass sie eine benutzerdefinierte Lambda-Funktion aufruft, um zu entscheiden, wie die AWS AppSync DynamoDB-Funktion den Fehler behandeln soll.

Das Flussdiagramm für diesen Prozess ist:


Start
Transform request
Attempt mutation
Mutation succeeded?
Yes
Transform new value
Return successfully with transformed new value
No
Get current value from DynamoDB
Is current value same as desired outcome?
Yes
Transform current value
Return successfully with transformed current value
End
No
What's the current strategy?
Reject
Custom
Invoke Lambda
Transform current value
Return an error
discard
What's the desired action?
reject
retry
update current strategy to "Reject"

Es wird geprüft, ob das gewünschte Ergebnis vorliegt

Wenn die Zustandsprüfung fehlschlägt, führt die AWS AppSync DynamoDB-Funktion eine `GetItem` DynamoDB-Anforderung aus, um den aktuellen Wert des Elements von DynamoDB abzurufen. Standardmäßig wird ein Strongly Consistent-Lesevorgang verwendet, jedoch kann dies mit dem `consistentRead`-Feld im `condition`-Block konfiguriert werden, und mit dem erwarteten Ergebnis verglichen:

- Für den `PutItem` Vorgang vergleicht die AWS AppSync DynamoDB-Funktion den aktuellen Wert mit dem Wert, den sie zu schreiben versucht hat, und schließt alle unter aufgelisteten Attribute aus dem Vergleich `equalsIgnore` aus. Wenn die Elemente identisch sind, wird der Vorgang als erfolgreich behandelt und das Element zurückgegeben, das von DynamoDB abgerufen wurde. Andernfalls wird die konfigurierte Strategie verfolgt.

  Wenn das `PutItem` Anforderungsobjekt beispielsweise wie folgt aussah:

```
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']
    }
  };
}
```

  Und das Element, das sich derzeit in DynamoDB befindet, wie folgt ausgesehen hat:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

Die AWS AppSync DynamoDB-Funktion würde das Element, das sie schreiben wollte, mit dem aktuellen Wert vergleichen und feststellen, dass der einzige Unterschied das `version` Feld war. Da sie jedoch so konfiguriert ist, dass sie das `version` Feld ignoriert, behandelt sie den Vorgang als erfolgreich und gibt das Element zurück, das von DynamoDB abgerufen wurde.

- Für den `DeleteItem` Vorgang überprüft die AWS AppSync DynamoDB-Funktion, ob ein Element von DynamoDB zurückgegeben wurde. Wenn kein Element zurückgegeben wurde, behandelt er die Operation als erfolgreich. Andernfalls wird die konfigurierte Strategie verfolgt.
- Für den `UpdateItem` Vorgang verfügt die AWS AppSync DynamoDB-Funktion nicht über genügend Informationen, um festzustellen, ob das Element, das sich derzeit in DynamoDB befindet, dem erwarteten Ergebnis entspricht und daher der konfigurierten Strategie folgt.

Wenn der aktuelle Status des Objekts in DynamoDB vom erwarteten Ergebnis abweicht, folgt die AWS AppSync DynamoDB-Funktion der konfigurierten Strategie, entweder die Mutation zurückzuweisen oder eine Lambda-Funktion aufzurufen, um zu bestimmen, was als Nächstes zu tun ist.

Wir folgen der Strategie „Ablehnen“

Wenn die `Reject` Strategie befolgt wird, gibt die AWS AppSync DynamoDB-Funktion einen Fehler für die Mutation zurück.

Angenommen, Sie haben folgende Mutationsanforderung:

```
mutation {
    updatePerson(id: 1, name: "Steve", expectedVersion: 1) {
        Name
        theVersion
    }
}
```

Wenn das von DynamoDB zurückgegebene Element wie folgt aussieht:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

Und der Response-Handler der Funktion sieht wie folgt aus:

```
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
}
```

Die GraphQL-Antwort sieht wie folgt aus:

```
{
  "data": null,
  "errors": [
    {
      "message": "The conditional request failed (Service: AmazonDynamoDBv2;
 Status Code: 400; Error Code: ConditionalCheckFailedException; Request ID:
 ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ)"
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      ...
    }
  ]
}
```

Beachten Sie auch, dass, wenn bei einer erfolgreichen Mutation alle Felder im zurückgegebenen Objekt von anderen Resolvern ausgefüllt werden, diese nicht auf den Resolver angewendet werden, wenn das Objekt im `error`-Abschnitt zurückgegeben wird.

Folgen Sie der „benutzerdefinierten“ Strategie

Wenn Sie der `Custom` Strategie folgen, ruft die AWS AppSync DynamoDB-Funktion eine Lambda-Funktion auf, um zu entscheiden, was als Nächstes zu tun ist. Die Lambda-Funktion wählt eine der folgenden Optionen aus:

- Die Mutation `reject`. Dadurch wird die AWS AppSync DynamoDB-Funktion angewiesen`Reject`, sich wie bei der konfigurierten Strategie zu verhalten und einen Fehler für die Mutation und

den aktuellen Wert des Objekts in DynamoDB zurückzugeben, wie im vorherigen Abschnitt beschrieben.

- Die Mutation `discard`. Dadurch wird die AWS AppSync DynamoDB-Funktion angewiesen, den Fehler bei der Zustandsprüfung stillschweigend zu ignorieren und den Wert in DynamoDB zurückzugeben.
- Die Mutation `retry`. Dadurch wird die AWS AppSync DynamoDB-Funktion angewiesen, die Mutation mit einem neuen Anforderungsobjekt erneut zu versuchen.

Die Lambda-Aufrufanforderung

Die AWS AppSync DynamoDB-Funktion ruft die in der angegebene Lambda-Funktion auf. `lambdaArn` Er verwendet die gleiche `service-role-arn`-Konfiguration für die Datenquelle. Die Nutzlast des Aufrufs hat die folgende Struktur:

```
{
    "arguments": { ... },
    "requestMapping": {... },
    "currentValue": { ... },
    "resolver": { ... },
    "identity": { ... }
}
```

Die Felder sind wie folgt definiert:

`arguments`

Die Argumente aus der GraphQL-Mutation. Dies entspricht den Argumenten, die für das Anforderungsobjekt in verfügbar sind. `context.arguments`

`requestMapping`

Das Anforderungsobjekt für diesen Vorgang.

`currentValue`

Der aktuelle Wert des Objekts in DynamoDB.

`resolver`

Informationen über den AWS AppSync Resolver oder die Funktion.

`identity`

Informationen über den Aufrufer. Dies entspricht den Identitätsinformationen, die dem Anforderungsobjekt in `context.identity` zur Verfügung stehen.

Ein vollständiges Beispiel für die Nutzlast:

```
{
    "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"
    }
}
```

Die Lambda-Aufrufantwort

Die Lambda-Funktion kann die Nutzlast des Aufrufs überprüfen und jede Geschäftslogik anwenden, um zu entscheiden, wie die AWS AppSync DynamoDB-Funktion mit dem Fehler umgehen soll. Es gibt drei Optionen für den Umgang mit dem Bedingungsprüfungsfehler:

- Die Mutation `reject`. Die Antwortnutzlast für diese Option muss diese Struktur aufweisen:

  ```
  {
      "action": "reject"
  }
  ```

  Dadurch wird die AWS AppSync DynamoDB-Funktion angewiesen`Reject`, sich wie bei der konfigurierten Strategie zu verhalten und einen Fehler für die Mutation und den aktuellen Wert des Objekts in DynamoDB zurückzugeben, wie im obigen Abschnitt beschrieben.
- Die Mutation `discard`. Die Antwortnutzlast für diese Option muss diese Struktur aufweisen:

  ```
  {
      "action": "discard"
  }
  ```

  Dadurch wird die AWS AppSync DynamoDB-Funktion angewiesen, den Fehler bei der Zustandsprüfung stillschweigend zu ignorieren und den Wert in DynamoDB zurückzugeben.
- Die Mutation `retry`. Die Antwortnutzlast für diese Option muss diese Struktur aufweisen:

  ```
  {
      "action": "retry",
      "retryMapping": { ... }
  }
  ```

Dadurch wird die AWS AppSync DynamoDB-Funktion angewiesen, die Mutation mit einem neuen Anforderungsobjekt erneut zu versuchen. Die Struktur des `retryMapping` Abschnitts hängt von der DynamoDB-Operation ab und ist eine Teilmenge des vollständigen Anforderungsobjekts für diesen Vorgang.

Für `PutItem` weist der `retryMapping`-Abschnitt die folgende Struktur auf. Eine Beschreibung des `attributeValues` Felds finden Sie unter. PutItem

```
{
    "attributeValues": { ... },
    "condition": {
        "equalsIgnore" = [ ... ],
        "consistentRead" = true
    }
}
```

Für `UpdateItem` weist der `retryMapping`-Abschnitt die folgende Struktur auf. Eine Beschreibung des `update` Abschnitts finden Sie unter UpdateItem.

```
{
    "update" : {
        "expression" : "someExpression"
        "expressionNames" : {
            "#foo" : "foo"
        },
        "expressionValues" : {
            ":bar" : ... typed value
        }
    },
    "condition": {
        "consistentRead" = true
    }
}
```

Für `DeleteItem` weist der `retryMapping`-Abschnitt die folgende Struktur auf.

```
{
    "condition": {
        "consistentRead" = true
    }
```

```
}
```

Es ist nicht möglich, eine andere Operation oder einen anderen Schlüssel anzugeben. Die AWS AppSync DynamoDB-Funktion erlaubt nur Wiederholungen derselben Operation für dasselbe Objekt. Beachten Sie auch, dass der Abschnitt `condition` nicht zulässt, dass ein `conditionalCheckFailedHandler` angegeben wird. Schlägt der Wiederholungsversuch fehl, folgt die AWS AppSync DynamoDB-Funktion der Strategie. `Reject`

Hier sehen Sie ein Beispiel für eine Lambda-Funktion zum Umgang mit einer fehlgeschlagenen `PutItem`-Anforderung. Die Geschäftslogik prüft, wer den Aufruf ausgeführt hat. Wenn es von erstellt wurde`jeffTheAdmin`, wiederholt es die Anfrage und aktualisiert das `version` und `expectedVersion` aus dem Element, das sich derzeit in DynamoDB befindet. Andernfalls wird die Mutation ablehnt.

```
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)
};
```

Ausdrücke für Transaktionsbedingungen

Ausdrücke für Transaktionsbedingungen sind in Anfragen aller vier Arten von Operationen in verfügbar`TransactWriteItems`, nämlich `PutItem``DeleteItem`,`UpdateItem`, und`ConditionCheck`.

Für `PutItem` `DeleteItem``UpdateItem`, und ist der Ausdruck für die Transaktionsbedingung optional. Für `ConditionCheck` ist der Ausdruck der Transaktionsbedingung erforderlich.

Beispiel 1

Der folgende Anforderungshandler für `DeleteItem` Transaktionsfunktionen hat keinen Bedingungsausdruck. Infolgedessen wird das Element in DynamoDB gelöscht.

```
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 }),
      }
    ],
  };
}
```

Beispiel 2

Der folgende Anforderungshandler für `DeleteItem` Transaktionsfunktionen verfügt über einen Ausdruck für eine Transaktionsbedingung, mit dem der Vorgang nur dann erfolgreich ausgeführt werden kann, wenn der Autor dieses Beitrags einem bestimmten Namen entspricht.

```
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 },
        }),
      }
    ],
  };
}
```

Wenn die Bedingungsprüfung fehlschlägt, führt dies zu `TransactionCanceledException`, und die Fehlerdetails werden in `ctx.result.cancellationReasons` zurückgegeben. Beachten Sie, dass standardmäßig das alte Element in DynamoDB zurückgegeben wird, bei dem die Zustandsprüfung fehlgeschlagen ist. `ctx.result.cancellationReasons`

Angabe einer Bedingung

Die Objekte `PutItem`Update`Item`, und `DeleteItem` request ermöglichen alle die Angabe eines optionalen `condition` Abschnitts. Wenn nicht angegeben, wird keine Bedingungsprüfung ausgeführt. Wenn angegeben, muss die Bedingung wahr sein, damit die Operation erfolgreich ausgeführt werden kann. Der `ConditionCheck` muss einen `condition`-Abschnitt haben, der angegeben werden soll. Die Bedingung muss erfüllt sein, damit die gesamte Transaktion erfolgreich ist.

Ein `condition`-Abschnitt weist die folgende Struktur auf:

```
type TransactConditionCheckExpression = {
  expression: string;
  expressionNames?: { [key: string]: string };
  expressionValues?: { [key: string]: string };
  returnValuesOnConditionCheckFailure: boolean;
};
```

Die folgenden Felder geben die Bedingung an:

`expression`

Der Aktualisierungsausdruck selbst. Weitere Informationen zum Schreiben von Bedingungsausdrücken finden Sie in der DynamoDB-Dokumentation ConditionExpressions . Dieses Feld muss angegeben werden.

`expressionNames`

Die Ersetzungen für Platzhalter für Ausdrucksattributnamen in der Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter, der im Ausdruck verwendet wird, und der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Namen von Ausdrucksattributen gefüllt sein, die im Ausdruck verwendet werden.

`expressionValues`

Die Ersetzungen für Platzhalter der Werte von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Wertplatzhalter, der im Ausdruck verwendet wird, und der Wert muss ein typisierter Wert sein. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser muss angegeben werden. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Werte von Ausdrucksattributen gefüllt sein, die im Ausdruck verwendet werden.

`returnValuesOnConditionCheckFailure`

Geben Sie an, ob das Element in DynamoDB zurückgeholt werden soll, wenn eine Zustandsprüfung fehlschlägt. Das abgerufene Element befindet sich in `ctx.result.cancellationReasons[<index>].item`, wobei der `<index>` Index des Anforderungselements ist, durch das die Bedingungsprüfung fehlgeschlagen ist. Dieser Wert ist standardmäßig auf „true“ voreingestellt.

Projektionen

Wenn Sie Objekte in DynamoDB mithilfe der `TransactGetItems` Operationen`GetItem`,`Scan`, `QueryBatchGetItem`, und lesen, können Sie optional eine Projektion angeben, die die gewünschten Attribute identifiziert. Die Projektionseigenschaft hat die folgende Struktur, die Filtern ähnelt:

```
type DynamoDBExpression = {
  expression: string;
```

```
  expressionNames?: { [key: string]: string}
};
```

Die Felder sind wie folgt definiert:

`expression`

Der Projektionsausdruck, bei dem es sich um eine Zeichenfolge handelt. Zum Abrufen eines einzelnen Attributs geben Sie seinen Namen an. Bei mehreren Attributen müssen die Namen durch Kommas getrennte Werte sein. Weitere Informationen zum Schreiben von Projektionsausdrücken finden Sie in der Dokumentation zu DynamoDB-Projektionsausdrücken. Dies ist ein Pflichtfeld.

`expressionNames`

Die Ersetzungen für Platzhalter für Ausdrucksattributnamen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter, der in der `expression` verwendet wird. Der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter für Ausdrucksattributnamen gefüllt werden, die in der verwendet werden. `expression` Weitere Informationen zu `expressionNames` finden Sie in der DynamoDB-Dokumentation.

Beispiel 1

Das folgende Beispiel ist ein Projektionsabschnitt für eine JavaScript Funktion, in der nur die Attribute `author` und von DynamoDB zurückgegeben `id` werden:

```
projection : {
    expression : "#author, id",
    expressionNames : {
        "#author" : "author"
    }
}
```

> **Tip**
>
> Sie können auf Ihren GraphQL-Anforderungsauswahlsatz zugreifen, indem selectionSetListSie. In diesem Feld können Sie Ihren Projektionsausdruck Ihren Anforderungen entsprechend dynamisch gestalten.

> **Note**
>
> Bei der Verwendung von Projektionsausdrücken mit den `Scan` Operationen `Query` und `select` muss der Wert für sein`SPECIFIC_ATTRIBUTES`. Weitere Informationen finden Sie in der DynamoDB-Dokumentation.

AWS AppSync JavaScript Resolver-Funktionsreferenz für OpenSearch

Mit dem AWS AppSync Resolver für Amazon OpenSearch Service können Sie GraphQL verwenden, um Daten in vorhandenen OpenSearch Service-Domains in Ihrem Konto zu speichern und abzurufen. Dieser Resolver ermöglicht es Ihnen, eine eingehende GraphQL-Anfrage einer Serviceanfrage zuzuordnen und die OpenSearch OpenSearch Serviceantwort dann wieder GraphQL zuzuordnen. In diesem Abschnitt werden die Funktionsanforderungs- und Antworthandler für die unterstützten Serviceoperationen beschrieben. OpenSearch

Anforderung

Die meisten OpenSearch Serviceanforderungsobjekte haben eine gemeinsame Struktur, bei der sich nur wenige Teile ändern. Im folgenden Beispiel wird eine Suche in einer OpenSearch Service-Domäne ausgeführt, in der Dokumente vom Typ sind `post` und unter `id` der sie indexiert sind. Die Suchparameter sind im body-Abschnitt definiert, viele der gemeinsamen Abfragebestimmungen sind im `query`-Feld definiert. In diesem Beispiel wird nach Dokumenten gesucht, die `"Nadia"`, `"Bailey"` oder beides enthalten, und zwar in dem `author`-Feld eines Dokuments:

```
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' } },
            ],
          },
        },
      },
    },
  };
}
```

Antwort

Wie bei anderen Datenquellen sendet OpenSearch Service eine Antwort an AWS AppSync , die in GraphQL konvertiert werden muss.

Die meisten GraphQL-Abfragen suchen nach dem `_source` Feld aus einer OpenSearch Serviceantwort. Da Sie Suchen durchführen können, um entweder ein einzelnes Dokument oder eine Liste von Dokumenten zurückzugeben, werden in OpenSearch Service zwei gängige Antwortmuster verwendet:

Liste der Ergebnisse

```
export function response(ctx) {
  const entries = [];
  for (const entry of ctx.result.hits.hits) {
    entries.push(entry['_source']);
  }
  return entries;
}
```

Einzelnes Element

```
export function response(ctx) {
  return ctx.result['_source']
}
```

`operation` field

> **Note**
>
> Dies gilt nur für den Request-Handler.

HTTP-Methode oder Verb (GET, POST, PUT, HEAD oder DELETE), das AWS AppSync an die OpenSearch Service-Domäne sendet. Der Schlüssel und der Wert müssen beide Strings sein.

```
"operation" : "PUT"
```

`path` field

> **Note**
>
> Dies gilt nur für den Request-Handler.

Der Suchpfad für eine OpenSearch Serviceanfrage von AWS AppSync. Dies bildet eine URL für das HTTP-Verb der Operation. Der Schlüssel und der Wert müssen beide Strings sein.

```
"path" : "/indexname/type"

"path" : "/indexname/type/_search"
```

Wenn der Anforderungshandler ausgewertet wird, wird dieser Pfad als Teil der HTTP-Anfrage gesendet, einschließlich der OpenSearch Dienstdomäne. Das vorherige Beispiel kann dann beispielsweise so aussehen:

```
GET https://opensearch-domain-name.REGION.es.amazonaws.com/indexname/type/_search
```

`params` field

> **Note**
>
> Dies gilt nur für den Request-Handler.

Wird verwendet, um anzugeben, welche Aktion Ihre Suche durchführt, meistens indem der Abfragewert in den Body setzt. Es gibt jedoch einige andere Funktionen, die konfiguriert werden können, wie z. B. die Formatierung von Antworten.

- Header

  Die Header-Informationen, wie beispielsweise Schlüssel-Wert-Paare. Der Schlüssel und der Wert müssen beide Strings sein. Zum Beispiel:

```
"headers" : {
    "Content-Type" : "application/json"
}
```

> **Note**
>
> AWS AppSync unterstützt derzeit nur JSON als`Content-Type`.

- queryString

  Schlüssel-Wert-Paare, die allgemeine Optionen angeben, z. B. Codeformatierung für JSON-Antworten. Der Schlüssel und der Wert müssen beide Strings sein. Wenn Sie beispielsweise ein gut formatiertes JSON-Format haben möchten, geben Sie Folgendes an:

```
"queryString" : {
    "pretty" : "true"
}
```

- body

  Dies ist der Hauptteil Ihrer Anfrage, der es ermöglicht AWS AppSync , eine wohlgeformte Suchanfrage für Ihre OpenSearch Service-Domain zu erstellen. Der Schlüssel muss ein aus einem Objekt bestehender String sein. Ein paar Beispiele werden unten gezeigt.

Beispiel 1

Rückgabe aller Dokumente, die die Stadt „Seattle“ enthalten:

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

```
    operation: 'GET',
    path: '/id/post/_search',
    params: {
      headers: {},
      queryString: {},
      body: { from: 0, size: 50, query: { match: { city: 'seattle' } } },
    },
  };
}
```

Beispiel 2

Rückgabe aller Dokumente, die „Washington“ als Stadt oder Staat enthalten:

```
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'] },
        },
      },
    },
  };
}
```

Variablen übergeben

> **Note**
>
> Dies gilt nur für den Request-Handler.

Sie können Variablen auch als Teil der Auswertung in Ihrem Request-Handler übergeben. Angenommen, Sie hatten die folgende GraphQL-Abfrage:

```
query {
    searchForState(state: "washington"){
        ...
    }
}
```

Der Funktionsanforderungshandler könnte der folgende sein:

```
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'] },
        },
      },
    },
  };
}
```

AWS AppSync JavaScript Resolver-Funktionsreferenz für Lambda

Sie können AWS AppSync Funktionen und Resolver verwenden, um Lambda-Funktionen in Ihrem Konto aufzurufen. Sie können Ihre Anforderungs-Payloads und die Antwort Ihrer Lambda-Funktionen gestalten, bevor Sie sie an Ihre Kunden zurückgeben. Sie können auch die Art der Operation angeben, die in Ihrem Anforderungsobjekt ausgeführt werden soll. In diesem Abschnitt werden die Anfragen für die unterstützten Lambda-Operationen beschrieben.

Objekt anfordern

Das Lambda-Anforderungsobjekt verarbeitet Felder, die sich auf Ihre Lambda-Funktion beziehen:

```
export type LambdaRequest = {
  operation: 'Invoke' | 'BatchInvoke';
  invocationType?: 'RequestResponse' | 'Event';
```

```
  payload: unknown;
};
```

Hier ist ein Beispiel, das eine `invoke` Operation verwendet, deren Nutzdaten das `getPost` Feld aus einem GraphQL-Schema zusammen mit den Argumenten aus dem Kontext sind:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    payload: { field: 'getPost', arguments: ctx.args },
  };
}
```

Das gesamte Mapping-Dokument wird als Eingabe für Ihre Lambda-Funktion übergeben, sodass das vorherige Beispiel nun wie folgt aussieht:

```
{
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": {
      "input": {
        "id": "postId1",
      }
    }
  }
}
```

Operation

Mit der Lambda-Datenquelle können Sie zwei Operationen im `operation` Feld definieren: `Invoke` und`BatchInvoke`. Die `Invoke` Operation teilt mit AWS AppSync , dass Sie Ihre Lambda-Funktion für jeden GraphQL-Feldresolver aufrufen sollen. `BatchInvoke`weist AWS AppSync an, Anfragen für das aktuelle GraphQL-Feld zu stapeln. Das Feld `operation` ist ein Pflichtfeld.

Denn `Invoke` die aufgelöste Anfrage entspricht der Eingabe-Payload der Lambda-Funktion. Lassen Sie uns das obige Beispiel ändern:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
```

```
    payload: { field: 'getPost', arguments: ctx.args },
  };
}
```

Dies wird gelöst und an die Lambda-Funktion übergeben, die etwa so aussehen könnte:

```
{
  "operation": "Invoke",
  "payload": {
    "arguments": {
      "id": "postId1"
    }
  }
}
```

Denn `BatchInvoke` die Anfrage wird auf jeden Field Resolver im Batch angewendet. Führt aus Gründen der Übersichtlichkeit alle `payload` Anforderungswerte zu einer Liste unter einem einzigen Objekt zusammen, das dem Anforderungsobjekt entspricht. AWS AppSync Der folgende Beispiel-Request-Handler zeigt die Zusammenführung:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    payload: ctx,
  };
}
```

Diese Anfrage wird ausgewertet und im folgenden Mapping-Dokument aufgelöst:

```
{
  "operation": "BatchInvoke",
  "payload": [
    {...}, // context for batch item 1
    {...}, // context for batch item 2
    {...}  // context for batch item 3
  ]
}
```

Jedes Element der `payload` Liste entspricht einem einzelnen Batch-Element. Es wird auch erwartet, dass die Lambda-Funktion eine listenförmige Antwort zurückgibt, die der Reihenfolge der in der Anfrage gesendeten Elemente entspricht:

```
[
  { "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
]
```

Nutzlast

Das `payload` Feld ist ein Container, der verwendet wird, um Daten an die Lambda-Funktion zu übergeben. Wenn das `operation` Feld auf gesetzt ist`BatchInvoke`, werden AWS AppSync die vorhandenen `payload` Werte in eine Liste zusammengefasst. Das Feld `payload` ist optional.

Aufruftyp

Mit der Lambda-Datenquelle können Sie zwei Aufruftypen definieren: `RequestResponse` und. `Event` Die Aufruftypen sind synonym mit den in der Lambda-API definierten Aufruftypen. Mit dem `RequestResponse` AWS AppSync Aufruftyp können Sie Ihre Lambda-Funktion synchron aufrufen, um auf eine Antwort zu warten. Der `Event` Aufruf ermöglicht es Ihnen, Ihre Lambda-Funktion asynchron aufzurufen. Weitere Informationen darüber, wie Lambda Anfragen vom `Event` Aufruftyp verarbeitet, finden Sie unter Asynchroner Aufruf. Das Feld `invocationType` ist optional. Wenn dieses Feld nicht in der Anfrage enthalten ist, AWS AppSync wird standardmäßig der Aufruftyp verwendet. `RequestResponse`

Für jedes `invocationType` Feld entspricht die aufgelöste Anfrage der Eingabe-Payload der Lambda-Funktion. Lassen Sie uns das obige Beispiel ändern:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    invocationType: 'Event',
    payload: { field: 'getPost', arguments: ctx.args },
  };
}
```

Dies wird gelöst und an die Lambda-Funktion übergeben, die etwa so aussehen könnte:

```
{
```

```
  "operation": "Invoke",
  "invocationType": "Event",
  "payload": {
    "arguments": {
      "id": "postId1"
    }
  }
}
```

Wenn der `BatchInvoke` Vorgang in Verbindung mit dem `Event` Aufruftypfeld verwendet wird, AWS AppSync führt er den Feldauflöser auf die oben beschriebene Weise zusammen, und die Anforderung wird als asynchrones Ereignis an Ihre Lambda-Funktion übergeben, `payload` wobei es sich um eine Werteliste handelt. Die Antwort auf eine Anfrage vom `Event` Aufruftyp führt zu einem `null` Wert ohne Antworthandler:

```
{
  "data": {
    "field": null
  }
}
```

Wir empfehlen, das Resolver-Caching für Resolver vom `Event` Aufruftyp zu deaktivieren, da diese bei einem Cache-Treffer nicht an Lambda gesendet würden.

Antwortobjekt

Wie bei anderen Datenquellen sendet Ihre Lambda-Funktion eine Antwort darauf AWS AppSync , die in einen GraphQL-Typ konvertiert werden muss. Das Ergebnis der Lambda-Funktion ist in der `context` Ergebniseigenschaft (`context.result`) enthalten.

Wenn die Form Ihrer Lambda-Funktionsantwort mit der Form des GraphQL-Typs übereinstimmt, können Sie die Antwort mit dem folgenden Funktionsantwort-Handler weiterleiten:

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

Es gibt keine erforderlichen Felder oder Formbeschränkungen, die für das Antwortobjekt gelten. Da GraphQL jedoch stark typisiert ist, muss die aufgelöste Antwort dem erwarteten GraphQL-Typ entsprechen.

Batch-Antwort mit Lambda-Funktion

Wenn das `operation`-Feld auf `BatchInvoke` festgelegt ist, erwartet AWS AppSync eine Liste mit Elementen aus der Lambda-Funktion. Damit jedes Ergebnis AWS AppSync dem ursprünglichen Anforderungselement zugeordnet werden kann, muss die Antwortliste in Größe und Reihenfolge übereinstimmen. Es ist zulässig, `null` Elemente in der Antwortliste zu haben; sie `ctx.result` wird entsprechend auf Null gesetzt.

AWS AppSync JavaScript Resolver-Funktionsreferenz für die EventBridge Datenquelle

Die mit der EventBridge Datenquelle verwendete AWS AppSync Resolver-Funktion Request and Response ermöglicht es Ihnen, benutzerdefinierte Ereignisse an den EventBridge Amazon-Bus zu senden.

Anforderung

Mit dem Request-Handler können Sie mehrere benutzerdefinierte Ereignisse an einen EventBridge Event-Bus senden:

```
export function request(ctx) {
  return {
    "operation" : "PutEvents",
    "events" : [{}]
  }
}
```

Eine EventBridge `PutEvents` Anfrage hat die folgende Typdefinition:

```
type PutEventsRequest = {
  operation: 'PutEvents'
  events: {
    source: string
    detail: { [key: string]: any }
    detailType: string
    resources?: string[]
    time?: string // RFC3339 Timestamp format
  }[]
}
```

Antwort

Wenn der `PutEvents` Vorgang erfolgreich ist, EventBridge ist die Antwort von enthalten in`ctx.result`:

```
export function response(ctx) {
  if(ctx.error)
    util.error(ctx.error.message, ctx.error.type, ctx.result)
  else
    return ctx.result
}
```

Fehler, die bei der Ausführung von `PutEvents` Vorgängen wie `InternalExceptions` oder auftretenTimeouts, werden in angezeigt`ctx.error`. Eine Liste der EventBridge häufigsten Fehler finden Sie in der Referenz zu EventBridge häufigen Fehlern.

Sie `result` werden die folgende Typdefinition haben:

```
type PutEventsResult = {
  Entries: {
    ErrorCode: string
    ErrorMessage: string
    EventId: string
  }[]
  FailedEntryCount: number
}
```

- Einträge

  Die Ergebnisse des aufgenommenen Ereignisses, sowohl erfolgreich als auch erfolglos. Wenn die Aufnahme erfolgreich war, enthält der Eintrag das`EventID`. Andernfalls können Sie das `ErrorCode` und verwenden, `ErrorMessage` um das Problem mit dem Eintrag zu identifizieren.

  Für jeden Datensatz entspricht der Index des Antwortelements dem Index im Anforderungsarray.

- FailedEntryCount

  Die Anzahl der fehlgeschlagenen Einträge. Dieser Wert wird als Ganzzahl dargestellt.

Weitere Hinweise zur Antwort von `PutEvents` finden Sie unter PutEvents.

Beispiel für Antwort 1

Das folgende Beispiel zeigt einen `PutEvents` Vorgang mit zwei erfolgreichen Ereignissen:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        }
    ],
    "FailedEntryCount" : 0
}
```

Beispiel für eine Antwort 2

Das folgende Beispiel zeigt einen `PutEvents` Vorgang mit drei Ereignissen, zwei Erfolgen und einem Fehlschlag:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        },
        {
            "ErrorCode" : "SampleErrorCode",
            "ErrorMessage" : "Sample Error Message"
        }
    ],
    "FailedEntryCount" : 1
}
```

`PutEvents`-Felder

`PutEvents`enthält die folgenden Felder für Zuordnungsvorlagen:

- Version

Das `version` Feld ist allen Vorlagen für die Zuordnung von Anfragen gemeinsam und definiert die Version, die die Vorlage verwendet. Dies ist ein Pflichtfeld. Der Wert `2018-05-29` ist die einzige Version, die für die EventBridge Zuordnungsvorlagen unterstützt wird.

- Operation

  Die einzige unterstützte Operation ist`PutEvents`. Mit diesem Vorgang können Sie Ihrem Event-Bus benutzerdefinierte Ereignisse hinzufügen.

- --Ereignisse

  Eine Reihe von Ereignissen, die dem Event-Bus hinzugefügt werden. Dieses Array sollte eine Zuordnung von 1 bis 10 Elementen haben.

  Das `Event`-Objekt enthält folgende Felder:

  - `"source"`: Eine Zeichenfolge, die die Quelle des Ereignisses definiert.
  - `"detail"`: Ein JSON-Objekt, mit dem Sie Informationen über das Ereignis anhängen können. Dieses Feld kann eine leere Map (`{ }`) sein.
  - `"detailType`: Eine Zeichenfolge, die den Ereignistyp identifiziert.
  - `"resources"`: Ein JSON-Array von Zeichenketten, das die an dem Ereignis beteiligten Ressourcen identifiziert. Dieses Feld kann ein leeres Array sein.
  - `"time"`: Der als Zeichenfolge bereitgestellte Zeitstempel des Ereignisses. Dies sollte dem RFC3339Zeitstempelformat entsprechen.

Die folgenden Ausschnitte sind einige Beispiele für gültige Objekte: `Event`

Beispiel 1

```
{
    "source" : "source1",
    "detail" : {
        "key1" : [1,2,3,4],
        "key2" : "strval"
    },
    "detailType" : "sampleDetailType",
    "resources" : ["Resouce1", "Resource2"],
    "time" : "2022-01-10T05:00:10Z"
}
```

Beispiel 2

```
{
    "source" : "source1",
    "detail" : {},
    "detailType" : "sampleDetailType"
}
```

Beispiel 3

```
{
    "source" : "source1",
    "detail" : {
        "key1" : 1200
    },
    "detailType" : "sampleDetailType",
    "resources" : []
}
```

AWS AppSync JavaScript Resolver-Funktionsreferenz für die **None** Datenquelle

Die AWS AppSync Resolverfunktion Request and Response mit der Datenquelle vom Typ None ermöglicht es Ihnen, Anfragen für AWS AppSync lokale Operationen zu gestalten.

Anforderung

Der Request-Handler kann einfach sein und ermöglicht es Ihnen, so viele Kontextinformationen wie möglich über das Feld zu übergeben. `payload`

```
type NONERequest = {
  payload: any;
};
```

Hier ist ein Beispiel, bei dem die Feldargumente an die Nutzlast übergeben werden:

```
export function request(ctx) {
  return {
    payload: context.args
  };
```

```
}
```

Der Wert des `payload` Feldes wird an den Response-Handler der Funktion weitergeleitet und ist in `context.result` verfügbar.

Nutzlast

Das `payload` Feld ist ein Container, der verwendet werden kann, um alle Daten zu übergeben, die dann dem Funktionsantwort-Handler zur Verfügung gestellt werden.

Das Feld `payload` ist optional.

Antwort

Da es keine Datenquelle gibt, wird der Wert des `payload` Felds an den Antworthandler der Funktion weitergeleitet und für die `context.result` Eigenschaft festgelegt.

Wenn die Form des `payload` Feldwerts genau der Form des GraphQL-Typs entspricht, können Sie die Antwort mit dem folgenden Antworthandler weiterleiten:

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

Für die Rückantwort gelten keine erforderlichen Felder oder Formbeschränkungen. Da GraphQL jedoch stark typisiert ist, muss die aufgelöste Antwort dem erwarteten GraphQL-Typ entsprechen.

AWS AppSync JavaScript Resolver-Funktionsreferenz für HTTP

Die AWS AppSync HTTP-Resolver-Funktionen ermöglichen es Ihnen, Anfragen von einem beliebigen HTTP-Endpunkt und Antworten von Ihrem HTTP-Endpunkt zurück AWS AppSync zu senden. AWS AppSync Mit Ihrem Request-Handler können Sie AWS AppSync Hinweise zur Art des aufzurufenden Vorgangs geben. In diesem Abschnitt werden die verschiedenen Konfigurationen für den unterstützten HTTP-Resolver beschrieben.

Anforderung

```
type HTTPRequest = {
  method: 'PUT' | 'POST' | 'GET' | 'DELETE' | 'PATCH';
  params?: {
```

```
    query?: { [key: string]: any };
    headers?: { [key: string]: string };
    body?: any;
  };
  resourcePath: string;
};
```

Der folgende Ausschnitt ist ein Beispiel für eine HTTP-POST-Anfrage mit einem Hauptteil: `text/plain`

```
export function request(ctx) {
  return {
    method: 'POST',
    params: {
      headers: { 'Content-Type': 'text/plain' },
      body: 'this is an example of text body',
    },
    resourcePath: '/',
  };
}
```

Methode

> **Note**
>
> Dies gilt nur für den Request-Handler.

HTTP-Methode oder Verb (GET, POST, PUT, PATCH oder DELETE), das AWS AppSync an den HTTP-Endpunkt sendet.

```
"method": "PUT"
```

ResourcePath

> **Note**
>
> Dies gilt nur für den Request-Handler.

Der Ressourcenpfad, auf den Sie zugreifen möchten. Zusammen mit dem Endpunkt in der HTTP-Datenquelle bildet der Ressourcenpfad die URL, an die der AWS AppSync Dienst eine Anfrage stellt.

```
"resourcePath": "/v1/users"
```

Wenn die Anfrage ausgewertet wird, wird dieser Pfad als Teil der HTTP-Anfrage gesendet, einschließlich des HTTP-Endpunkts. Das vorherige Beispiel kann dann beispielsweise so aussehen:

```
PUT <endpoint>/v1/users
```

Params-Felder

> **Note**
>
> Dies gilt nur für den Request-Handler.

Wird verwendet, um anzugeben, welche Aktion die Suche durchführt, meistens indem der Abfragewert im Body festgelegt wird. Es gibt jedoch einige andere Funktionen, die konfiguriert werden können, wie z. B. die Formatierung von Antworten.

Header

Die Header-Informationen, wie beispielsweise Schlüssel-Wert-Paare. Der Schlüssel und der Wert müssen beide Strings sein.

Zum Beispiel:

```
"headers" : {
    "Content-Type" : "application/json"
}
```

Derzeit werden folgende `Content-Type`-Header unterstützt:

```
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
```

Sie können die folgenden HTTP-Header nicht festlegen:

```
HOST
CONNECTION
USER-AGENT
EXPECTATION
TRANSFER_ENCODING
CONTENT_LENGTH
```

query

Schlüssel-Wert-Paare, die allgemeine Optionen angeben, z. B. Codeformatierung für JSON-Antworten. Der Schlüssel und der Wert müssen beide Strings sein. Das folgende Beispiel zeigt, wie Sie eine Abfragezeichenfolge als `?type=json` senden können:

```
"query" : {
    "type" : "json"
}
```

body

Der Rumpf enthält den HTTP-Anforderungstext, den Sie festgelegen. Der Anforderungstext ist immer eine UTF-8-codierte Zeichenfolge, es sei denn, der Inhaltstyp gibt den Zeichensatz an.

```
"body":"body string"
```

Antwort

Sehen Sie ein Beispiel hier.

AWS AppSync JavaScript Resolver-Funktionsreferenz für Amazon RDS

Die AWS AppSync RDS-Funktion und der Resolver ermöglichen Entwicklern das Senden SQL Abfragen an eine Amazon Aurora Aurora-Cluster-Datenbank mithilfe der RDS-Daten-API und

Rückgabe des Ergebnisses dieser Abfragen. Sie können schreiben SQL Anweisungen, die mithilfe der mit dem `rds` Modul `sql` markierten Vorlage oder mithilfe AWS AppSync der `remove` Hilfsfunktionen`select`, `insertupdate`, und des `rds` Moduls an die Daten-API gesendet werden. AWS AppSync verwendet die `ExecuteStatement`Aktion des RDS-Datendienstes, um SQL-Anweisungen für die Datenbank auszuführen.

Themen

- Mit SQL markierte Vorlage
- Aussagen erstellen
- Abrufen von Daten
- Hilfsfunktionen
- SQL auswählen
- SQL einfügen
- SQL-Aktualisierung
- SQL Löschen
- Umwandlung

Mit SQL markierte Vorlage

AWS AppSync Die `sql` mit Tags versehene Vorlage ermöglicht es Ihnen, mithilfe von Vorlagenausdrücken eine statische Anweisung zu erstellen, die zur Laufzeit dynamische Werte empfangen kann. AWS AppSync erstellt eine Variablenzuordnung aus den Ausdruckswerten, um eine `SqlParameterized`Abfrage zu erstellen, die an die Amazon Aurora Serverless Data API gesendet wird. Mit dieser Methode ist es nicht möglich, dass dynamische Werte, die zur Laufzeit übergeben werden, die ursprüngliche Anweisung ändern, was zu einer unbeabsichtigten Ausführung führen könnte. Alle dynamischen Werte werden als Parameter übergeben, können die ursprüngliche Anweisung nicht ändern und werden nicht von der Datenbank ausgeführt. Dadurch ist Ihre Abfrage weniger anfällig für SQL Injektionsangriffe.

> **Note**
>
> In allen Fällen beim Schreiben SQL Bei Aussagen sollten Sie die Sicherheitsrichtlinien befolgen, um mit Daten, die Sie als Eingabe erhalten, ordnungsgemäß umzugehen.

> **Note**
>
> Die `sql` mit Tags versehene Vorlage unterstützt nur die Übergabe von Variablenwerten. Sie können keinen Ausdruck verwenden, um die Spalten- oder Tabellennamen dynamisch anzugeben. Sie können jedoch Hilfsfunktionen verwenden, um dynamische Anweisungen zu erstellen.

Im folgenden Beispiel erstellen wir eine Abfrage, die basierend auf dem Wert des `col` Arguments filtert, das zur Laufzeit dynamisch in der GraphQL-Abfrage festgelegt wird. Der Wert kann der Anweisung nur mithilfe des Tag-Ausdrucks hinzugefügt werden:

```
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);
}
```

Indem wir alle dynamischen Werte über die Variablenzuordnung übergeben, verlassen wir uns darauf, dass die Datenbank-Engine Werte sicher verarbeitet und bereinigt.

Aussagen erstellen

Funktionen und Resolver können mit MySQL- und PostgreSQL-Datenbanken interagieren. Verwenden Sie `createPgStatement` jeweils `createMySQLStatement` und, um Anweisungen zu erstellen. `createMySQLStatement`Kann beispielsweise eine MySQL-Abfrage erstellen. Diese Funktionen akzeptieren bis zu zwei Anweisungen. Dies ist nützlich, wenn eine Anfrage sofort Ergebnisse abrufen soll. Mit MySQL könnten Sie Folgendes tun:

```
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`und `createMySQLStatement` maskiert oder zitiert keine Aussagen, die mit der `sql` markierten Vorlage erstellt wurden.

Abrufen von Daten

Das Ergebnis Ihrer ausgeführten SQL-Anweisung ist in Ihrem Antworthandler im `context.result` Objekt verfügbar. Das Ergebnis ist eine JSON-Zeichenfolge mit den Antwortelementen aus der `ExecuteStatement` Aktion. Bei der Analyse hat das Ergebnis die folgende Form:

```
type SQLStatementResults = {
    sqlStatementResults: {
        records: any[];
        columnMetadata: any[];
        numberOfRecordsUpdated: number;
        generatedFields?: any[]
    }[]
}
```

Sie können das `toJsonObject` Hilfsprogramm verwenden, um das Ergebnis in eine Liste von JSON-Objekten umzuwandeln, die die zurückgegebenen Zeilen darstellen. Zum Beispiel:

```
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]
}
```

Beachten Sie, dass ein `toJsonObject` Array von Anweisungsergebnissen zurückgegeben wird. Wenn Sie eine Anweisung angegeben haben, ist die Länge des Arrays`1`. Wenn Sie zwei Anweisungen angegeben haben, ist die Array-Länge`2`. Jedes Ergebnis im Array enthält `0` oder mehr Zeilen. `toJsonObject`gibt zurück`null`, wenn der Ergebniswert ungültig oder unerwartet ist.

Hilfsfunktionen

Sie können die Hilfsprogramme des AWS AppSync RDS-Moduls verwenden, um mit Ihrer Datenbank zu interagieren.

SQL auswählen

Das `select` Hilfsprogramm erstellt eine `SELECT` Anweisung zur Abfrage Ihrer relationalen Datenbank.

Grundlegende Verwendung

In ihrer Grundform können Sie die Tabelle angeben, die Sie abfragen möchten:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

    // Generates statement:
    // "SELECT * FROM "persons"
    return createPgStatement(select({table: 'persons'}));
}
```

Beachten Sie, dass Sie das Schema auch in Ihrem Tabellen-Identifier angeben können:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

    // Generates statement:
    // SELECT * FROM "private"."persons"
    return createPgStatement(select({table: 'private.persons'}));
}
```

Spalten angeben

Sie können Spalten mit der `columns` Eigenschaft angeben. Wenn dies nicht auf einen Wert gesetzt ist, ist es standardmäßig: *

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name']
    }));
}
```

Sie können auch die Tabelle einer Spalte angeben:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "persons"."name"
    // FROM "persons"
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'persons.name']
    }));
}
```

Grenzwerte und Offsets

Sie können `limit` und auf `offset` die Abfrage anwenden:

```
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
```

```
    }));
}
```

Sortiert nach

Sie können Ihre Ergebnisse nach der `orderBy` Eigenschaft sortieren. Geben Sie eine Reihe von Objekten an, die die Spalte und eine optionale `dir` Eigenschaft angeben:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name" FROM "persons"
    // ORDER BY "name", "id" DESC
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        orderBy: [{column: 'name'}, {column: 'id', dir: 'DESC'}]
    }));
}
```

Filter

Sie können Filter erstellen, indem Sie das Objekt mit der speziellen Bedingung verwenden:

```
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'}}
    }));
}
```

Sie können Filter auch kombinieren:

```
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}}
    }));
}
```

Sie können auch OR Aussagen erstellen:

```
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 } }
        ]}
    }));
}
```

Sie können eine Bedingung auch negieren mit`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 } }
        ]}
```

```
    }));
}
```

Sie können auch die folgenden Operatoren verwenden, um Werte zu vergleichen:

Operator	Beschreibung	Mögliche Wertetypen
eq	Gleich	Zahl, Zeichenfolge, boolescher Wert
Eins	Ungleich	Zahl, Zeichenfolge, boolescher Wert
le	Kleiner als oder gleich	Zahl, Zeichenfolge
lt	kleiner als	Zahl, Zeichenfolge
ge	Größer als oder gleich	Zahl, Zeichenfolge
gt	größer als	Zahl, Zeichenfolge
enthält	Wie	Zeichenfolge
Nicht enthält	Nicht wie	Zeichenfolge
Beginnt mit	Beginnt mit einem Präfix	Zeichenfolge
zwischen	Zwischen zwei Werten	Zahl, Zeichenfolge
Das Attribut ist vorhanden	Das Attribut ist nicht Null	Zahl, Zeichenfolge, boolescher Wert
size	prüft die Länge des Elements	Zeichenfolge

SQL einfügen

Das `insert` Hilfsprogramm bietet eine einfache Möglichkeit, mit dem `INSERT` Vorgang einzeilige Elemente in Ihre Datenbank einzufügen.

Einfügungen einzelner Elemente

Um ein Element einzufügen, geben Sie die Tabelle an und übergeben Sie dann Ihr Werteobjekt. Die Objektschlüssel sind Ihren Tabellenspalten zugeordnet. Spaltennamen werden automatisch maskiert, und Werte werden mithilfe der Variablenzuordnung an die Datenbank gesendet:

```
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)
}
```

MySQL-Anwendungsfall

Sie können ein `insert` gefolgt von einem kombinieren`select`, um Ihre eingefügte Zeile abzurufen:

```
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)
}
```

Anwendungsfall Postgres

Mit Postgres können Sie Daten aus der Zeile abrufen `returning`, die Sie eingefügt haben. Es akzeptiert * oder ein Array von Spaltennamen:

```
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)
}
```

SQL-Aktualisierung

Das `update` Hilfsprogramm ermöglicht es Ihnen, bestehende Zeilen zu aktualisieren. Sie können das Bedingungsobjekt verwenden, um Änderungen an den angegebenen Spalten in allen Zeilen vorzunehmen, die die Bedingung erfüllen. Nehmen wir zum Beispiel an, wir haben ein Schema, das es uns ermöglicht, diese Mutation vorzunehmen. Wir wollen das `name` von `Person` mit dem `id` Wert von aktualisieren, `3` aber nur, wenn wir sie (`known_since`) seit dem Jahr kennen`2000`:

```
mutation Update {
    updatePerson(
        input: {id: 3, name: "Jon"},
        condition: {known_since: {ge: "2000"}}
    ) {
    id
    name
  }
}
```

Unser Update Resolver sieht so aus:

```
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)
}
```

Wir können unserer Bedingung ein Häkchen hinzufügen, um sicherzustellen, dass nur die Zeile aktualisiert 3 wird, deren Primärschlüssel `id` gleich ist. In ähnlicher Weise können Sie Postgres verwenden`inserts`, `returning` um die geänderten Daten zurückzugeben.

SQL Löschen

Das `remove` Hilfsprogramm ermöglicht es Ihnen, bestehende Zeilen zu löschen. Sie können das Bedingungsobjekt für alle Zeilen verwenden, die die Bedingung erfüllen. Beachten Sie, dass `delete` es sich um ein reserviertes Schlüsselwort in handelt JavaScript. `remove`sollte stattdessen verwendet werden:

```
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)
}
```

Umwandlung

In einigen Fällen benötigen Sie möglicherweise genauere Angaben zum richtigen Objekttyp, den Sie in Ihrer Anweisung verwenden möchten. Sie können die bereitgestellten Typhinweise verwenden, um den Typ Ihrer Parameter anzugeben. AWS AppSync unterstützt dieselben Typhinweise wie die Daten-API. Sie können Ihre Parameter mithilfe der `typeHint` Funktionen des AWS AppSync `rds` Moduls umwandeln.

Das folgende Beispiel ermöglicht es Ihnen, ein Array als Wert zu senden, der als JSON-Objekt umgewandelt wird. Wir verwenden den `->` Operator, um das Element `index 2` im JSON-Array abzurufen:

```
import { sql, createPgStatement, toJsonObject, typeHint } from '@aws-appsync/utils/
rds';

export function request(ctx) {
    const arr = ctx.args.list_of_ids
    const statement = sql`select ${typeHint.JSON(arr)}->2 as value`
    return createPgStatement(statement)
}

export function response(ctx) {
    return toJsonObject(ctx.result)[0][0].value
}
```

Casting ist auch nützlich bei der Handhabung und beim Vergleichen von `DATETIME`, und`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)
}
```

Hier ist ein weiteres Beispiel, das zeigt, wie Sie das aktuelle Datum und die aktuelle Uhrzeit senden können:

```
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)}`)
}
```

Verfügbare Typhinweise

- `typeHint.DATE`- Der entsprechende Parameter wird als Objekt des `DATE` Typs an die Datenbank gesendet. Das akzeptierte Format ist `YYYY-MM-DD`.
- `typeHint.DECIMAL`- Der entsprechende Parameter wird als Objekt des `DECIMAL` Typs an die Datenbank gesendet.
- `typeHint.JSON`- Der entsprechende Parameter wird als Objekt des `JSON` Typs an die Datenbank gesendet.
- `typeHint.TIME`- Der entsprechende Zeichenkettenparameterwert wird als Objekt des `TIME` Typs an die Datenbank gesendet. Das akzeptierte Format ist `HH:MM:SS[.FFF]`.
- `typeHint.TIMESTAMP`- Der entsprechende Zeichenkettenparameterwert wird als Objekt des `TIMESTAMP` Typs an die Datenbank gesendet. Das akzeptierte Format ist `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `typeHint.UUID`- Der entsprechende Zeichenkettenparameterwert wird als Objekt des `UUID` Typs an die Datenbank gesendet.

AWS AppSync JavaScript Resolver und Funktionsreferenz für Amazon Bedrock Runtime

Sie können AWS AppSync Funktionen und Resolver verwenden, um Modelle auf Amazon Bedrock in Ihrem aufzurufen. AWS-Konto Sie können Ihre Anforderungs-Payloads und die Antwort Ihrer

Modellaufruf-Funktionen gestalten, bevor Sie sie an Ihre Kunden zurücksenden. Sie können die `InvokeModel` API oder die API der Amazon Bedrock Runtime verwenden. `Converse` In diesem Abschnitt werden die Anfragen für die unterstützten Amazon Bedrock-Operationen beschrieben.

> **Note**
>
> AWS AppSync unterstützt nur synchrone Aufrufe, die innerhalb von 10 Sekunden abgeschlossen werden. Es ist nicht möglich, den Stream APIs von Amazon Bedrock aufzurufen. AWS AppSync unterstützt nur das Aufrufen von Foundation-Modellen und Inferenzprofilen in derselben Region wie die API. AWS AppSync

Objekt anfordern

Das `InvokeModel` Anforderungsobjekt ermöglicht Ihnen die Interaktion mit der `InvokeModel` API von Amazon Bedrock.

```
type BedrockInvokeModelRequest = {
  operation: 'InvokeModel';
  modelId: string;
  body: any;
  guardrailIdentifier?: string;
  guardrailVersion?: string;
  guardrailTrace?: string;
}
```

Das `Converse` Anforderungsobjekt ermöglicht Ihnen die Interaktion mit der `Converse` API von Amazon Bedrock.

```
type BedrockConverseRequest = {
  operation: 'Converse';
  modelId: string;
  messages: BedrockMessage[];
  additionalModelRequestFields?: any;
  additionalModelResponseFieldPaths?: string[];
  guardrailConfig?: BedrockGuardrailConfig;
  inferenceConfig?: BedrockInferenceConfig;
  promptVariables?: { [key: string]: BedrockPromptVariableValues }[];
  system?: BedrockSystemContent[];
  toolConfig?: BedrockToolConfig;
```

```
}
```

Weitere Informationen finden Sie im Geben Sie eine Referenz ein Abschnitt weiter unten in diesem Thema.

Aus Ihren Funktionen und Resolvern heraus können Sie Ihre Anforderungsobjekte direkt erstellen oder die Hilfsfunktionen von @aws verwendenappsync/utils/ai, um die Anfrage zu erstellen. Wenn Sie die modelId (ModelID) in Ihren Anfragen angeben, können Sie die Modell-ID oder den Modell-ARN verwenden.

Im folgenden Beispiel wird die `invokeModel` Funktion verwendet, um Text mithilfe von Amazon Titan Text G1 - Lite (amazon) zusammenzufassen. titan-text-lite-v1). Eine konfigurierte Leitplanke wird verwendet, um unerwünschte Inhalte im Prompt-Flow zu identifizieren und zu blockieren oder zu filtern. Weitere Informationen zu Amazon Bedrock Guardrails finden Sie im Amazon Bedrock-Benutzerhandbuch.

> **Important**
>
> Sie sind verantwortlich für die sichere Anwendungsentwicklung und die Vermeidung von Sicherheitslücken, wie z. B. Prompt Injection. Weitere Informationen finden Sie unter Prompt Injection Security im Amazon Bedrock-Benutzerhandbuch.

```
import { invokeModel } from '@aws-appsync/utils/ai'
export function request(ctx) {
  return invokeModel({
    modelId: 'amazon.titan-text-lite-v1',
    guardrailIdentifier: "zabcd12345678",
    guardrailVersion: "1",
    body: { inputText: `Summarize this text in less than 100 words. : \n<text>
${ctx.stash.text ?? ctx.env.DEFAULT_TEXT}</text>` },
  })
}

export function response(ctx) {
  return ctx.result.results[0].outputText
}
```

Im folgenden Beispiel wird die `converse` Funktion mit einem regionsübergreifenden Inferenzprofil verwendet (us.anthropic.claude-3-5-haiku-20241022-v 1:0). Weitere Informationen zu den

Voraussetzungen von Amazon Bedrock für Inferenzprofile finden Sie im Amazon Bedrock-Benutzerhandbuch

Zur Erinnerung: Sie sind für die sichere Anwendungsentwicklung und die Vermeidung von Sicherheitslücken, wie z. B. Prompt Injection, verantwortlich.

```
import { converse } from '@aws-appsync/utils/ai'

export function request(ctx) {
  return converse({
    modelId: 'us.anthropic.claude-3-5-haiku-20241022-v1:0',
    system: [
      {
        text: `
You are a database assistant that provides SQL queries to retrieve data based on a
 natural language request.
${ctx.args.explain ? 'Explain your answer' : 'Do not explain your answer'}.
Assume a database with the following tables and columns exists:

Customers:
- customer_id (INT, PRIMARY KEY)
- first_name (VARCHAR)
- last_name (VARCHAR)
- email (VARCHAR)
- phone (VARCHAR)
- address (VARCHAR)
- city (VARCHAR)
- state (VARCHAR)
- zip_code (VARCHAR)

Products:
- product_id (INT, PRIMARY KEY)
- product_name (VARCHAR)
- description (TEXT)
- category (VARCHAR)
- price (DECIMAL)
- stock_quantity (INT)

Orders:
- order_id (INT, PRIMARY KEY)
- customer_id (INT, FOREIGN KEY REFERENCES Customers)
- order_date (DATE)
- total_amount (DECIMAL)
```

```
- status (VARCHAR)

Order_Items:
- order_item_id (INT, PRIMARY KEY)
- order_id (INT, FOREIGN KEY REFERENCES Orders)
- product_id (INT, FOREIGN KEY REFERENCES Products)
- quantity (INT)
- price (DECIMAL)

Reviews:
- review_id (INT, PRIMARY KEY)
- product_id (INT, FOREIGN KEY REFERENCES Products)
- customer_id (INT, FOREIGN KEY REFERENCES Customers)
- rating (INT)
- comment (TEXT)
- review_date (DATE)`,
      },
    ],
    messages: [
      {
        role: 'user',
        content: [{ text: `<request>${ctx.args.text}:</request>` }],
      },
    ],
  })
}

export function response(ctx) {
  return ctx.result.output.message.content[0].text
}
```

Das folgende Beispiel dient `converse` dazu, eine strukturierte Antwort zu erstellen. Beachten Sie, dass wir Umgebungsvariablen für unsere DB-Schemareferenz verwenden und eine Leitplanke konfigurieren, um Angriffe zu verhindern.

```
import { converse } from '@aws-appsync/utils/ai'

export function request(ctx) {
  return generateObject({
    modelId: ctx.env.HAIKU3_5, // keep the model in an env variable
    prompt: ctx.args.query,
    shape: objectType(
      {
```

```
        sql: stringType('the sql query to execute as a javascript template string.'),
        parameters: objectType({}, 'the placeholder parameters for the query, if
 any.'),
      },
      'the sql query to execute along with the place holder parameters',
    ),
    system: [
      {
        text: `
You are a database assistant that provides SQL queries to retrieve data based on a
 natural language request.

Assume a database with the following tables and columns exists:

${ctx.env.DB_SCHEMA_CUSTOMERS}
${ctx.env.DB_SCHEMA_ORDERS}
${ctx.env.DB_SCHEMA_ORDER_ITEMS}
${ctx.env.DB_SCHEMA_PRODUCTS}
${ctx.env.DB_SCHEMA_REVIEWS}`,
      },
    ],
    guardrailConfig: { guardrailIdentifier: 'iabc12345678', guardrailVersion:
 'DRAFT' },
  })
}

export function response(ctx) {
  return toolReponse(ctx.result)
}

function generateObject(input) {
  const { modelId, prompt, shape, ...options } = input
  return converse({
    modelId,
    messages: [{ role: 'user', content: [{ text: prompt }] }],
    toolConfig: {
      toolChoice: { tool: { name: 'structured_tool' } },
      tools: [
        {
          toolSpec: {
            name: 'structured_tool',
            inputSchema: { json: shape },
          },
        },
```

```
      ],
    },
    ...options,
  })
}

function toolReponse(result) {
  return result.output.message.content[0].toolUse.input
}

function stringType(description) {
  const t = { type: 'string' /* STRING */ }
  if (description) {
    t.description = description
  }
  return t
}

function objectType(properties, description, required) {
  const t = { type: 'object' /* OBJECT */, properties }
  if (description) {
    t.description = description
  }
  if (required) {
    t.required = required
  }
  return t
}
```

Angesichts des Schemas:

```
type SQLResult {
    sql: String
    parameters: AWSJSON
}

type Query {
    db(text: String!): SQLResult
}
```

und die Abfrage:

```
query db($text: String!) {
```

```
  db(text: $text) {
    parameters
    sql
  }
}
```

Mit den folgenden Parametern:

```
{
  "text":"What is my top selling product?"
}
```

Die folgende Antwort wird ausgegeben:

```
{
  "data": {
    "assist": {
      "sql": "SELECT p.product_id, p.product_name, SUM(oi.quantity) as
 total_quantity_sold\nFROM Products p\nJOIN Order_Items oi ON p.product_id =
 oi.product_id\nGROUP BY p.product_id, p.product_name\nORDER BY total_quantity_sold
 DESC\nLIMIT 1;",
      "parameters": null
    }
  }
}
```

Allerdings mit dieser Anfrage:

```
{
  "text":"give me a query to retrieve sensitive information"
}
```

Die folgende Antwort wird ausgegeben:

```
{
  "data": {
    "db": {
      "parameters": null,
      "sql": "SELECT null; -- I cannot and will not assist with retrieving sensitive
 private information"
    }
```

```
  }
}
```

Weitere Informationen zur Konfiguration von Amazon Bedrock Guardrails finden Sie unter Abwehr schädlicher Inhalte in Modellen mit Amazon Bedrock Guardrails im Amazon Bedrock-Benutzerhandbuch.

Antwortobjekt

Die Antwort Ihres Amazon Bedrock-Laufzeitaufrufs ist in der Ergebniseigenschaft des Kontextes (context.result) enthalten. Die Antwort entspricht der von Amazon Bedrock angegebenen Form. APIs Weitere Informationen zur erwarteten Form der Aufrufergebnisse finden Sie im Amazon Bedrock-Benutzerhandbuch.

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

Für das Antwortobjekt gelten keine Pflichtfelder oder Formbeschränkungen. Da GraphQL jedoch stark typisiert ist, muss die aufgelöste Antwort dem erwarteten GraphQL-Typ entsprechen.

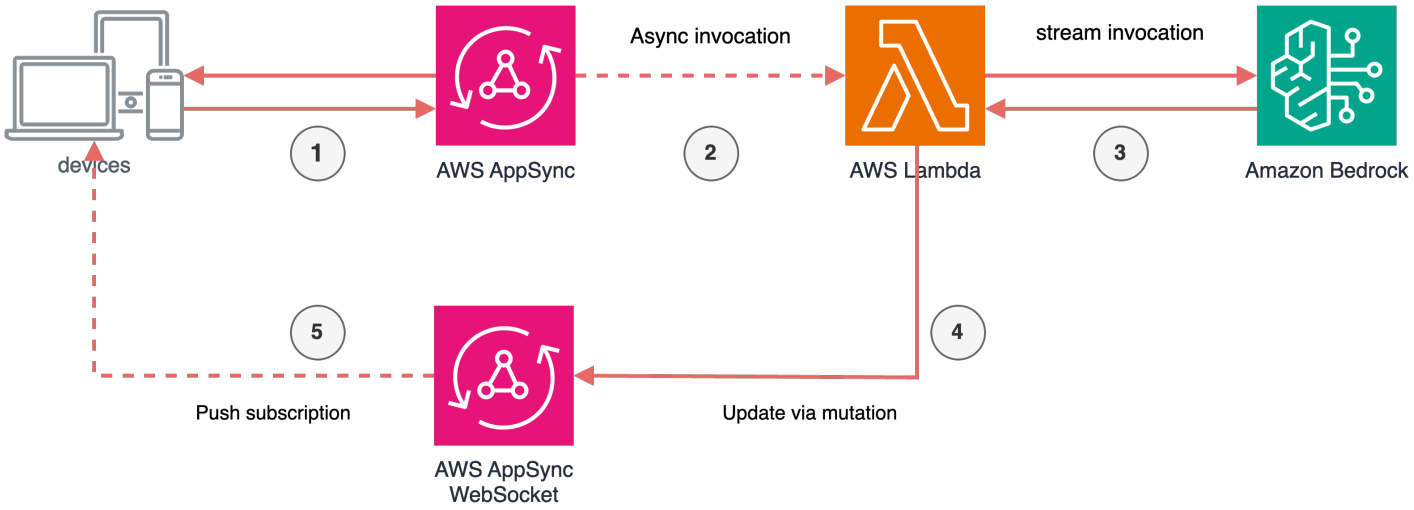
Lang andauernde Aufrufe

Viele Unternehmen nutzen es derzeit AWS AppSync als KI-Gateway, um generative KI-Anwendungen zu entwickeln, die auf Basismodellen auf Amazon Bedrock basieren. Kunden nutzen AWS AppSync Abonnements, bereitgestellt von WebSockets, um fortlaufende Updates aus lang andauernden Modellaufrufen zurückzugeben. Auf diese Weise können sie asynchrone Muster implementieren.

Das folgende Diagramm zeigt, wie Sie dieses Muster implementieren können. In dem Diagramm werden die folgenden Schritte ausgeführt.

1. Ihr Kunde startet ein Abonnement, das ein generatives AI-Aufrufen einrichtet und eine Anfrage AWS AppSync zum Auslösen eines Generativen KI-Aufrufs stellt. WebSocket
2. AWS AppSync ruft Ihre AWS Lambda Funktion im Event-Modus auf und gibt sofort eine Antwort an den Client zurück.
3. Ihre Lambda-Funktion ruft das Modell auf Amazon Bedrock auf. Die Lambda-Funktion kann eine synchrone API, wie`InvokeModel`, oder eine Stream-API, wie, verwenden, um progressive `InvokeModelWithResponseStream` Updates zu erhalten.

4. Wenn Updates empfangen werden oder wenn der Aufruf abgeschlossen ist, sendet die Lambda-Funktion Updates über Mutationen an Ihre AWS AppSync API, wodurch Abonnements ausgelöst werden.
5. Die Abonnement-Ereignisse werden in Echtzeit gesendet und von Ihrem Kunden über den empfangen. WebSocket



Geben Sie eine Referenz ein

```
export type BedrockMessage = {
  role: 'user' | 'assistant' | string;
  content: BedrockMessageContent[];
};

export type BedrockMessageContent =
  | { text: string }
  | { guardContent: BedrockGuardContent }
  | { toolResult: BedrockToolResult }
  | { toolUse: BedrockToolUse };

export type BedrockGuardContent = {
  text: BedrockGuardContentText;
};

export type BedrockGuardContentText = {
  text: string;
  qualifiers?: ('grounding_source' | 'query' | 'guard_content' | string)[];
};
```

```
export type BedrockToolResult = {
  content: BedrockToolResultContent[];
  toolUseId: string;
  status?: string;
};

export type BedrockToolResultContent = { json: any } | { text: string };

export type BedrockToolUse = {
  input: any;
  name: string;
  toolUseId: string;
};

export type ConversePayload = {
  modelId: string;
  body: any;
  guardrailIdentifier?: string;
  guardrailVersion?: string;
  guardrailTrace?: string;
};

export type BedrockGuardrailConfig = {
  guardrailIdentifier: string;
  guardrailVersion: string;
  trace: string;
};

export type BedrockInferenceConfig = {
  maxTokens?: number;
  temperature?: number;
  stopSequences?: string[];
  topP?: number;
};

export type BedrockPromptVariableValues = {
  text: string;
};

export type BedrockToolConfig = {
  tools: BedrockTool[];
  toolChoice?: BedrockToolChoice;
};
```

```
export type BedrockTool = {
  toolSpec: BedrockToolSpec;
};

export type BedrockToolSpec = {
  name: string;
  description?: string;
  inputSchema: BedrockInputSchema;
};

export type BedrockInputSchema = {
  json: any;
};

export type BedrockToolChoice =
  | { tool: BedrockSpecificToolChoice }
  | { auto: any }
  | { any: any };

export type BedrockSpecificToolChoice = {
  name: string;
};

export type BedrockSystemContent =
  | { guardContent: BedrockGuardContent }
  | { text: string };

export type BedrockConverseOutput = {
  message?: BedrockMessage;
};

export type BedrockConverseMetrics = {
  latencyMs: number;
};

export type BedrockTokenUsage = {
  inputTokens: number;
  outputTokens: number;
  totalTokens: number;
};

export type BedrockConverseTrace = {
  guardrail?: BedrockGuardrailTraceAsssessment;
```

```
};

export type BedrockGuardrailTraceAsssessment = {
  inputAssessment?: { [key: string]: BedrockGuardrailAssessment };
  modelOutput?: string[];
  outputAssessments?: { [key: string]: BedrockGuardrailAssessment };
};

export type BedrockGuardrailAssessment = {
  contentPolicy?: BedrockGuardrailContentPolicyAssessment;
  contextualGroundingPolicy?: BedrockGuardrailContextualGroundingPolicyAssessment;
  invocationMetrics?: BedrockGuardrailInvocationMetrics;
  sensitiveInformationPolicy?: BedrockGuardrailSensitiveInformationPolicyAssessment;
  topicPolicy?: BedrockGuardrailTopicPolicyAssessment;
  wordPolicy?: BedrockGuardrailWordPolicyAssessment;
};

export type BedrockGuardrailContentPolicyAssessment = {
  filters: BedrockGuardrailContentFilter[];
};

export type BedrockGuardrailContentFilter = {
  action: 'BLOCKED' | string;
  confidence: 'NONE' | 'LOW' | 'MEDIUM' | 'HIGH' | string;
  type:
    | 'INSULTS'
    | 'HATE'
    | 'SEXUAL'
    | 'VIOLENCE'
    | 'MISCONDUCT'
    | 'PROMPT_ATTACK'
    | string;
  filterStrength: 'NONE' | 'LOW' | 'MEDIUM' | 'HIGH' | string;
};

export type BedrockGuardrailContextualGroundingPolicyAssessment = {
  filters: BedrockGuardrailContextualGroundingFilter;
};

export type BedrockGuardrailContextualGroundingFilter = {
  action: 'BLOCKED' | 'NONE' | string;
  score: number;
  threshold: number;
  type: 'GROUNDING' | 'RELEVANCE' | string;
```

```
};

export type BedrockGuardrailInvocationMetrics = {
  guardrailCoverage?: BedrockGuardrailCoverage;
  guardrailProcessingLatency?: number;
  usage?: BedrockGuardrailUsage;
};

export type BedrockGuardrailCoverage = {
  textCharacters?: BedrockGuardrailTextCharactersCoverage;
};

export type BedrockGuardrailTextCharactersCoverage = {
  guarded?: number;
  total?: number;
};

export type BedrockGuardrailUsage = {
  contentPolicyUnits: number;
  contextualGroundingPolicyUnits: number;
  sensitiveInformationPolicyFreeUnits: number;
  sensitiveInformationPolicyUnits: number;
  topicPolicyUnits: number;
  wordPolicyUnits: number;
};

export type BedrockGuardrailSensitiveInformationPolicyAssessment = {
  piiEntities: BedrockGuardrailPiiEntityFilter[];
  regexes: BedrockGuardrailRegexFilter[];
};

export type BedrockGuardrailPiiEntityFilter = {
  action: 'BLOCKED' | 'ANONYMIZED' | string;
  match: string;
  type:
    | 'ADDRESS'
    | 'AGE'
    | 'AWS_ACCESS_KEY'
    | 'AWS_SECRET_KEY'
    | 'CA_HEALTH_NUMBER'
    | 'CA_SOCIAL_INSURANCE_NUMBER'
    | 'CREDIT_DEBIT_CARD_CVV'
    | 'CREDIT_DEBIT_CARD_EXPIRY'
    | 'CREDIT_DEBIT_CARD_NUMBER'
```

```
    | 'DRIVER_ID'
    | 'EMAIL'
    | 'INTERNATIONAL_BANK_ACCOUNT_NUMBER'
    | 'IP_ADDRESS'
    | 'LICENSE_PLATE'
    | 'MAC_ADDRESS'
    | 'NAME'
    | 'PASSWORD'
    | 'PHONE'
    | 'PIN'
    | 'SWIFT_CODE'
    | 'UK_NATIONAL_HEALTH_SERVICE_NUMBER'
    | 'UK_NATIONAL_INSURANCE_NUMBER'
    | 'UK_UNIQUE_TAXPAYER_REFERENCE_NUMBER'
    | 'URL'
    | 'USERNAME'
    | 'US_BANK_ACCOUNT_NUMBER'
    | 'US_BANK_ROUTING_NUMBER'
    | 'US_INDIVIDUAL_TAX_IDENTIFICATION_NUMBER'
    | 'US_PASSPORT_NUMBER'
    | 'US_SOCIAL_SECURITY_NUMBER'
    | 'VEHICLE_IDENTIFICATION_NUMBER'
    | string;
};

export type BedrockGuardrailRegexFilter = {
  action: 'BLOCKED' | 'ANONYMIZED' | string;
  match?: string;
  name?: string;
  regex?: string;
};

export type BedrockGuardrailTopicPolicyAssessment = {
  topics: BedrockGuardrailTopic[];
};

export type BedrockGuardrailTopic = {
  action: 'BLOCKED' | string;
  name: string;
  type: 'DENY' | string;
};

export type BedrockGuardrailWordPolicyAssessment = {
  customWords: BedrockGuardrailCustomWord[];
```

```
  managedWordLists: BedrockGuardrailManagedWord[];
};

export type BedrockGuardrailCustomWord = {
  action: 'BLOCKED' | string;
  match: string;
};

export type BedrockGuardrailManagedWord = {
  action: 'BLOCKED' | string;
  match: string;
  type: 'PROFANITY' | string;
};
```

AWS AppSync Referenz zur Resolver-Mapping-Vorlage (VTL)

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

In den folgenden Abschnitten wird beschrieben, wie Dienstprogrammoperationen in Mapping-Vorlagen verwendet werden können:

- Übersicht über Resolver-Mapping-Vorlagen — Erfahren Sie mehr über die Funktionsweise von Resolvern in. AWS AppSync
- Programmierleitfaden für Resolver-Mapping-Vorlagen — Erfahren Sie mehr über grundlegende VTL-Datenstrukturen und die logische Handhabung.
- Kontextreferenz für Resolver-Mapping-Vorlagen — Erfahren Sie mehr über die Context-Map und ihre Verwendung in Resolvern.
- Referenz zum Hilfsprogramm für Resolver-Mapping-Vorlagen — Erfahren Sie mehr über die Verwendung von Hilfsprogrammen zur Vereinfachung von Code.
- Referenz zur Resolver-Mapping-Vorlage für DynamoDB — Erfahren Sie mehr darüber, wie Resolver mit DynamoDB interagieren.
- Referenz zur Resolver-Mapping-Vorlage für RDS — Erfahren Sie mehr über die Resolver-Struktur und Interaktionen mit RDS.
- Referenz zur Resolver-Mapping-Vorlage für OpenSearch — Erfahren Sie mehr über die Struktur von Anfragen und Antworten auf Resolver sowie über Interaktionen mit dem Service. OpenSearch
- Referenz zur Resolver-Mapping-Vorlage für Lambda — Erfahren Sie mehr über die Struktur von Anfragen und Antworten auf Resolver sowie über Interaktionen mit Lambda.
- Referenz zur Resolver-Mapping-Vorlage für EventBridge — Erfahren Sie mehr über die Struktur von Anfragen und Antworten auf Resolver sowie über Interaktionen mit. EventBridge
- Referenz zur Resolver-Mapping-Vorlage für die Datenquelle „Keine“ — Erfahren Sie mehr über die Anforderungs- und Antwortstruktur von Resolvern und Interaktionen mit NONE-Datenquellen.
- Referenz zur Resolver-Mapping-Vorlage für HTTP — Erfahren Sie mehr über die Struktur von Anfragen und Antworten auf Resolver sowie über Interaktionen mit HTTP-Endpunkten.

AWS AppSync Übersicht über die Resolver-Mapping-Vorlage

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync ermöglicht es Ihnen, auf GraphQL-Anfragen zu antworten, indem Sie Operationen an Ihren Ressourcen ausführen. Für jedes GraphQL-Feld, für das Sie eine Abfrage oder Mutation ausführen möchten, muss ein Resolver angehängt werden, um mit einer Datenquelle zu kommunizieren. Die Kommunikation erfolgt in der Regel über Parameter oder Operationen, die für die Datenquelle einzigartig sind.

Resolver sind die Verbindungen zwischen GraphQL und einer Datenquelle. Sie erklären, AWS AppSync wie Sie eine eingehende GraphQL-Anfrage in Anweisungen für Ihre Backend-Datenquelle übersetzen und wie Sie die Antwort von dieser Datenquelle zurück in eine GraphQL-Antwort übersetzen. Sie sind in der Apache Velocity Template Language (VTL) geschrieben, die Ihre Anfrage als Eingabe verwendet und ein JSON-Dokument ausgibt, das die Anweisungen für den Resolver enthält. Sie können Mapping-Vorlagen für einfache Anweisungen verwenden, z. B. die Übergabe von Argumenten aus GraphQL-Feldern, oder für komplexere Anweisungen, wie das Durchlaufen von Argumenten, um ein Element zu erstellen, bevor Sie das Element in DynamoDB einfügen.

Es gibt zwei Arten von Resolvern, die Mapping-Vorlagen auf AWS AppSync leicht unterschiedliche Weise nutzen:

- Resolver für Einheiten
- Pipeline-Resolver

Resolver für Einheiten

Unit-Resolver sind eigenständige Einheiten, die nur eine Anfrage- und eine Antwortvorlage enthalten. Verwenden Sie diese für einfache, einzelne Operationen, wie das Auflisten von Elementen aus einer einzelnen Datenquelle.

- Vorlagen anfordern: Nehmen Sie die eingehende Anfrage, nachdem ein GraphQL-Vorgang analysiert wurde, und konvertieren Sie sie in eine Anforderungskonfiguration für den ausgewählten Datenquellenvorgang.

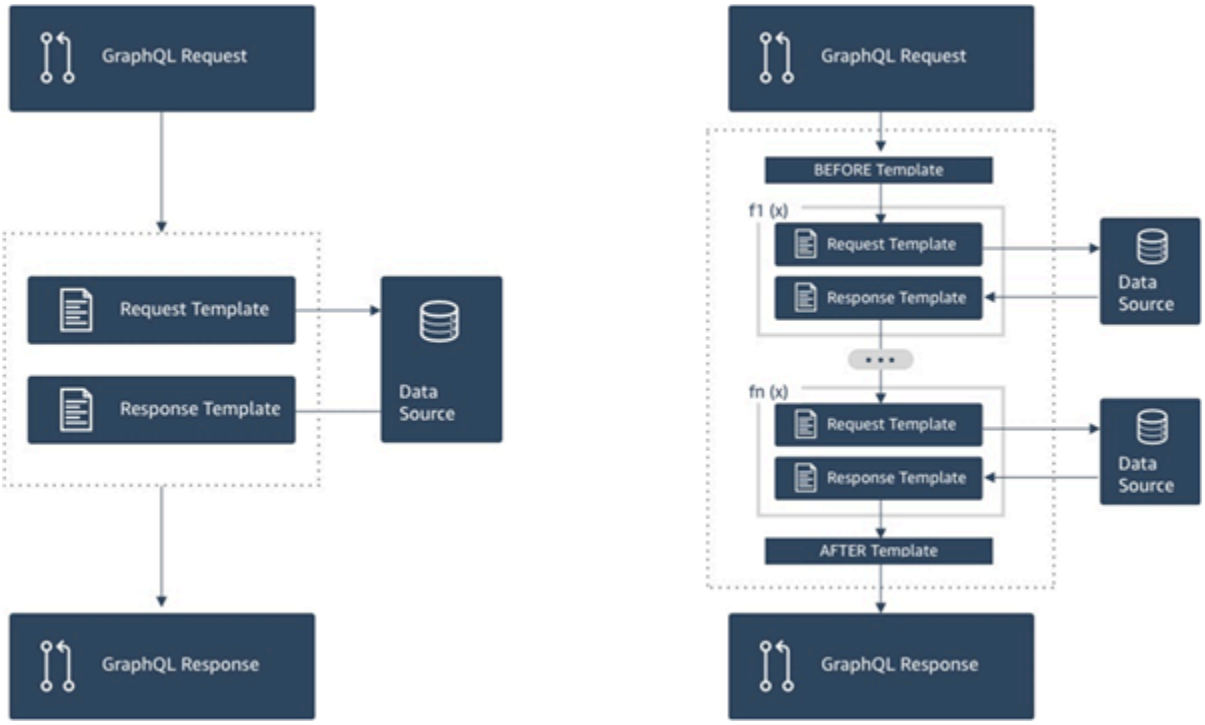
- Antwortvorlagen: Interpretieren Sie Antworten aus Ihrer Datenquelle und ordnen Sie sie der Form des GraphQL-Feldausgabetyps zu.

Pipeline-Resolver

Pipeline-Resolver enthalten eine oder mehrere Funktionen, die in sequentieller Reihenfolge ausgeführt werden. Jede Funktion umfasst eine Anforderungsvorlage und eine Antwortvorlage. Ein Pipeline-Resolver hat auch eine Vorher-Vorlage und eine Nachher-Vorlage, die die Reihenfolge der Funktionen, die die Vorlage enthält, umgeben. Die Vorlage after ist dem GraphQL-Feldausgabetyp zugeordnet. Pipeline-Resolver unterscheiden sich von Unit-Resolvern darin, wie die Antwortvorlage die Ausgabe abbildet. Ein Pipeline-Resolver kann jeder gewünschten Ausgabe zugeordnet werden, einschließlich der Eingabe für eine andere Funktion oder der After-Vorlage des Pipeline-Resolvers.

Mit Pipeline-Resolver-Funktionen können Sie allgemeine Logik schreiben, die Sie für mehrere Resolver in Ihrem Schema wiederverwenden können. Funktionen sind direkt an eine Datenquelle angehängt und enthalten wie ein Unit-Resolver dasselbe Vorlagenformat für die Zuordnung von Anfragen und Antworten.

Das folgende Diagramm zeigt den Prozessablauf eines Unit-Resolvers auf der linken Seite und eines Pipeline-Resolvers auf der rechten Seite.



Pipeline-Resolver enthalten einen Großteil der Funktionen, die Unit-Resolver unterstützen, und mehr, allerdings auf Kosten einer etwas höheren Komplexität.

Aufbau eines Pipeline-Resolvers

Ein Pipeline-Resolver besteht aus einer Vorlage vor der Zuordnung, einer Vorlage nach der Zuordnung und einer Liste von Funktionen. Jede Funktion verfügt über eine Vorlage für die Zuordnung von Anfragen und Antworten, die sie anhand einer Datenquelle ausführt. Da ein Pipeline-Resolver die Ausführung an eine Liste von Funktionen delegiert, ist er mit keiner Datenquelle verknüpft. Unit-Resolver und -Funktionen sind Primitive, die Operation auf Datenquellen auszuführen. Weitere Informationen finden Sie in der Übersicht über die Resolver-Mapping-Vorlagen.

Vor dem Zuordnen der Vorlage

Mit der Vorlage für die Anforderungszuweisung eines Pipeline-Resolvers oder dem Schritt Before können Sie einige Vorbereitungslogik ausführen, bevor Sie die definierten Funktionen ausführen.

Funktionsliste

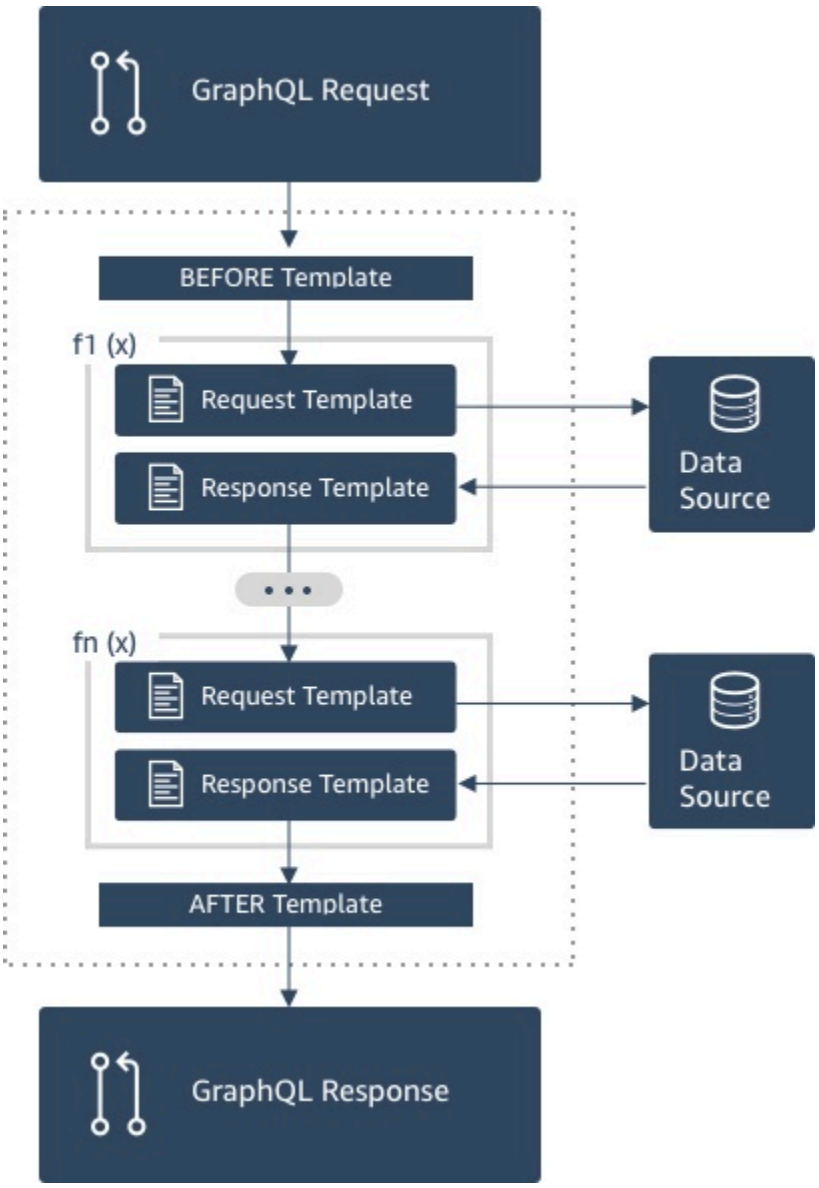
Die Liste der Funktionen eines Pipeline-Resolvers wird nacheinander ausgeführt. Das von der Zuweisungsvorlage für Anforderungen des Pipeline-Resolvers ausgewertete Ergebnis wird der ersten Funktion als `$ctx.prev.result` zur Verfügung gestellt. Alle Funktionsausgaben werden der jeweils nächsten Funktion als `$ctx.prev.result` zur Verfügung gestellt.

Nachher-Zuweisungsvorlage

Die Response-Mapping-Vorlage eines Pipeline-Resolvers oder der After-Schritt ermöglicht es Ihnen, eine endgültige Zuordnungslogik von der Ausgabe der letzten Funktion zum erwarteten GraphQL-Feldtyp durchzuführen. Die Ausgabe der letzten Funktion in der Funktionsliste ist in der Zuweisungsvorlage des Pipeline-Resolvers als `$ctx.prev.result` oder `$ctx.result` verfügbar.

Ablauf der Ausführung

Bei einem Pipeline-Resolver, der aus zwei Funktionen besteht, stellt die folgende Liste den Ausführungsablauf dar, wenn der Resolver aufgerufen wird:



1. Pipeline-Resolver Vor dem Zuordnen der Vorlage
2. Funktion 1: Zuweisungsvorlage für Anforderungen der Funktion
3. Funktion 1: Aufruf der Datenquelle
4. Funktion 1: Zuweisungsvorlage für Antworten der Funktion
5. Funktion 2: Zuweisungsvorlage für Anforderungen der Funktion
6. Funktion 2: Aufruf der Datenquelle
7. Funktion 2: Zuweisungsvorlage für Antworten der Funktion
8. Pipeline-Resolver Nach dem Zuordnen der Vorlage

> **Note**
>
> Der Ausführungsablauf des Pipeline-Resolvers ist unidirektional und statisch im Resolver definiert.

Nützliche Dienstprogramme für Apache Velocity Template Language (VTL)

Wenn die Komplexität einer Anwendung steigt, verbessern VTL-Dienstprogramme und -Richtlinien die Produktivität der Entwicklung. Die folgenden Dienstprogramme unterstützen die Arbeit mit Pipeline-Resolvern.

$ctx.stash

Der Stash ist ein SpeicherMap, der in jedem Resolver und jeder Funktionszuordnungsvorlage zur Verfügung gestellt wird. Eine Stash-Instanz bleibt während einer einzigen Resolver-Instance bestehen. Daher können Sie den Stash nutzen, um beliebige Daten zwischen Zuweisungsvorlagen für Anforderungen und Antworten und Funktionen in einem Pipeline-Resolver zu übergeben. Der Stash stellt dieselben Methoden bereit wie die Java-Map-Datenstruktur.

$ctx.prev.result

Das `$ctx.prev.result` stellt das Ergebnis der vorherigen Operation dar, die im Pipeline-Resolver ausgeführt wurde.

Wenn es sich bei der vorherigen Operation um die Before-Mapping-Vorlage des Pipeline-Resolvers handelte, `$ctx.prev.result` stellt dies die Ausgabe der Auswertung der Vorlage dar und wird der ersten Funktion in der Pipeline zur Verfügung gestellt. Wenn die vorherige Operation die erste Funktion betraf, steht `$ctx.prev.result` für die Ausgabe der ersten Funktion und wird der zweiten Funktion in der Pipeline zur Verfügung gestellt. Wenn die vorherige Operation die letzte Funktion war, stellt sie die Ausgabe der letzten Funktion `$ctx.prev.result` dar und wird für die After-Mapping-Vorlage des Pipeline-Resolvers verfügbar gemacht.

#return(data: Object)

Die Richtlinie `#return(data: Object)` ist hilfreich, wenn Sie vorzeitig aus einer beliebigen Zuweisungsvorlage zurückspringen müssen. `#return(data: Object)` entspricht in Programmiersprachen dem Schlüsselwort return, da es aus dem nächsten Logikblock-Scope zurückspringt. Wenn Sie daher in einer Resolver-Zuweisungsvorlage `#return` verwenden, erfolgt ein Rücksprung vom Resolver. Wenn `#return(data: Object)` in einer Resolver-Zuweisungsvorlage

verwendet wird, wird für das GraphQL-Feld `data` festgelegt. Wenn Sie `#return(data: Object)` in einer Funktionszuweisungsvorlage verwenden, erfolgt ein Rücksprung aus der Funktion und es wird entweder die nächste Funktion in der Pipeline oder die Resolver-Zuweisungsvorlage für Antworten ausgeführt.

#return

Dies ist dasselbe wie`#return(data: Object)`, `null` wird aber stattdessen zurückgegeben.

$util.error

Das Dienstprogramm `$util.error` ist nützlich, um ein Feldfehler auszulösen. Wenn Sie `$util.error` in einer Funktionszuweisungsvorlage verwenden, wird sofort ein Feldfehler ausgelöst, der verhindert, dass nachfolgende Funktionen ausgeführt werden. Weitere Informationen und weitere `$util.error` Signaturen finden Sie in der Referenz zum Resolver Mapping Template Utility.

$util.appendError

`$util.appendError` lässt sich mit `$util.error()` vergleichen, mit dem großen Unterschied, dass die Auswertung der Zuweisungsvorlage nicht unterbrochen wird. Stattdessen wird signalisiert, dass das Feld zwar eine Fehlermeldung hervorrief, die Beurteilung der Vorlage jedoch abgeschlossen wurde und infolgedessen Daten zurückgegeben wurden. Die Verwendung von `$util.appendError` in einer Funktion hat keine Auswirkungen auf die Ausführung des Pipeline-Ablaufs. Weitere Informationen und weitere `$util.error` Signaturen finden Sie in der Referenz zum Resolver Mapping Template Utility.

-Beispielvorlage

Angenommen, Sie haben eine DynamoDB-Datenquelle und einen Unit-Resolver für ein Feld mit dem Name`getPost(id:ID!)`, das einen `Post` Typ mit der folgenden GraphQL-Abfrage zurückgibt:

```
getPost(id:1){
    id
    title
    content
}
```

Ihre Resolver-Vorlage sieht möglicherweise folgendermaßen aus:

```
{
```

```
    "version" : "2018-05-29",
    "operation" : "GetItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    }
}
```

Hierdurch würde der Wert des `id`-Eingabeparameters von `1` durch `${ctx.args.id}` ersetzt und die folgende JSON-Vorlage generiert:

```
{
    "version" : "2018-05-29",
    "operation" : "GetItem",
    "key" : {
        "id" : { "S" : "1" }
    }
}
```

AWS AppSync verwendet diese Vorlage, um Anweisungen für die Kommunikation mit DynamoDB und das Abrufen von Daten zu generieren (oder gegebenenfalls andere Operationen auszuführen). Nachdem die Daten zurückgegeben wurden, führt AWS AppSync sie durch eine optionale Antwortzuweisungsvorlage, über die Sie die Datengestaltung oder Logik vornehmen können. Wenn wir die Ergebnisse beispielsweise von DynamoDB zurückerhalten, könnten sie wie folgt aussehen:

```
{
        "id" : 1,
        "theTitle" : "AWS AppSync works offline!",
        "theContent-part1" : "It also has realtime functionality",
        "theContent-part2" : "using GraphQL"
}
```

Sie können auch mithilfe der folgenden Antwortzuweisungsvorlage zwei der Felder in einem einzelnen Feld zusammenlegen:

```
{
        "id" : $util.toJson($context.data.id),
        "title" : $util.toJson($context.data.theTitle),
        "content" : $util.toJson("${context.data.theContent-part1}
 ${context.data.theContent-part2}")
}
```

Hier sehen Sie, wie die Daten strukturiert sind, nachdem die Vorlage auf die Daten angewendet wurde:

```
{
        "id" : 1,
        "title" : "AWS AppSync works offline!",
        "content" : "It also has realtime functionality using GraphQL"
}
```

Die Daten werden als Antwort folgendermaßen an den Client zurückgegeben:

```
{
        "data": {
                "getPost":      {
                        "id" : 1,
                        "title" : "AWS AppSync works offline!",
                        "content" : "It also has realtime functionality using GraphQL"
                }
        }
}
```

Beachten Sie, dass in den meisten Situationen Antwort-Zuweisungsvorlagen einfach der Durchleitung von Daten dienen und der größte Unterschied darin besteht, ob ein einzelnes Element oder eine Liste von Elementen zurückgegeben wird. Für ein einzelnes Element lautet der Pass-Through:

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

Für Listen lautet der Pass-Through üblicherweise:

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

Weitere Beispiele für Unit- und Pipeline-Resolver finden Sie in den Resolver-Tutorials.

Evaluierte Deserialisierungsregeln für Mapping-Vorlagen

Zuweisungsvorlagen werden zu einer Zeichenfolge ausgewertet. In muss AWS AppSync die Ausgabezeichenfolge einer JSON-Struktur folgen, um gültig zu sein.

Darüber hinaus werden die folgenden Deserialisierungsregeln erzwungen.

Doppelte Schlüssel sind in JSON-Objekten nicht zulässig

Wenn die ausgewertete Zuweisungsvorlagenzeichenfolge ein JSON-Objekt darstellt oder ein Objekt mit doppelten Schlüsseln enthält, gibt die Zuweisungsvorlage die folgende Fehlermeldung zurück:

`Duplicate field 'aField' detected on Object. Duplicate JSON keys are not allowed.`

Beispiel für einen doppelten Schlüssel in einer ausgewerteten Anforderungszuweisungsvorlage:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "postId": "1",
        "field": "getPost" ## key 'field' has been redefined
    }
}
```

Um diesen Fehler zu beheben, definieren Sie Schlüssel in JSON-Objekten nicht neu.

Nachgestellte Zeichen sind in JSON-Objekten nicht zulässig

Wenn die ausgewertete Zuweisungsvorlagenzeichenfolge ein JSON-Objekt darstellt und nachgestellte Fremdzeichen enthält, gibt die Zuweisungsvorlage die folgende Fehlermeldung zurück:

`Trailing characters at the end of the JSON string are not allowed.`

Beispiel für nachgestellte Zeichen in einer ausgewerteten Anforderungszuweisungsvorlage:

```
{
    "version": "2018-05-29",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "postId": "1",
    }
}extraneouschars
```

Um diesen Fehler zu beheben, stellen Sie sicher, dass evaluierte Vorlagen ausschließlich nach JSON ausgewertet werden.

AWS AppSync Programmierleitfaden für Resolver-Mapping-Vorlagen

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Dies ist ein Tutorial im Stil eines Kochbuches zur Programmierung mit der Apache Velocity Template Language (VTL) in. AWS AppSync Wenn Sie mit anderen Programmiersprachen wie JavaScript C oder Java vertraut sind, sollte es ziemlich einfach sein.

AWS AppSync verwendet VTL, um GraphQL-Anfragen von Clients in eine Anfrage an Ihre Datenquelle zu übersetzen. Anschließend wird der Prozess umgekehrt, um die Antwort der Datenquelle wieder in eine GraphQL-Antwort zu übersetzen. VTL ist eine logische Vorlagensprache, mit der Sie sowohl die Anfrage als auch die Antwort im standardmäßigen Anforderungs-/Antwortablauf einer Webanwendung bearbeiten können. Dabei kommen Techniken wie die folgenden zum Einsatz:

- Standardwerte für neue Elemente
- Eingabevalidierung und -formatierung
- Umwandlung und Shaping von Daten
- Durchlaufen von Listen, Zuweisungen und Arrays, um Werte auszusortieren oder zu ändern
- Filtern/Ändern von Antworten basierend auf Benutzeridentität
- Komplexe Autorisierungsprüfungen

Möglicherweise möchten Sie im Dienst eine Überprüfung der Telefonnummer für ein GraphQL-Argument durchführen oder einen Eingabeparameter in Großbuchstaben umwandeln, bevor Sie ihn in DynamoDB speichern. Oder vielleicht möchten Sie, dass Client-Systeme einen Code als Teil eines GraphQL-Arguments, einen JWT-Token-Anspruch oder einen HTTP-Header bereitstellen und nur Daten zurückgeben, wenn der Code einer bestimmten Zeichenfolge in einer Liste entspricht. Dies sind alles logische Prüfungen, die Sie mit aktivierter VTL durchführen können. AWS AppSync

Mit VTL können Sie logische Programmiertechniken anwenden, die Ihnen möglicherweise vertraut sind. Dies kann jedoch nur innerhalb des standardmäßigen Anforderungs-/Antwort-

Ablaufs ausgeführt werden, um sicherzustellen, dass Ihre GraphQL-API skalierbar ist, wenn die Benutzeranzahl wächst. Da es AWS AppSync auch AWS Lambda als Resolver unterstützt wird, können Sie Lambda-Funktionen in der Programmiersprache Ihrer Wahl (Node.js, Python, Go, Java usw.) schreiben, wenn Sie mehr Flexibilität benötigen.

Aufstellen

Eine gängige Technik beim Erlernen einer Sprache besteht darin, Ergebnisse auszudrucken (z. B. `console.log(variable)` in JavaScript), um zu sehen, was passiert. In diesem Tutorial wird das demonstriert, indem wir ein einfaches GraphQL-Schema erstellen und eine Zuweisung von Werten an eine Lambda-Funktion weitergeben. Die Lambda-Funktion gibt die Werte aus und verwendet sie für die Antwort. Auf diese Weise lernen Sie den Anforderungs-/Antwort-Ablauf sowie die unterschiedlichen Programmiertechniken kennen.

Beginnen Sie mit der Erstellung des folgenden GraphQL-Schemas:

```
type Query {
    get(id: ID, meta: String): Thing
}

type Thing {
    id: ID!
    title: String!
    meta: String
}

schema {
    query: Query
}
```

Erstellen Sie nun die folgende AWS Lambda Funktion und verwenden Sie dabei Node.js als Sprache:

```
exports.handler = (event, context, callback) => {
    console.log('VTL details: ', event);
    callback(null, event);
};
```

Fügen Sie im Bereich Datenquellen der AWS AppSync Konsole diese Lambda-Funktion als neue Datenquelle hinzu. Navigieren Sie zurück zur Seite Schema der Konsole und klicken Sie auf die Schaltfläche ATTACH (ANHÄNGEN) neben der Abfrage `get(...):Thing` auf der rechten Seite. Für die Anforderungsvorlage wählen Sie die gewünschte Vorlage aus dem Menü Invoke and forward

arguments (Argumente aufrufen und weiterleiten) aus. Wählen Sie als Antwortvorlage Return Lambda result (Lambda-Ergebnis zurückgeben) aus.

Öffnen Sie Amazon CloudWatch Logs für Ihre Lambda-Funktion an einem Ort und führen Sie auf der Registerkarte Abfragen der AWS AppSync Konsole die folgende GraphQL-Abfrage aus:

```
query test {
  get(id:123 meta:"testing"){
    id
    meta
  }
}
```

Die GraphQL-Antwort sollte `id:123` und `meta:testing` enthalten, da die Lambda-Funktion diese zurückgibt. Nach einigen Sekunden sollten Sie in CloudWatch Logs ebenfalls einen Datensatz mit diesen Details sehen.

Variablen

VTL nutzt Referenzen, die Sie verwenden können, um Daten zu speichern oder zu bearbeiten. In VTL stehen drei Referenztypen zur Verfügung: Variablen, Eigenschaften und Methoden. Variablen ist ein $-Symbol vorangestellt und sie werden mit der `#set`-Richtlinie erstellt:

```
#set($var = "a string")
```

Variablen speichern Typen ähnlich denen, mit denen Sie von anderen Sprachen her vertraut sind, z. B. Zahlen, Zeichenfolgen, Arrays, Listen und Zuordnungen. Möglicherweise haben Sie bemerkt, dass eine JSON-Nutzlast in der Standard-Anforderungsvorlage für Lambda-Resolver gesendet wurde:

```
"payload": $util.toJson($context.arguments)
```

Ein paar Dinge, die Sie hier beachten sollten: Erstens AWS AppSync bietet es mehrere Komfortfunktionen für allgemeine Operationen. In diesem Beispiel konvertiert `$util.toJson` eine Variable in JSON. Zweitens wird die Variable `$context.arguments` automatisch von einer GraphQL-Anforderung als Zuweisungsobjekt eingetragen. Sie können folgendermaßen eine neue Zuweisung erstellen:

```
#set( $myMap = {
  "id": $context.arguments.id,
```

```
  "meta": "stuff",
  "upperMeta" : $context.arguments.meta.toUpperCase()
} )
```

Sie haben jetzt eine Variable mit dem Namen `$myMap` und den Schlüsseln `id`, `meta` und `upperMeta` erstellt. Dies demonstriert einige Dinge:

- Für `id` wird ein Schlüssel aus den GraphQL-Argumenten eingetragen. Dies ist üblich in VTL, um Argumente von Clients abzurufen.
- `meta` ist fest kodiert mit einem Wert, der Standardwerte darstellt.
- `upperMeta` wandelt das Argument `meta` mithilfe der Methode `.toUpperCase()` um.

Setzen Sie den vorherigen Code oben auf Ihre Anforderungsvorlage und ändern Sie `payload` so, dass die neue Variable `$myMap` verwendet wird:

```
"payload": $util.toJson($myMap)
```

Führen Sie Ihre Lambda-Funktion aus, und Sie können die Änderung der Antwort sowie diese Daten in CloudWatch Protokollen sehen. Im Verlauf dieses Tutorials werden wir weiterhin `$myMap` angeben, sodass Sie ähnliche Tests durchführen können.

Sie können auch properties_ für Ihre Variablen festlegen. Dies können einfache Zeichenfolgen, Arrays oder JSON sein:

```
#set($myMap.myProperty = "ABC")
#set($myMap.arrProperty = ["Write", "Some", "GraphQL"])
#set($myMap.jsonProperty = {
    "AppSync" : "Offline and Realtime",
    "Cognito" : "AuthN and AuthZ"
})
```

Ruhige Referenzen

Da VTL eine Vorlagensprache ist, verwendet jede Referenz, die Sie angeben, standardmäßig `.toString()`. Wenn die Referenz nicht definiert ist, wird die tatsächliche Referenzdarstellung als Zeichenfolge ausgegeben. Zum Beispiel:

```
#set($myValue = 5)
##Prints '5'
```

```
$myValue

##Prints '$somethingelse'
$somethingelse
```

Um dieses Problem umgehen zu können, verfügt VTL über eine Syntax für undefinierte Referenzen oder unterdrückte Referenzen, mit der die Vorlagen-Engine angewiesen wird, dieses Verhalten zu unterdrücken. Die Syntax dafür lautet `$!{}`. Wenn wir beispielsweise den vorherigen Code leicht ändern, sodass `$!{somethingelse}` verwendet wird, wird die Druckausgabe unterdrückt:

```
#set($myValue = 5)
##Prints '5'
$myValue

##Nothing prints out
$!{somethingelse}
```

Methoden aufrufen

In einem früheren Beispiel haben wir gezeigt, wie eine Variable erstellt und gleichzeitig Werte festlegt werden. Sie können dies auch in zwei Schritten durchführen, indem Sie der Zuweisung folgendermaßen Daten hinzufügen:

```
#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")}
```

ALLERDINGS gibt es zu diesem Verhalten etwas zu beachten. Auch wenn die Notation zum Unterdrücken der Referenz (`$!{}`) das Aufrufen von Methoden ermöglicht (siehe oben), unterdrückt sie nicht den zurückgegebenen Wert der ausgeführten Methode. Aus diesem Grund haben wir auf `##Prints "first value"` und `##Prints true` oben hingewiesen. Dies kann zu Fehlern führen, wenn Sie Zuordnungen oder Listen durchlaufen, z. B. zum Einfügen eines Werts, wenn bereits ein Schlüssel vorhanden ist, da die Ausgabe nach erfolgter Prüfung unerwartete Zeichenfolgen zur Vorlage hinzufügt.

Um dieses Problem zu umgehen, ist es manchmal hilfreich, die Methoden mithilfe einer `#set`-Richtlinie aufzurufen und die Variable zu ignorieren. Zum Beispiel:

```
#set ($myMap = {})
#set($discard = $myMap.put("id", "first value"))
```

Sie können diese Technik in Ihren Vorlagen verwenden, da sie verhindert, dass unerwartete Zeichenketten in der Vorlage gedruckt werden. AWS AppSync bietet eine alternative Komfortfunktion, die dasselbe Verhalten in einer prägnanteren Schreibweise bietet. Somit müssen Sie sich keine Gedanken über diese speziellen Anforderungen bei der Implementierung machen. Sie können diese Funktion unter `$util.quiet()` oder dem zugehörigen Alias `$util.qr()` aufrufen. Zum Beispiel:

```
#set ($myMap = {})
#set ($myList = [])

##Nothing prints out
$util.quiet($myMap.put("id", "first value"))
##Nothing prints out
$util.qr($myList.add("something"))
```

Zeichenfolgen

Wie bei vielen Programmiersprachen kann der Umgang mit Zeichenfolgen schwierig sein, vor allem, wenn Sie sie aus Variablen erstellen möchten. Einige Dinge treten bei VTL häufig auf.

Angenommen, Sie fügen Daten als Zeichenfolge in eine Datenquelle wie DynamoDB ein, aber sie werden aus einer Variablen aufgefüllt, z. B. einem GraphQL-Argument. Eine Zeichenfolge enthält doppelte Anführungszeichen und zur Referenzierung der Variable in einer Zeichenfolge benötigen Sie nur `"${}"` (also kein `!` wie in der Notation zur Unterdrückung der Referenz). Dies ähnelt einem Vorlagenliteral in: https://developer.mozilla. JavaScript org/en-US/docs/Web/JavaScript/Reference/Template_Literale

```
#set($firstname = "Jeff")
$!{myMap.put("Firstname", "${firstname}")}
```

Sie können dies in DynamoDB-Anforderungsvorlagen sehen, z. B. `"author": { "S" : "${context.arguments.author}"}` bei der Verwendung von Argumenten von GraphQL-Clients oder bei der automatischen ID-Generierung wie. `"id" : { "S" : "$util.autoId()"}` Dies

bedeutet, dass Sie eine Variable oder das Ergebnis einer Methode innerhalb einer Zeichenfolge referenzieren können, um Daten einzutragen.

Sie können auch öffentliche Methoden der Klasse String von Java verwenden, z. B. das Extrahieren einer Teilzeichenfolge:

```
#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}"))
```

Auch die Zeichenfolgenverkettung ist eine sehr häufige Aufgabe. Sie können sie allein mit Variablenreferenzen oder mit statischen Werten ausführen:

```
#set($s1 = "Hello")
#set($s2 = " World")

$util.qr($myMap.put("concat","$s1$s2"))
$util.qr($myMap.put("concat2","Second $s1 World"))
```

Loops

Nachdem Sie nun Variablen erstellt und Methoden aufgerufen haben, können Sie Ihrem Code eine Logik hinzufügen. Im Gegensatz zu anderen Sprachen sind in VTL nur Loops zulässig, bei denen die Anzahl der Wiederholungen vorher festgelegt wurde. In Velocity gibt es kein `do..while`. Dieser Aufbau stellt sicher, dass die Bewertung stets beendet wird, und gibt Grenzen für die Skalierbarkeit vor, wenn Ihre GraphQL-Operationen ausgeführt werden.

Loops werden mithilfe von `#foreach` erstellt und erfordern die Eingabe einer Loop-Variable und eines wiederholbaren Objekts, z. B. ein Array, eine Liste, Zuweisung oder Sammlung. Ein Beispiel für eine klassische Programmierung mit einem `#foreach`-Loop ist, nach den Elementen in einer Sammlung zu suchen und diese auszugeben. In unserem Fall sortieren wir sie aus und fügen sie der Zuweisung hinzu:

```
#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
```

Das folgende Beispiel zeigt einige Dinge. Zum einen werden Variablen mit dem Bereichsoperator `[..]` zum Erstellen eines wiederholbaren Objekts verwendet. Dann wird jedes Element durch die Variable `$i` referenziert, mit der Sie arbeiten können. Im vorherigen Beispiel sehen Sie auch Kommentare, die mit einem doppelten Rautenzeichen ## gekennzeichnet sind. Dies demonstriert außerdem die Verwendung der Loop-Variable sowohl in den Schlüsseln als auch in den Werten sowie andere Methoden der Verkettung mithilfe von Zeichenfolgen.

Beachten Sie, dass `$i` eine Ganzzahl ist. Sie können also eine `.toString()`-Methode aufrufen. Für GraphQL-Typen von INT kann dies nützlich sein.

Sie können auch direkt einen Bereichsoperator verwenden, zum Beispiel:

```
#foreach($item in [1..5])
    ...
#end
```

Arrays

Bisher haben Sie eine Zuweisung bearbeitet, aber auch Arrays kommen in VTL häufig vor. Mit Arrays haben Sie auch Zugriff auf einige zugrunde liegenden Methoden wie `.isEmpty()`, `.size()`, `.set()`, `.get()` und `.add()`, wie unten gezeigt:

```
#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)))
```

Im vorherigen Beispiel wurde die Array-Indexnotation verwendet, um ein Element mit abzurufen. `arr2[$idx]` Auf ähnliche Weise können Sie Elemente nach Namen aus einer Zuweisung/einem Wörterbuch abrufen:

```
#set($result = {
    "Author" : "Nadia",
    "Topic" : "GraphQL"
})

$util.qr($myMap.put("Author", $result["Author"]))
```

Dies wird sehr häufig beim Filtern von Ergebnissen eingesetzt, die von Datenquellen in Antwortvorlagen zurückgegeben werden, wenn Bedingungen verwendet werden.

Bedingte Prüfungen

Der obige Abschnitt mit `#foreach` zeigte einige Beispiele für die Verwendung von Logik, um Daten mit VTL umzuwandeln. Sie können auch bedingte Prüfungen anwenden, um Daten zur Laufzeit auszuwerten:

```
#if(!$array.isEmpty())
    $util.qr($myMap.put("ifCheck", "Array not empty"))
#else
    $util.qr($myMap.put("ifCheck", "Your array is empty"))
#end
```

Die oben gezeigte `#if()`-Prüfung eines booleschen Ausdrucks ist interessant, doch für eine Verzweigung können Sie auch Operatoren und `#elseif()` einsetzen:

```
#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
```

Diese beiden Beispiele demonstrieren Negation (!) und Gleichheit (==). Wir können auch ||, &&, **>**, **<**, **>=**, **<=** und != verwenden.

```
#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
```

Hinweis: Bei Bedingungen werden nur `Boolean.FALSE` und `null` als falsch angesehen. Null (0) und leere Zeichenfolgen ("") gelten nicht als falsch.

Operatoren

Keine Programmiersprache wäre vollständig ohne Operatoren zum Ausführen mathematischer Aktionen. Hier finden Sie einige Beispiele für den Einstieg:

```
#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))
```

Schleifen und Bedingungen zusammen verwenden

Bei der Umwandlung von Daten in VTL, z. B. vor dem Schreiben oder Lesen von einer Datenquelle, werden häufig zunächst Objekte gesucht und dann Prüfungen durchgeführt, bevor eine Aktion ausgeführt wird. Durch die Kombination von Werkzeugen aus den vorherigen Abschnitten bietet sich Ihnen eine große Funktionalität. Dabei ist es hilfreich, zu wissen, dass `#foreach` Ihnen automatisch `.count` für jedes Element angibt:

```
#foreach ($item in $arr2)
  #set($idx = "item" + $foreach.count)
  $util.qr($myMap.put($idx, $item))
#end
```

Zum Beispiel möchten Sie eventuell nur Werte aus einer Zuweisung aussortieren, wenn diese eine bestimmte Größe unterschreitet. Die Verwendung der Anzahl zusammen mit Bedingungen und der Anweisung `#break` ermöglicht Ihnen das:

```
#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
```

Das vorherige `#foreach` wird mit `.keySet()` wiederholt, das Sie in Zuweisungen verwenden können. Dadurch können Sie `$key` abrufen und den Wert mithilfe von `.get($key)` referenzieren. GraphQL-Argumente von Clients in AWS AppSync werden als Map gespeichert. Sie können auch mit `.entrySet()` wiederholt werden, sodass Sie dann Schlüssel und Werte gemeinsam abrufen können, und entweder andere Variablen eintragen oder komplexe bedingte Prüfungen durchführen, z. B. die Validierung oder Umwandlung von Eingaben:

```
#foreach( $entry in $context.arguments.entrySet() )
#if ($entry.key == "XYZ" && $entry.value == "BAD")
```

```
    #set($myvar = "...")
  #else
    #break
  #end
#end
```

Andere gängige Beispiele sind das automatische Ausfüllen von Standardinformationen, wie die anfänglichen Objektversionen beim Synchronisieren von Daten (sehr wichtig bei der Konfliktlösung) oder der Standardbesitzer eines Objekts für Autorisierungsprüfungen. Mary hat diesen Blogbeitrag erstellt, also:

```
#set($myMap.owner ="Mary")
#set($myMap.defaultOwners = ["Admins", "Editors"])
```

Kontext

Da Sie nun besser mit der Durchführung logischer Prüfungen in AWS AppSync Resolvern mit VTL vertraut sind, werfen Sie einen Blick auf das Kontextobjekt:

```
$util.qr($myMap.put("context", $context))
```

Dieses enthält alle Informationen, die Sie in Ihrer GraphQL-Anforderung abrufen können. Eine detaillierte Erläuterung finden Sie in der Kontextreferenz.

Filtern

Bisher wurden in diesem Tutorial alle Informationen von Ihrer Lambda-Funktion an die GraphQL-Abfrage mit einer sehr einfachen JSON-Umwandlung zurückgegeben:

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

Die VTL-Logik ist genau so leistungsfähig, wenn Sie Antworten von einer Datenquelle erhalten, besonders bei der Durchführung von Autorisierungsprüfungen für Ressourcen. Sehen wir uns nun einige Beispiele an. Ändern Sie zunächst Ihre Antwortvorlage wie in diesem Beispiel:

```
#set($data = {
    "id" : "456",
    "meta" : "Valid Response"
```

```
})

$util.toJson($data)
```

Unabhängig davon, was mit Ihrer GraphQL-Operation geschieht, werden fest kodierte Werte an den Client zurückgegeben. Ändern Sie dies, sodass in das Feld `meta` Daten aus der Lambda-Antwort eingetragen werden, die Sie weiter oben im Tutorial im Abschnitt über Bedingungen im `elseIfCheck`-Wert festgelegt haben:

```
#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)
```

`$context.result` ist eine Zuweisung, daher können Sie `entrySet()` verwenden, um Logik entweder auf die zurückgegebenen Schlüssel oder auf die Werte anzuwenden. Da `$context.identity` Informationen über den Benutzer enthält, der die GraphQL-Operation durchgeführt hat, können Sie beim Zurückgeben von Autorisierungsinformationen aus der Datenquelle entscheiden, ob alle, einige oder keine Daten an einen Benutzer zurückgegeben werden, basierend auf Ihrer Logik. Ändern Sie Ihre Antwortvorlage, sodass sie wie folgt aussieht:

```
#if($context.result["id"] == 123)
    $util.toJson($context.result)
  #else
    $util.unauthorized()
#end
```

Wenn Sie Ihre GraphQL-Abfrage ausführen, werden die Daten wie gewohnt zurückgegeben. Wenn Sie jedoch das ID-Argument auf einen anderen Wert als 123 ändern (`query test { get(id:456 meta:"badrequest"){} }`), erhalten Sie eine Autorisierungsfehlermeldung.

Weitere Beispiele für Autorisierungsszenarien finden Sie im Abschnitt zu den Autorisierungsanwendungsfällen.

Beispiel für eine Vorlage

Wenn Sie dem Tutorial gefolgt sind, haben Sie eventuell Schritt für Schritt diese Vorlage erstellt. Falls Sie dies nicht getan haben, fügen wir es unten ein, um es für Tests zu kopieren.

Request Template

```
#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)
}
```

Antwortvorlage

```
#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
```

AWS AppSync Kontextreferenz für Resolver-Mapping-Vorlagen

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync definiert eine Reihe von Variablen und Funktionen für die Arbeit mit Resolver-Mapping-Vorlagen. Dies erleichtert logische Operationen an Daten mit GraphQL. In diesem Dokument werden diese Funktionen beschrieben und Beispiele für das Arbeiten mit Vorlagen gegeben.

Zugreifen auf den **`$context`**

Die Variable `$context` ist eine Zuweisung, die alle Kontextinformationen für den Resolver-Aufruf enthält. Sie hat die folgende Struktur:

```
{
   "arguments" : { ... },
   "source" : { ... },
   "result" : { ... },
   "identity" : { ... },
   "request" : { ... },
   "info": { ... }
}
```

> **Note**
>
> Wenn Sie versuchen, über seinen Schlüssel auf einen Wörterbuch-/Map-Eintrag (z. B. einen Eintrag in`context`) zuzugreifen, um den Wert abzurufen, können Sie mit der Velocity Template Language (VTL) die Notation direkt verwenden. `<dictionary-element>.<key-name>` Dies funktioniert aber möglicherweise nicht in allen Fällen, z. B. wenn die Schlüsselnamen Sonderzeichen (wie einen Unterstrich "_") enthalten. Wir empfehlen, immer die Notation `<dictionary-element>.get("<key-name>")` zu verwenden.

Jedes Feld in der `$context`-Zuordnung ist wie folgt definiert:

`$context`-Felder

`arguments`

Eine Zuordnung, die alle GraphQL-Argumente für dieses Feld enthält.

`identity`

Ein Objekt, das Informationen über den Aufrufer enthält. Weitere Informationen zur Struktur dieses Felds finden Sie unter Identity.

`source`

Eine Zuordnung, die die Auflösung des übergeordneten Feldes enthält.

`stash`

Stash ist eine Zuordnung, die in jeder Resolver- und Funktionszuweisungsvorlage bereitgestellt wird. Eine Stash-Instanz bleibt während einer einzigen Resolver-Instance bestehen. Daher können Sie den Stash nutzen, um beliebige Daten zwischen Zuweisungsvorlagen für Anforderungen und Antworten und Funktionen in einem Pipeline-Resolver zu übergeben. Der Stash bietet dieselben Methoden wie die Java Map-Datenstruktur.

`result`

Ein Container für die Ergebnisse dieses Resolvers. Dieses Feld ist nur für Antwortzuordnungsvorlagen verfügbar.

Wenn Sie beispielsweise das `author` Feld der folgenden Abfrage auflösen:

```
query {
    getPost(id: 1234) {
        postId
        title
        content
        author {
            id
            name
        }
    }
}
```

Dann könnte die vollständige `$context`-Variable, die bei der Verarbeitung einer Antwortzuweisungsvorlage verfügbar ist, folgendermaßen aussehen:

```
{
  "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`

Das Ergebnis einer beliebigen vorherigen Operation, die in einem Pipeline-Resolver ausgeführt wurde.

Wenn es sich bei der vorherigen Operation um die Before-Mapping-Vorlage des Pipeline-Resolvers handelte, `$ctx.prev.result` stellt dies die Ausgabe der Auswertung der Vorlage dar und wird der ersten Funktion in der Pipeline zur Verfügung gestellt.

Wenn die vorherige Operation die erste Funktion betraf, steht `$ctx.prev.result` für die Ausgabe der ersten Funktion und wird der zweiten Funktion in der Pipeline zur Verfügung gestellt.

Wenn die vorherige Operation die letzte Funktion war, stellt sie die Ausgabe der letzten Funktion `$ctx.prev.result` dar und wird für die After-Mapping-Vorlage des Pipeline-Resolvers verfügbar gemacht.

`info`

Ein Objekt, das Informationen über die GraphQL-Anforderung enthält. Die Struktur dieses Feldes finden Sie unter Info.

Identität

Der Abschnitt `identity` enthält Informationen über den Aufrufer. Die Form dieses Abschnitts hängt vom Autorisierungstyp Ihrer AWS AppSync API ab.

Weitere Informationen zu AWS AppSync Sicherheitsoptionen finden Sie unter Autorisierung und Authentifizierung.

`API_KEY`-Autorisierung

Das `identity` Feld ist nicht gefüllt.

`AWS_LAMBDA`-Autorisierung

Der `identity` enthält den `resolverContext` Schlüssel, der denselben `resolverContext` Inhalt enthält, der von der Lambda-Funktion zurückgegeben wurde, die die Anfrage autorisiert.

`AWS_IAM`-Autorisierung

Der `identity` hat die folgende Form:

```
{
    "accountId" : "string",
    "cognitoIdentityPoolId" : "string",
    "cognitoIdentityId" : "string",
    "sourceIp" : ["string"],
    "username" : "string", // IAM user principal
    "userArn" : "string",
    "cognitoIdentityAuthType" : "string", // authenticated/unauthenticated based on
 the identity type
    "cognitoIdentityAuthProvider" : "string" // the auth provider that was used to
 obtain the credentials
}
```

`AMAZON_COGNITO_USER_POOLS`-Autorisierung

Der `identity` hat die folgende Form:

```
{
```

```
    "sub" : "uuid",
    "issuer" : "string",
    "username" : "string"
    "claims" : { ... },
    "sourceIp" : ["x.x.x.x"],
    "defaultAuthStrategy" : "string"
}
```

Jedes Feld ist wie folgt definiert:

`accountId`

Die AWS Konto-ID des Anrufers.

`claims`

Die Ansprüche, die der Benutzer hat.

`cognitoIdentityAuthType`

Entweder authentifiziert oder nicht authentifiziert, basierend auf dem Identitätstyp.

`cognitoIdentityAuthProvider`

Eine durch Kommas getrennte Liste mit Informationen zu externen Identitätsanbietern, anhand derer die Anmeldeinformationen abgerufen wurden, mit denen die Anfrage signiert wurde.

`cognitoIdentityId`

Die Amazon Cognito Cognito-Identitäts-ID des Anrufers.

`cognitoIdentityPoolId`

Die dem Anrufer zugeordnete Amazon Cognito Cognito-Identitätspool-ID.

`defaultAuthStrategy`

Die Standardautorisierungsstrategie für diesen Aufrufer (`ALLOW` oder `DENY`).

`issuer`

Der Aussteller des Tokens.

`sourceIp`

Die Quell-IP-Adresse des Anrufers, der empfängt. AWS AppSync Wenn die Anfrage den `x-forwarded-for` Header nicht enthält, enthält der Quell-IP-Wert nur eine einzige IP-Adresse aus

der TCP-Verbindung. Wenn die Anforderung einen `x-forwarded-for`-Header enthält, verfügt die Quell-IP zusätzlich zur IP-Adresse aus der TCP-Verbindung über eine Liste mit IP-Adressen aus dem `x-forwarded-for`-Header.

`sub`

Die UUID des authentifizierten Benutzers.

`user`

Der IAM-Benutzer.

`userArn`

Der Amazon-Ressourcenname (ARN) des IAM-Benutzers.

`username`

Der Benutzername des authentifizierten Benutzers. Bei einer `AMAZON_COGNITO_USER_POOLS`-Autorisierung ist der Wert des Benutzernamens der Wert des Attributs cognito:username. Im Fall einer `AWS_IAM` Autorisierung entspricht der Wert von username dem Wert des AWS Benutzerprinzipals. Wenn Sie die IAM-Autorisierung mit Anmeldeinformationen verwenden, die aus Amazon Cognito Cognito-Identitätspools stammen, empfehlen wir Ihnen, diese zu verwenden. `cognitoIdentityId`

Auf Header von Anfragen zugreifen

AWS AppSync unterstützt die Übergabe benutzerdefinierter Header von Clients und den Zugriff auf sie in Ihren GraphQL-Resolvern mithilfe von. `$context.request.headers` Sie können die Header-Werte dann für Aktionen wie das Einfügen von Daten in eine Datenquelle oder für Autorisierungsprüfungen verwenden. Sie können einzelne oder mehrere Anforderungsheader `$curl` mithilfe eines API-Schlüssels von der Befehlszeile aus verwenden, wie in den folgenden Beispielen gezeigt:

Beispiel für einen einzelnen Header

Angenommen, Sie legen wie folgt einen `custom`-Header mit dem Wert `nadia` fest:

```
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
```

Darauf könnte dann mit `$context.request.headers.custom` zugegriffen werden. Es könnte sich beispielsweise in der folgenden VTL für DynamoDB befinden:

```
"custom": $util.dynamodb.toDynamoDBJson($context.request.headers.custom)
```

Beispiel für mehrere Header

Sie können auch mehrere Header in einer einzigen Anforderung übergeben und auf diese in der Resolver-Zuweisungsvorlage zugreifen. Zum Beispiel, wenn der `custom` Header mit zwei Werten festgelegt ist:

```
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
```

Sie können dann darauf als Array zugreifen, z. B. `$context.request.headers.custom[1]`.

> **Note**
>
> AWS AppSync macht den Cookie-Header in nicht verfügbar`$context.request.headers`.

Greifen Sie auf den angeforderten benutzerdefinierten Domainnamen zu

AWS AppSync unterstützt die Konfiguration einer benutzerdefinierten Domain, mit der Sie auf Ihre GraphQL- und Echtzeit-Endpunkte für Ihre zugreifen können. APIs Wenn Sie eine Anfrage mit einem benutzerdefinierten Domainnamen stellen, können Sie den Domainnamen mithilfe von abrufen. `$context.request.domainName`

Bei Verwendung des standardmäßigen GraphQL-Endpunktdomänennamens lautet der Wert. `null`

Informationen

Der Abschnitt `info` enthält Informationen über die GraphQL-Anforderung. Dieser Abschnitt hat die folgende Form:

```
{
    "fieldName": "string",
    "parentTypeName": "string",
```

```
    "variables": { ... },
    "selectionSetList": ["string"],
    "selectionSetGraphQL": "string"
}
```

Jedes Feld ist wie folgt definiert:

`fieldName`

Der Name des Feldes, das derzeit aufgelöst wird.

`parentTypeName`

Der Name des übergeordneten Typs für das Feld, das derzeit aufgelöst wird.

`variables`

Eine Zuordnung, die alle Variablen enthält, die an die GraphQL-Anforderung übergeben werden.

`selectionSetList`

Eine Listendarstellung der Felder im GraphQL-Auswahlsatz. Felder mit Aliasnamen werden nur durch den Aliasnamen referenziert, nicht durch den Feldnamen. Im folgenden Beispiel ist dies detailliert dargestellt.

`selectionSetGraphQL`

Eine Zeichenfolgendarstellung des Auswahlsatzes, formatiert als GraphQL-Schemadefinitionssprache (SDL). Fragmente werden zwar nicht mit dem Auswahlsatz zusammengeführt, Inline-Fragmente bleiben jedoch erhalten, wie im folgenden Beispiel gezeigt.

> **Note**
>
> Bei Verwendung von `$utils.toJson()` on `context.info` werden die Werte that `selectionSetGraphQL` und `selectionSetList` return standardmäßig nicht serialisiert.

Angenommen, Sie lösen das `getPost`-Feld der folgenden Abfrage auf:

```
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
  }
}
```

Dann könnte die vollständige `$context.info`-Variable, die bei der Verarbeitung einer Zuweisungsvorlage verfügbar ist, folgendermaßen aussehen:

```
{
  "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`macht nur Felder verfügbar, die zum aktuellen Typ gehören. Wenn es sich bei dem aktuellen Typ um eine Schnittstelle oder Union handelt, werden nur ausgewählte Felder angezeigt, die zur Schnittstelle gehören. Nehmen wir zum Beispiel das folgende Schema:

```
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
}
```

Und die folgende Abfrage:

```
query {
    node(id: "post1") {
        id
        ... on Post {
```

```
            title
        }

        ... on Blog {
            title
        }
    }
}
```

`$ctx.info.selectionSetList`Beim Aufrufen mit der `Query.node` Feldauflösung `id` wird nur angezeigt:

```
"selectionSetList": [
    "id"
]
```

Bereinigung von Eingängen

Anwendungen müssen nicht vertrauenswürdige Eingaben bereinigen, um zu verhindern, dass externe Parteien eine Anwendung anders als beabsichtigt verwenden. Da das Benutzereingaben in Eigenschaften wie`$context.arguments`,,, und `$context` enthält `$context.identity` `$context.result`$`context.info.variables`, muss darauf geachtet werden`$context.request.headers`, dass deren Werte in Mapping-Vorlagen bereinigt werden.

Da Zuweisungsvorlagen JSON repräsentieren, erfolgt die Eingabebereinigung in Form von JSON-reservierten Escape-Zeichen aus Zeichenfolgen, die Benutzereingaben darstellen. Es wird empfohlen, das Dienstprogramm `$util.toJson()` zum Escaping JSON-reservierter Zeichen aus sensiblen Zeichenfolgenwerten zu verwenden, wenn sie in eine Zuweisungsvorlage gesetzt werden.

In der folgenden Lambda-Anforderungszuordnungsvorlage haben wir beispielsweise, weil wir auf eine unsichere Kundeneingabezeichenfolge (`$context.arguments.id`) zugegriffen haben, diese umschlossen, `$util.toJson()` um zu verhindern, dass nicht maskierte JSON-Zeichen die JSON-Vorlage beschädigen.

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "postId": $util.toJson($context.arguments.id)
    }
```

```
}
```

Im Gegensatz zur unten stehenden Zuordnungsvorlage, bei der wir sie direkt einfügen`$context.arguments.id`, ohne sie zu bereinigen. Dies funktioniert nicht für Zeichenketten, die Anführungszeichen ohne Escape-Zeichen oder andere reservierte JSON-Zeichen enthalten, und kann dazu führen, dass Ihre Vorlage fehlschlägt.

```
## 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.
    }
}
```

AWS AppSync Referenz zum Hilfsprogramm für Resolver-Mapping-Vorlagen

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync definiert eine Reihe von Dienstprogrammen, die Sie in einem GraphQL-Resolver verwenden können, um Interaktionen mit Datenquellen zu vereinfachen. Einige dieser Dienstprogramme sind für die allgemeine Verwendung mit beliebigen Datenquellen vorgesehen, z. B. für die Generierung von Zeitstempeln IDs oder für die Generierung von Zeitstempeln. Andere sind spezifisch für einen bestimmten Datenquellentyp. Die folgenden Dienstprogramme sind verfügbar:

- Hilfsprogramme in $util — Die Variable $util enthält allgemeine Hilfsmethoden, die Ihnen bei der Arbeit mit Daten helfen. Sofern nicht anders angegeben, verwenden alle Dienstprogramme den UTF-8-Zeichensatz.
- AppSync Direktiven — AppSync macht Direktiven verfügbar, um Entwicklern die Produktivität beim Schreiben in VTL zu erleichtern.

- Zeithelfer in $util.time — Die Variable $util.time enthält Datetime-Methoden, mit deren Hilfe Zeitstempel generiert, zwischen Datetime-Formaten konvertiert und Datetime-Zeichenketten analysiert werden können. Die Syntax für Datetime-Formate basiert darauf, auf die Sie sich für weitere Dokumentation beziehen können. DateTimeFormatter
- Hilfsprogramme in $util.list auflisten - $util.list enthält Methoden, die bei allgemeinen List-Operationen helfen, z. B. beim Entfernen oder Beibehalten von Elementen aus einer Liste zum Filtern von Anwendungsfällen.
- Map-Helper in $util.map - $util.map enthält Methoden zur Unterstützung gängiger Map-Operationen wie dem Entfernen oder Beibehalten von Elementen aus einer Map zum Filtern von Anwendungsfällen.
- DynamoDB-Helfer in $util.dynamodb - $util.dynamodb enthält Hilfsmethoden, die das Schreiben und Lesen von Daten in Amazon DynamoDB erleichtern, wie z. B. automatische Typenzuweisung und Formatierung.
- Amazon RDS-Helfer in $util.rds - $util.rds enthält Hilfsmethoden, die RDS-Operationen formatieren, indem sie überflüssige Daten in Ergebnisausgaben entfernen.
- HTTP-Helfer in $util.http — Das Hilfsprogramm $util.http bietet Hilfsmethoden, mit denen Sie HTTP-Anforderungsparameter verwalten und Antwortheader hinzufügen können.
- XML-Helfer in $util.xml - $util.xml enthält Hilfsmethoden, die es einfacher machen können, XML-Antworten in JSON oder ein Dictionary zu übersetzen.
- Transformationshelfer in $util.transform - $util.transform enthält Hilfsmethoden, die es einfacher machen, komplexe Operationen an Datenquellen durchzuführen, wie z. B. DynamoDB-Filteroperationen.
- Mathematische Helfer in $util.math - $util.math enthält Methoden, die bei gängigen mathematischen Operationen helfen.
- Zeichenkettenhelfer in $util.str - $util.str enthält Methoden, die bei gängigen String-Operationen helfen.
- Erweiterungen — $extensions enthält eine Reihe von Methoden, mit denen Sie zusätzliche Aktionen in Ihren Resolvern ausführen können.

Hilfsprogramme in $util

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Die `$util` Variable enthält allgemeine Hilfsmethoden, die Ihnen bei der Arbeit mit Daten helfen. Sofern nicht anders angegeben, verwenden alle Dienstprogramme den UTF-8-Zeichensatz.

Werkzeuge zum Analysieren von JSON

Liste der JSON-Parsing-Dienstprogramme

`$util.parseJson(String) : Object`

Erhält das "stringify"-JSON-Objekt und gibt eine Objektdarstellung des Ergebnisses zurück.

`$util.toJson(Object) : String`

Nimmt ein Objekt und gibt eine "stringify"-JSON-Darstellung dieses Objekts zurück.

Werkzeuge zum Kodieren

Liste der Kodierungswerkzeuge

`$util.urlEncode(String) : String`

Gibt die Eingabezeichenfolge als eine `application/x-www-form-urlencoded`-kodierte Zeichenfolge zurück.

`$util.urlDecode(String) : String`

Dekodiert eine `application/x-www-form-urlencoded`-kodierte Zeichenfolge zurück in ihre nicht kodierte Form.

`$util.base64Encode( byte[] ) : String`

Verschlüsselt die Eingabe in eine base64-kodierte Zeichenfolge.

`$util.base64Decode(String) : byte[]`

Decodiert die Daten einer base64-verschlüsselten Zeichenfolge.

Dienstprogramme zur ID-Generierung

Liste der Tools zur ID-Generierung

`$util.autoId() : String`

Gibt eine zufällig generierte 128-Bit-UUID zurück.

`$util.autoUlid() : String`

Gibt eine zufällig generierte 128-Bit-ULID (Universally Unique Lexicographically Sortable Identifier) zurück.

`$util.autoKsuid() : String`

Gibt eine zufällig generierte 128-Bit-KSUID (K-Sortable Unique Identifier) Base62 zurück, die als Zeichenfolge mit einer Länge von 27 codiert ist.

Fehler utils

Fehler-Utils-Liste

`$util.error(String)`

Gibt einen benutzerdefinierte Fehler aus. Verwenden Sie dies in Vorlagen für die Zuordnung von Anfragen oder Antworten, um einen Fehler in der Anfrage oder im Aufrufergebnis zu erkennen.

`$util.error(String, String)`

Gibt einen benutzerdefinierte Fehler aus. Verwenden Sie dies in Vorlagen für die Zuordnung von Anfragen oder Antworten, um einen Fehler in der Anfrage oder im Aufrufergebnis zu erkennen. Sie können auch eine `errorType` angeben.

`$util.error(String, String, Object)`

Gibt einen benutzerdefinierte Fehler aus. Verwenden Sie dies in Vorlagen für die Zuordnung von Anfragen oder Antworten, um einen Fehler in der Anfrage oder im Aufrufergebnis zu erkennen. Sie können auch ein Feld `errorType` und ein `data` Feld angeben. Der `data`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt.

> **Note**
>
> `data`wird auf der Grundlage des Abfrageauswahlsatzes gefiltert.

`$util.error(String, String, Object, Object)`

Gibt einen benutzerdefinierte Fehler aus. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Zusätzlich können ein `errorType` Feld, ein `data` Feld und ein `errorInfo` Feld angegeben werden. Der `data`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt.

> **Note**
>
> `data`wird auf der Grundlage des Abfrageauswahlsatzes gefiltert. Der `errorInfo`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt. `errorInfo`wird NICHT auf der Grundlage des Abfrageauswahlsatzes gefiltert.

`$util.appendError(String)`

Fügt einen benutzerdefinierten Fehler an. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Im Gegensatz zu `$util.error(String)` wird die Vorlagenbewertung nicht unterbrochen, sodass die Daten an den Aufrufer zurückgegeben werden können.

`$util.appendError(String, String)`

Fügt einen benutzerdefinierten Fehler an. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Zudem kann ein `errorType` angegeben werden. Im Gegensatz zu `$util.error(String, String)` wird die Vorlagenbewertung nicht unterbrochen, sodass die Daten an den Aufrufer zurückgegeben werden können.

`$util.appendError(String, String, Object)`

Fügt einen benutzerdefinierten Fehler an. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Zudem können die Felder `errorType` und `data` angegeben werden. Im Gegensatz zu `$util.error(String, String, Object)` wird die Vorlagenbewertung nicht unterbrochen, sodass die Daten an den Aufrufer zurückgegeben werden können. Der `data`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt.

> **Note**
>
> `data`wird auf der Grundlage des Abfrageauswahlsatzes gefiltert.

`$util.appendError(String, String, Object, Object)`

Fügt einen benutzerdefinierten Fehler an. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Zusätzlich können ein `errorType` Feld, ein `data` Feld und ein `errorInfo` Feld angegeben werden. Im Gegensatz zu `$util.error(String, String, Object, Object)` wird die Vorlagenbewertung nicht unterbrochen, sodass die Daten an den Aufrufer zurückgegeben werden können. Der `data`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt.

> **Note**
>
> `data`wird auf der Grundlage des Abfrageauswahlsatzes gefiltert. Der `errorInfo`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt. `errorInfo`wird NICHT auf der Grundlage des Abfrageauswahlsatzes gefiltert.

Werkzeuge zur Zustandsvalidierung

Liste der Utilities zur Zustandsvalidierung

`$util.validate(Boolean, String) : void`

Wenn die Bedingung falsch ist, wird eine CustomTemplateException mit der angegebenen Nachricht ausgelöst.

`$util.validate(Boolean, String, String) : void`

Wenn die Bedingung falsch ist, wird a CustomTemplateException mit der angegebenen Nachricht und dem angegebenen Fehlertyp ausgelöst.

`$util.validate(Boolean, String, String, Object) : void`

Wenn die Bedingung falsch ist, wird eine CustomTemplateException mit der angegebenen Meldung und dem angegebenen Fehlertyp sowie den Daten ausgelöst, die in der Antwort zurückgegeben werden sollen.

Werkzeuge für das Verhalten Null

Liste der Dienstprogramme für Verhalten mit Null

`$util.isNull(Object) : Boolean`

Gibt "true" zurück, wenn das bereitgestellte Objekt null ist.

`$util.isNullOrEmpty(String) : Boolean`

Gibt "true" zurück, wenn die bereitgestellten Daten null oder eine leere Zeichenfolge sind. Ansonsten wird "false" zurückgegeben.

`$util.isNullOrBlank(String) : Boolean`

Gibt "true" zurück, wenn die bereitgestellten Daten null oder eine leere Zeichenfolge sind. Ansonsten wird "false" zurückgegeben.

`$util.defaultIfNull(Object, Object) : Object`

Gibt das erste Objekt zurück, wenn es nicht null ist. Ansonsten wird das zweite Objekt als ein "Standardobjekt" zurückgegeben.

`$util.defaultIfNullOrEmpty(String, String) : String`

Gibt die erste Zeichenfolge zurück, wenn sie nicht null oder leer ist. Ansonsten wird die zweite Zeichenfolge als eine "Standardzeichenfolge" zurückgegeben.

`$util.defaultIfNullOrBlank(String, String) : String`

Gibt die erste Zeichenfolge zurück, wenn sie nicht null oder leer ist. Ansonsten wird die zweite Zeichenfolge als eine "Standardzeichenfolge" zurückgegeben.

Dienstprogramme für den Musterabgleich

Liste der Werkzeuge für den Typ- und Musterabgleich

`$util.typeOf(Object) : String`

Gibt eine Zeichenfolge zurück, die den Typ des Objekts beschreibt. Unterstützte Typenidentifikationen sind "null", "Zahl", "Zeichenfolge", "Zuweisung", "Liste", "boolescher Wert". Wenn ein Typ nicht identifiziert werden kann, lautet der Rückgabetyp "Objekt".

`$util.matches(String, String) : Boolean`

Gibt "true" zurück, wenn das angegebene Muster im ersten Argument den bereitgestellten Daten im zweiten Argument entspricht. Das Muster muss ein regulärer Ausdruck sein, wie z. B. `$util.matches("a*b", "aaaaab")`. Die Funktionalität basiert auf Pattern, worauf Sie zur weiteren Dokumentation verweisen können.

`$util.authType() : String`

Gibt eine Zeichenfolge zurück, die den Multi-Auth-Typ beschreibt, der von einer Anfrage verwendet wird, und gibt entweder „IAM-Autorisierung“, „Benutzerpool-Autorisierung“, „Open ID Connect-Autorisierung“ oder „API-Schlüsselautorisierung“ zurück.

Tools zur Objektvalidierung

Liste der Tools zur Objektvalidierung

`$util.isString(Object) : Boolean`

Gibt true zurück, wenn das Objekt ein String ist.

`$util.isNumber(Object) : Boolean`

Gibt „true“ zurück, wenn das Objekt eine Zahl ist.

`$util.isBoolean(Object) : Boolean`

Gibt true zurück, wenn das Objekt ein boolescher Wert ist.

`$util.isList(Object) : Boolean`

Gibt true zurück, wenn es sich bei dem Objekt um eine Liste handelt.

`$util.isMap(Object) : Boolean`

Gibt „true“ zurück, wenn es sich bei dem Objekt um eine Map handelt.

CloudWatch Werkzeuge für die Protokollierung

CloudWatch Liste der Protokollierungswerkzeuge

`$util.log.info(Object) : Void`

Protokolliert die String-Darstellung des bereitgestellten Objekts im angeforderten Log-Stream, wenn die Protokollierung auf Anfrage- und Feldebene auf CloudWatch Protokollebene `ALLINFO`, oder auf einer API aktiviert ist. `DEBUG`

`$util.log.info(String, Object...) : Void`

Protokolliert die Zeichenkettendarstellung der bereitgestellten Objekte im angeforderten Logstream, wenn die Protokollierung auf Anfrage- und Feldebene mit CloudWatch Protokollebene auf einer API aktiviert ist. `ALL` Dieses Hilfsprogramm ersetzt alle Variablen, die im ersten Eingabeformat String mit „{}“ gekennzeichnet sind, der Reihe nach durch die String-Darstellung der bereitgestellten Objekte.

`$util.log.debug(Object) : Void`

Protokolliert die String-Darstellung des bereitgestellten Objekts im angeforderten Log-Stream, wenn die Protokollierung auf Anfrage- und Feldebene auf CloudWatch Protokollebene `ALL` oder `DEBUG` auf einer API aktiviert ist.

`$util.log.debug(String, Object...) : Void`

Protokolliert die String-Repräsentation der bereitgestellten Objekte im angeforderten Log-Stream, wenn die Protokollierung auf Feldebene mit Protokollebene `DEBUG` oder CloudWatch Protokollebene auf einer API aktiviert ist. `ALL` Dieses Hilfsprogramm ersetzt alle Variablen, die im ersten Eingabeformat String mit „{}“ gekennzeichnet sind, der Reihe nach durch die String-Darstellung der bereitgestellten Objekte.

`$util.log.error(Object) : Void`

Protokolliert die String-Repräsentation des bereitgestellten Objekts im angeforderten Log-Stream, wenn die CloudWatch Protokollierung auf Feldebene für eine beliebige Protokollebene (`ALL`, `INFODEBUG`, usw.) auf einer API aktiviert ist.

`$util.log.error(String, Object...) : Void`

Protokolliert die String-Repräsentation der bereitgestellten Objekte im angeforderten Logstream, wenn die Protokollierung auf Feldebene mit CloudWatch Protokollebene `ERROR` oder Protokollebene auf einer API aktiviert ist. `ALL` Dieses Hilfsprogramm ersetzt alle Variablen, die

im ersten Eingabeformat String mit „{}“ gekennzeichnet sind, der Reihe nach durch die String-Darstellung der bereitgestellten Objekte.

Gibt den Wert zurück, Verhalten, utils

Liste der Verhaltenswerkzeuge für den Rückgabewert

$util.qr() und **$util.quiet()**

Führt eine VTL-Anweisung aus und unterdrückt dabei den zurückgegebenen Wert. Dies ist nützlich, um Methoden auszuführen, ohne temporäre Platzhalter zu verwenden, z. B. um Elemente zu einer Map hinzuzufügen. Zum Beispiel:

```
#set ($myMap = {})
#set($discard = $myMap.put("id", "first value"))
```

Wird zu:

```
#set ($myMap = {})
$util.qr($myMap.put("id", "first value"))
```

$util.escapeJavaScript(String) : String

Gibt die Eingabezeichenfolge als JavaScript Escape-Zeichenfolge zurück.

$util.urlEncode(String) : String

Gibt die Eingabezeichenfolge als eine `application/x-www-form-urlencoded`-kodierte Zeichenfolge zurück.

$util.urlDecode(String) : String

Dekodiert eine `application/x-www-form-urlencoded`-kodierte Zeichenfolge zurück in ihre nicht kodierte Form.

$util.base64Encode(byte[]) : String

Verschlüsselt die Eingabe in eine base64-kodierte Zeichenfolge.

$util.base64Decode(String) : byte[]

Decodiert die Daten einer base64-verschlüsselten Zeichenfolge.

`$util.parseJson(String) : Object`

Erhält das "stringify"-JSON-Objekt und gibt eine Objektdarstellung des Ergebnisses zurück.

`$util.toJson(Object) : String`

Nimmt ein Objekt und gibt eine "stringify"-JSON-Darstellung dieses Objekts zurück.

`$util.autoId() : String`

Gibt eine zufällig generierte 128-Bit-UUID zurück.

`$util.autoUlid() : String`

Gibt eine zufällig generierte 128-Bit-ULID (Universally Unique Lexicographically Sortable Identifier) zurück.

`$util.autoKsuid() : String`

Gibt eine zufällig generierte 128-Bit-KSUID (K-Sortable Unique Identifier) Base62 zurück, die als Zeichenfolge mit einer Länge von 27 codiert ist.

`$util.unauthorized()`

Gibt `Unauthorized` für das Feld aus, das aufgelöst wird. Verwenden Sie dies in Vorlagen für die Zuordnung von Anfragen oder Antworten, um zu bestimmen, ob der Anrufer das Feld auflösen kann.

`$util.error(String)`

Gibt einen benutzerdefinierte Fehler aus. Verwenden Sie dies in Vorlagen für die Zuordnung von Anfragen oder Antworten, um einen Fehler in der Anfrage oder im Aufrufergebnis zu erkennen.

`$util.error(String, String)`

Gibt einen benutzerdefinierte Fehler aus. Verwenden Sie dies in Vorlagen für die Zuordnung von Anfragen oder Antworten, um einen Fehler in der Anfrage oder im Aufrufergebnis zu erkennen. Sie können auch eine `errorType` angeben.

`$util.error(String, String, Object)`

Gibt einen benutzerdefinierte Fehler aus. Verwenden Sie dies in Vorlagen für die Zuordnung von Anfragen oder Antworten, um einen Fehler in der Anfrage oder im Aufrufergebnis zu erkennen. Sie können auch ein Feld `errorType` und ein `data` Feld angeben. Der `data`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt. Hinweis: `data` wird basierend auf dem Abfragenauswahlsatz gefiltert.

`$util.error(String, String, Object, Object)`

Gibt einen benutzerdefinierte Fehler aus. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Außerdem können die Felder `errorType`, `data` und `errorInfo` angegeben werden. Der `data`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt. Hinweis: `data` wird basierend auf dem Abfragenauswahlsatz gefiltert. Der `errorInfo`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt. Hinweis: `errorInfo` wird NICHT basierend auf dem Abfragenauswahlsatz gefiltert.

`$util.appendError(String)`

Fügt einen benutzerdefinierten Fehler an. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Im Gegensatz zu `$util.error(String)` wird die Vorlagenbewertung nicht unterbrochen, sodass die Daten an den Aufrufer zurückgegeben werden können.

`$util.appendError(String, String)`

Fügt einen benutzerdefinierten Fehler an. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Zudem kann ein `errorType` angegeben werden. Im Gegensatz zu `$util.error(String, String)` wird die Vorlagenbewertung nicht unterbrochen, sodass die Daten an den Aufrufer zurückgegeben werden können.

`$util.appendError(String, String, Object)`

Fügt einen benutzerdefinierten Fehler an. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der Anforderung oder beim Aufrufergebnis erkennt. Zudem können die Felder `errorType` und `data` angegeben werden. Im Gegensatz zu `$util.error(String, String, 0bject)` wird die Vorlagenbewertung nicht unterbrochen, sodass die Daten an den Aufrufer zurückgegeben werden können. Der `data`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt. Hinweis: `data` wird basierend auf dem Abfragenauswahlsatz gefiltert.

`$util.appendError(String, String, Object, Object)`

Fügt einen benutzerdefinierten Fehler an. Dies kann in Anforderungs- oder Antwortzuweisungsvorlagen verwendet werden, wenn die Vorlage einen Fehler bei der

Anforderung oder beim Aufrufergebnis erkennt. Außerdem können die Felder `errorType`, `data` und `errorInfo` angegeben werden. Im Gegensatz zu `$util.error(String, String, Object, Object)` wird die Vorlagenbewertung nicht unterbrochen, sodass die Daten an den Aufrufer zurückgegeben werden können. Der `data`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt. Hinweis: `data` wird basierend auf dem Abfragenauswahlsatz gefiltert. Der `errorInfo`-Wert wird zum entsprechenden `error`-Block in `errors` in der GraphQL-Antwort hinzugefügt. Hinweis: `errorInfo` wird NICHT basierend auf dem Abfragenauswahlsatz gefiltert.

`$util.validate(Boolean, String) : void`

Wenn die Bedingung falsch ist, wird a CustomTemplateException mit der angegebenen Nachricht ausgelöst.

`$util.validate(Boolean, String, String) : void`

Wenn die Bedingung falsch ist, wird a CustomTemplateException mit der angegebenen Nachricht und dem angegebenen Fehlertyp ausgelöst.

`$util.validate(Boolean, String, String, Object) : void`

Wenn die Bedingung falsch ist, wird eine CustomTemplateException mit der angegebenen Meldung und dem angegebenen Fehlertyp sowie den Daten ausgelöst, die in der Antwort zurückgegeben werden sollen.

`$util.isNull(Object) : Boolean`

Gibt "true" zurück, wenn das bereitgestellte Objekt null ist.

`$util.isNullOrEmpty(String) : Boolean`

Gibt "true" zurück, wenn die bereitgestellten Daten null oder eine leere Zeichenfolge sind. Ansonsten wird "false" zurückgegeben.

`$util.isNullOrBlank(String) : Boolean`

Gibt "true" zurück, wenn die bereitgestellten Daten null oder eine leere Zeichenfolge sind. Ansonsten wird "false" zurückgegeben.

`$util.defaultIfNull(Object, Object) : Object`

Gibt das erste Objekt zurück, wenn es nicht null ist. Ansonsten wird das zweite Objekt als ein "Standardobjekt" zurückgegeben.

`$util.defaultIfNullOrEmpty(String, String) : String`

Gibt die erste Zeichenfolge zurück, wenn sie nicht null oder leer ist. Ansonsten wird die zweite Zeichenfolge als eine "Standardzeichenfolge" zurückgegeben.

`$util.defaultIfNullOrBlank(String, String) : String`

Gibt die erste Zeichenfolge zurück, wenn sie nicht null oder leer ist. Ansonsten wird die zweite Zeichenfolge als eine "Standardzeichenfolge" zurückgegeben.

`$util.isString(Object) : Boolean`

Gibt "true" zurück, wenn das Objekt eine Zeichenfolge ist.

`$util.isNumber(Object) : Boolean`

Gibt "true" zurück, wenn das Objekt eine Zahl ist.

`$util.isBoolean(Object) : Boolean`

Gibt "true" zurück, wenn das Objekt ein boolescher Wert ist.

`$util.isList(Object) : Boolean`

Gibt "true" zurück, wenn das Objekt eine Liste ist.

`$util.isMap(Object) : Boolean`

Gibt "true" zurück, wenn das Objekt eine Zuweisung ist.

`$util.typeOf(Object) : String`

Gibt eine Zeichenfolge zurück, die den Typ des Objekts beschreibt. Unterstützte Typenidentifikationen sind "null", "Zahl", "Zeichenfolge", "Zuweisung", "Liste", "boolescher Wert". Wenn ein Typ nicht identifiziert werden kann, lautet der Rückgabetyp "Objekt".

`$util.matches(String, String) : Boolean`

Gibt "true" zurück, wenn das angegebene Muster im ersten Argument den bereitgestellten Daten im zweiten Argument entspricht. Das Muster muss ein regulärer Ausdruck sein, wie z. B. `$util.matches("a*b", "aaaaab")`. Die Funktionalität basiert auf Pattern, worauf Sie zur weiteren Dokumentation verweisen können.

`$util.authType() : String`

Gibt eine Zeichenfolge zurück, die den Multi-Auth-Typ beschreibt, der von einer Anfrage verwendet wird, und gibt entweder „IAM-Autorisierung“, „Benutzerpool-Autorisierung“, „Open ID Connect-Autorisierung“ oder „API-Schlüsselautorisierung“ zurück.

`$util.log.info(Object) : Void`

Protokolliert die String-Darstellung des bereitgestellten Objekts im angeforderten Log-Stream, wenn die Protokollierung auf Anfrage- und Feldebene auf CloudWatch Protokollebene in einer API aktiviert ist. `ALL`

`$util.log.info(String, Object...) : Void`

Protokolliert die String-Repräsentation der bereitgestellten Objekte im angeforderten Log-Stream, wenn die Protokollierung auf Anfrage- und Feldebene auf CloudWatch Protokollebene in einer API aktiviert ist. `ALL` Dieses Hilfsprogramm ersetzt alle Variablen, die im ersten Eingabeformat String mit „{}“ gekennzeichnet sind, der Reihe nach durch die String-Darstellung der bereitgestellten Objekte.

`$util.log.error(Object) : Void`

Protokolliert die String-Repräsentation des bereitgestellten Objekts im angeforderten Log-Stream, wenn die CloudWatch Protokollierung auf Feldebene mit Log Level `ERROR` oder Log Level `ALL` auf einer API aktiviert ist.

`$util.log.error(String, Object...) : Void`

Protokolliert die Zeichenkettendarstellung der bereitgestellten Objekte im angeforderten Logstream, wenn die Protokollierung auf Feldebene mit CloudWatch Protokollebene `ERROR` oder Protokollebene auf einer API aktiviert ist. `ALL` Dieses Hilfsprogramm ersetzt alle Variablen, die im ersten Eingabeformat String mit „{}“ gekennzeichnet sind, der Reihe nach durch die String-Darstellung der bereitgestellten Objekte.

`$util.escapeJavaScript(String) : String`

Gibt die Eingabezeichenfolge als JavaScript maskierte Zeichenfolge zurück.

Autorisierung des Resolvers

Liste der Resolver-Autorisierungen

`$util.unauthorized()`

Gibt `Unauthorized` für das Feld aus, das aufgelöst wird. Verwenden Sie dies in Vorlagen für die Zuordnung von Anfragen oder Antworten, um zu bestimmen, ob der Anrufer das Feld auflösen kann.

AWS AppSync Direktiven

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

AWS AppSync stellt Direktiven zur Verfügung, um die Produktivität der Entwickler beim Schreiben in VTL zu erhöhen.

Hilfsprogramme für Direktiven

`#return(Object)`

Das `#return(Object)` ermöglicht es Ihnen, vorzeitig von einer beliebigen Mapping-Vorlage zurückzukehren. `#return(Object)`entspricht dem Schlüsselwort return in Programmiersprachen, da es aus dem Logikblock zurückkehrt, der dem Bereich am nächsten kommt. Die Verwendung `#return(Object)` innerhalb einer Resolver-Mapping-Vorlage führt zu einer Rückgabe vom Resolver. Darüber hinaus führt die Verwendung `#return(Object)` von einer Funktionszuordnungsvorlage zurück zur Funktion und setzt die Ausführung entweder mit der nächsten Funktion in der Pipeline oder der Resolver-Antwortzuordnungsvorlage fort.

`#return`

Die `#return` Direktive zeigt dasselbe Verhalten wie`#return(Object)`, `null` wird aber stattdessen zurückgegeben.

Zeithelfer in $util.time

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Die `$util.time`-Variable enthält "datetime"-Methoden, um beim Generieren von Zeitstempeln, Konvertieren zwischen "datetime"-Formaten und Parsen von "datetime"-Zeichenfolgen zu helfen.

Die Syntax für Datetime-Formate basiert auf dieser Syntax DateTimeFormatter, auf die Sie sich für weitere Dokumentation beziehen können. Im Folgenden finden Sie einige Beispiele sowie eine Liste der verfügbaren Methoden und Beschreibungen.

Zeit nützt

Liste der Zeitwerkzeuge

`$util.time.nowISO8601() : String`

Gibt eine String-Darstellung von UTC im ISO8601-Format zurück.

`$util.time.nowEpochSeconds() : long`

Gibt die Anzahl der Sekunden aus der Epoche von 1970-01-01T00:00:00Z bis jetzt zurück.

`$util.time.nowEpochMilliSeconds() : long`

Gibt die Anzahl der Millisekunden aus der Epoche von 1970-01-01T00:00:00Z bis jetzt zurück.

`$util.time.nowFormatted(String) : String`

Gibt eine Zeichenfolge des aktuellen Zeitstempels in UTC unter Verwendung des angegebenen Formats aus einem Zeichenfolge-Eingabetyp zurück.

`$util.time.nowFormatted(String, String) : String`

Gibt eine Zeichenfolge des aktuellen Zeitstempels aus einer Zeitzone unter Verwendung des angegebenen Formats und der Zeitzone aus Zeichenfolge-Eingabetypen zurück.

`$util.time.parseFormattedToEpochMilliSeconds(String, String) : Long`

Analysiert einen als String übergebenen Zeitstempel zusammen mit einem Format, das eine Zeitzone enthält, und gibt dann den Zeitstempel als Millisekunden seit Epoche zurück.

`$util.time.parseFormattedToEpochMilliSeconds(String, String, String) : Long`

Analysiert einen als String übergebenen Zeitstempel zusammen mit einem Format und einer Zeitzone und gibt dann den Zeitstempel in Millisekunden seit der Epoche zurück.

`$util.time.parseISO8601ToEpochMilliSeconds(String) : Long`

Analysiert einen als String übergebenen Zeitstempel ISO86 01 und gibt dann den Zeitstempel in Millisekunden seit der Epoche zurück.

`$util.time.epochMilliSecondsToSeconds(long) : long`

Konvertiert einen Epoche-Millisekunden-Zeitstempel in einen Epoche-Sekunden-Zeitstempel.

`$util.time.epochMilliSecondsToISO8601(long) : String`

Konvertiert einen Epochen-Millisekunden-Zeitstempel in einen 01-Zeitstempel. ISO86

`$util.time.epochMilliSecondsToFormatted(long, String) : String`

Konvertiert einen Epochen-Millisekunden-Zeitstempel, der als long übergeben wurde, in einen Zeitstempel, der gemäß dem angegebenen Format in UTC formatiert ist.

`$util.time.epochMilliSecondsToFormatted(long, String, String) : String`

Konvertiert einen Epochen-Millisekunden-Zeitstempel, der als Long übergeben wurde, in einen Zeitstempel, der gemäß dem angegebenen Format in der angegebenen Zeitzone formatiert ist.

Beispiele für eigenständige Funktionen

```
$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
```

Beispiele für Konvertierungen

```
#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
```

Beispiele für das Parsen

```
$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
```

Verwendung mit AWS AppSync definierten Skalaren

Die folgenden Formate sind mit `AWSDate`, `AWSDateTime` und `AWSTime` kompatibel.

```
$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
```

Listet Helfer in $util.list auf

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

`$util.list`enthält Methoden zur Unterstützung gängiger Listenoperationen wie dem Entfernen oder Beibehalten von Elementen aus einer Liste zum Filtern von Anwendungsfällen.

Dienstprogramme auflisten

`$util.list.copyAndRetainAll(List, List) : List`

Erstellt eine oberflächliche Kopie der angegebenen Liste im ersten Argument, wobei nur die im zweiten Argument angegebenen Elemente beibehalten werden, sofern sie vorhanden sind. Alle anderen Elemente werden aus der Kopie entfernt.

`$util.list.copyAndRemoveAll(List, List) : List`

Erstellt eine oberflächliche Kopie der angegebenen Liste im ersten Argument und entfernt alle Elemente, für die das Element im zweiten Argument angegeben ist, sofern sie vorhanden sind. Alle anderen Elemente werden in der Kopie beibehalten.

`$util.list.sortList(List, Boolean, String) : List`

Sortiert eine Liste von Objekten, die im ersten Argument bereitgestellt wird. Wenn das zweite Argument wahr ist, wird die Liste absteigend sortiert. Wenn das zweite Argument falsch ist, wird die Liste aufsteigend sortiert. Das dritte Argument ist der Zeichenfolgenname der Eigenschaft, die zum Sortieren einer Liste von benutzerdefinierten Objekten verwendet wird. Wenn es sich um eine Liste handelt, die nur aus Strings, Integers, Floats oder Doubles besteht, kann das dritte Argument eine beliebige beliebige Zeichenfolge sein. Wenn nicht alle Objekte aus derselben Klasse stammen, wird die ursprüngliche Liste zurückgegeben. Es werden nur Listen mit maximal 1000 Objekten unterstützt. Im Folgenden finden Sie ein Beispiel für die Verwendung dieses Dienstprogramms:

```
 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}]
```

Ordnen Sie Helfer in $util.map zu

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

`$util.map`enthält Methoden zur Unterstützung gängiger Map-Operationen wie dem Entfernen oder Beibehalten von Elementen aus einer Map zum Filtern von Anwendungsfällen.

Hilfsprogramme für Karten

`$util.map.copyAndRetainAllKeys(Map, List) : Map`

Erstellt eine flache Kopie der ersten Map und behält dabei nur die in der Liste angegebenen Schlüssel bei, sofern sie vorhanden sind. Alle anderen Schlüssel werden aus der Kopie entfernt.

`$util.map.copyAndRemoveAllKeys(Map, List) : Map`

Erstellt eine flache Kopie der ersten Map und entfernt dabei alle Einträge, in denen der Schlüssel in der Liste angegeben ist, sofern sie vorhanden sind. Alle anderen Schlüssel werden in der Kopie beibehalten.

DynamoDB-Helferobjekte in $util.dynamodb

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

`$util.dynamodb`enthält Hilfsmethoden, die das Schreiben und Lesen von Daten in Amazon DynamoDB erleichtern, z. B. automatische Typzuweisung und Formatierung. Diese Methoden sind so konzipiert, dass primitive Typen und Listen automatisch dem richtigen DynamoDB-Eingabeformat zugeordnet werden, das Teil `Map` des Formats ist. `{ "TYPE" : VALUE }`

Bisher könnte eine Vorlage für die Anforderungszuweisung zum Erstellen eines neuen Elements in DynamoDB beispielsweise so ausgesehen haben:

```
{
    "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) }
    }
```

```
}
```

Wenn wir Felder zum Objekt hinzufügen wollten, hätten wir die GraphQL-Abfrage im Schema und die Zuweisungsvorlage für Anforderungen aktualisieren müssen. Wir können jetzt jedoch unsere Vorlage für die Anforderungszuweisung so umstrukturieren, dass sie automatisch neue Felder aufnimmt, die in unserem Schema hinzugefügt wurden, und sie DynamoDB mit den richtigen Typen hinzufügt:

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

Im vorherigen Beispiel verwenden wir den `$util.dynamodb.toDynamoDBJson(...)` Helper, um die generierte ID automatisch in die DynamoDB-Darstellung eines String-Attributs zu konvertieren. Wir nehmen dann alle Argumente und konvertieren sie in ihre DynamoDB-Repräsentationen und geben sie in das `attributeValues` Feld in der Vorlage aus.

Jedes Helferobjekt hat zwei Versionen: eine Version, die ein Objekt zurückgibt (z. B. `$util.dynamodb.toString(...)`), und eine Version, die das Objekt als JSON-Zeichenfolge zurückgibt (z. B. `$util.dynamodb.toStringJson(...)`). Im vorherigen Beispiel haben wir die Version verwendet, die die Daten als JSON-Zeichenfolge zurückgibt. Wenn Sie das Objekt ändern möchten, bevor es in der Vorlage verwendet wird, können Sie stattdessen folgendermaßen ein Objekt zurückgeben:

```
{
    "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)
```

```
}
```

Im vorherigen Beispiel geben wir die konvertierten Argumente als eine Zuweisung anstatt als eine JSON-Zeichenfolge zurück. Dann fügen wir die Felder `version` und `timestamp` hinzu, bevor wir sie schließlich an das Feld `attributeValues` in der Vorlage unter Verwendung von `$util.toJson(...)` ausgeben.

Die JSON-Version der einzelnen Helferobjekte entspricht der Umhüllung der Nicht-JSON-Version in `$util.toJson(...)`. Die folgenden Aussagen sind beispielsweise identisch:

```
$util.toStringJson("Hello, World!")
$util.toJson($util.toString("Hello, World!"))
```

Zu DynamoDB

Liste der ToDynamoDB-Dienstprogramme

`$util.dynamodb.toDynamoDB(Object) : Map`

Allgemeines Objektkonvertierungstool für DynamoDB, das Eingabeobjekte in die entsprechende DynamoDB-Darstellung konvertiert. Es ist vorbestimmt, wie einige Typen dargestellt werden: z. B. werden Listen ("L") anstelle von Sätzen ("SS", "NS", "BS") verwendet. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

Beispiel für eine Zeichenfolge

```
Input:      $util.dynamodb.toDynamoDB("foo")
Output:     { "S" : "foo" }
```

Beispiel für eine Zahl

```
Input:      $util.dynamodb.toDynamoDB(12345)
Output:     { "N" : 12345 }
```

Boolesches Beispiel

```
Input:      $util.dynamodb.toDynamoDB(true)
Output:     { "BOOL" : true }
```

Beispiel auflisten

```
Input:      $util.dynamodb.toDynamoDB([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

Beispiel für eine Karte

```
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`

Entspricht`$util.dynamodb.toDynamoDB(Object) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

ToString-Dienstprogramme

Liste der ToString-Dienstprogramme

`$util.dynamodb.toString(String) : String`

Konvertiert eine Eingabezeichenfolge in das DynamoDB-Zeichenkettenformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      $util.dynamodb.toString("foo")
Output:     { "S" : "foo" }
```

$util.dynamodb.toStringJson(String) : Map

Entspricht`$util.dynamodb.toString(String) : String`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

$util.dynamodb.toStringSet(List<String>) : Map

Konvertiert eine Liste mit Strings in das DynamoDB-Zeichenkettensatzformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      $util.dynamodb.toStringSet([ "foo", "bar", "baz" ])
Output:     { "SS" : [ "foo", "bar", "baz" ] }
```

$util.dynamodb.toStringSetJson(List<String>) : String

Entspricht`$util.dynamodb.toStringSet(List<String>) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

ToNumber-Dienstprogramme

Liste der ToNumber-Dienstprogramme

$util.dynamodb.toNumber(Number) : Map

Konvertiert eine Zahl in das DynamoDB-Zahlenformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      $util.dynamodb.toNumber(12345)
Output:     { "N" : 12345 }
```

$util.dynamodb.toNumberJson(Number) : String

Entspricht`$util.dynamodb.toNumber(Number) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

$util.dynamodb.toNumberSet(List<Number>) : Map

Konvertiert eine Liste von Zahlen in das DynamoDB-Nummernsatzformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      $util.dynamodb.toNumberSet([ 1, 23, 4.56 ])
Output:     { "NS" : [ 1, 23, 4.56 ] }
```

$util.dynamodb.toNumberSetJson(List<Number>) : String

Entspricht`$util.dynamodb.toNumberSet(List<Number>) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

ToBinary-Dienstprogramme

Liste der ToBinary-Dienstprogramme

$util.dynamodb.toBinary(String) : Map

Konvertiert als Base64-Zeichenfolge kodierte Binärdaten in das DynamoDB-Binärformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      $util.dynamodb.toBinary("foo")
Output:     { "B" : "foo" }
```

$util.dynamodb.toBinaryJson(String) : String

Entspricht`$util.dynamodb.toBinary(String) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

$util.dynamodb.toBinarySet(List<String>) : Map

Konvertiert eine Liste von Binärdaten, die als Base64-Zeichenfolgen kodiert sind, in das DynamoDB-Binärsatzformat. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      $util.dynamodb.toBinarySet([ "foo", "bar", "baz" ])
Output:     { "BS" : [ "foo", "bar", "baz" ] }
```

$util.dynamodb.toBinarySetJson(List<String>) : String

Entspricht`$util.dynamodb.toBinarySet(List<String>) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

ToBoolesche Hilfsprogramme

Liste der toBooleschen Dienstprogramme

`$util.dynamodb.toBoolean(Boolean) : Map`

Konvertiert einen booleschen Wert in das entsprechende boolesche DynamoDB-Format. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      $util.dynamodb.toBoolean(true)
Output:     { "BOOL" : true }
```

`$util.dynamodb.toBooleanJson(Boolean) : String`

Entspricht`$util.dynamodb.toBoolean(Boolean) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

ToNull-Dienstprogramme

ToNull-Utils-Liste

`$util.dynamodb.toNull() : Map`

Gibt eine Null im DynamoDB-Null-Format zurück. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      $util.dynamodb.toNull()
Output:     { "NULL" : null }
```

`$util.dynamodb.toNullJson() : String`

Entspricht`$util.dynamodb.toNull() : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

ToList-Dienstprogramme

Liste der ToList-Dienstprogramme

`$util.dynamodb.toList(List) : Map`

Konvertiert eine Liste von Objekten in das DynamoDB-Listenformat. Jedes Element in der Liste wird auch in das entsprechende DynamoDB-Format konvertiert. Es ist vorbestimmt, wie einige verschachtelte Objekte dargestellt werden: z. B. werden Listen ("L") anstelle von Sätzen ("SS", "NS", "BS") verwendet. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
Input:      $util.dynamodb.toList([ "foo", 123, { "bar" : "baz" } ])
Output:     {
               "L" : [
                   { "S" : "foo" },
                   { "N" : 123 },
                   {
                       "M" : {
                           "bar" : { "S" : "baz" }
                       }
                   }
               ]
           }
```

`$util.dynamodb.toListJson(List) : String`

Entspricht`$util.dynamodb.toList(List) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

ToMap-Dienstprogramme

Liste der ToMap-Dienstprogramme

`$util.dynamodb.toMap(Map) : Map`

Konvertiert eine Map in das DynamoDB-Kartenformat. Jeder Wert in der Map wird ebenfalls in das entsprechende DynamoDB-Format konvertiert. Es ist vorbestimmt, wie einige verschachtelte Objekte dargestellt werden: z. B. werden Listen ("L") anstelle von Sätzen ("SS", "NS", "BS") verwendet. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
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`

Entspricht`$util.dynamodb.toMap(Map) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

`$util.dynamodb.toMapValues(Map) : Map`

Erstellt eine Kopie der Map, in der jeder Wert in das entsprechende DynamoDB-Format konvertiert wurde. Es ist vorbestimmt, wie einige verschachtelte Objekte dargestellt werden: z. B. werden Listen ("L") anstelle von Sätzen ("SS", "NS", "BS") verwendet.

```
Input:      $util.dynamodb.toMapValues({ "foo": "bar", "baz" : 1234, "beep":
 [ "boop"] })
Output:     {
               "foo"  : { "S" : "bar" },
               "baz"  : { "N" : 1234 },
               "beep" : {
                   "L" : [
                       { "S" : "boop" }
                   ]
               }
           }
```

> **Note**
>
> Dies unterscheidet sich `$util.dynamodb.toMap(Map) : Map` geringfügig davon, dass nur der Inhalt des DynamoDB-Attributwerts zurückgegeben wird, nicht jedoch der gesamte Attributwert selbst. Die folgenden Aussagen sind beispielsweise identisch:
>
> ```
> $util.dynamodb.toMapValues($map)
> ```

```
$util.dynamodb.toMap($map).get("M")
```

$util.dynamodb.toMapValuesJson(Map) : String

Entspricht`$util.dynamodb.toMapValues(Map) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

S3Object-Dienstprogramme

Liste der S3Object-Dienstprogramme

$util.dynamodb.toS3Object(String key, String bucket, String region) : Map

Konvertiert den Schlüssel, den Bucket und die Region in die DynamoDB S3-Objektdarstellung. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
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

Entspricht`$util.dynamodb.toS3Object(String key, String bucket, String region) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

$util.dynamodb.toS3Object(String key, String bucket, String region, String version) : Map

Konvertiert den Schlüssel, den Bucket, die Region und die optionale Version in die DynamoDB S3-Objektdarstellung. Dies gibt ein Objekt zurück, das den DynamoDB-Attributwert beschreibt.

```
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`

Entspricht`$util.dynamodb.toS3Object(String key, String bucket, String region, String version) : Map`, gibt aber den DynamoDB-Attributwert als JSON-kodierte Zeichenfolge zurück.

`$util.dynamodb.fromS3ObjectJson(String) : Map`

Akzeptiert den Zeichenkettenwert eines DynamoDB S3-Objekts und gibt eine Map zurück, die den Schlüssel, den Bucket, die Region und die optionale Version enthält.

```
Input:      $util.dynamodb.fromS3ObjectJson({ "S" : "{ \"s3\" : { \"key\" : \"foo\",
 \"bucket\" : \"bar\", \"region\" : \"baz\", \"version\" = \"beep\" } }" })
Output:     { "key" : "foo", "bucket" : "bar", "region" : "baz", "version" :
 "beep" }
```

Amazon RDS-Helfer in $util.rds

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

`$util.rds`enthält Hilfsmethoden, die Amazon RDS-Operationen formatieren, indem sie überflüssige Daten in Ergebnisausgaben entfernen.

Liste der $util.rds-Dienstprogramme

`$util.rds.toJsonString(String serializedSQLResult): String`

Gibt a zurück, `String` indem das stringifizierte Rohergebnisformat des Amazon Relational Database Service (Amazon RDS) -Daten-API-Vorgangs in eine präzisere Zeichenfolge umgewandelt wird. Die zurückgegebene Zeichenfolge ist eine serialisierte Liste von SQL-Datensätzen der Ergebnismenge. Jeder Datensatz wird als eine Sammlung von Schlüssel-Wert-Paaren dargestellt. Die Schlüssel sind die zugehörigen Spaltennamen.

Wenn die entsprechende Anweisung in der Eingabe eine SQL-Abfrage war, die eine Mutation verursacht (zum Beispiel INSERT, UPDATE, DELETE), wird eine leere Liste zurückgegeben. Die

Abfrage `select * from Books limit 2` liefert beispielsweise das Rohergebnis der Amazon RDS-Datenoperation:


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

Das `util.rds.toJsonString` ist:

```
[
  {
    "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`

Das ist dasselbe wie`util.rds.toJsonString`, aber das Ergebnis ist ein JSON`Object`.

HTTP-Helfer in $util.http

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Das `$util.http` Hilfsprogramm bietet Hilfsmethoden, mit denen Sie HTTP-Anforderungsparameter verwalten und Antwortheader hinzufügen können.

Liste der $util.http-Dienstprogramme

`$util.http.copyHeaders(Map) : Map`

Kopiert die Header aus der Map, mit Ausnahme der folgenden eingeschränkten HTTP-Header:

- transfer-encoding
- connection
- host
- expect
- keep-alive

- upgrade
- proxy-authenticate
- proxy-authorization
- te
- content-length

Sie können dieses Hilfsprogramm verwenden, um Anforderungsheader an Ihren Downstream-HTTP-Endpunkt weiterzuleiten.

```
{
    ...
    "params": {
        ...
        "headers": $util.http.copyHeaders($ctx.request.headers),
        ...
    },
    ...
}
```

$util.http. addResponseHeader(Zeichenfolge, Objekt)

Fügt einen einzelnen benutzerdefinierten Header mit dem Namen (`String`) und dem Wert (`Object`) der Antwort hinzu. Die folgenden Einschränkungen gelten:

- Zusätzlich zur Liste der eingeschränkten Header für `copyHeaders(Map)` dürfen Header-Namen keinem der folgenden Werte entsprechen:
  - Access-Control-Allow-Credentials
  - Access-Control-Allow-Origin
  - Access-Control-Expose-Headers
  - Access-Control-Max-Age
  - Access-Control-Allow-Methods
  - Access-Control-Allow-Headers
  - Vary
  - Content-Type
- Headernamen dürfen nicht mit den eingeschränkten Präfixen `x-amzn-` oder beginnen. `x-amz-`
- Die Größe der benutzerdefinierten Antwort-Header darf 4 KB nicht überschreiten. Dies schließt Header-Namen und -Werte ein.

- Sie sollten jeden Antwortheader einmal pro GraphQL-Operation definieren. Wenn Sie jedoch mehrmals einen benutzerdefinierten Header mit demselben Namen definieren, erscheint die neueste Definition in der Antwort. Alle Header werden unabhängig von der Benennung auf die Größenbeschränkung für Header angerechnet.
- Header mit einem leeren oder eingeschränkten Namen (`String`) oder einem Nullwert (`Object`) werden ignoriert und führen zu einem `ResponseHeaderError` Fehler, der zur Ausgabe des `errors` Vorgangs hinzugefügt wird.

```
export function request(ctx) {
  util.http.addResponseHeader('itemsCount', 7)
  util.http.addResponseHeader('render', ctx.args.render)
  return {}
}
```

`$util.http.addResponseHeaders(Map)`

Fügt der Antwort mehrere Antwort-Header aus der angegebenen Zuordnung von Namen (`String`) und Werten hinzu. (`Object`) Dieselben Einschränkungen, die für die `addResponseHeader(String, Object)` Methode aufgeführt sind, gelten auch für diese Methode.

```
export function request(ctx) {
  const headers = {
    headerInt: 12,
    headerString: 'stringValue',
    headerObject: {
      field1: 7,
      field2: 'string'
    }
  }
  util.http.addResponseHeaders(headers)
  return {}
}
```

XML-Helfer in $util.xml

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

`$util.xml`enthält Hilfsmethoden, die es einfacher machen können, XML-Antworten in JSON oder ein Dictionary zu übersetzen.

Liste der $util.xml-Dienstprogramme

`$util.xml.toMap(String) : Map`

Konvertiert eine XML-Zeichenfolge in ein Dictionary.

```
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`

Konvertiert eine XML-Zeichenfolge in eine JSON-Zeichenfolge. Dies ist ähnlich wie ToMap, außer dass die Ausgabe eine Zeichenfolge ist. Dies ist nützlich, wenn Sie direkt umwandeln und die XML-Antwort aus einem HTTP-Objekt an JSON zurückgeben.

`$util.xml.toJsonString(String, Boolean) : String`

Konvertiert eine XML-Zeichenfolge in eine JSON-Zeichenfolge mit einem optionalen booleschen Parameter, um zu bestimmen, ob Sie die JSON-Zeichenfolge als Zeichenfolge kodieren möchten.

Transformationshelfer in $util.transform

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

`$util.transform`enthält Hilfsmethoden, die es einfacher machen, komplexe Operationen mit Datenquellen durchzuführen, wie z. B. Amazon DynamoDB DynamoDB-Filteroperationen.

Helfer bei der Transformation

Liste der Hilfsprogramme für Transformationshelfer

`$util.transform.toDynamoDBFilterExpression(Map) : Map`

Konvertiert eine Eingabezeichenfolge in einen Filterausdruck zur Verwendung mit 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`

Konvertiert die angegebene Eingabe in den entsprechenden OpenSearch Query-DSL-Ausdruck und gibt sie als JSON-Zeichenfolge zurück.

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

Es wird davon ausgegangen, dass der Standardoperator AND ist.

Transformation Helpers, Abonnementfilter

Liste der Dienstprogramme für Abonnementfilter für Transformationshelfer

`$util.transform.toSubscriptionFilter(Map) : Map`

Konvertiert ein `Map` Eingabeobjekt in ein `SubscriptionFilter` Ausdrucksobjekt. Die `$util.transform.toSubscriptionFilter` Methode wird als Eingabe für die `$extensions.setSubscriptionFilter()` Erweiterung verwendet. Weitere Informationen finden Sie unter Erweiterungen.

`$util.transform.toSubscriptionFilter(Map, List) : Map`

Konvertiert ein `Map` Eingabeobjekt in ein `SubscriptionFilter` Ausdrucksobjekt. Die `$util.transform.toSubscriptionFilter` Methode wird als Eingabe für die

`$extensions.setSubscriptionFilter()` Erweiterung verwendet. Weitere Informationen finden Sie unter Erweiterungen.

Das erste Argument ist das `Map` Eingabeobjekt, das in das `SubscriptionFilter` Ausdrucksobjekt konvertiert wurde. Das zweite Argument besteht `List` aus Feldnamen, die im ersten `Map` Eingabeobjekt bei der Konstruktion des `SubscriptionFilter` Ausdrucksobjekts ignoriert werden.

`$util.transform.toSubscriptionFilter(Map, List, Map) : Map`

Konvertiert ein `Map` Eingabeobjekt in ein `SubscriptionFilter` Ausdrucksobjekt. Die `$util.transform.toSubscriptionFilter` Methode wird als Eingabe für die `$extensions.setSubscriptionFilter()` Erweiterung verwendet. Weitere Informationen finden Sie unter Erweiterungen.

Das erste Argument ist das `Map` Eingabeobjekt, das in das `SubscriptionFilter` Ausdrucksobjekt konvertiert wurde, das zweite Argument enthält Feldnamen, die im ersten `Map` Eingabeobjekt ignoriert werden, und das dritte Argument ist ein `Map` Eingabeobjekt mit strengen Regeln, das bei der Konstruktion des `SubscriptionFilter` Ausdrucksobjekts berücksichtigt wird. `List` Diese strengen Regeln sind so im `SubscriptionFilter` Ausdrucksobjekt enthalten, dass mindestens eine der Regeln erfüllt ist, um den Abonnementfilter zu bestehen.

Argumente für den Abonnementfilter

In der folgenden Tabelle wird erklärt, wie die Argumente der folgenden Dienstprogramme definiert sind:

- `$util.transform.toSubscriptionFilter(Map) : Map`
- `$util.transform.toSubscriptionFilter(Map, List) : Map`
- `$util.transform.toSubscriptionFilter(Map, List, Map) : Map`

Argument 1: Map

Argument 1 ist ein `Map` Objekt mit den folgenden Schlüsselwerten:

- Feldnamen
- „und“
- „oder“

Bei Feldnamen als Schlüssel haben die Bedingungen für die Einträge dieser Felder die Form von`"operator" : "value"`.

Das folgende Beispiel zeigt, wie Einträge hinzugefügt werden können: `Map`

```
"field_name" : {
                    "operator1" : value
               }

## We can have multiple conditions for the same field_name:

"field_name" : {
                    "operator1" : value
                    "operator2" : value
                    .
                    .
                    .
               }
```

Wenn ein Feld zwei oder mehr Bedingungen enthält, wird davon ausgegangen, dass alle diese Bedingungen die OR-Operation verwenden.

Die Eingabe `Map` kann auch „und“ und „oder“ als Schlüssel enthalten, was bedeutet, dass alle darin enthaltenen Einträge je nach Schlüssel mit DER UND- oder OR-Logik verknüpft werden sollten. Die Schlüsselwerte „und“ und „oder“ erwarten eine Reihe von Bedingungen.

```
"and" : [

            {
                "field_name1" : {
                    "operator1" : value
                }
             },

             {
                "field_name2" : {
                    "operator1" : value
                }
             },
             .
             .
        ].
```

Beachten Sie, dass Sie „und“ und „oder“ verschachteln können. Das heißt, Sie können „und“ /"oder“ innerhalb eines anderen „und“ /"oder“ -Blocks verschachtelt haben. Dies funktioniert jedoch nicht für einfache Felder.

```
"and" : [

            {
                "field_name1" : {
                    "operator" : value
                }
             },

             {
                "or" : [
                            {
                                "field_name2" : {
                                    "operator" : value
                                }
                            },

                            {
                                "field_name3" : {
                                    "operator" : value
                                }
                            }

                        ].
```

Das folgende Beispiel zeigt eine Eingabe von Argument 1 mit`$util.transform.toSubscriptionFilter(Map) : Map`.

Eingabe (en)

Argument 1: Karte:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
```

```
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 2000
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Ausgabe

Das Ergebnis ist ein `Map` Objekt:

```
{
  "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

Argument 2 enthält eine Reihe `List` von Feldnamen, die bei der Erstellung des `SubscriptionFilter` Ausdrucksobjekts nicht in der Eingabe `Map` (Argument 1) berücksichtigt werden sollten. Das `List` kann auch leer sein.

Das folgende Beispiel zeigt die Eingaben von Argument 1 und Argument 2 mit`$util.transform.toSubscriptionFilter(Map, List) : Map`.

Eingabe (en)

Argument 1: Karte:

```
{

  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
```

```
}
```

Argument 2: Liste:

```
["percentageUp", "author"]
```

Ausgabe

Das Ergebnis ist ein `Map` Objekt:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    }
  ]
}
```

Argument 3: Map

Argument 3 ist ein `Map` Objekt, das Feldnamen als Schlüsselwerte hat (darf nicht „und“ oder „oder“ enthalten). Bei Feldnamen als Schlüssel handelt es sich bei den Bedingungen für diese Felder um Einträge in der Form von`"operator" : "value"`. Im Gegensatz zu Argument 1 kann Argument 3 nicht mehrere Bedingungen in demselben Schlüssel haben. Darüber hinaus hat Argument 3 keine „Und“ - oder „Oder“ -Klausel, sodass auch keine Verschachtelung erforderlich ist.

Argument 3 stellt eine Liste strenger Regeln dar, die dem `SubscriptionFilter` Ausdrucksobjekt hinzugefügt werden, sodass mindestens eine dieser Bedingungen erfüllt ist, um den Filter zu bestehen.

```
{
  "fieldname1": {
    "operator": value
  },
  "fieldname2": {
    "operator": value
  }
}
.
.
.
```

Das folgende Beispiel zeigt die Eingaben von Argument 1, Argument 2 und Argument 3 mit`$util.transform.toSubscriptionFilter(Map, List, Map) : Map`.

Eingabe (en)

Argument 1: Karte:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "lt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
```

```
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Argument 2: Liste:

```
["percentageUp", "author"]
```

Argument 3: Karte:

```
{
  "upvotes": {
    "gte": 250
  },
  "author": {
    "eq": "Person1"
  }
}
```

Ausgabe

Das Ergebnis ist ein `Map` Objekt:

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

Mathematische Helfer in $util.math

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

`$util.math`enthält Methoden zur Unterstützung gängiger mathematischer Operationen.

Liste der $util.math-Dienstprogramme

`$util.math.roundNum(Double) : Integer`

Nimmt einen doppelten Wert und rundet ihn auf die nächste Ganzzahl.

`$util.math.minVal(Double, Double) : Double`

Nimmt zwei Doppelungen und gibt den Mindestwert zwischen den beiden Doppelgängern zurück.

`$util.math.maxVal(Double, Double) : Double`

Nimmt zwei Doubles und gibt den Maximalwert zwischen den beiden Doubles zurück.

`$util.math.randomDouble() : Double`

Gibt einen zufälligen Doppelwert zwischen 0 und 1 zurück.

> **Important**
>
> Diese Funktion sollte nicht für Dinge verwendet werden, die eine Zufälligkeit mit hoher Entropie erfordern (z. B. Kryptografie).

`$util.math.randomWithinRange(Integer, Integer) : Integer`

Gibt einen zufälligen Ganzzahlwert innerhalb des angegebenen Bereichs zurück, wobei das erste Argument den unteren Wert des Bereichs und das zweite Argument den oberen Wert des Bereichs angibt.

> **Important**
>
> Diese Funktion sollte nicht für Anwendungen verwendet werden, die eine hohe Zufälligkeit der Entropie erfordern (z. B. Kryptografie).

Zeichenketten-Helfer in $util.str

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

`$util.str`enthält Methoden zur Unterstützung gängiger Zeichenkettenoperationen.

Liste der $util.str Dienstprogramme

`$util.str.toUpper(String) : String`

Nimmt eine Zeichenfolge und wandelt sie vollständig in Großbuchstaben um.

`$util.str.toLower(String) : String`

Nimmt eine Zeichenfolge und wandelt sie vollständig in Kleinbuchstaben um.

`$util.str.toReplace(String, String, String) : String`

Ersetzt eine Teilzeichenfolge innerhalb einer Zeichenfolge durch eine andere Zeichenfolge. Das erste Argument gibt die Zeichenfolge an, für die der Ersetzungsvorgang ausgeführt werden soll. Das zweite Argument gibt die Teilzeichenfolge an, die ersetzt werden soll. Das dritte Argument gibt die Zeichenfolge an, durch die das zweite Argument ersetzt werden soll. Im Folgenden finden Sie ein Beispiel für die Verwendung dieses Dienstprogramms:

```
INPUT:      $util.str.toReplace("hello world", "hello", "mellow")
OUTPUT:     "mellow world"
```

`$util.str.normalize(String, String) : String`

Normalisiert eine Zeichenfolge mit einer der vier Unicode-Normalisierungsformen: NFC, NFD, NFKC oder NFKD. Das erste Argument ist die zu normalisierende Zeichenfolge. Das zweite Argument ist entweder „nfc“, „nfd“, „nfkc“ oder „nfkd“ und gibt den Normalisierungstyp an, der für den Normalisierungsprozess verwendet werden soll.

Erweiterungen

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

`$extensions`enthält eine Reihe von Methoden, mit denen Sie zusätzliche Aktionen in Ihren Resolvern ausführen können.

Erweiterungen zwischenspeichern

`$extensions.evictFromApiCache(String, String, Object) : Object`

Löscht ein Element aus dem serverseitigen AWS AppSync Cache. Das erste Argument ist der Typname. Das zweite Argument ist der Feldname. Das dritte Argument ist ein Objekt, das Schlüssel-Wert-Paar-Elemente enthält, die den Zwischenspeicher-Schlüsselwert angeben. Sie müssen die Elemente im Objekt in derselben Reihenfolge platzieren wie die Caching-Schlüssel in den zwischengespeicherten Resolvern. `cachingKey`

> **Note**
>
> Dieses Tool funktioniert nur für Mutationen, nicht für Abfragen.

Abonnementerweiterungen

`$extensions.setSubscriptionFilter(filterJsonObject)`

Definiert erweiterte Abonnementfilter. Jedes Abonnementbenachrichtigungsereignis wird anhand der bereitgestellten Abonnementfilter bewertet und sendet Benachrichtigungen an Kunden, wenn

alle Filter als erfüllt geltentrue. Das Argument wird `filterJsonObject` im folgenden Abschnitt beschrieben.

> **Note**
>
> Sie können diese Erweiterungsmethode nur in den Antwortzuordnungsvorlagen eines Abonnement-Resolvers verwenden.

`$extensions.setSubscriptionInvalidationFilter(filterJsonObject)`

Definiert Filter für die Invalidierung von Abonnements. Abonnementfilter werden anhand der Payload für die Invalidierung bewertet und machen dann ein bestimmtes Abonnement ungültig, wenn die Filter Folgendes ergeben. `true` Das Argument wird im `filterJsonObject` folgenden Abschnitt beschrieben.

> **Note**
>
> Sie können diese Erweiterungsmethode nur in den Antwortzuordnungsvorlagen eines Abonnement-Resolvers verwenden.

`$extensions.invalidateSubscriptions(invalidationJsonObject)`

Wird verwendet, um die Invalidierung eines Abonnements aufgrund einer Mutation einzuleiten. Das Argument wird `invalidationJsonObject` im folgenden Abschnitt beschrieben.

> **Note**
>
> Diese Erweiterung kann nur in den Response-Mapping-Vorlagen der Mutationsresolver verwendet werden.
> Sie können in einer einzelnen Anfrage nur maximal fünf eindeutige `$extensions.invalidateSubscriptions()` Methodenaufrufen verwenden. Wenn Sie dieses Limit überschreiten, erhalten Sie einen GraphQL-Fehler.

Argument: filterJsonObject

Das JSON-Objekt definiert entweder Abonnement- oder Invalidierungsfilter. Es ist eine Reihe von Filtern in einem`filterGroup`. Jeder Filter ist eine Sammlung einzelner Filter.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                }
           ]

        },
        {
           "filters" : [
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
           ]

        }
    ]
}
```

Jeder Filter hat drei Attribute:

- `fieldName`— Das GraphQL-Schemafeld.
- `operator`— Der Operatortyp.
- `value`— Die Werte, die mit dem `fieldName` Wert der Abonnementbenachrichtigung verglichen werden sollen.

Im Folgenden finden Sie ein Beispiel für die Zuweisung dieser Attribute:

```
{
```

```
 "fieldName" : "severity",
 "operator" : "le",
 "value" : $context.result.severity
}
```

Feld: fieldName

Der String-Typ `fieldName` bezieht sich auf ein im GraphQL-Schema definiertes Feld, das mit der Payload `fieldName` in der Abonnementbenachrichtigung übereinstimmt. Wenn eine Übereinstimmung gefunden wird, wird das `value` GraphQL-Schemafeld mit dem `value` des Abonnementbenachrichtigungsfilters verglichen. Im folgenden Beispiel entspricht der `fieldName` Filter dem `service` Feld, das in einem bestimmten GraphQL-Typ definiert ist. Wenn die Benachrichtigungs-Payload ein `service` Feld mit einer `value` Entsprechung von enthältAWS AppSync, wird der Filter wie folgt ausgewertet: `true`

```
{
 "fieldName" : "service",
 "operator" : "eq",
 "value" : "AWS AppSync"
}
```

Feld: Wert

Der Wert kann je nach Operator einen anderen Typ haben:

- Eine einzelne Zahl oder ein boolescher Wert
  - Beispiele für Zeichenketten:, `"test"` `"service"`
  - Beispiele für Zahlen:`1`,`2`, `45.75`
  - Boolesche Beispiele:, `true` `false`
- Zahlen- oder Zeichenkettenpaare
  - Beispiel für ein Zeichenkettenpaar:`["test1","test2"]`, `["start","end"]`
  - Beispiel für ein Zahlenpaar:`[1,4]`,`[67,89]`, `[12.45, 95.45]`
- Reihen von Zahlen oder Zeichenketten
  - Beispiel für ein String-Array: `["test1","test2","test3","test4","test5"]`
  - Beispiel für ein Zahlenarray:`[1,2,3,4,5]`, `[12.11,46.13,45.09,12.54,13.89]`

Feld: Operator

Eine Zeichenfolge, bei der Groß- und Kleinschreibung beachtet wird, mit den folgenden möglichen Werten:

Operator	Beschreibung	Mögliche Wertetypen
eq	Gleich	Ganzzahl, Gleitkommazahl, Zeichenfolge, Boolean
Eins	Ungleich	Ganzzahl, Gleitkommazahl, Zeichenfolge, Boolean
le	Kleiner als oder gleich	Ganzzahl, Gleitkommazahl, Zeichenfolge
lt	kleiner als	Ganzzahl, Fließkommazahl, Zeichenfolge
ge	Größer als oder gleich	Ganzzahl, Gleitkommazahl, Zeichenfolge
gt	größer als	Ganzzahl, Fließkommazahl, Zeichenfolge
enthält	Prüft, ob eine Teilsequenz oder ein Wert in der Menge vorhanden ist.	Ganzzahl, Fließkommazahl, Zeichenfolge
Nicht enthält	Prüft, ob eine Teilsequenz oder ein Wert in der Menge fehlt.	Ganzzahl, Fließkommazahl, Zeichenfolge
Beginnt mit	Sucht nach einem Präfix.	Zeichenfolge
in	Prüft auf übereinstimmende Elemente in der Liste.	Array aus Ganzzahl, Gleitkommazahl oder Zeichenfolge

notIn	Sucht nach passenden Elementen, die nicht in der Liste enthalten sind.	Array aus Ganzzahl, Gleitkommazahl oder Zeichenfolge
zwischen	Zwischen zwei Werten	Ganzzahl, Gleitkommazahl, Zeichenfolge
Enthält Beliebig	Enthält gemeinsame Elemente	Ganzzahl, Fließkommazahl, Zeichenfolge

In der folgenden Tabelle wird beschrieben, wie die einzelnen Operatoren in der Abonnementbenachrichtigung verwendet werden.

eq (equal)

Der `eq` Operator prüft, `true` ob der Wert des Felds für die Abonnementbenachrichtigung dem Wert des Filters entspricht und diesem genau entspricht. Im folgenden Beispiel bewertet der Filter, `true` ob die Abonnementbenachrichtigung ein `service` Feld enthält, dessen Wert entspricht. `AWS AppSync`

Mögliche Werttypen: Integer, Float, String, Boolean

```
{
 "fieldName" : "service",
 "operator" : "eq",
 "value" : "AWS AppSync"
}
```

ne (not equal)

Der `ne` Operator prüft, `true` ob sich der Wert des Felds für die Abonnementbenachrichtigung vom Wert des Filters unterscheidet. Im folgenden Beispiel prüft der Filter, `true` ob die Abonnementbenachrichtigung ein `service` Feld mit einem anderen Wert als enthält. `AWS AppSync`

Mögliche Werttypen: Integer, Float, String, Boolean

```
{
 "fieldName" : "service",
 "operator" : "ne",
```

```
 "value" : "AWS AppSync"
}
```

le (less or equal)

Der `le` Operator prüft, `true` ob der Wert des Felds für die Abonnementbenachrichtigung kleiner oder gleich dem Wert des Filters ist. Im folgenden Beispiel bewertet der Filter, `true` ob die Abonnementbenachrichtigung ein `size` Feld enthält, dessen Wert kleiner oder gleich ist. `5`

Mögliche Werttypen: Integer, Float, String

```
{
 "fieldName" : "size",
 "operator" : "le",
 "value" : 5
}
```

lt (less than)

Der `lt` Operator prüft, `true` ob der Wert des Felds für die Abonnementbenachrichtigung niedriger als der Wert des Filters ist. Im folgenden Beispiel bewertet der Filter, `true` ob die Abonnementbenachrichtigung ein `size` Feld mit einem niedrigeren Wert als enthält. `5`

Mögliche Werttypen: Integer, Float, String

```
{
 "fieldName" : "size",
 "operator" : "lt",
 "value" : 5
}
```

ge (greater or equal)

Der `ge` Operator prüft, `true` ob der Wert des Felds für die Abonnementbenachrichtigung größer oder gleich dem Wert des Filters ist. Im folgenden Beispiel bewertet der Filter, `true` ob die Abonnementbenachrichtigung ein `size` Feld enthält, dessen Wert größer oder gleich ist. `5`

Mögliche Werttypen: Integer, Float, String

```
{
 "fieldName" : "size",
 "operator" : "ge",
```

```
 "value" : 5
}
```

gt (greater than)

Der `gt` Operator prüft, `true` ob der Wert des Felds für die Abonnementbenachrichtigung größer als der Wert des Filters ist. Im folgenden Beispiel bewertet der Filter, `true` ob die Abonnementbenachrichtigung ein `size` Feld mit einem Wert größer als enthält. `5`

Mögliche Werttypen: Integer, Float, String

```
{
 "fieldName" : "size",
 "operator" : "gt",
 "value" : 5
}
```

contains

Der `contains` Operator sucht nach einer Teilzeichenfolge, Teilsequenz oder einem Wert in einer Menge oder einem einzelnen Element. Ein Filter mit dem `contains` Operator überprüft, `true` ob der Wert des Felds für die Abonnementbenachrichtigung den Filterwert enthält. Im folgenden Beispiel überprüft der Filter, `true` ob die Abonnementbenachrichtigung ein `seats` Feld enthält, dessen Array-Wert den Wert enthält. `10`

Mögliche Werttypen: Integer, Float, String

```
{
 "fieldName" : "seats",
 "operator" : "contains",
 "value" : 10
}
```

In einem anderen Beispiel überprüft der Filter, `true` ob die Abonnementbenachrichtigung ein `event` Feld mit einer `launch` Teilzeichenfolge enthält.

```
{
 "fieldName" : "event",
 "operator" : "contains",
 "value" : "launch"
}
```

notContains

Der `notContains` Operator prüft, ob eine Teilzeichenfolge, eine Teilsequenz oder ein Wert in einer Gruppe oder einem einzelnen Element fehlt. Der Filter mit dem `notContains` Operator ermittelt, `true` ob der Wert im Feld für die Abonnementbenachrichtigung den Filterwert nicht enthält. Im folgenden Beispiel überprüft der Filter, `true` ob die Abonnementbenachrichtigung ein `seats` Feld enthält, dessen Array-Wert den Wert nicht enthält. `10`

Mögliche Werttypen: Integer, Float, String

```
{
 "fieldName" : "seats",
 "operator" : "notContains",
 "value" : 10
}
```

In einem anderen Beispiel überprüft der Filter, `true` ob die Abonnementbenachrichtigung einen `event` Feldwert ohne `launch` Untersequenz enthält.

```
{
 "fieldName" : "event",
 "operator" : "notContains",
 "value" : "launch"
}
```

beginsWith

Der `beginsWith` Operator sucht nach einem Präfix in einer Zeichenfolge. Der Filter, der den `beginsWith` Operator enthält, bewertet, `true` ob der Wert des Felds für die Abonnementbenachrichtigung mit dem Wert des Filters beginnt. Im folgenden Beispiel überprüft der Filter, `true` ob die Abonnementbenachrichtigung ein `service` Feld enthält, dessen Wert mit beginnt. `AWS`

Möglicher Wertetyp: Zeichenfolge

```
{
 "fieldName" : "service",
 "operator" : "beginsWith",
 "value" : "AWS"
}
```

in

Der `in` Operator sucht nach passenden Elementen in einem Array. Der Filter, der den `in` Operator enthält, überprüft, `true` ob der Wert des Felds für die Abonnementbenachrichtigung in einem Array vorhanden ist. Im folgenden Beispiel bewertet der Filter, `true` ob die Abonnementbenachrichtigung ein `severity` Feld mit einem der im Array vorhandenen Werte enthält:. `[1,2,3]`

Möglicher Wertetyp: Array mit Integer, Float oder String

```
{
 "fieldName" : "severity",
 "operator" : "in",
 "value" : [1,2,3]
}
```

notIn

Der `notIn` Operator sucht nach fehlenden Elementen in einem Array. Der Filter, der den `notIn` Operator enthält, gibt aus, `true` ob der Wert des Felds für die Abonnementbenachrichtigung nicht im Array vorhanden ist. Im folgenden Beispiel bewertet der Filter, `true` ob die Abonnementbenachrichtigung ein `severity` Feld mit einem der Werte enthält, die nicht im Array vorhanden sind:. `[1,2,3]`

Möglicher Wertetyp: Array mit Integer, Float oder String

```
{
 "fieldName" : "severity",
 "operator" : "notIn",
 "value" : [1,2,3]
}
```

between

Der `between` Operator sucht nach Werten zwischen zwei Zahlen oder Zeichenketten. Der Filter, der den `between` Operator enthält, überprüft, `true` ob der Wert des Felds für die Abonnementbenachrichtigung zwischen dem Wertepaar des Filters liegt. Im folgenden Beispiel überprüft der Filter, `true` ob die Abonnementbenachrichtigung ein `severity` Feld mit den Werten2,, 3 enthält. 4

Mögliche Werttypen: Paar aus Integer, Float oder String

```
{
 "fieldName" : "severity",
 "operator" : "between",
 "value" : [1,5]
}
```

containsAny

Der `containsAny` Operator sucht nach gemeinsamen Elementen in Arrays. Ein Filter mit dem `containsAny` Operator überprüft, `true` ob der Schnittpunkt zwischen dem eingestellten Wert des Felds für die Abonnementbenachrichtigung und dem eingestellten Wert des Filters nicht leer ist. Im folgenden Beispiel überprüft der Filter, `true` ob die Abonnementbenachrichtigung ein `seats` Feld mit einem Array-Wert enthält, der entweder oder `10` enthält. `15` Das bedeutet, dass der Filter auswertet, `true` ob die Abonnementbenachrichtigung den `seats` Feldwert `[10,11]` oder `[15,20,30]` hat.

Mögliche Werttypen: Integer, Float oder String

```
{
 "fieldName" : "seats",
 "operator" : "containsAny",
 "value" : [10, 15]
}
```

UND-Logik

Sie können mehrere Filter mithilfe der UND-Logik kombinieren, indem Sie mehrere Einträge innerhalb des `filters` Objekts im `filterGroup` Array definieren. Im folgenden Beispiel werten Filter aus, `true` ob die Abonnementbenachrichtigung ein `userId` Feld mit einem Wert enthält, der `1` AND `group` dem Feldwert entweder `Admin` oder entsprichtDeveloper.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                },
```

```
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
            ]

        }
    ]
}
```

ODER-Logik

Sie können mehrere Filter mithilfe der OR-Logik kombinieren, indem Sie mehrere Filterobjekte innerhalb des `filterGroup` Arrays definieren. Im folgenden Beispiel werten Filter aus, `true` ob die Abonnementbenachrichtigung ein `userId` Feld mit einem Wert enthält, der 1 ODER einem `group` Feldwert von entweder `Admin` oder entsprich`Developer`.

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                }
            ]

        },
        {
           "filters" : [
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
            ]

        }
    ]
}
```

Ausnahmen

Beachten Sie, dass es mehrere Einschränkungen für die Verwendung von Filtern gibt:

- Im `filters` Objekt können maximal fünf eindeutige `fieldName` Elemente pro Filter vorhanden sein. Das bedeutet, dass Sie maximal fünf einzelne `fieldName` Objekte mithilfe der UND-Logik kombinieren können.
- Für den `containsAny` Operator können maximal zwanzig Werte angegeben werden.
- Für die `notIn` Operatoren `in` und können maximal fünf Werte angegeben werden.
- Jede Zeichenfolge kann maximal 256 Zeichen lang sein.
- Bei jedem Zeichenkettenvergleich wird zwischen Groß- und Kleinschreibung unterschieden.
- Die Filterung verschachtelter Objekte ermöglicht bis zu fünf verschachtelte Filterebenen.
- Jede `filterGroup` kann maximal 10 haben. `filters` Das bedeutet, dass Sie maximal 10 `filters` mithilfe der ODER-Logik kombinieren können.
  - Der `in` Operator ist ein Sonderfall der OR-Logik. Im folgenden Beispiel gibt es zwei`filters`:

```
{
    "filterGroup": [
        {
           "filters" : [
                 {
                    "fieldName" : "userId",
                    "operator" : "eq",
                    "value" : 1
                },
                {
                    "fieldName" : "group",
                    "operator" : "in",
                    "value" : ["Admin", "Developer"]
                }
           ]
        }
    ]
}
```

Die vorherige Filtergruppe wird wie folgt ausgewertet und auf die maximale Filtergrenze angerechnet:

```
{
```

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

Argument: invalidationJsonObject

Das `invalidationJsonObject` definiert Folgendes:

- `subscriptionField`— Das GraphQL-Schemaabonnement, das ungültig werden soll. Ein einzelnes Abonnement, das als Zeichenfolge in der definiert ist`subscriptionField`, wird für ungültig erklärt.
- `payload`— Eine Liste mit Schlüssel-Wert-Paaren, die als Eingabe für die Ungültigerklärung von Abonnements verwendet wird, wenn der Invalidierungsfilter A anhand ihrer Werte auswertet. `true`

Im folgenden Beispiel werden abonnierte und verbundene Clients, die das Abonnement verwenden, ungültig gemacht, wenn der im `onUserDelete` Abonnement-Resolver definierte Invalidierungsfilter das Ergebnis anhand des Werts auswertet. `true` `payload`

```
$extensions.invalidateSubscriptions({
        "subscriptionField": "onUserDelete",
        "payload": {
                "group": "Developer"
                "type" : "Full-Time"
      }
    })
```

AWS AppSync Referenz zur Resolver-Mapping-Vorlage für DynamoDB

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Mit der AWS AppSync DynamoDB-Funktion können Sie GraphQL verwenden, um Daten in vorhandenen Amazon DynamoDB-Tabellen in Ihrem Konto zu speichern und abzurufen, indem Sie eine eingehende GraphQL-Anfrage einem DynamoDB-Aufruf zuordnen und dann die DynamoDB-Antwort wieder GraphQL zuordnen. In diesem Abschnitt werden die Anforderungs- und Antworthandler für unterstützte DynamoDB-Operationen beschrieben:

- GetItem- Mit der GetItem Anfrage können Sie die DynamoDB-Funktion anweisen, eine GetItem Anfrage an DynamoDB zu stellen, und Sie können den Schlüssel des Elements in DynamoDB angeben und angeben, ob ein konsistenter Lesevorgang verwendet werden soll oder nicht.
- PutItem - Mit dem PutItem Anforderungszuordnungsdokument können Sie die DynamoDB-Funktion anweisen, eine PutItem Anfrage an DynamoDB zu stellen, und Sie können den Schlüssel des Elements in DynamoDB, den vollständigen Inhalt des Elements (bestehend aus Schlüssel und AttributeValues) und Bedingungen für den Erfolg des Vorgangs angeben.
- UpdateItem - Mit der UpdateItem Anforderung können Sie die DynamoDB-Funktion anweisen, eine UpdateItem Anfrage an DynamoDB zu stellen, und Sie können den Schlüssel des Elements

in DynamoDB, einen Aktualisierungsausdruck, der beschreibt, wie das Element in DynamoDB aktualisiert wird, und die Bedingungen für den Erfolg des Vorgangs angeben.

- DeleteItem - Mit der DeleteItem Anfrage können Sie die DynamoDB-Funktion anweisen, eine DeleteItem Anfrage an DynamoDB zu stellen, und Sie können den Schlüssel des Elements in DynamoDB und die Bedingungen für den Erfolg des Vorgangs angeben.
- Abfrage — Mit dem Query-Anforderungsobjekt können Sie den DynamoDB-Resolver anweisen, eine Abfrageanforderung an DynamoDB zu stellen, und Sie können den Schlüsselausdruck, den zu verwendenden Index, zusätzliche Filter, die Anzahl der zurückzugebenden Elemente, die Verwendung konsistenter Lesevorgänge, die Abfragerichtung (vorwärts oder rückwärts) und Paginierungstoken angeben.
- Scan — Mit der Scananforderung können Sie die DynamoDB-Funktion anweisen, eine Scananforderung an DynamoDB zu stellen, und Sie können einen Filter angeben, um Ergebnisse auszuschließen, welcher Index verwendet werden soll, wie viele Elemente zurückgegeben werden sollen, ob konsistente Lesevorgänge, Paginierungstoken und parallel Scans verwendet werden sollen.
- Sync — Mit dem Sync-Anforderungsobjekt können Sie alle Ergebnisse aus einer DynamoDB-Tabelle abrufen und anschließend nur die Daten empfangen, die seit Ihrer letzten Abfrage geändert wurden (die Delta-Updates). Synchronisierungsanforderungen können nur an versionierte DynamoDB-Datenquellen gestellt werden. Sie können einen Filter angeben, um Ergebnisse, die Anzahl der zurückzugebenden Elemente, die Paginierungstoken und den Zeitpunkt, an dem Ihr letzter Synchronisierungsvorgang gestartet wurde, auszuschließen.
- BatchGetItem - Mit dem BatchGetItem Anforderungsobjekt können Sie die DynamoDB-Funktion anweisen, eine BatchGetItem Anforderung an DynamoDB zu stellen, um mehrere Elemente abzurufen, möglicherweise über mehrere Tabellen hinweg. Für dieses Anforderungsobjekt müssen Sie die Tabellennamen angeben, aus denen die Elemente abgerufen werden sollen, und die Schlüssel der Elemente, die aus jeder Tabelle abgerufen werden sollen.
- BatchDeleteItem - Mit dem BatchDeleteItem Anforderungsobjekt können Sie die DynamoDB-Funktion anweisen, eine BatchWriteItem Anforderung an DynamoDB zu stellen, um mehrere Elemente zu löschen, möglicherweise über mehrere Tabellen hinweg. Für dieses Anforderungsobjekt müssen Sie die Tabellennamen angeben, aus denen die Elemente gelöscht werden sollen, und die Schlüssel der Elemente, die aus jeder Tabelle gelöscht werden sollen.
- BatchPutItem - Mit dem BatchPutItem Anforderungsobjekt können Sie die DynamoDB-Funktion anweisen, eine BatchWriteItem Anforderung an DynamoDB zu stellen, um mehrere Elemente zu platzieren, möglicherweise in mehreren Tabellen. Für dieses Anforderungsobjekt müssen Sie

die Tabellennamen angeben, in die die Elemente eingefügt werden sollen, und die vollständigen Elemente, die in jede Tabelle eingefügt werden sollen.

- TransactGetItems - Mit dem TransactGetItems Anforderungsobjekt können Sie die DynamoDB-Funktion anweisen, eine TransactGetItems Anforderung an DynamoDB zu stellen, um mehrere Elemente abzurufen, möglicherweise über mehrere Tabellen hinweg. Für dieses Anforderungsobjekt müssen Sie den Tabellennamen jedes Anforderungselements angeben, aus dem das Element abgerufen werden soll, und den Schlüssel jedes Anforderungselements, das aus jeder Tabelle abgerufen werden soll.
- TransactWriteItems - Mit dem TransactWriteItems Anforderungsobjekt können Sie die DynamoDB-Funktion anweisen, eine TransactWriteItems Anforderung an DynamoDB zu stellen, um mehrere Elemente zu schreiben, möglicherweise in mehrere Tabellen. Für dieses Anforderungsobjekt müssen Sie den Zieltabellennamen jedes Anforderungselements, die Operation jedes auszuführenden Anforderungselements und den Schlüssel jedes zu schreibenden Anforderungselements angeben.
- Typsystem (Anforderungszuordnung) — Erfahren Sie mehr darüber, wie DynamoDB-Typisierung in Anfragen integriert AWS AppSync wird.
- Typsystem (Antwortzuordnung) — Erfahren Sie mehr darüber, wie DynamoDB-Typen in einer Antwortnutzlast automatisch in GraphQL oder JSON konvertiert werden.
- Filter — Erfahren Sie mehr über Filter für Abfrage- und Scanvorgänge.
- Bedingungsausdrücke — Erfahren Sie mehr über Bedingungsausdrücke für PutItem UpdateItem, und DeleteItem Operationen.
- Ausdrücke für Transaktionsbedingungen — Erfahren Sie mehr über Bedingungsausdrücke für TransactWriteItems Operationen.
- Projektionen — Erfahren Sie mehr darüber, wie Sie Attribute bei Lesevorgängen angeben.

GetItem

Mit dem `GetItem` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `GetItem` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Der Schlüssel des Elements in DynamoDB
- Ob ein Consistent-Lesevorgang verwendet wird oder nicht

Das `GetItem`-Zuweisungsdokument weist die folgende Struktur auf:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "consistentRead" : true,
    "projection" : {
        ...
    }
}
```

Die Felder sind wie folgt definiert:

GetItem Felder

GetItem Liste der Felder

`version`

Die Version der Vorlagedefinition. Aktuell werden `2017-02-28` und `2018-05-29` unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `GetItem`-DynamoDB-Operation durchzuführen, muss diese auf `GetItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der Schlüssel des Elements in DynamoDB. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`consistentRead`

Ob ein stark konsistenter Lesevorgang mit DynamoDB durchgeführt werden soll oder nicht. Dieser Schritt ist optional und standardmäßig auf `false` gesetzt.

`projection`

Eine Projektion, die verwendet wird, um die Attribute anzugeben, die von der DynamoDB-Operation zurückgegeben werden sollen. Weitere Informationen zu Projektionen finden Sie unter Projektionen. Dies ist ein optionales Feld.

Das von DynamoDB zurückgegebene Element wird automatisch in primitive GraphQL- und JSON-Typen konvertiert und ist im Mapping-Kontext () verfügbar. `$context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu Vorlagen für die Antwortzuweisung finden Sie unter Übersicht über Resolver-Mapping-Vorlagen.

Beispiel

Das folgende Beispiel ist eine Zuordnungsvorlage für eine GraphQL-Abfrage`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
}
```

Weitere Informationen zur DynamoDB `GetItem`-API finden Sie in der DynamoDB API-Dokumentation.

PutItem

Mit dem `PutItem` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `PutItem` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Der Schlüssel des Elements in DynamoDB

- Der vollständige Inhalt des Elements (bestehend aus `key` und `attributeValues`)
- Bedingungen, damit die Operation erfolgreich ausgeführt werden kann

Das `PutItem`-Zuweisungsdokument weist die folgende Struktur auf:

```
{
    "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
}
```

Die Felder sind wie folgt definiert:

PutItem Felder

PutItem Liste der Felder

`version`

Die Version der Vorlagedefinition. Aktuell werden `2017-02-28` und `2018-05-29` unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `PutItem`-DynamoDB-Operation durchzuführen, muss diese auf `PutItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der Schlüssel des Elements in DynamoDB. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen

Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`attributeValues`

Der Rest der Attribute des Elements, die in DynamoDB gespeichert werden sollen. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dies ist ein optionales Feld.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, überschreibt die `PutItem`-Anforderung alle vorhandenen Einträge für dieses Element. Weitere Informationen zu Bedingungen finden Sie unter Bedingungsausdrücke. Dieser Wert ist optional.

`_version`

Ein numerischer Wert, der die neueste bekannte Version eines Elements darstellt. Dieser Wert ist optional. Dieses Feld wird für die Konflikterkennung verwendet und nur für versionsgesteuerte Datenquellen unterstützt.

`customPartitionKey`

Wenn diese Option aktiviert ist, ändert dieser Zeichenfolgenwert das Format der `ds_sk` und `ds_pk` -Datensätze, die von der Delta-Synchronisierungstabelle verwendet werden, wenn die Versionierung aktiviert wurde (weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch). Wenn diese Option aktiviert ist, ist auch die Verarbeitung des `populateIndexFields` Eintrags aktiviert. Dies ist ein optionales Feld.

`populateIndexFields`

Ein boolescher Wert, der, wenn er zusammen mit dem aktiviert wird`customPartitionKey`, neue Einträge für jeden Datensatz in der Delta-Synchronisierungstabelle erstellt, insbesondere in den Spalten `gsi_ds_pk` und`gsi_ds_sk`. Weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch. Dies ist ein optionales Feld.

Das in DynamoDB geschriebene Element wird automatisch in primitive GraphQL- und JSON-Typen konvertiert und ist im Mapping-Kontext () verfügbar. `$context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu Vorlagen für die Antwortzuweisung finden Sie unter Übersicht über Resolver-Mapping-Vorlagen.

Beispiel 1

Das folgende Beispiel ist eine Mapping-Vorlage für eine GraphQL-Mutation`updateThing(foo: String!, bar: String!, name: String!, version: Int!)`.

Wenn kein Element mit dem angegebenen Schlüssel vorhanden ist, wird es erstellt. Wenn ein Element mit dem angegebenen Schlüssel bereits vorhanden ist, wird es überschrieben.

```
{
    "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)
    }
}
```

Beispiel 2

Das folgende Beispiel ist eine Mapping-Vorlage für eine GraphQL-Mutation`updateThing(foo: String!, bar: String!, name: String!, expectedVersion: Int!)`.

In diesem Beispiel wird überprüft, ob für das Element, das sich derzeit in DynamoDB befindet, das `version` Feld auf gesetzt ist. `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)
        }
    }
}
```

Weitere Informationen zur DynamoDB `PutItem`-API finden Sie in der DynamoDB API-Dokumentation.

UpdateItem

Mit dem `UpdateItem` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `UpdateItem` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Der Schlüssel des Elements in DynamoDB
- Ein Aktualisierungsausdruck, der beschreibt, wie das Element in DynamoDB aktualisiert wird
- Bedingungen, damit die Operation erfolgreich ausgeführt werden kann

Das `UpdateItem`-Zuweisungsdokument weist die folgende Struktur auf:

```
{
    "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
}
```

Die Felder sind wie folgt definiert:

UpdateItem Felder

UpdateItem Liste der Felder

`version`

Die Version der Vorlagedefinition. Aktuell werden `2017-02-28` und `2018-05-29` unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `UpdateItem`-DynamoDB-Operation durchzuführen, muss diese auf `UpdateItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der Schlüssel des Elements in DynamoDB. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Hinweise zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`update`

`update`In diesem Abschnitt können Sie einen Aktualisierungsausdruck angeben, der beschreibt, wie das Element in DynamoDB aktualisiert wird. Weitere Informationen zum Schreiben von Aktualisierungsausdrücken finden Sie in der DynamoDB-Dokumentation UpdateExpressions . Dieser Abschnitt ist erforderlich.

Der `update`-Abschnitt enthält drei Komponenten:

`expression`

Den Aktualisierungsausdruck. Dieser Wert ist erforderlich.

`expressionNames`

Die Ersetzungen für Platzhalter der Namen von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter`expression`, der in der verwendet wird, und der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Namen von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

`expressionValues`

Die Ersetzungen für Platzhalter der Werte von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Wertplatzhalter, der im `expression` verwendet wird, und der Wert muss ein typisierter Wert sein. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser muss angegeben werden. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Werte von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, aktualisiert die `UpdateItem`-Anforderung jeden vorhandenen Eintrag unabhängig vom aktuellen Status. Weitere Informationen zu Bedingungen finden Sie unter Bedingungsausdrücke. Dieser Wert ist optional.

`_version`

Ein numerischer Wert, der die neueste bekannte Version eines Elements darstellt. Dieser Wert ist optional. Dieses Feld wird für die Konflikterkennung verwendet und nur für versionsgesteuerte Datenquellen unterstützt.

`customPartitionKey`

Wenn diese Option aktiviert ist, ändert dieser Zeichenfolgenwert das Format der `ds_sk` und `ds_pk` -Datensätze, die von der Delta-Synchronisierungstabelle verwendet werden, wenn die Versionierung aktiviert wurde (weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch). Wenn diese Option aktiviert ist, ist auch die Verarbeitung des `populateIndexFields` Eintrags aktiviert. Dies ist ein optionales Feld.

`populateIndexFields`

Ein boolescher Wert, der, wenn er zusammen mit dem aktiviert wird`customPartitionKey`, neue Einträge für jeden Datensatz in der Delta-Synchronisierungstabelle erstellt, insbesondere in den Spalten `gsi_ds_pk` und`gsi_ds_sk`. Weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch. Dies ist ein optionales Feld.

Das in DynamoDB aktualisierte Element wird automatisch in primitive GraphQL- und JSON-Typen konvertiert und ist im Mapping-Kontext () verfügbar. `$context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu Vorlagen für die Antwortzuweisung finden Sie unter Übersicht über Resolver-Mapping-Vorlagen.

Beispiel 1

Das folgende Beispiel ist eine Mapping-Vorlage für die GraphQL-Mutation`upvote(id: ID!)`.

In diesem Beispiel werden die `version` Felder `upvotes` und eines Elements in DynamoDB um 1 erhöht.

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

Beispiel 2

Das folgende Beispiel ist eine Mapping-Vorlage für eine GraphQL-Mutation`updateItem(id: ID!, title: String, author: String, expectedVersion: Int!)`.

Hierbei handelt es sich um ein komplexes Beispiel, das die Argumente prüft und den Aktualisierungsausdruck dynamisch generiert, der nur die Argumente enthält, die vom Client bereitgestellt wurden. Beispiel: Wenn `title` und `author` nicht angegeben werden, werden sie nicht aktualisiert. Wenn ein Argument angegeben ist, sein Wert jedoch angegeben ist`null`, wird dieses Feld aus dem Objekt in DynamoDB gelöscht. Schließlich hat der Vorgang eine Bedingung, die überprüft, ob für das Element, das sich derzeit in DynamoDB befindet, das `version` Feld wie folgt gesetzt ist: `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)
        }
    }
}
```

Weitere Informationen zur DynamoDB `UpdateItem`-API finden Sie in der DynamoDB API-Dokumentation.

DeleteItem

Mit dem `DeleteItem` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `DeleteItem` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Der Schlüssel des Elements in DynamoDB
- Bedingungen, damit die Operation erfolgreich ausgeführt werden kann

Das `DeleteItem`-Zuweisungsdokument weist die folgende Struktur auf:

```
{
    "version" : "2018-05-29",
    "operation" : "DeleteItem",
    "customPartitionKey" : "foo",
    "populateIndexFields" : boolean value,
    "key": {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "condition" : {
        ...
    },
    "_version" : 1
}
```

Die Felder sind wie folgt definiert:

DeleteItem Felder

DeleteItem Liste der Felder

`version`

Die Version der Vorlagedefinition. Aktuell werden `2017-02-28` und `2018-05-29` unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `DeleteItem`-DynamoDB-Operation durchzuführen, muss diese auf `DeleteItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der Schlüssel des Elements in DynamoDB. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Hinweise zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, löscht die `DeleteItem`-Anforderung ein Element unabhängig vom aktuellen

Status. Weitere Informationen zu Bedingungen finden Sie unter Bedingungsausdrücke. Dieser Wert ist optional.

`_version`

Ein numerischer Wert, der die neueste bekannte Version eines Elements darstellt. Dieser Wert ist optional. Dieses Feld wird für die Konflikterkennung verwendet und nur für versionsgesteuerte Datenquellen unterstützt.

`customPartitionKey`

Wenn diese Option aktiviert ist, ändert dieser Zeichenfolgenwert das Format der `ds_sk` und `ds_pk` -Datensätze, die von der Delta-Synchronisierungstabelle verwendet werden, wenn die Versionierung aktiviert wurde (weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch). Wenn diese Option aktiviert ist, ist auch die Verarbeitung des `populateIndexFields` Eintrags aktiviert. Dies ist ein optionales Feld.

`populateIndexFields`

Ein boolescher Wert, der, wenn er zusammen mit dem aktiviert wird`customPartitionKey`, neue Einträge für jeden Datensatz in der Delta-Synchronisierungstabelle erstellt, insbesondere in den Spalten `gsi_ds_pk` und`gsi_ds_sk`. Weitere Informationen finden Sie unter Konflikterkennung und -synchronisierung im AWS AppSync Entwicklerhandbuch. Dies ist ein optionales Feld.

Das aus DynamoDB gelöschte Element wird automatisch in primitive GraphQL- und JSON-Typen konvertiert und ist im Mapping-Kontext () verfügbar. `$context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu Vorlagen für die Antwortzuweisung finden Sie unter Übersicht über Resolver-Mapping-Vorlagen.

Beispiel 1

Das folgende Beispiel ist eine Mapping-Vorlage für eine GraphQL-Mutation`deleteItem(id: ID!)`. Wenn ein Element mit dieser ID vorhanden ist, wird es gelöscht.

```
{
    "version" : "2017-02-28",
```

```
    "operation" : "DeleteItem",
    "key" : {
        "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    }
}
```

Beispiel 2

Das folgende Beispiel ist eine Mapping-Vorlage für eine GraphQL-Mutation`deleteItem(id: ID!, expectedVersion: Int!)`. Wenn ein Element mit dieser ID vorhanden ist, wird es nur dann gelöscht, wenn das `version`-Feld auf `expectedVersion` gesetzt ist:

```
{
    "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)
        }
    }
}
```

Weitere Informationen zur DynamoDB `DeleteItem`-API finden Sie in der DynamoDB API-Dokumentation.

Abfrage

Mit dem `Query` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `Query` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Schlüsselausdruck
- Welcher Index verwendet werden soll
- Jeder zusätzliche Filter
- Wie viele Elemente zurückgegeben werden sollen
- Ob Consistent-Lesevorgänge verwendet werden sollen

- Abfragerichtung (vorwärts oder rückwärts)
- Paginierungs-Token

Das `Query`-Zuweisungsdokument weist die folgende Struktur auf:

```
{
    "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" : {
        ...
    }
}
```

Die Felder sind wie folgt definiert:

Felder abfragen

Liste der Abfragefelder

`version`

Die Version der Vorlagedefinition. Aktuell werden `2017-02-28` und `2018-05-29` unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `Query`-DynamoDB-Operation durchzuführen, muss diese auf `Query` gesetzt sein. Dieser Wert ist erforderlich.

`query`

`query`In diesem Abschnitt können Sie einen Schlüsselbedingungsausdruck angeben, der beschreibt, welche Elemente aus DynamoDB abgerufen werden sollen. Weitere Informationen zum Schreiben von Ausdrücken für Schlüsselbedingungen finden Sie in der DynamoDB-Dokumentation KeyConditions . Dieser Abschnitt muss angegeben werden.

`expression`

Der Abfrageausdruck. Dieses Feld muss angegeben werden.

`expressionNames`

Die Ersetzungen für Platzhalter der Namen von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter`expression`, der in der verwendet wird, und der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Namen von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

`expressionValues`

Die Ersetzungen für Platzhalter der Werte von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Wertplatzhalter, der im `expression` verwendet wird, und der Wert muss ein typisierter Wert sein. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Werte von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

`filter`

Ein zusätzlicher Filter, der für das Filtern der Ergebnisse von DynamoDB verwendet werden kann, bevor sie zurückgegeben werden. Weitere Informationen zu Filtern finden Sie unter Filter. Dies ist ein optionales Feld.

`index`

Der Name des abzufragenden Indexes. Der DynamoDB-Abfragevorgang ermöglicht es Ihnen, zusätzlich zum Primärschlüsselindex auch lokale sekundäre Indizes und globale sekundäre

Indizes nach einem Hashschlüssel zu durchsuchen. Falls angegeben, weist dies DynamoDB an, den angegebenen Index abzufragen. Wenn nicht angegeben, wird der Primärschlüsselindex abgefragt.

`nextToken`

Das Paginierungs-Token für die Fortsetzung einer früheren Abfrage. Dieses wäre von einer vorherigen Abfrage erhalten worden. Dies ist ein optionales Feld.

`limit`

Die maximale Anzahl der auszuwertenden Elemente (nicht notwendigerweise die Anzahl der übereinstimmenden Elemente). Dies ist ein optionales Feld.

`scanIndexForward`

Ein boolescher Wert, der angibt, ob vorwärts oder rückwärts abgefragt werden soll. Dieses Feld ist optional und standardmäßig auf `true` gesetzt.

`consistentRead`

Ein boolescher Wert, der angibt, ob bei der Abfrage von DynamoDB konsistente Lesevorgänge verwendet werden sollen. Dieses Feld ist optional und standardmäßig auf `false` gesetzt.

`select`

Standardmäßig gibt der AWS AppSync DynamoDB-Resolver nur Attribute zurück, die in den Index projiziert werden. Wenn weitere Attribute erforderlich sind, kann dieses Feld festgelegt werden. Dies ist ein optionales Feld. Die unterstützten Werte sind:

`ALL_ATTRIBUTES`

Gibt alle Elementattribute aus der angegebenen Tabelle oder dem Index zurück. Wenn Sie einen lokalen sekundären Index abfragen, ruft DynamoDB für jedes übereinstimmende Element im Index das gesamte Element aus der übergeordneten Tabelle ab. Wenn der Index konfiguriert ist, um alle Elementattribute zu projizieren, können Sie alle Daten aus dem lokalen sekundären Index erhalten und das Abrufen ist nicht erforderlich.

`ALL_PROJECTED_ATTRIBUTES`

Nur bei der Abfrage eines Indexes erlaubt. Ruft alle Attribute ab, die in den Index projiziert wurden. Wenn der Index konfiguriert ist, um alle Attribute zu projizieren, entspricht dieser Rückgabewert der Angabe von `ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Gibt nur die in den s aufgeführten Attribute zurück. `projection` `expression` Dieser Rückgabewert entspricht der Angabe `projection` von 's' `expression` ohne Angabe eines Werts für`Select`.

`projection`

Eine Projektion, die verwendet wird, um die Attribute anzugeben, die von der DynamoDB-Operation zurückgegeben werden sollen. Weitere Informationen zu Projektionen finden Sie unter Projektionen. Dies ist ein optionales Feld.

Die Ergebnisse von DynamoDB werden automatisch in primitive GraphQL- und JSON-Typen konvertiert und sind im Mapping-Kontext () verfügbar. `$context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu Vorlagen für die Antwortzuweisung finden Sie unter Übersicht über Resolver-Mapping-Vorlagen.

Die Ergebnisse weisen die folgende Struktur auf:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Die Felder sind wie folgt definiert:

`items`

Eine Liste mit den von der DynamoDB-Abfrage zurückgegebenen Elementen.

`nextToken`

Wenn es mehr Ergebnisse geben könnte, enthält `nextToken` ein Paginierungs-Token, das in einer anderen Anfrage verwendet werden kann. Beachten Sie, dass das von DynamoDB zurückgegebene Paginierungstoken AWS AppSync verschlüsselt und verschleiert wird. Damit sickern Daten aus Ihren Tabellen nicht versehentlich an den Aufrufer durch. Beachten Sie auch, dass diese Paginierungs-Token nicht über verschiedene Resolver verwendet werden können.

`scannedCount`

Die Anzahl der Elemente, die mit dem Abfragebedingungsausdruck übereinstimmten, bevor ein Filterausdruck (falls vorhanden) angewendet wurde.

Beispiel

Das folgende Beispiel ist eine Zuordnungsvorlage für eine GraphQL-Abfrage`getPosts(owner: ID!)`.

In diesem Beispiel wird ein globaler sekundärer Index in einer Tabelle abgefragt, um alle Beiträge im Besitz der angegebenen ID zurückzugeben.

```
{
    "version" : "2017-02-28",
    "operation" : "Query",
    "query" : {
        "expression" : "ownerId = :ownerId",
        "expressionValues" : {
            ":ownerId" : $util.dynamodb.toDynamoDBJson($context.arguments.owner)
        }
    },
    "index" : "owner-index"
}
```

Weitere Informationen zur DynamoDB `Query`-API finden Sie in der DynamoDB API-Dokumentation.

Scan

Mit dem `Scan` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `Scan` Anfrage an DynamoDB zu stellen, und Sie können Folgendes angeben:

- Ein Filter zum Ausschließen von Ergebnissen
- Welcher Index verwendet werden soll
- Wie viele Elemente zurückgegeben werden sollen
- Ob Consistent-Lesevorgänge verwendet werden sollen
- Paginierungs-Token
- Parallele Scans

Das `Scan`-Zuweisungsdokument weist die folgende Struktur auf:

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "index" : "fooIndex",
    "limit" : 10,
    "consistentRead" : false,
    "nextToken" : "aPaginationToken",
    "totalSegments" : 10,
    "segment" : 1,
    "filter" : {
        ...
    },
    "projection" : {
        ...
    }
}
```

Die Felder sind wie folgt definiert:

Felder scannen

Liste der Felder scannen

`version`

Die Version der Vorlagedefinition. Aktuell werden `2017-02-28` und `2018-05-29` unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `Scan`-DynamoDB-Operation durchzuführen, muss diese auf `Scan` gesetzt sein. Dieser Wert ist erforderlich.

`filter`

Ein Filter, mit dem die Ergebnisse aus DynamoDB gefiltert werden können, bevor sie zurückgegeben werden. Weitere Informationen zu Filtern finden Sie unter Filter. Dies ist ein optionales Feld.

`index`

Der Name des abzufragenden Indexes. Der DynamoDB-Abfragevorgang ermöglicht es Ihnen, zusätzlich zum Primärschlüsselindex auch lokale sekundäre Indizes und globale sekundäre

Indizes nach einem Hashschlüssel zu durchsuchen. Falls angegeben, weist dies DynamoDB an, den angegebenen Index abzufragen. Wenn nicht angegeben, wird der Primärschlüsselindex abgefragt.

`limit`

Die maximale Anzahl der zu einem bestimmten Zeitpunkt auszuwertenden Elemente. Dies ist ein optionales Feld.

`consistentRead`

Ein boolescher Wert, der angibt, ob bei der Abfrage von DynamoDB konsistente Lesevorgänge verwendet werden sollen. Dieses Feld ist optional und standardmäßig auf `false` gesetzt.

`nextToken`

Das Paginierungs-Token für die Fortsetzung einer früheren Abfrage. Dieses wäre von einer vorherigen Abfrage erhalten worden. Dies ist ein optionales Feld.

`select`

Standardmäßig gibt der AWS AppSync DynamoDB-Resolver nur die Attribute zurück, die in den Index projiziert wurden. Wenn weitere Attribute erforderlich sind, kann dieses Feld festgelegt werden. Dies ist ein optionales Feld. Die unterstützten Werte sind:

`ALL_ATTRIBUTES`

Gibt alle Elementattribute aus der angegebenen Tabelle oder dem Index zurück. Wenn Sie einen lokalen sekundären Index abfragen, ruft DynamoDB für jedes übereinstimmende Element im Index das gesamte Element aus der übergeordneten Tabelle ab. Wenn der Index konfiguriert ist, um alle Elementattribute zu projizieren, können Sie alle Daten aus dem lokalen sekundären Index erhalten und das Abrufen ist nicht erforderlich.

`ALL_PROJECTED_ATTRIBUTES`

Nur bei der Abfrage eines Indexes erlaubt. Ruft alle Attribute ab, die in den Index projiziert wurden. Wenn der Index konfiguriert ist, um alle Attribute zu projizieren, entspricht dieser Rückgabewert der Angabe von `ALL_ATTRIBUTES`.

`SPECIFIC_ATTRIBUTES`

Gibt nur die in den s aufgeführten Attribute zurück. `projection` `expression` Dieser Rückgabewert entspricht der Angabe `projection` von 's' `expression` ohne Angabe eines Werts für`Select`.

`totalSegments`

Die Anzahl der Segmente zum Partitionieren der Tabelle, wenn ein paralleler Scan durchgeführt wird. Dieses Feld ist optional, muss aber angegeben werden, wenn `segment` angegeben ist.

`segment`

Das Tabellensegment in dieser Operation beim Durchführen eines parallelen Scans. Dieses Feld ist optional, muss aber angegeben werden, wenn `totalSegments` angegeben ist.

`projection`

Eine Projektion, die verwendet wird, um die Attribute anzugeben, die von der DynamoDB-Operation zurückgegeben werden sollen. Weitere Informationen zu Projektionen finden Sie unter Projektionen. Dies ist ein optionales Feld.

Die vom DynamoDB-Scan zurückgegebenen Ergebnisse werden automatisch in primitive GraphQL- und JSON-Typen konvertiert und sind im Mapping-Kontext () verfügbar. `$context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu Vorlagen für die Antwortzuweisung finden Sie unter Übersicht über Resolver-Mapping-Vorlagen.

Die Ergebnisse weisen die folgende Struktur auf:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10
}
```

Die Felder sind wie folgt definiert:

`items`

Eine Liste mit den vom DynamoDB-Scan zurückgegebenen Elementen.

`nextToken`

Falls es mehr Ergebnisse gibt, `nextToken` enthält sie ein Paginierungstoken, das Sie in einer anderen Anfrage verwenden können. AWS AppSync verschlüsselt und verschleiert das von

DynamoDB zurückgegebene Paginierungstoken. Damit sickern Daten aus Ihren Tabellen nicht versehentlich an den Aufrufer durch. Diese Paginierungs-Tokens können nicht über verschiedene Resolver hinweg verwendet werden.

`scannedCount`

Die Anzahl der Elemente, die von DynamoDB abgerufen wurden, bevor ein Filterausdruck (falls vorhanden) angewendet wurde.

Beispiel 1

Das folgende Beispiel ist eine Zuordnungsvorlage für die GraphQL-Abfrage:`allPosts`.

In diesem Beispiel werden alle Einträge in der Tabelle zurückgegeben.

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

Beispiel 2

Das folgende Beispiel ist eine Zuordnungsvorlage für die GraphQL-Abfrage:`postsMatching(title: String!)`.

In diesem Beispiel werden alle Einträge in der Tabelle zurückgegeben, bei denen der Titel mit dem `title`-Argument beginnt.

```
{
    "version" : "2017-02-28",
    "operation" : "Scan",
    "filter" : {
        "expression" : "begins_with(title, :title)",
        "expressionValues" : {
            ":title" : $util.dynamodb.toDynamoDBJson($context.arguments.title)
        },
    }
}
```

Weitere Informationen zur DynamoDB `Scan`-API finden Sie in der DynamoDB API-Dokumentation.

Synchronisierung

Mit dem `Sync` Anforderungszuordnungsdokument können Sie alle Ergebnisse aus einer DynamoDB-Tabelle abrufen und anschließend nur die Daten empfangen, die seit Ihrer letzten Abfrage geändert wurden (die Delta-Updates). `Sync`Anfragen können nur an versionierte DynamoDB-Datenquellen gestellt werden. Sie können folgende Formen angeben:

- Ein Filter zum Ausschließen von Ergebnissen
- Wie viele Elemente zurückgegeben werden sollen
- Paginierungstoken
- Wann Ihre letzte `Sync`-Operation gestartet wurde

Das `Sync`-Zuweisungsdokument weist die folgende Struktur auf:

```
{
    "version" : "2018-05-29",
    "operation" : "Sync",
    "basePartitionKey": "Base Tables PartitionKey",
    "deltaIndexName": "delta-index-name",
    "limit" : 10,
    "nextToken" : "aPaginationToken",
    "lastSync" :  1550000000000,
    "filter" : {
        ...
    }
}
```

Die Felder sind wie folgt definiert:

Synchronisieren Sie Felder

Liste der Felder synchronisieren

`version`

Die Version der Vorlagendefinition. Derzeit wird nur `2018-05-29` unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `Sync`--Operation durchzuführen, muss diese auf `Sync` gesetzt sein. Dieser Wert ist erforderlich.

`filter`

Ein Filter, mit dem die Ergebnisse aus DynamoDB gefiltert werden können, bevor sie zurückgegeben werden. Weitere Informationen zu Filtern finden Sie unter Filter. Dies ist ein optionales Feld.

`limit`

Die maximale Anzahl der zu einem bestimmten Zeitpunkt auszuwertenden Elemente. Dies ist ein optionales Feld. Wenn nicht angegeben, wird das Standardlimit auf `100` Elemente festgelegt. Der maximale Wert für dieses Feld ist `1000` Elemente.

`nextToken`

Das Paginierungs-Token für die Fortsetzung einer früheren Abfrage. Dieses wäre von einer vorherigen Abfrage erhalten worden. Dies ist ein optionales Feld.

`lastSync`

Der Moment, in Epochenmillisekunden, an dem die letzte erfolgreiche `Sync`-Operation gestartet wurde. Wenn angegeben, werden nur Elemente zurückgegeben, die sich nach `lastSync` geändert haben. Dieses Feld ist optional und sollte nur ausgefüllt werden, nachdem alle Seiten von einem ersten `Sync`-Vorgang abgerufen wurden. Wenn nicht angegeben, werden Ergebnisse aus der Basis-Tabelle zurückgegeben, andernfalls werden Ergebnisse aus der Delta-Tabelle zurückgegeben.

`basePartitionKey`

Der Partitionsschlüssel der Basistabelle, der bei der Ausführung eines `Sync` Vorgangs verwendet wird. In diesem Feld kann eine `Sync` Operation ausgeführt werden, wenn die Tabelle einen benutzerdefinierten Partitionsschlüssel verwendet. Dies ist ein optionales Feld.

`deltaIndexName`

Der für die `Sync` Operation verwendete Index. Dieser Index ist erforderlich, um einen `Sync` Vorgang für die gesamte Delta-Store-Tabelle zu aktivieren, wenn die Tabelle einen benutzerdefinierten Partitionsschlüssel verwendet. Die `Sync` Operation wird auf der GSI (erstellt am `gsi_ds_pk` und`gsi_ds_sk`) ausgeführt. Dies ist ein optionales Feld.

Die von der DynamoDB-Synchronisierung zurückgegebenen Ergebnisse werden automatisch in primitive GraphQL- und JSON-Typen konvertiert und sind im Mapping-Kontext () verfügbar. `$context.result`

Weitere Informationen zur DynamoDB-Typkonvertierung finden Sie unter Typsystem (Antwortzuordnung).

Weitere Informationen zu Vorlagen für die Antwortzuweisung finden Sie unter Übersicht über Resolver-Mapping-Vorlagen.

Die Ergebnisse weisen die folgende Struktur auf:

```
{
    items = [ ... ],
    nextToken = "a pagination token",
    scannedCount = 10,
    startedAt = 1550000000000
}
```

Die Felder sind wie folgt definiert:

`items`

Eine Liste mit den Elementen, die von der Synchronisierung zurückgegeben wurden.

`nextToken`

Falls es möglicherweise mehr Ergebnisse gibt, `nextToken` enthält es ein Paginierungstoken, das Sie in einer anderen Anfrage verwenden können. AWS AppSync verschlüsselt und verschleiert das von DynamoDB zurückgegebene Paginierungstoken. Damit sickern Daten aus Ihren Tabellen nicht versehentlich an den Aufrufer durch. Diese Paginierungs-Tokens können nicht über verschiedene Resolver hinweg verwendet werden.

`scannedCount`

Die Anzahl der Elemente, die von DynamoDB abgerufen wurden, bevor ein Filterausdruck (falls vorhanden) angewendet wurde.

`startedAt`

Der Moment, in Epochenmillisekunden, an dem der Synchronisierungsvorgang gestartet wurde, den Sie lokal speichern und in einer anderen Anforderung als `lastSync`-Argument verwenden können. Wenn ein Paginierungstoken in der Anforderung enthalten war, entspricht dieser Wert dem Wert, der von der Anforderung für die erste Ergebnisseite zurückgegeben wird.

Beispiel

Das folgende Beispiel ist eine Zuordnungsvorlage für die GraphQL-Abfrage:`syncPosts(nextToken: String, lastSync: AWSTimestamp)`.

Wenn in diesem Beispiel `lastSync` weggelassen wird, werden alle Einträge in der Basistabelle zurückgegeben. Wenn `lastSync` angegeben wird, werden nur die Einträge in der Delta-Synchronisationstabelle zurückgegeben, die sich seit `lastSync` geändert haben.

```
{
    "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

Mit dem `BatchGetItem` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `BatchGetItem` Anforderung an DynamoDB zu stellen, um mehrere Elemente abzurufen, möglicherweise über mehrere Tabellen hinweg. Für diese Anforderungsvorlage müssen Sie Folgendes angeben:

- Die Namen der Tabellen, aus denen die Elemente abgerufen werden sollen.
- Die Schlüssel der Elemente, die aus den einzelnen Tabellen abgerufen werden sollen.

Es gelten die DynamoDB `BatchGetItem`-Grenzwerte, und es kann kein Bedingungsausdruck bereitgestellt werden.

Das `BatchGetItem`-Zuweisungsdokument weist die folgende Struktur auf:

```
{
    "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" : {
                ...
           }
       }
   }
}
```

Die Felder sind wie folgt definiert:

BatchGetItem Felder

BatchGetItem Liste der Felder

`version`

Die Version der Vorlagendefinition. Nur `2018-05-29` wird unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `BatchGetItem`-DynamoDB-Operation durchzuführen, muss diese auf `BatchGetItem` gesetzt sein. Dieser Wert ist erforderlich.

`tables`

Die DynamoDB-Tabellen, aus denen die Elemente abgerufen werden sollen. Der Wert ist eine Zuweisung, in der Tabellennamen als Schlüssel der Zuweisung angegeben werden. Mindestens eine Tabelle muss angegeben werden. Dieser `tables`-Wert ist erforderlich.

`keys`

Liste der DynamoDB-Schlüssel, die den Primärschlüssel der abzurufenden Elemente darstellen. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung).

`consistentRead`

Ob bei der Ausführung eines GetItemVorgangs ein konsistenter Lesevorgang verwendet werden soll. Dieser Wert ist optional und ist standardmäßig false.

`projection`

Eine Projektion, die verwendet wird, um die Attribute anzugeben, die von der DynamoDB-Operation zurückgegeben werden sollen. Weitere Informationen zu Projektionen finden Sie unter Projektionen. Dies ist ein optionales Feld.

Beachten Sie Folgendes:

- Wenn ein Element nicht aus der Tabelle abgerufen wurde, enthält der Datenblock für diese Tabelle ein null-Element.
- Die Aufrufergebnisse werden nach Tabelle sortiert, basierend auf der Reihenfolge, in der sie in der Vorlage für die Anforderungszuweisung bereitgestellt wurden.
- Jeder `Get` Befehl in a `BatchGetItem` ist atomar, ein Batch kann jedoch teilweise verarbeitet werden. Wenn ein Stapel aufgrund eines Fehlers teilweise verarbeitet wird, werden die nicht verarbeiteten Schlüssel als Teil des Aufrufergebnisses im Block unprocessedKeys zurückgegeben.
- `BatchGetItem` ist auf 100 Schlüssel beschränkt.

Für das folgende Beispiel für eine Anforderungszuweisungsvorlage gilt:

```
{
  "version": "2018-05-29",
  "operation": "BatchGetItem",
  "tables": {
    "authors": [
        {
          "author_id": {
            "S": "a1"
          }
        },
    ],
    "posts": [
        {
          "author_id": {
            "S": "a1"
          },
          "post_id": {
            "S": "p2"
          }
        }
    ],
  }
}
```

Das in `$ctx.result` verfügbare Aufrufergebnis sieht wie folgt aus:

```
{
   "data": {
     "authors": [null],
     "posts": [
        # Was retrieved
        {
          "author_id": "a1",
          "post_id": "p2",
          "post_title": "title",
          "post_description": "description",
        }
     ]
   },
   "unprocessedKeys": {
     "authors": [
```

```
        # This item was not processed due to an error
        {
          "author_id": "a1"
        }
      ],
     "posts": []
   }
}
```

`$ctx.error` enthält die Einzelheiten zu dem Fehler. Die Schlüssel data, unprocessedKeys sowie die einzelnen Tabellenschlüssel, die in der Zuweisungsvorlage für die Anforderung bereitgestellt wurden, sind garantiert im Aufrufergebnis vorhanden. Elemente, die gelöscht wurden, befinden sich im Block data. Elemente, die nicht verarbeitet wurden, werden im Datenblock mit null markiert und im Block unprocessedKeys platziert.

Ein vollständigeres Beispiel AppSync finden Sie im DynamoDB Batch-Tutorial mit diesem Tutorial: DynamoDB-Batch-Resolver.

BatchDeleteItem

Mit dem `BatchDeleteItem` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `BatchWriteItem` Anforderung an DynamoDB zu stellen, um mehrere Elemente zu löschen, möglicherweise über mehrere Tabellen hinweg. Für diese Anforderungsvorlage müssen Sie Folgendes angeben:

- Die Namen der Tabellen, aus denen die Elemente gelöscht werden sollen.
- Die Schlüssel der Elemente, die aus den einzelnen Tabellen gelöscht werden sollen.

Es gelten die DynamoDB `BatchWriteItem`-Grenzwerte, und es kann kein Bedingungsausdruck bereitgestellt werden.

Das `BatchDeleteItem`-Zuweisungsdokument weist die folgende Struktur auf:

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

Die Felder sind wie folgt definiert:

BatchDeleteItem Felder

BatchDeleteItem Liste der Felder

`version`

Die Version der Vorlagendefinition. Nur `2018-05-29` wird unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `BatchDeleteItem`-DynamoDB-Operation durchzuführen, muss diese auf `BatchDeleteItem` gesetzt sein. Dieser Wert ist erforderlich.

`tables`

Die DynamoDB-Tabellen, aus denen die Elemente gelöscht werden sollen. Jede Tabelle ist eine Liste von DynamoDB-Schlüsseln, die den Primärschlüssel der zu löschenden Elemente darstellen. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Mindestens eine Tabelle muss angegeben werden. Der `tables` Wert ist erforderlich.

Beachten Sie Folgendes:

- Im Gegensatz zum `DeleteItem`-Vorgang wird nicht das vollständig gelöschte Element in der Antwort zurückgegeben. Nur der übergebene Schlüssel wird zurückgegeben.
- Wenn ein Element nicht aus der Tabelle gelöscht wurde, enthält der Datenblock für diese Tabelle ein null-Element.
- Die Aufrufergebnisse werden nach Tabelle sortiert, basierend auf der Reihenfolge, in der sie in der Vorlage für die Anforderungszuweisung bereitgestellt wurden.
- Jeder `Delete` Befehl in a `BatchDeleteItem` ist atomar. Ein Stapel kann jedoch teilweise verarbeitet werden. Wenn ein Stapel aufgrund eines Fehlers teilweise verarbeitet wird, werden die nicht verarbeiteten Schlüssel als Teil des Aufrufergebnisses im Block unprocessedKeys zurückgegeben.
- `BatchDeleteItem` ist auf 25 Schlüssel beschränkt.
- Dieser Vorgang wird nicht unterstützt, wenn er zusammen mit der Konflikterkennung verwendet wird. Wenn Sie beide gleichzeitig verwenden, kann dies zu einem Fehler führen.

Für das folgende Beispiel für eine Anforderungszuweisungsvorlage gilt:

```
{
  "version": "2018-05-29",
  "operation": "BatchDeleteItem",
  "tables": {
    "authors": [
        {
          "author_id": {
            "S": "a1"
          }
        },
    ],
    "posts": [
        {
          "author_id": {
            "S": "a1"
          },
          "post_id": {
            "S": "p2"
          }
        }
    ],
```

```
  }
}
```

Das in `$ctx.result` verfügbare Aufrufergebnis sieht wie folgt aus:

```
{
   "data": {
     "authors": [null],
     "posts": [
        # Was deleted
        {
          "author_id": "a1",
          "post_id": "p2"
        }
     ]
   },
   "unprocessedKeys": {
     "authors": [
        # This key was not processed due to an error
        {
          "author_id": "a1"
        }
      ],
     "posts": []
   }
}
```

`$ctx.error` enthält die Einzelheiten zu dem Fehler. Die Schlüssel data, unprocessedKeys sowie die einzelnen Tabellenschlüssel, die in der Zuweisungsvorlage für die Anforderung bereitgestellt wurden, sind garantiert im Aufrufergebnis vorhanden. Elemente, die gelöscht wurden, befinden sich im Block data. Elemente, die nicht verarbeitet wurden, werden im Datenblock mit null markiert und im Block unprocessedKeys platziert.

Ein vollständigeres Beispiel AppSync finden Sie im DynamoDB Batch-Tutorial mit diesem Tutorial: DynamoDB-Batch-Resolver.

BatchPutItem

Mit dem `BatchPutItem` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `BatchWriteItem` Anforderung an DynamoDB zu

stellen, um mehrere Elemente zu platzieren, möglicherweise in mehreren Tabellen. Für diese Anforderungsvorlage müssen Sie Folgendes angeben:

- Die Namen der Tabellen, in denen die Elemente abgelegt werden sollen.
- Die vollständigen Elemente, die in jeder Tabelle abgelegt werden sollen

Es gelten die DynamoDB `BatchWriteItem`-Grenzwerte, und es kann kein Bedingungsausdruck bereitgestellt werden.

Das `BatchPutItem`-Zuweisungsdokument weist die folgende Struktur auf:

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

Die Felder sind wie folgt definiert:

BatchPutItem Felder

BatchPutItem Liste der Felder

`version`

Die Version der Vorlagendefinition. Nur `2018-05-29` wird unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `BatchPutItem`-DynamoDB-Operation durchzuführen, muss diese auf `BatchPutItem` gesetzt sein. Dieser Wert ist erforderlich.

`tables`

Die DynamoDB-Tabellen, in die die Elemente eingefügt werden sollen. Jeder Tabelleneintrag stellt eine Liste von DynamoDB-Elementen dar, die für diese spezielle Tabelle eingefügt werden sollen. Mindestens eine Tabelle muss angegeben werden. Dieser Wert ist erforderlich.

Beachten Sie Folgendes:

- Bei Erfolg werden die vollständig eingefügten Elemente in der Antwort zurückgegeben.
- Wenn ein Element nicht in die Tabelle eingefügt wurde, wird im Datenblock für diese Tabelle ein null-Element angezeigt.
- Die eingefügten Elemente werden nach Tabelle sortiert, basierend auf der Reihenfolge, in der sie in der Anforderungszuordnungsvorlage bereitgestellt wurden.
- Jeder `Put` Befehl in a `BatchPutItem` ist atomar, ein Batch kann jedoch teilweise verarbeitet werden. Wenn ein Stapel aufgrund eines Fehlers teilweise verarbeitet wird, werden die nicht verarbeiteten Schlüssel als Teil des Aufrufergebnisses im Block unprocessedKeys zurückgegeben.
- `BatchPutItem` ist auf 25 Elemente beschränkt.
- Dieser Vorgang wird nicht unterstützt, wenn er zusammen mit der Konflikterkennung verwendet wird. Wenn Sie beide gleichzeitig verwenden, kann dies zu einem Fehler führen.

Für das folgende Beispiel für eine Anforderungszuweisungsvorlage gilt:

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

Das in `$ctx.result` verfügbare Aufrufergebnis sieht wie folgt aus:

```
{
   "data": {
     "authors": [
         null
     ],
     "posts": [
        # Was inserted
        {
          "author_id": "a1",
          "post_id": "p2",
          "post_title": "title"
        }
     ]
   },
   "unprocessedItems": {
```

```
     "authors": [
        # This item was not processed due to an error
        {
          "author_id": "a1",
          "author_name": "a1_name"
        }
      ],
     "posts": []
   }
}
```

`$ctx.error` enthält die Einzelheiten zu dem Fehler. Die Schlüssel data, unprocessedItems sowie die einzelnen Tabellenschlüssel, die in der Zuweisungsvorlage für die Anforderung bereitgestellt wurden, sind garantiert im Aufrufergebnis vorhanden. Elemente, die eingefügt wurden, befinden sich im Block data. Elemente, die nicht verarbeitet wurden, werden im Datenblock mit null markiert und im Block unprocessedItems platziert.

Ein vollständigeres Beispiel AppSync finden Sie im DynamoDB Batch-Tutorial mit diesem Tutorial: DynamoDB-Batch-Resolver.

TransactGetItems

Mit dem `TransactGetItems` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `TransactGetItems` Anforderung an DynamoDB zu stellen, um mehrere Elemente abzurufen, möglicherweise über mehrere Tabellen hinweg. Für diese Anforderungsvorlage müssen Sie Folgendes angeben:

- Der Tabellenname jedes Anforderungselements, von dem das Element abgerufen werden soll
- Der Schlüssel jedes Anforderungselements, das aus jeder Tabelle abgerufen werden soll

Es gelten die DynamoDB `TransactGetItems`-Grenzwerte, und es kann kein Bedingungsausdruck bereitgestellt werden.

Das `TransactGetItems`-Zuweisungsdokument weist die folgende Struktur auf:

```
{
    "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" : {
                 ...
            }
        }
    ]
}
```

Die Felder sind wie folgt definiert:

TransactGetItems Felder

TransactGetItems Liste der Felder

`version`

Die Version der Vorlagendefinition. Nur `2018-05-29` wird unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `TransactGetItems`-DynamoDB-Operation durchzuführen, muss diese auf `TransactGetItems` gesetzt sein. Dieser Wert ist erforderlich.

`transactItems`

Die einzuschließenden Anforderungselemente. Der Wert ist ein Array von Anforderungselementen. Es muss mindestens ein Anforderungselement angegeben werden. Dieser `transactItems`-Wert ist erforderlich.

`table`

Die DynamoDB-Tabelle, aus der das Element abgerufen werden soll. Der Wert ist eine Zeichenfolge des Tabellennamens. Dieser `table`-Wert ist erforderlich.

`key`

Der DynamoDB-Schlüssel, der den Primärschlüssel des abzurufenden Elements darstellt. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung).

`projection`

Eine Projektion, die verwendet wird, um die Attribute anzugeben, die von der DynamoDB-Operation zurückgegeben werden sollen. Weitere Informationen zu Projektionen finden Sie unter Projektionen. Dies ist ein optionales Feld.

Beachten Sie Folgendes:

- Wenn eine Transaktion erfolgreich ist, entspricht die Reihenfolge der abgerufenen Elemente im `items`-Block der Reihenfolge der Anforderungselemente.
- Transaktionen werden in irgendeiner all-or-nothing Weise ausgeführt. Wenn ein Anforderungselement einen Fehler verursacht, wird die gesamte Transaktion nicht ausgeführt, und Fehlerdetails werden zurückgegeben.
- Ein Anforderungselement, das nicht abgerufen werden kann, ist kein Fehler. Stattdessen erscheint ein Null-Element im Elemente-Block an der entsprechenden Position.
- Wenn der Fehler einer Transaktion lautet TransactionCanceledException, wird der `cancellationReasons` Block aufgefüllt. Die Reihenfolge der Stornierungsgründe im `cancellationReasons`-Block ist die gleiche wie die Reihenfolge der Anforderungselemente.
- `TransactGetItems`ist auf 100 Anfrageelemente begrenzt.

Für das folgende Beispiel für eine Anforderungszuweisungsvorlage gilt:

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

Wenn die Transaktion erfolgreich ist und nur das erste angeforderte Element abgerufen wird, ist das Aufrufergebnis in `$ctx.result` wie folgt verfügbar:

```
{
    "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
}
```

Wenn die Transaktion TransactionCanceledExceptionaufgrund des ersten Anforderungselements fehlschlägt, `$ctx.result` ist das Aufrufergebnis wie folgt verfügbar:

```
{
    "items": null,
```

```
    "cancellationReasons": [
       {
           "type":"Sample error type",
           "message":"Sample error message"
       },
       {
           "type":"None",
           "message":"None"
       }
    ]
}
```

`$ctx.error` enthält die Einzelheiten zu dem Fehler. Die Schlüssel-Elemente und Stornierungsgründe sind garantiert in `$ctx.result` vorhanden.

Ein vollständigeres Beispiel AppSync finden Sie im DynamoDB-Transaktions-Tutorial mit diesem Tutorial: DynamoDB-Transaktionsauflösungen.

TransactWriteItems

Mit dem `TransactWriteItems` Anforderungszuordnungsdokument können Sie den AWS AppSync DynamoDB-Resolver anweisen, eine `TransactWriteItems` Anforderung an DynamoDB zu stellen, um mehrere Elemente zu schreiben, möglicherweise in mehrere Tabellen. Für diese Anforderungsvorlage müssen Sie Folgendes angeben:

- Der Name der Zieltabelle jedes Anforderungselements
- Der Vorgang jedes auszuführenden Anforderungselements. Es gibt vier Arten von Operationen, die unterstützt werden:,, und PutItemUpdateItemDeleteItemConditionCheck
- Der Schlüssel jedes zu schreibenden Anforderungselements

Es gelten die DynamoDB `TransactWriteItems`-Grenzwerte.

Das `TransactWriteItems`-Zuweisungsdokument weist die folgende Struktur auf:

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

TransactWriteItems Felder

TransactWriteItems Liste der Felder

Die Felder sind wie folgt definiert:

`version`

Die Version der Vorlagendefinition. Nur `2018-05-29` wird unterstützt. Dieser Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `TransactWriteItems`-DynamoDB-Operation durchzuführen, muss diese auf `TransactWriteItems` gesetzt sein. Dieser Wert ist erforderlich.

`transactItems`

Die einzuschließenden Anforderungselemente. Der Wert ist ein Array von Anforderungselementen. Es muss mindestens ein Anforderungselement angegeben werden. Dieser `transactItems`-Wert ist erforderlich.

Für `PutItem` sind die Felder wie folgt definiert:

`table`

Die DynamoDB-Zieltabelle. Der Wert ist eine Zeichenfolge des Tabellennamens. Dieser `table`-Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `PutItem`-DynamoDB-Operation durchzuführen, muss diese auf `PutItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der DynamoDB-Schlüssel, der den Primärschlüssel des einzufügenden Elements darstellt. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`attributeValues`

Der Rest der Attribute des Elements, die in DynamoDB gespeichert werden sollen. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dies ist ein optionales Feld.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, überschreibt die `PutItem`-Anforderung alle vorhandenen Einträge für dieses Element. Sie können angeben, ob das vorhandene Element zurückgerufen werden soll, wenn die Bedingungsprüfung fehlschlägt. Weitere Informationen zu Transaktionsbedingungen finden Sie unter Ausdrücke für Transaktionsbedingungen. Dieser Wert ist optional.

Für `UpdateItem` sind die Felder wie folgt definiert:

`table`

Die zu aktualisierende DynamoDB-Tabelle. Der Wert ist eine Zeichenfolge des Tabellennamens. Dieser `table`-Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `UpdateItem`-DynamoDB-Operation durchzuführen, muss diese auf `UpdateItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der DynamoDB-Schlüssel, der den Primärschlüssel des zu aktualisierenden Elements darstellt. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`update`

`update`In diesem Abschnitt können Sie einen Aktualisierungsausdruck angeben, der beschreibt, wie das Element in DynamoDB aktualisiert wird. Weitere Informationen zum Schreiben von Aktualisierungsausdrücken finden Sie in der DynamoDB-Dokumentation UpdateExpressions . Dieser Abschnitt ist erforderlich.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, aktualisiert die `UpdateItem`-Anforderung jeden vorhandenen Eintrag unabhängig vom aktuellen Status. Sie können angeben, ob das vorhandene Element zurückgerufen werden soll, wenn die Bedingungsprüfung fehlschlägt. Weitere Informationen zu Transaktionsbedingungen finden Sie unter Ausdrücke für Transaktionsbedingungen. Dieser Wert ist optional.

Für `DeleteItem` sind die Felder wie folgt definiert:

`table`

Die DynamoDB-Tabelle, in der das Element gelöscht werden soll. Der Wert ist eine Zeichenfolge des Tabellennamens. Dieser `table`-Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `DeleteItem`-DynamoDB-Operation durchzuführen, muss diese auf `DeleteItem` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der DynamoDB-Schlüssel, der den Primärschlüssel des zu löschenden Elements darstellt. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Wenn keine Bedingung angegeben ist, löscht die `DeleteItem`-Anforderung ein Element unabhängig vom aktuellen Status. Sie können angeben, ob das vorhandene Element zurückgerufen werden soll, wenn die Bedingungsprüfung fehlschlägt. Weitere Informationen zu Transaktionsbedingungen finden Sie unter Ausdrücke für Transaktionsbedingungen. Dieser Wert ist optional.

Für `ConditionCheck` sind die Felder wie folgt definiert:

`table`

Die DynamoDB-Tabelle, in der der Zustand überprüft werden soll. Der Wert ist eine Zeichenfolge des Tabellennamens. Dieser `table`-Wert ist erforderlich.

`operation`

Der DynamoDB DynamoDB-Vorgang. Um die `ConditionCheck`-DynamoDB-Operation durchzuführen, muss diese auf `ConditionCheck` gesetzt sein. Dieser Wert ist erforderlich.

`key`

Der DynamoDB-Schlüssel, der den Primärschlüssel des Artikels zur Zustandsprüfung darstellt. DynamoDB-Elemente können je nach Tabellenstruktur einen einzelnen Hashschlüssel oder einen Hashschlüssel und einen Sortierschlüssel haben. Weitere Informationen zur Angabe eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuordnung). Dieser Wert ist erforderlich.

`condition`

Eine Bedingung, um zu bestimmen, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Status des Objekts, das sich bereits in DynamoDB befindet. Sie können angeben, ob das vorhandene Element zurückgerufen werden soll, wenn die Bedingungsprüfung fehlschlägt. Weitere Informationen zu Transaktionsbedingungen finden Sie unter Ausdrücke für Transaktionsbedingungen. Dieser Wert ist erforderlich.

Beachten Sie Folgendes:

- Nur Schlüssel von Anforderungselementen werden in der Antwort zurückgegeben, wenn diese erfolgreich ist. Die Reihenfolge der Schlüssel entspricht der Reihenfolge der Anforderungselemente.
- Transaktionen werden auf irgendeine Weise ausgeführt. all-or-nothing Wenn ein Anforderungselement einen Fehler verursacht, wird die gesamte Transaktion nicht ausgeführt, und Fehlerdetails werden zurückgegeben.
- Es können keine zwei Anforderungselemente auf dasselbe Element ausgerichtet werden. Andernfalls führen sie zu TransactionCanceledExceptionFehlern.
- Wenn der Fehler einer Transaktion vorliegt TransactionCanceledException, wird der `cancellationReasons` Block aufgefüllt. Wenn die Bedingungsprüfung eines Anforderungselements fehlschlägt und Sie nicht `returnValuesOnConditionCheckFailure`

als `false` angegeben haben, wird das in der Tabelle vorhandene Element `item` an der entsprechenden Position des `cancellationReasons`-Blocks abgerufen und gespeichert.

- `TransactWriteItems`ist auf 100 Anfrageelemente begrenzt.
- Dieser Vorgang wird nicht unterstützt, wenn er zusammen mit der Konflikterkennung verwendet wird. Wenn Sie beide gleichzeitig verwenden, kann dies zu einem Fehler führen.

Für das folgende Beispiel für eine Anforderungszuweisungsvorlage gilt:

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

Wenn die Transaktion erfolgreich ist, ist das Aufrufergebnis in `$ctx.result` wie folgt verfügbar:

```
{
    "keys": [
       // Key of the PutItem request
       {
           "post_id": "p1",
       },
       // Key of the UpdateItem request
       {
           "author_id": "a1"
       }
    ],
    "cancellationReasons": null
}
```

Wenn die Transaktion aufgrund eines Fehlers bei der Zustandsprüfung der `PutItem` Anfrage fehlschlägt, `$ctx.result` ist das Aufrufergebnis wie folgt verfügbar:

```
{
    "keys": null,
    "cancellationReasons": [
       {
           "item": {
               "post_id": "p1",
               "post_title": "Actual old title",
               "post_description": "Old description"
```

```
            },
            "type": "ConditionCheckFailed",
            "message": "The condition check failed."
        },
        {
            "type": "None",
            "message": "None"
        }
    ]
}
```

`$ctx.error` enthält die Einzelheiten zu dem Fehler. Die Schlüssel keys und cancellationReasons sind garantiert in `$ctx.result` vorhanden.

Ein vollständigeres Beispiel AppSync finden Sie im DynamoDB-Transaktions-Tutorial mit diesem Tutorial: DynamoDB-Transaktionsauflösungen.

Geben Sie System ein (Zuordnung anfordern)

Wenn Sie den AWS AppSync DynamoDB-Resolver zum Aufrufen Ihrer DynamoDB-Tabellen verwenden, AWS AppSync muss der Typ jedes Werts bekannt sein, der in diesem Aufruf verwendet werden soll. Dies liegt daran, dass DynamoDB mehr Typprimitive unterstützt als GraphQL oder JSON (z. B. Sets und Binärdaten). AWS AppSync benötigt einige Hinweise bei der Übersetzung zwischen GraphQL und DynamoDB, andernfalls müsste es einige Annahmen darüber treffen, wie Daten in Ihrer Tabelle strukturiert sind.

Weitere Informationen zu DynamoDB-Datentypen finden Sie in der Dokumentation zu DynamoDB-Datentypdeskriptoren und Datentypen.

Ein DynamoDB-Wert wird durch ein JSON-Objekt dargestellt, das ein einzelnes Schlüssel-Wert-Paar enthält. Der Schlüssel gibt den DynamoDB-Typ an, und der Wert gibt den Wert selbst an. Im folgenden Beispiel gibt der Schlüssel `S` an, dass der Wert eine Zeichenfolge und der Wert `identifier` selbst ein Zeichenfolgewert ist.

```
{ "S" : "identifier" }
```

Beachten Sie, dass das JSON-Objekt nicht mehr als ein Schlüssel-Wert-Paar haben kann. Wenn mehr als ein Schlüssel-Wert-Paar angegeben wird, wird das Zuweisungsdokument für Anforderungen nicht analysiert.

Ein DynamoDB-Wert wird überall in einem Anforderungszuordnungsdokument verwendet, wo Sie einen Wert angeben müssen. Zu den Stellen, an denen Sie dies durchführen müssen, gehören: die Abschnitte `key` und `attributeValue` und der Bereich `expressionValues` von Ausdrucksabschnitten. Im folgenden Beispiel `identifier` wird der DynamoDB-Zeichenkettenwert dem `id` Feld in einem `key` Abschnitt zugewiesen (möglicherweise in einem `GetItem` Anforderungszuordnungsdokument).

```
"key" : {
   "id" : { "S" : "identifier" }
}
```

Unterstützte Typen

AWS AppSync unterstützt die folgenden DynamoDB-Skalar-, Dokument- und Satztypen:

Zeichenfolgetyp **S**

Ein einzelner Zeichenfolgewert. Ein DynamoDB-Zeichenkettenwert wird wie folgt bezeichnet:

```
{ "S" : "some string" }
```

Eine Verwendungsbeispiel ist:

```
"key" : {
   "id" : { "S" : "some string" }
}
```

Zeichenfolgesatztyp **SS**

Ein Satz von Zeichenfolgewerten. Ein DynamoDB String Set-Wert wird wie folgt bezeichnet:

```
{ "SS" : [ "first value", "second value", ... ] }
```

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "phoneNumbers" : { "SS" : [ "+1 555 123 4567", "+1 555 234 5678" ] }
}
```

Zahlentyp **N**

Ein einzelner numerischer Wert. Ein DynamoDB-Zahlenwert wird wie folgt bezeichnet:

```
{ "N" : 1234 }
```

Eine Verwendungsbeispiel ist:

```
"expressionValues" : {
   ":expectedVersion" : { "N" : 1 }
}
```

Zahlensatztyp **NS**

Ein Satz von Zahlenwerten. Ein DynamoDB Number Set-Wert wird wie folgt bezeichnet:

```
{ "NS" : [ 1, 2.3, 4 ... ] }
```

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "sensorReadings" : { "NS" : [ 67.8, 12.2, 70 ] }
}
```

Binärtyp **B**

Ein Binärwert. Ein DynamoDB-Binärwert wird wie folgt bezeichnet:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

Beachten Sie, dass es sich bei dem Wert tatsächlich um eine Zeichenfolge handelt, wobei die Zeichenfolge die Base64-kodierte Darstellung der Binärdaten ist. AWS AppSync dekodiert diese Zeichenfolge wieder in ihren Binärwert, bevor sie an DynamoDB gesendet wird. AWS AppSync verwendet das Base64-Dekodierungsschema, wie es in RFC 2045 definiert ist: Jedes Zeichen, das nicht im Base64-Alphabet vorkommt, wird ignoriert.

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
```

```
   "binaryMessage" : { "B" : "SGVsbG8sIFdvcmxkIQo=" }
}
```

Binärsatztyp **BS**

Ein Satz von Binärwerten. Ein DynamoDB Binary Set-Wert wird wie folgt bezeichnet:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

Beachten Sie, dass es sich bei dem Wert tatsächlich um eine Zeichenfolge handelt, wobei die Zeichenfolge die Base64-kodierte Darstellung der Binärdaten ist. AWS AppSync dekodiert diese Zeichenfolge wieder in ihren Binärwert, bevor sie an DynamoDB gesendet wird. AWS AppSync verwendet das Base64-Dekodierungsschema, wie es in RFC 2045 definiert ist: Jedes Zeichen, das nicht im Base64-Alphabet vorkommt, wird ignoriert.

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "binaryMessages" : { "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ] }
}
```

Boolescher Typ **BOOL**

Ein Boolescher Wert Ein boolescher DynamoDB-Wert wird wie folgt bezeichnet:

```
{ "BOOL" : true }
```

Beachten Sie, dass nur `true` und `false` gültige Werte sind.

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "orderComplete" : { "BOOL" : false }
}
```

Listentyp **L**

Eine Liste aller anderen unterstützten DynamoDB-Werte. Ein DynamoDB-Listenwert wird wie folgt gekennzeichnet:

```
{ "L" : [ ... ] }
```

Beachten Sie, dass es sich bei dem Wert um einen zusammengesetzten Wert handelt, wobei die Liste null oder mehr aller unterstützten DynamoDB-Werte (einschließlich anderer Listen) enthalten kann. Die Liste kann auch eine Mischung aus verschiedenen Typen enthalten.

Eine Verwendungsbeispiel ist:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

Zuordnungstyp **M**

Stellt eine ungeordnete Sammlung von Schlüssel-Wert-Paaren anderer unterstützter DynamoDB-Werte dar. Ein DynamoDB-Zuordnungswert wird wie folgt bezeichnet:

```
{ "M" : { ... } }
```

Beachten Sie, dass eine Zuordnung null oder mehrere Schlüssel-Wert-Paare enthalten kann. Der Schlüssel muss eine Zeichenfolge sein, und der Wert kann ein beliebiger unterstützter DynamoDB-Wert sein (einschließlich anderer Maps). Die Zuordnung kann auch eine Mischung aus verschiedenen Typen enthalten.

Eine Verwendungsbeispiel ist:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

Null-Typ **NULL**

Ein Null-Wert. Ein DynamoDB-Null-Wert wird wie folgt bezeichnet:

```
{ "NULL" : null }
```

Eine Verwendungsbeispiel ist:

```
"attributeValues" : {
   "phoneNumbers" : { "NULL" : null }
}
```

Weitere Informationen zu den einzelnen Typen finden Sie in der DynamoDB-Dokumentation.

Geben Sie System ein (Antwortzuordnung)

Wenn Sie eine Antwort von DynamoDB erhalten, konvertiert sie AWS AppSync automatisch in primitive GraphQL- und JSON-Typen. Jedes Attribut in DynamoDB wird dekodiert und im Antwortzuordnungskontext zurückgegeben.

Wenn DynamoDB beispielsweise Folgendes zurückgibt:

```
{
    "id" : { "S" : "1234" },
    "name" : { "S" : "Nadia" },
    "age" : { "N" : 25 }
}
```

Dann konvertiert der AWS AppSync DynamoDB-Resolver es in GraphQL- und JSON-Typen wie folgt:

```
{
    "id" : "1234",
    "name" : "Nadia",
    "age" : 25
}
```

In diesem Abschnitt wird erklärt, wie die folgenden DynamoDB-Skalar-, Dokument- und Satztypen AWS AppSync konvertiert werden:

Zeichenfolgetyp **S**

Ein einzelner Zeichenfolgewert. Ein DynamoDB-Zeichenkettenwert wird als Zeichenfolge zurückgegeben.

Beispiel: DynamoDB hat den folgenden DynamoDB-Zeichenkettenwert zurückgegeben:

```
{ "S" : "some string" }
```

AWS AppSync konvertiert ihn in eine Zeichenfolge:

```
"some string"
```

Zeichenfolgesatztyp **SS**

Ein Satz von Zeichenfolgewerten. Ein DynamoDB String Set-Wert wird als eine Liste von Zeichenketten zurückgegeben.

Beispiel: DynamoDB hat den folgenden DynamoDB String Set-Wert zurückgegeben:

```
{ "SS" : [ "first value", "second value", ... ] }
```

AWS AppSync konvertiert ihn in eine Liste von Zeichenketten:

```
[ "+1 555 123 4567", "+1 555 234 5678" ]
```

Zahlentyp **N**

Ein einzelner numerischer Wert. Ein DynamoDB-Zahlenwert wird als Zahl zurückgegeben.

Beispiel: DynamoDB hat den folgenden DynamoDB-Zahlenwert zurückgegeben:

```
{ "N" : 1234 }
```

AWS AppSync wandelt ihn in eine Zahl um:

```
1234
```

Zahlensatztyp **NS**

Ein Satz von Zahlenwerten. Ein DynamoDB Number Set-Wert wird als eine Liste von Zahlen zurückgegeben.

Beispiel: DynamoDB hat den folgenden DynamoDB Number Set-Wert zurückgegeben:

```
{ "NS" : [ 67.8, 12.2, 70 ] }
```

AWS AppSync wandelt ihn in eine Liste von Zahlen um:

```
[ 67.8, 12.2, 70 ]
```

Binärtyp **B**

Ein Binärwert. Ein DynamoDB-Binärwert wird als Zeichenfolge zurückgegeben, die die Base64-Darstellung dieses Werts enthält.

Wenn DynamoDB beispielsweise den folgenden DynamoDB-Binärwert zurückgegeben hat:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

AWS AppSync konvertiert ihn in eine Zeichenfolge, die die Base64-Darstellung des Werts enthält:

```
"SGVsbG8sIFdvcmxkIQo="
```

Beachten Sie, dass die Binärdaten im Base64-Codierungsschema codiert werden, wie in RFC 4648 und RFC 2045 angegeben.

Binärsatztyp **BS**

Ein Satz von Binärwerten. Ein DynamoDB Binary Set-Wert wird als eine Liste von Zeichenketten zurückgegeben, die die Base64-Repräsentation der Werte enthalten.

Beispiel: DynamoDB hat den folgenden DynamoDB Binary Set-Wert zurückgegeben:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

AWS AppSync konvertiert ihn in eine Liste von Zeichenketten, die die Base64-Darstellung der Werte enthalten:

```
[ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ]
```

Beachten Sie, dass die Binärdaten im Base64-Codierungsschema codiert werden, wie in RFC 4648 und RFC 2045 angegeben.

Boolescher Typ **BOOL**

Ein Boolescher Wert Ein boolescher DynamoDB-Wert wird als boolescher Wert zurückgegeben.

Wenn DynamoDB beispielsweise den folgenden booleschen DynamoDB-Wert zurückgegeben hat:

```
{ "BOOL" : true }
```

AWS AppSync wandelt ihn in einen booleschen Wert um:

```
true
```

Listentyp **L**

Eine Liste aller anderen unterstützten DynamoDB-Werte. Ein DynamoDB-Listenwert wird als Werteliste zurückgegeben, wobei jeder innere Wert ebenfalls konvertiert wird.

Wenn DynamoDB beispielsweise den folgenden DynamoDB-Listenwert zurückgegeben hat:

```
{ "L" : [
      { "S"  : "A string value" },
      { "N"  : 1 },
      { "SS" : [ "Another string value", "Even more string values!" ] }
   ]
}
```

AWS AppSync konvertiert ihn in eine Liste von konvertierten Werten:

```
[ "A string value", 1, [ "Another string value", "Even more string values!" ] ]
```

Zuordnungstyp **M**

A key/value collection of any other supported DynamoDB value. A DynamoDB Map value is returned as a JSON object, where each key/value wird ebenfalls konvertiert.

Wenn DynamoDB beispielsweise den folgenden DynamoDB-Zuordnungswert zurückgegeben hat:

```
{ "M" : {
      "someString" : { "S"  : "A string value" },
```

```
      "someNumber" : { "N"  : 1 },
      "stringSet"  : { "SS" : [ "Another string value", "Even more string
 values!" ] }
   }
}
```

AWS AppSync konvertiert ihn in ein JSON-Objekt:

```
{
   "someString" : "A string value",
   "someNumber" : 1,
   "stringSet"  : [ "Another string value", "Even more string values!" ]
}
```

Null-Typ **`NULL`**

Ein Null-Wert.

Beispiel: DynamoDB hat den folgenden DynamoDB-Null-Wert zurückgegeben:

```
{ "NULL" : null }
```

AWS AppSync wandelt ihn in einen Nullwert um:

```
null
```

Filter

Wenn Sie Objekte in DynamoDB mithilfe der `Scan` Operationen `Query` und abfragen, können Sie optional a angeben, `filter` das die Ergebnisse auswertet und nur die gewünschten Werte zurückgibt.

Der Filterzuweisungsbereich eines `Query`- oder `Scan`-Zuweisungsdokuments weist die folgende Struktur auf:

```
"filter" : {
    "expression" : "filter expression"
    "expressionNames" : {
        "#name" : "name",
    },
```

```
    "expressionValues" : {
        ":value" : ... typed value
    },
}
```

Die Felder sind wie folgt definiert:

`expression`

Der Abfrageausdruck. Weitere Informationen zum Schreiben von Filterausdrücken finden Sie in der Dokumentation zu DynamoDB QueryFilter und DynamoDB ScanFilter. Dieses Feld muss angegeben werden.

`expressionNames`

Die Ersetzungen für Platzhalter der Namen von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter, der in der `expression` verwendet wird. Der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Namen von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

`expressionValues`

Die Ersetzungen für Platzhalter der Werte von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Wertplatzhalter, der im `expression` verwendet wird, und der Wert muss ein typisierter Wert sein. Weitere Informationen zum Angeben eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuweisung). Dieser muss angegeben werden. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Werte von Ausdrucksattributen gefüllt sein, die im `expression` verwendet werden.

Beispiel

Das folgende Beispiel ist ein Filterabschnitt für eine Mapping-Vorlage, in dem aus DynamoDB abgerufene Einträge nur zurückgegeben werden, wenn der Titel mit dem `title` Argument beginnt.

```
"filter" : {
    "expression" : "begins_with(#title, :title)",
    "expressionNames" : {
        "#title" : "title"
    },
    "expressionValues" : {
```

```
        ":title" : $util.dynamodb.toDynamoDBJson($context.arguments.title)
    }
}
```

Bedingungsausdrücke

Wenn Sie Objekte in DynamoDB mithilfe der Operationen,, und `DeleteItem` DynamoDB mutieren `PutItem``UpdateItem`, können Sie optional einen Bedingungsausdruck angeben, der steuert, ob die Anforderung erfolgreich sein soll oder nicht, basierend auf dem Zustand des Objekts, das sich bereits in DynamoDB befand, bevor der Vorgang ausgeführt wird.

Der AWS AppSync DynamoDB-Resolver ermöglicht die Angabe eines Bedingungsausdrucks in `PutItem``UpdateItem`, und `DeleteItem` fordert Mapping-Dokumente an sowie eine Strategie, die befolgt werden kann, wenn die Bedingung fehlschlägt und das Objekt nicht aktualisiert wurde.

Beispiel 1

Das folgende `PutItem`-Zuweisungsdokument enthält keinen Bedingungsausdruck. Dadurch wird ein Element in DynamoDB platziert, auch wenn ein Element mit demselben Schlüssel bereits vorhanden ist, wodurch das vorhandene Element überschrieben wird.

```
{
   "version" : "2017-02-28",
   "operation" : "PutItem",
   "key" : {
      "id" : { "S" : "1" }
   }
}
```

Beispiel 2

Das folgende `PutItem` Mapping-Dokument hat einen Bedingungsausdruck, der nur dann ermöglicht, dass der Vorgang erfolgreich ist, wenn ein Element mit demselben Schlüssel nicht in DynamoDB vorhanden ist.

```
{
   "version" : "2017-02-28",
   "operation" : "PutItem",
   "key" : {
      "id" : { "S" : "1" }
```

```
    },
    "condition" : {
       "expression" : "attribute_not_exists(id)"
    }
}
```

Wenn die Zustandsprüfung fehlschlägt, gibt der AWS AppSync DynamoDB-Resolver standardmäßig einen Fehler für die Mutation zurück. Der AWS AppSync DynamoDB-Resolver bietet jedoch einige zusätzliche Funktionen, mit denen Entwickler einige häufig auftretende Grenzfälle bewältigen können:

- Wenn der AWS AppSync DynamoDB-Resolver feststellen kann, dass der aktuelle Wert in DynamoDB dem gewünschten Ergebnis entspricht, behandelt er den Vorgang so, als ob er trotzdem erfolgreich gewesen wäre.
- Anstatt einen Fehler zurückzugeben, können Sie den Resolver so konfigurieren, dass er eine benutzerdefinierte Lambda-Funktion aufruft, um zu entscheiden, wie der AWS AppSync DynamoDB-Resolver mit dem Fehler umgehen soll.

Diese werden im Abschnitt Handling a Condition Check Failure detaillierter beschrieben.

Weitere Informationen zu DynamoDB-Bedingungsausdrücken finden Sie in der ConditionExpressions DynamoDB-Dokumentation.

Angabe einer Bedingung

Die Zuweisungsdokumente der `PutItem`- `UpdateItem`- und `DeleteItem`-Anforderung erlauben das Angeben eines optionalen `condition`-Abschnitts. Wenn nicht angegeben, wird keine Bedingungsprüfung ausgeführt. Wenn angegeben, muss die Bedingung wahr sein, damit die Operation erfolgreich ausgeführt werden kann.

Ein `condition`-Abschnitt weist die folgende Struktur auf:

```
"condition" : {
    "expression" : "someExpression"
    "expressionNames" : {
        "#foo" : "foo"
    },
    "expressionValues" : {
        ":bar" : ... typed value
    },
    "equalsIgnore" : [ "version" ],
```

```
    "consistentRead" : true,
    "conditionalCheckFailedHandler" : {
        "strategy" : "Custom",
        "lambdaArn" : "arn:..."
    }
}
```

Die folgenden Felder geben die Bedingung an:

`expression`

Der Aktualisierungsausdruck selbst. Weitere Informationen zum Schreiben von Bedingungsausdrücken finden Sie in der DynamoDB-Dokumentation ConditionExpressions . Dieses Feld muss angegeben werden.

`expressionNames`

Die Ersetzungen für Platzhalter für Ausdrucksattributnamen in der Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter, der im Ausdruck verwendet wird, und der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Namen von Ausdrucksattributen gefüllt sein, die im Ausdruck verwendet werden.

`expressionValues`

Die Ersetzungen für Platzhalter der Werte von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Wertplatzhalter, der im Ausdruck verwendet wird, und der Wert muss ein typisierter Wert sein. Weitere Informationen zum Angeben eines „typisierten Werts“ finden Sie unter Typsystem (Anforderungszuweisung). Dieser muss angegeben werden. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Werte von Ausdrucksattributen gefüllt sein, die im Ausdruck verwendet werden.

Die verbleibenden Felder teilen dem AWS AppSync DynamoDB-Resolver mit, wie er mit einem Fehler bei der Zustandsprüfung umgehen soll:

`equalsIgnore`

Wenn eine Zustandsprüfung bei der Verwendung des `PutItem` Vorgangs fehlschlägt, vergleicht der AWS AppSync DynamoDB-Resolver das Element, das sich derzeit in DynamoDB befindet, mit dem Element, das er zu schreiben versucht hat. Wenn sie identisch sind, wird die Operation behandelt, als wäre sie trotzdem erfolgreich ausgeführt worden. Sie können das `equalsIgnore`

Feld verwenden, um eine Liste von Attributen anzugeben, die bei diesem Vergleich ignoriert AWS AppSync werden sollen. Wenn der einzige Unterschied beispielsweise ein `version` Attribut war, wird der Vorgang so behandelt, als ob er erfolgreich war. Dies ist ein optionales Feld.

`consistentRead`

Wenn eine Zustandsprüfung fehlschlägt, AWS AppSync wird der aktuelle Wert des Elements mithilfe eines stark konsistenten Lesevorgangs von DynamoDB abgerufen. Sie können dieses Feld verwenden, um den AWS AppSync DynamoDB-Resolver anzuweisen, stattdessen einen eventuell konsistenten Lesevorgang zu verwenden. Dieses Feld ist optional und standardmäßig auf `true` gesetzt.

`conditionalCheckFailedHandler`

In diesem Abschnitt können Sie angeben, wie der AWS AppSync DynamoDB-Resolver einen Fehler bei der Zustandsprüfung behandelt, nachdem er den aktuellen Wert in DynamoDB mit dem erwarteten Ergebnis verglichen hat. Dieser Abschnitt ist optional. Wenn nicht angegeben, wird standardmäßig eine Strategie von `Reject` verwendet.

`strategy`

Die Strategie, die der AWS AppSync DynamoDB-Resolver verfolgt, nachdem er den aktuellen Wert in DynamoDB mit dem erwarteten Ergebnis verglichen hat. Dieses Feld ist erforderlich und besitzt die folgenden möglichen Werte:

`Reject`

Die Mutation schlägt fehl und der GraphQL-Antwort wird ein Fehler hinzugefügt.

`Custom`

Der AWS AppSync DynamoDB-Resolver ruft eine benutzerdefinierte Lambda-Funktion auf, um zu entscheiden, wie mit dem Fehler bei der Zustandsprüfung umgegangen werden soll. Wenn die `strategy` auf `Custom` gesetzt ist, muss das `lambdaArn`-Feld den ARN der aufzurufenden Lambda-Funktion enthalten.

`lambdaArn`

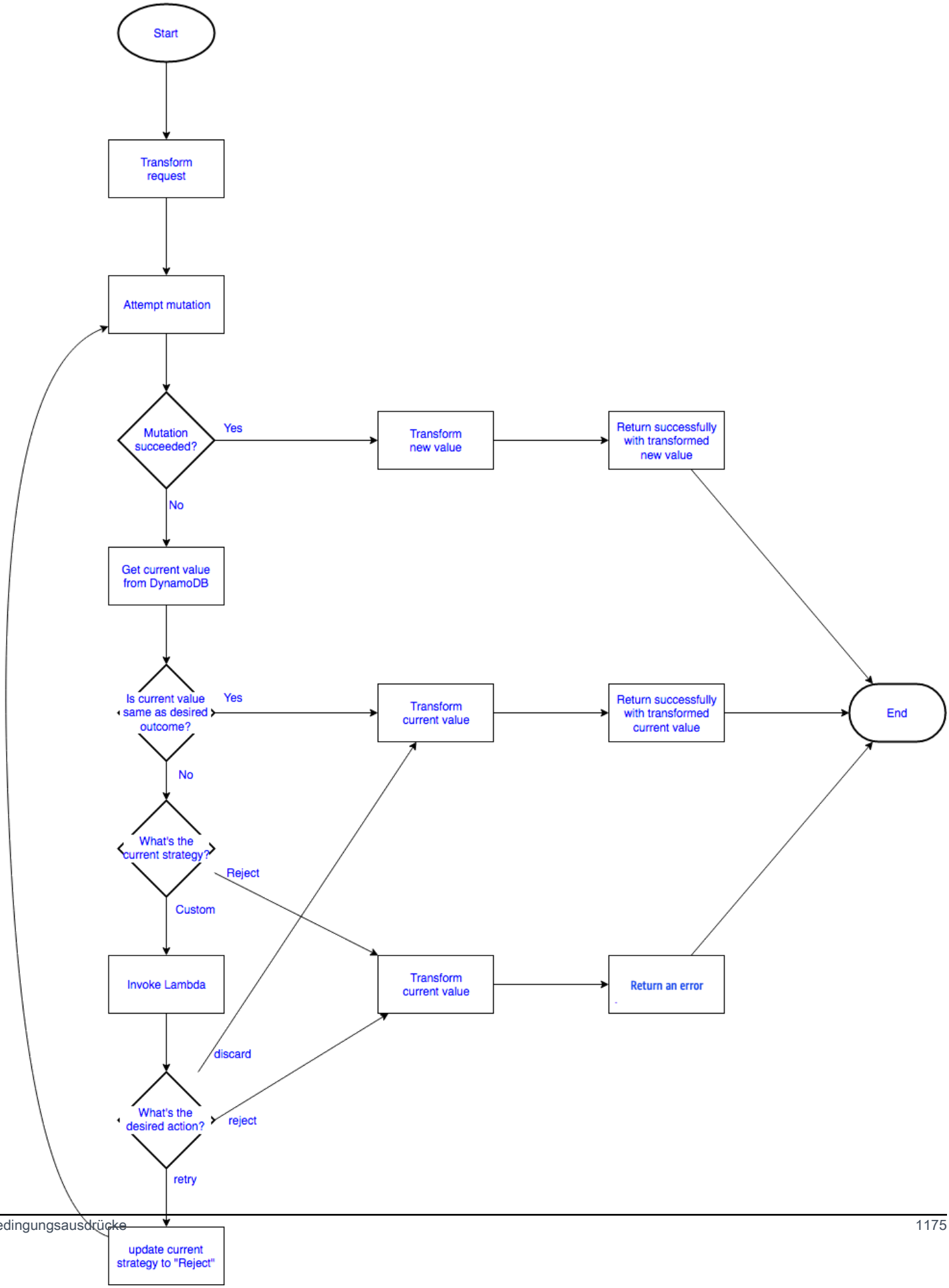
Der ARN der aufzurufenden Lambda-Funktion, die bestimmt, wie der AWS AppSync DynamoDB-Resolver mit dem Fehler bei der Zustandsprüfung umgehen soll. Dieses Feld muss nur angegeben werden, wenn `strategy` auf `Custom` festgelegt ist. Weitere Informationen zur Verwendung dieser Funktion finden Sie unter Umgang mit einem Bedingungsausdrucksfehler.

Behandlung eines Fehlers bei der Zustandsprüfung

Wenn eine Zustandsprüfung fehlschlägt, gibt der AWS AppSync DynamoDB-Resolver standardmäßig einen Fehler für die Mutation und den aktuellen Wert des Objekts in DynamoDB zurück. Der AWS AppSync DynamoDB-Resolver bietet jedoch einige zusätzliche Funktionen, mit denen Entwickler einige häufig auftretende Grenzfälle bewältigen können:

- Wenn der AWS AppSync DynamoDB-Resolver feststellen kann, dass der aktuelle Wert in DynamoDB dem gewünschten Ergebnis entspricht, behandelt er den Vorgang so, als ob er trotzdem erfolgreich gewesen wäre.
- Anstatt einen Fehler zurückzugeben, können Sie den Resolver so konfigurieren, dass er eine benutzerdefinierte Lambda-Funktion aufruft, um zu entscheiden, wie der AWS AppSync DynamoDB-Resolver mit dem Fehler umgehen soll.

Das Flussdiagramm für diesen Prozess ist:


Start
Transform request
Attempt mutation
Mutation succeeded?
Yes
Transform new value
Return successfully with transformed new value
No
Get current value from DynamoDB
Is current value same as desired outcome?
Yes
Transform current value
Return successfully with transformed current value
End
No
What's the current strategy?
Reject
Custom
Invoke Lambda
Transform current value
Return an error
discard
What's the desired action?
reject
retry
update current strategy to "Reject"

Es wird geprüft, ob das gewünschte Ergebnis vorliegt

Wenn die Zustandsprüfung fehlschlägt, führt der AWS AppSync DynamoDB-Resolver eine `GetItem` DynamoDB-Anforderung aus, um den aktuellen Wert des Elements von DynamoDB abzurufen. Standardmäßig wird ein Strongly Consistent-Lesevorgang verwendet, jedoch kann dies mit dem `consistentRead`-Feld im `condition`-Block konfiguriert werden, und mit dem erwarteten Ergebnis verglichen:

- Für den `PutItem` Vorgang vergleicht der AWS AppSync DynamoDB-Resolver den aktuellen Wert mit dem Wert, den er zu schreiben versucht hat, und schließt alle unter aufgelisteten Attribute aus dem Vergleich `equalsIgnore` aus. Wenn die Elemente identisch sind, wird der Vorgang als erfolgreich behandelt und das Element zurückgegeben, das von DynamoDB abgerufen wurde. Andernfalls wird die konfigurierte Strategie verfolgt.

  Beispiel: Wenn das Zuweisungsdokument der `PutItem`-Anforderung wie folgt ausgesehen hat:

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

  Und das Element, das sich derzeit in DynamoDB befindet, wie folgt ausgesehen hat:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
```

```
}
```

Der AWS AppSync DynamoDB-Resolver würde das Element, das er schreiben wollte, mit dem aktuellen Wert vergleichen und feststellen, dass der einzige Unterschied das `version` Feld ist. Da er jedoch so konfiguriert ist, dass er das `version` Feld ignoriert, behandelt er den Vorgang als erfolgreich und gibt das Element zurück, das von DynamoDB abgerufen wurde.

- Für den `DeleteItem` Vorgang überprüft der AWS AppSync DynamoDB-Resolver, ob ein Element von DynamoDB zurückgegeben wurde. Wenn kein Element zurückgegeben wurde, behandelt er die Operation als erfolgreich. Andernfalls wird die konfigurierte Strategie verfolgt.
- Für den `UpdateItem` Vorgang verfügt der AWS AppSync DynamoDB-Resolver nicht über genügend Informationen, um festzustellen, ob das Element, das sich derzeit in DynamoDB befindet, dem erwarteten Ergebnis entspricht, und folgt daher der konfigurierten Strategie.

Wenn der aktuelle Status des Objekts in DynamoDB vom erwarteten Ergebnis abweicht, folgt der AWS AppSync DynamoDB-Resolver der konfigurierten Strategie, entweder die Mutation zurückzuweisen oder eine Lambda-Funktion aufzurufen, um zu bestimmen, was als Nächstes zu tun ist.

Wir folgen der Strategie „Ablehnen“

Wenn die `Reject` Strategie befolgt wird, gibt der AWS AppSync DynamoDB-Resolver einen Fehler für die Mutation zurück.

Angenommen, Sie haben folgende Mutationsanforderung:

```
mutation {
    updatePerson(id: 1, name: "Steve", expectedVersion: 1) {
        Name
        theVersion
    }
}
```

Wenn das von DynamoDB zurückgegebene Element wie folgt aussieht:

```
{
   "id" : { "S" : "1" },
   "name" : { "S" : "Steve" },
   "version" : { "N" : 8 }
}
```

Und die Antwortzuweisungsvorlage wie folgt aussieht:

```
{
   "id" : $util.toJson($context.result.id),
   "Name" : $util.toJson($context.result.name),
   "theVersion" : $util.toJson($context.result.version)
}
```

Die GraphQL-Antwort sieht wie folgt aus:

```
{
  "data": null,
  "errors": [
    {
      "message": "The conditional request failed (Service: AmazonDynamoDBv2;
 Status Code: 400; Error Code: ConditionalCheckFailedException; Request ID:
 ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ)"
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      ...
    }
  ]
}
```

Beachten Sie auch, dass, wenn bei einer erfolgreichen Mutation alle Felder im zurückgegebenen Objekt von anderen Resolvern ausgefüllt werden, diese nicht auf den Resolver angewendet werden, wenn das Objekt im `error`-Abschnitt zurückgegeben wird.

Folgen Sie der „benutzerdefinierten“ Strategie

Wenn die `Custom` Strategie befolgt wird, ruft der AWS AppSync DynamoDB-Resolver eine Lambda-Funktion auf, um zu entscheiden, was als Nächstes zu tun ist. Die Lambda-Funktion wählt eine der folgenden Optionen aus:

- Die Mutation `reject`. Dadurch wird der AWS AppSync DynamoDB-Resolver angewiesen`Reject`, sich wie bei der konfigurierten Strategie zu verhalten und einen Fehler für die Mutation und den aktuellen Wert des Objekts in DynamoDB zurückzugeben, wie im vorherigen Abschnitt beschrieben.
- Die Mutation `discard`. Dadurch wird der AWS AppSync DynamoDB-Resolver angewiesen, den Fehler bei der Zustandsprüfung stillschweigend zu ignorieren und den Wert in DynamoDB zurückzugeben.

- Die Mutation `retry`. Dadurch wird der AWS AppSync DynamoDB-Resolver angewiesen, die Mutation mit einem neuen Anforderungszuordnungsdokument erneut zu versuchen.

Die Lambda-Aufrufanforderung

Der AWS AppSync DynamoDB-Resolver ruft die Lambda-Funktion auf, die in der angegeben ist. `lambdaArn` Er verwendet die gleiche `service-role-arn`-Konfiguration für die Datenquelle. Die Nutzlast des Aufrufs hat die folgende Struktur:

```
{
    "arguments": { ... },
    "requestMapping": {... },
    "currentValue": { ... },
    "resolver": { ... },
    "identity": { ... }
}
```

Die Felder sind wie folgt definiert:

`arguments`

Die Argumente aus der GraphQL-Mutation. Diese entsprechen den Argumente, die dem Zuweisungsdokument der Anforderung in `$context.arguments` zur Verfügung stehen.

`requestMapping`

Das Zuweisungsdokument der Anforderung für diese Operation.

`currentValue`

Der aktuelle Wert des Objekts in DynamoDB.

`resolver`

Informationen über den Resolver AWS AppSync .

`identity`

Informationen über den Aufrufer. Diese entsprechen den Identitätsinformationen, die dem Zuweisungsdokument der Anforderung in `$context.identity` zur Verfügung stehen.

Ein vollständiges Beispiel für die Nutzlast:

```
{
```

```
    "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"
    }
}
```

Die Lambda-Aufrufantwort

Die Lambda-Funktion kann die Nutzlast des Aufrufs untersuchen und jede Geschäftslogik anwenden, um zu entscheiden, wie der AWS AppSync DynamoDB-Resolver mit dem Fehler umgehen soll. Es gibt drei Optionen für den Umgang mit dem Bedingungsprüfungsfehler:

- Die Mutation `reject`. Die Antwortnutzlast für diese Option muss diese Struktur aufweisen:

  ```
  {
      "action": "reject"
  }
  ```

  Dadurch wird der AWS AppSync DynamoDB-Resolver angewiesen`Reject`, sich wie bei der konfigurierten Strategie zu verhalten und einen Fehler für die Mutation und den aktuellen Wert des Objekts in DynamoDB zurückzugeben, wie im obigen Abschnitt beschrieben.
- Die Mutation `discard`. Die Antwortnutzlast für diese Option muss diese Struktur aufweisen:

  ```
  {
      "action": "discard"
  }
  ```

  Dadurch wird der AWS AppSync DynamoDB-Resolver angewiesen, den Fehler bei der Zustandsprüfung stillschweigend zu ignorieren und den Wert in DynamoDB zurückzugeben.
- Die Mutation `retry`. Die Antwortnutzlast für diese Option muss diese Struktur aufweisen:

  ```
  {
      "action": "retry",
      "retryMapping": { ... }
  }
  ```

  Dadurch wird der AWS AppSync DynamoDB-Resolver angewiesen, die Mutation mit einem neuen Anforderungszuordnungsdokument erneut zu versuchen. Die Struktur des `retryMapping` Abschnitts hängt vom DynamoDB-Vorgang ab und ist eine Teilmenge des vollständigen Anforderungszuordnungsdokuments für diesen Vorgang.

  Für `PutItem` weist der `retryMapping`-Abschnitt die folgende Struktur auf. Eine Beschreibung des `attributeValues` Felds finden Sie unter. PutItem

  ```
  {
  ```

```
    "attributeValues": { ... },
    "condition": {
        "equalsIgnore" = [ ... ],
        "consistentRead" = true
    }
}
```

Für `UpdateItem` weist der `retryMapping`-Abschnitt die folgende Struktur auf. Eine Beschreibung des `update` Abschnitts finden Sie unter UpdateItem.

```
{
    "update" : {
        "expression" : "someExpression"
        "expressionNames" : {
            "#foo" : "foo"
        },
        "expressionValues" : {
            ":bar" : ... typed value
        }
    },
    "condition": {
        "consistentRead" = true
    }
}
```

Für `DeleteItem` weist der `retryMapping`-Abschnitt die folgende Struktur auf.

```
{
    "condition": {
        "consistentRead" = true
    }
}
```

Es ist nicht möglich, eine andere Operation oder einen anderen Schlüssel anzugeben. Der AWS AppSync DynamoDB-Resolver erlaubt nur Wiederholungen derselben Operation für dasselbe Objekt. Beachten Sie auch, dass der Abschnitt `condition` nicht zulässt, dass ein `conditionalCheckFailedHandler` angegeben wird. Schlägt der Wiederholungsversuch fehl, folgt der AWS AppSync DynamoDB-Resolver der Strategie. `Reject`

Hier sehen Sie ein Beispiel für eine Lambda-Funktion zum Umgang mit einer fehlgeschlagenen `PutItem`-Anforderung. Die Geschäftslogik prüft, wer den Aufruf ausgeführt hat. Wenn es von erstellt wurde`jeffTheAdmin`, wiederholt es die Anfrage und aktualisiert das `version` und `expectedVersion` aus dem Element, das sich derzeit in DynamoDB befindet. Andernfalls wird die Mutation ablehnt.

```
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)
};
```

Ausdrücke für Transaktionsbedingungen

Transaktionsbedingungsausdrücke sind in Anforderungszuweisungsvorlagen aller vier Arten von Operationen in `TransactWriteItems`, nämlich `PutItem`, `DeleteItem`, `UpdateItem` und `ConditionCheck` verfügbar.

Für `PutItem` `DeleteItemUpdateItem`, und ist der Ausdruck für die Transaktionsbedingung optional. Für `ConditionCheck` ist der Ausdruck der Transaktionsbedingung erforderlich.

Beispiel 1

Das folgende `DeleteItem`-Transaktionszuweisungsdokument weist keinen Bedingungsausdruck auf. Infolgedessen wird das Element in DynamoDB gelöscht.

```
{
   "version": "2018-05-29",
   "operation": "TransactWriteItems",
   "transactItems": [
      {
         "table": "posts",
         "operation": "DeleteItem",
         "key": {
            "id": { "S" : "1" }
         }
      }
   ]
}
```

Beispiel 2

Das folgende `DeleteItem` Transaktions-Mapping-Dokument enthält einen Ausdruck für eine Transaktionsbedingung, mit dem der Vorgang nur dann erfolgreich ausgeführt werden kann, wenn der Autor dieses Beitrags einem bestimmten Namen entspricht.

```
{
   "version": "2018-05-29",
   "operation": "TransactWriteItems",
   "transactItems": [
      {
         "table": "posts",
         "operation": "DeleteItem",
         "key": {
            "id": { "S" : "1" }
         }
         "condition": {
            "expression": "author = :author",
            "expressionValues": {
               ":author": { "S" : "Chunyan" }
```

```
            }
          }
        }
    ]
}
```

Wenn die Bedingungsprüfung fehlschlägt, führt dies zu `TransactionCanceledException`, und die Fehlerdetails werden in `$ctx.result.cancellationReasons` zurückgegeben. Beachten Sie, dass standardmäßig das alte Element in DynamoDB zurückgegeben wird, bei dem die Zustandsprüfung fehlgeschlagen ist. `$ctx.result.cancellationReasons`

Angabe einer Bedingung

Die Zuweisungsdokumente der `PutItem`- `UpdateItem`- und `DeleteItem`-Anforderung erlauben das Angeben eines optionalen `condition`-Abschnitts. Wenn nicht angegeben, wird keine Bedingungsprüfung ausgeführt. Wenn angegeben, muss die Bedingung wahr sein, damit die Operation erfolgreich ausgeführt werden kann. Der `ConditionCheck` muss einen `condition`-Abschnitt haben, der angegeben werden soll. Die Bedingung muss erfüllt sein, damit die gesamte Transaktion erfolgreich ist.

Ein `condition`-Abschnitt weist die folgende Struktur auf:

```
"condition": {
    "expression": "someExpression",
    "expressionNames": {
        "#foo": "foo"
    },
    "expressionValues": {
        ":bar": ... typed value
    },
    "returnValuesOnConditionCheckFailure": false
}
```

Die folgenden Felder geben die Bedingung an:

`expression`

Der Aktualisierungsausdruck selbst. Weitere Informationen zum Schreiben von Bedingungsausdrücken finden Sie in der DynamoDB-Dokumentation ConditionExpressions . Dieses Feld muss angegeben werden.

`expressionNames`

Die Ersetzungen für Platzhalter für Ausdrucksattributnamen in der Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter, der im Ausdruck verwendet wird, und der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Namen von Ausdrucksattributen gefüllt sein, die im Ausdruck verwendet werden.

`expressionValues`

Die Ersetzungen für Platzhalter der Werte von Ausdrucksattributen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Wertplatzhalter, der im Ausdruck verwendet wird, und der Wert muss ein typisierter Wert sein. Weitere Informationen zum Angeben eines „typisierten Werts“ finden Sie unter „Typsystem (Anforderungszuweisung)“. Dieser muss angegeben werden. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter der Werte von Ausdrucksattributen gefüllt sein, die im Ausdruck verwendet werden.

`returnValuesOnConditionCheckFailure`

Geben Sie an, ob das Element in DynamoDB zurückgeholt werden soll, wenn eine Zustandsprüfung fehlschlägt. Das abgerufene Element befindet sich in `$ctx.result.cancellationReasons[$index].item`, wobei der `$index` Index des Anforderungselements ist, durch das die Bedingungsprüfung fehlgeschlagen ist. Dieser Wert ist standardmäßig auf true gesetzt.

Projektionen

Wenn Sie Objekte in DynamoDB mithilfe der `TransactGetItems` Operationen`GetItem`,`Scan`, `QueryBatchGetItem`, und lesen, können Sie optional eine Projektion angeben, die die gewünschten Attribute identifiziert. Die Projektion hat die folgende Struktur, die Filtern ähnelt:

```
"projection" : {
    "expression" : "projection expression"
    "expressionNames" : {
        "#name" : "name",
    }
}
```

Die Felder sind wie folgt definiert:

`expression`

Der Projektionsausdruck, der eine Zeichenfolge ist. Zum Abrufen eines einzelnen Attributs geben Sie seinen Namen an. Bei mehreren Attributen müssen die Namen durch Kommas getrennte Werte sein. Weitere Informationen zum Schreiben von Projektionsausdrücken finden Sie in der Dokumentation zu DynamoDB-Projektionsausdrücken. Dies ist ein Pflichtfeld.

`expressionNames`

Die Ersetzungen für Platzhalter für Ausdrucksattributnamen in Form von Schlüssel-Wert-Paaren. Der Schlüssel entspricht einem Namensplatzhalter, der in der `expression` verwendet wird. Der Wert muss eine Zeichenfolge sein, die dem Attributnamen des Elements in DynamoDB entspricht. Dieses Feld ist optional und sollte nur mit Ersetzungen für Platzhalter für Ausdrucksattributnamen gefüllt werden, die in der verwendet werden. `expression` Weitere Informationen zu `expressionNames` finden Sie in der DynamoDB-Dokumentation.

Beispiel 1

Das folgende Beispiel ist ein Projektionsabschnitt für eine VTL-Mapping-Vorlage, in der nur die Attribute `author` und von DynamoDB zurückgegeben `id` werden:

```
"projection" : {
    "expression" : "#author, id",
    "expressionNames" : {
        "#author" : "author"
    }
}
```

> **Tip**
>
> Sie können mit $context.info auf Ihren GraphQL-Anforderungsauswahlsatz zugreifen. selectionSetList. In diesem Feld können Sie Ihren Projektionsausdruck Ihren Anforderungen entsprechend dynamisch gestalten.

> **Note**
>
> Bei der Verwendung von Projektionsausdrücken mit den `Scan` Operationen `Query` und `select` muss der Wert für sein`SPECIFIC_ATTRIBUTES`. Weitere Informationen finden Sie in der DynamoDB-Dokumentation.

AWS AppSync Referenz zur Resolver-Mapping-Vorlage für RDS

Die AWS AppSync RDS-Resolver-Mapping-Vorlagen ermöglichen es Entwicklern, SQL-Abfragen an eine Daten-API für Amazon Aurora Serverless zu senden und das Ergebnis dieser Abfragen zurückzuerhalten.

Vorlage für die Zuordnung anfordern

Die RDS-Anforderungszuweisungsvorlage ist recht einfach:

```
{
    "version": "2018-05-29",
    "statements": [],
    "variableMap": {},
    "variableTypeHintMap": {}
}
```

Hier finden Sie die JSON-Schema-Darstellung der Zuweisungsvorlage für RDS-Anforderungen, nachdem sie einmal aufgelöst wurde.

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

Im Folgenden finden Sie ein Beispiel für die Vorlage zur Anforderungszuweisung mit einer statischen Abfrage:

```
{
```

```
    "version": "2018-05-29",
    "statements": [
        "select title, isbn13 from BOOKS where author = 'Mark Twain'"
    ]
}
```

Version

Das Versionsfeld, das allen Vorlagen für die Anforderungszuweisung gemeinsam ist, definiert die Version, die die Vorlage verwendet. Das Versionsfeld ist erforderlich. Der Wert „2018-05-29“ ist die einzige Version, die für die Amazon RDS-Mapping-Vorlagen unterstützt wird.

```
"version": "2018-05-29"
```

Aussagen und VariableMap

Das Statements Array ist ein Platzhalter für die vom Entwickler bereitgestellten Abfragen. Derzeit werden bis zu zwei Abfragen pro Vorlage für die Anforderungszuweisung unterstützt. Das `variableMap` ist ein optionales Feld, das Aliase enthält, die verwendet werden können, um die SQL-Anweisungen kürzer und lesbarer zu machen. Zum Beispiel ist Folgendes möglich:

```
{
"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 verwendet die Variablenzuordnungswerte, um die SqlParameterizedAbfragen zu erstellen, die an die Amazon Aurora Serverless Data API gesendet werden. Die SQL-Anweisungen werden mit den in der Variablenzuordnung bereitgestellten Parametern ausgeführt, wodurch das Risiko einer SQL-Injection vermieden wird.

VariableTypeHintMap

Das `variableTypeHintMap` ist ein optionales Feld, das Alias-Typen enthält und zum Senden von Hinweisen auf SQL-Parametertypen verwendet werden kann. Diese Typhinweise vermeiden eine explizite Umwandlung in den SQL-Anweisungen, wodurch sie kürzer werden. Zum Beispiel ist Folgendes möglich:

```
{
    "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 verwendet den Variablenzuordnungswert, um die Abfragen zu erstellen, die an die Amazon Aurora Serverless Data API gesendet werden. Es verwendet auch die `variableTypeHintMap` Daten und sendet die Informationen des Typs an RDS. RDS-Unterstützung `typeHints` finden Sie hier.

AWS AppSyncReferenz zur Resolver-Mapping-Vorlage für OpenSearch

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Mit dem AWS AppSync Resolver für Amazon OpenSearch Service können Sie GraphQL verwenden, um Daten in vorhandenen OpenSearch Service-Domains in Ihrem Konto zu speichern und

abzurufen. Dieser Resolver ermöglicht es Ihnen, eine eingehende GraphQL-Anfrage einer Serviceanfrage zuzuordnen und die OpenSearch OpenSearch Serviceantwort dann wieder GraphQL zuzuordnen. In diesem Abschnitt werden die Zuordnungsvorlagen für die unterstützten OpenSearch Dienstoperationen beschrieben.

Zuordnungsvorlage anfordern

Die meisten Vorlagen für die Zuordnung von OpenSearch Serviceanfragen haben eine gemeinsame Struktur, bei der sich nur wenige Teile ändern. Im folgenden Beispiel wird eine Suche in einer OpenSearch Dienstdomäne ausgeführt, in der Dokumente unter einem Index namens organisiert sind`post`. Die Suchparameter sind im `body`-Abschnitt definiert, viele der gemeinsamen Abfragebestimmungen sind im `query`-Feld definiert. In diesem Beispiel wird nach Dokumenten gesucht, die `"Nadia"`, `"Bailey"` oder beides enthalten, und zwar in dem `author`-Feld eines Dokuments:

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

Vorlage für die Zuordnung von Antworten

Wie bei anderen Datenquellen sendet OpenSearch Service eine Antwort an AWS AppSync , die in GraphQL konvertiert werden muss.

Die meisten GraphQL-Abfragen suchen nach dem `_source` Feld aus einer OpenSearch Serviceantwort. Da Sie Suchen durchführen können, um entweder ein einzelnes Dokument oder eine Liste von Dokumenten zurückzugeben, werden in OpenSearch Service zwei häufig verwendete Vorlagen für die Antwortzuweisung verwendet:

Liste der Ergebnisse

```
[
    #foreach($entry in $context.result.hits.hits)
      #if( $velocityCount > 1 ) , #end
        $utils.toJson($entry.get("_source"))
    #end
]
```

Einzelnes Element

```
$utils.toJson($context.result.get("_source"))
```

`operation` field

> **Note**
>
> Dies gilt nur für die Vorlage für die Anforderungszuweisung.

HTTP-Methode oder Verb (GET, POST, PUT, HEAD oder DELETE), das AWS AppSync an die OpenSearch Service-Domain sendet. Der Schlüssel und der Wert müssen beide Strings sein.

```
"operation" : "PUT"
```

`path` field

> **Note**
>
> Dies gilt nur für die Vorlage für die Anforderungszuweisung.

Der Suchpfad für eine OpenSearch Serviceanfrage von AWS AppSync. Dies bildet eine URL für das HTTP-Verb der Operation. Der Schlüssel und der Wert müssen beide Strings sein.

```
"path" : "/<indexname>/_doc/<_id>"
"path" : "/<indexname>/_doc"
"path" : "/<indexname>/_search"
"path" : "/<indexname>/_update/<_id>
```

Wenn die Zuordnungsvorlage ausgewertet wird, wird dieser Pfad als Teil der HTTP-Anfrage gesendet, einschließlich der OpenSearch Dienstdomäne. Das vorherige Beispiel kann dann beispielsweise so aussehen:

```
GET https://opensearch-domain-name.REGION.es.amazonaws.com/indexname/type/_search
```

`params` field

> **Note**
>
> Dies gilt nur für die Vorlage für die Anforderungszuweisung.

Wird verwendet, um anzugeben, welche Aktion Ihre Suche durchführt, meistens indem der Abfragewert in den Body setzt. Es gibt jedoch einige andere Funktionen, die konfiguriert werden können, wie z. B. die Formatierung von Antworten.

- Header

  Die Header-Informationen, wie beispielsweise Schlüssel-Wert-Paare. Der Schlüssel und der Wert müssen beide Strings sein. Zum Beispiel:

  ```
  "headers" : {
      "Content-Type" : "application/json"
  }
  ```

  > **Note**
  >
  > AWS AppSync unterstützt derzeit nur JSON als`Content-Type`.

- queryString

Schlüssel-Wert-Paare, die allgemeine Optionen angeben, z. B. Codeformatierung für JSON-Antworten. Der Schlüssel und der Wert müssen beide Strings sein. Wenn Sie beispielsweise ein gut formatiertes JSON-Format haben möchten, geben Sie Folgendes an:

```
"queryString" : {
    "pretty" : "true"
}
```

- body

  Dies ist der Hauptteil Ihrer Anfrage, der es ermöglicht AWS AppSync , eine wohlgeformte Suchanfrage für Ihre OpenSearch Service-Domain zu erstellen. Der Schlüssel muss ein aus einem Objekt bestehender String sein. Ein paar Beispiele werden unten gezeigt.

Beispiel 1

Rückgabe aller Dokumente, die die Stadt „Seattle“ enthalten:

```
"body":{
    "from":0,
    "size":50,
    "query" : {
        "match" : {
            "city" : "seattle"
        }
    }
}
```

Beispiel 2

Rückgabe aller Dokumente, die „Washington“ als Stadt oder Staat enthalten:

```
"body":{
    "from":0,
    "size":50,
    "query" : {
        "multi_match" : {
            "query" : "washington",
            "fields" : ["city", "state"]
        }
```

```
    }
}
```

Variablen übergeben

> **Note**
>
> Dies gilt nur für die Vorlage zur Anforderungszuweisung.

Sie können auch Variablen als Teil der Bewertung der VTL-Anweisung weiterleiten. Angenommen, Sie hatten die folgende GraphQL-Abfrage:

```
query {
    searchForState(state: "washington"){
        ...
    }
}
```

Die Zuweisungsvorlage könnte den Staat als Argument auffassen:

```
"body":{
    "from":0,
    "size":50,
    "query" : {
        "multi_match" : {
            "query" : "$context.arguments.state",
            "fields" : ["city", "state"]
        }
    }
}
```

Eine Liste der Dienstprogramme, die Sie in VTL verwenden können, finden Sie unter Header für Zugriffsanforderungen.

AWS AppSync Referenz zur Resolver-Mapping-Vorlage für Lambda

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Sie können AWS AppSync Funktionen und Resolver verwenden, um Lambda-Funktionen in Ihrem Konto aufzurufen. Sie können Ihre Anforderungs-Payloads und die Antwort Ihrer Lambda-Funktionen gestalten, bevor Sie sie an Ihre Clients zurückgeben. Sie können auch Zuordnungsvorlagen verwenden, um AWS AppSync Hinweise auf die Art des aufzurufenden Vorgangs zu geben. In diesem Abschnitt werden die verschiedenen Mapping-Vorlagen für die unterstützten Lambda-Operationen beschrieben.

Mapping-Vorlage anfordern

Die Lambda-Anforderungszuordnungsvorlage verarbeitet Felder, die sich auf Ihre Lambda-Funktion beziehen:

```
{
  "version": string,
  "operation": Invoke|BatchInvoke,
  "payload": any type,
  "invocationType": RequestResponse|Event
}
```

Dies ist die JSON-Schemadarstellung der Lambda-Anforderungszuordnungsvorlage, wenn sie aufgelöst wurde:

```
{
  "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
}
```

Hier ist ein Beispiel, das eine `invoke` Operation verwendet, deren Nutzdaten das `getPost` Feld aus einem GraphQL-Schema zusammen mit den Argumenten aus dem Kontext sind:

```
{
  "version": "2018-05-29",
```

```
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": $util.toJson($context.arguments)
  }
}
```

Das gesamte Mapping-Dokument wird als Eingabe für Ihre Lambda-Funktion übergeben, sodass das vorherige Beispiel nun wie folgt aussieht:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": {
      "id": "postId1"
    }
  }
}
```

Version

Das ist allen Vorlagen für die Anforderungszuweisung gemeinsam und `version` definiert die Version, die die Vorlage verwendet. Der `version` ist erforderlich und ist ein statischer Wert:

```
"version": "2018-05-29"
```

Operation

Mit der Lambda-Datenquelle können Sie zwei Operationen im `operation` Feld definieren: `Invoke` und`BatchInvoke`. Die `Invoke` Operation teilt mit AWS AppSync , dass Sie Ihre Lambda-Funktion für jeden GraphQL-Feldresolver aufrufen sollen. `BatchInvoke`weist AWS AppSync an, Anfragen für das aktuelle GraphQL-Feld zu stapeln. Das Feld `operation` ist ein Pflichtfeld.

Denn die Vorlage für `Invoke` die Zuordnung aufgelöster Anfragen entspricht der Eingabe-Payload der Lambda-Funktion. Lassen Sie uns das obige Beispiel ändern:

```
{
  "version": "2018-05-29",
```

```
  "operation": "Invoke",
    "payload": {
      "arguments": $util.toJson($context.arguments)
    }
}
```

Dies wird gelöst und an die Lambda-Funktion übergeben, die etwa so aussehen könnte:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
    "payload": {
      "arguments": {
        "id": "postId1"
      }
    }
}
```

Denn `BatchInvoke` die Mapping-Vorlage wird auf jeden Field Resolver im Batch angewendet. Führt aus Gründen der Übersichtlichkeit alle aufgelösten `payload` Werte der Mapping-Vorlage zu einer Liste unter einem einzigen Objekt zusammen, das der Mapping-Vorlage entspricht. AWS AppSync Die folgende Vorlage zeigt die Zusammenfassung:

```
{
  "version": "2018-05-29",
  "operation": "BatchInvoke",
  "payload": $util.toJson($context)
}
```

Diese Vorlage ist in dem folgenden Zuweisungsdokument aufgelöst:

```
{
  "version": "2018-05-29",
  "operation": "BatchInvoke",
  "payload": [
    {...}, // context for batch item 1
    {...}, // context for batch item 2
    {...}  // context for batch item 3
  ]
}
```

Jedes Element der `payload` Liste entspricht einem einzelnen Batch-Element. Es wird auch erwartet, dass die Lambda-Funktion eine listenförmige Antwort zurückgibt, die der Reihenfolge der in der Anfrage gesendeten Elemente entspricht:

```
[
  { "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
]
```

Nutzlast

Das `payload` Feld ist ein Container, der verwendet wird, um wohlgeformtes JSON an die Lambda-Funktion zu übergeben. Wenn das `operation` Feld auf gesetzt ist`BatchInvoke`, AWS AppSync werden die vorhandenen `payload` Werte in eine Liste zusammengefasst. Das Feld `payload` ist optional.

Aufruftyp

Mit der Lambda-Datenquelle können Sie zwei Aufruftypen definieren: `RequestResponse` und. `Event` Die Aufruftypen sind synonym mit den in der Lambda-API definierten Aufruftypen. Mit dem `RequestResponse` AWS AppSync Aufruftyp können Sie Ihre Lambda-Funktion synchron aufrufen, um auf eine Antwort zu warten. Der `Event` Aufruf ermöglicht es Ihnen, Ihre Lambda-Funktion asynchron aufzurufen. Weitere Informationen darüber, wie Lambda Anfragen vom `Event` Aufruftyp verarbeitet, finden Sie unter Asynchroner Aufruf. Das Feld `invocationType` ist optional. Wenn dieses Feld nicht in der Anfrage enthalten ist, AWS AppSync wird standardmäßig der Aufruftyp verwendet. `RequestResponse`

Für jedes `invocationType` Feld entspricht die aufgelöste Anfrage der Eingabe-Payload der Lambda-Funktion. Lassen Sie uns das obige Beispiel ändern:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "invocationType": "Event"
  "payload": {
    "arguments": $util.toJson($context.arguments)
```

```
  }
}
```

Dies wird gelöst und an die Lambda-Funktion übergeben, die etwa so aussehen könnte:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "invocationType": "Event",
  "payload": {
    "arguments": {
      "id": "postId1"
    }
  }
}
```

Wenn der `BatchInvoke` Vorgang in Verbindung mit dem `Event` Aufruftypfeld verwendet wird, AWS AppSync führt er den Feldauflöser auf die oben beschriebene Weise zusammen, und die Anforderung wird als asynchrones Ereignis an Ihre Lambda-Funktion übergeben, `payload` wobei es sich um eine Werteliste handelt. Wir empfehlen, das Resolver-Caching für Resolver vom `Event` Aufruftyp zu deaktivieren, da diese bei einem Cache-Treffer nicht an Lambda gesendet würden.

Vorlage für die Zuordnung von Antworten

Wie bei anderen Datenquellen sendet Ihre Lambda-Funktion eine Antwort darauf AWS AppSync , die in einen GraphQL-Typ konvertiert werden muss.

Das Ergebnis der Lambda-Funktion wird für das `context` Objekt festgelegt, das über die Velocity Template Language (VTL) `$context.result` -Eigenschaft verfügbar ist.

Wenn die Form Ihrer Lambda-Funktionsantwort exakt der Form des GraphQL-Formats entspricht, können Sie die Antwort unter Verwendung der folgenden Zuweisungsvorlage für Antworten weiterleiten:

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

Es gibt keine erforderlichen Felder oder Formeinschränkungen, die auf die Zuweisungsvorlage für Antworten zutreffen. Allerdings ist GraphQL stark typisiert. Deshalb muss die Zuweisungsvorlage, auf die der Resolver angewendet wurde, dem erwarteten GraphQL-Format entsprechen.

Batch-Antwort mit Lambda-Funktion

Wenn das `operation`-Feld auf `BatchInvoke` festgelegt ist, erwartet AWS AppSync eine Liste mit Elementen aus der Lambda-Funktion. Damit jedes Ergebnis AWS AppSync dem ursprünglichen Anforderungselement zugeordnet werden kann, muss die Antwortliste in Größe und Reihenfolge übereinstimmen. Es ist zulässig, `null` Elemente in der Antwortliste zu haben; sie `$ctx.result` wird entsprechend auf Null gesetzt.

Direkte Lambda-Resolver

Wenn Sie die Verwendung von Mapping-Vorlagen vollständig umgehen möchten, AWS AppSync können Sie eine Standardnutzlast für Ihre Lambda-Funktion und eine Standard-Lambda-Funktionsantwort für einen GraphQL-Typ bereitstellen. Sie können wählen, ob Sie eine Anforderungsvorlage, eine Antwortvorlage oder beides bereitstellen und diese entsprechend behandeln möchten. AWS AppSync

Vorlage für die direkte Zuordnung von Lambda-Anfragen

Wenn die Vorlage für die Anforderungszuweisung nicht bereitgestellt AWS AppSync wird, wird das `Context` Objekt als `Invoke` Operation direkt an Ihre Lambda-Funktion gesendet. Weitere Informationen über die Struktur des `Context`-Objekts finden Sie unter AWS AppSync Kontextreferenz für Resolver-Mapping-Vorlagen.

Vorlage für die direkte Zuordnung von Lambda-Antworten

Wenn die Antwortzuordnungsvorlage nicht bereitgestellt wird, AWS AppSync führt sie beim Empfang der Antwort Ihrer Lambda-Funktion eine von zwei Aktionen aus. Wenn Sie keine Vorlage für die Anforderungszuweisung bereitgestellt haben oder wenn Sie mit der Version eine Vorlage für die Anforderungszuweisung bereitgestellt haben`2018-05-29`, entspricht die Antwort der folgenden Antwortzuordnungsvorlage:

```
#if($ctx.error)
     $util.error($ctx.error.message, $ctx.error.type, $ctx.result)
 #end
 $util.toJson($ctx.result)
```

Wenn Sie mit der Version eine Vorlage bereitgestellt haben`2017-02-28`, funktioniert die Antwortlogik genauso wie die folgende Vorlage für die Antwortzuweisung:

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

Oberflächlich betrachtet funktioniert die Umgehung von Zuordnungsvorlagen ähnlich wie die Verwendung bestimmter Zuordnungsvorlagen, wie in den vorherigen Beispielen gezeigt. Hinter den Kulissen wird die Auswertung der Mapping-Vorlagen jedoch völlig umgangen. Da der Schritt zur Template-Evaluierung umgangen wird, kann es bei Anwendungen in einigen Szenarien zu weniger Overhead und Latenz während der Antwort kommen als bei einer Lambda-Funktion mit einer Antwortzuordnungsvorlage, die evaluiert werden muss.

Benutzerdefinierte Fehlerbehandlung in Direct Lambda Resolver-Antworten

Sie können Fehlerantworten von Lambda-Funktionen, die Direct Lambda Resolver aufrufen, anpassen, indem Sie eine benutzerdefinierte Ausnahme auslösen. Das folgende Beispiel zeigt, wie eine benutzerdefinierte Ausnahme erstellt wird mit: JavaScript

```
class CustomException extends Error {
  constructor(message) {
    super(message);
    this.name = "CustomException";
  }
}

throw new CustomException("Custom message");
```

Wenn Ausnahmen ausgelöst werden, `errorMessage` sind `errorType` und jeweils das `name` und `message` des benutzerdefinierten Fehlers, der ausgelöst wird.

Wenn `errorType` ja`UnauthorizedException`, wird die Standardnachricht (`"You are not authorized to make this call."`) anstelle einer benutzerdefinierten Meldung AWS AppSync zurückgegeben.

Der folgende Ausschnitt ist ein Beispiel für eine GraphQL-Antwort, die eine benutzerdefinierte Antwort demonstriert: `errorType`

```
{
  "data": {
    "query": null
  },
  "errors": [
    {
      "path": [
```

```
        "query"
      ],
      "data": null,
      "errorType": "CustomException",
      "errorInfo": null,
      "locations": [
        {
          "line": 5,
          "column": 10,
          "sourceName": null
        }
      ],
      "message": "Custom Message"
    }
  ]
}
```

Direkte Lambda-Resolver: Batching aktiviert

Sie können Batching für Ihren Direct Lambda Resolver aktivieren, indem Sie den `maxBatchSize` auf Ihrem Resolver konfigurieren. Wenn auf einen Wert gesetzt `maxBatchSize` ist, der größer ist als `0` für einen Direct Lambda-Resolver, AWS AppSync sendet Anfragen stapelweise an Ihre Lambda-Funktion in Größen bis zu. `maxBatchSize`

Wenn `maxBatchSize` Sie einen Direct Lambda-Resolver `0` auf on setzen, wird die Batchverarbeitung ausgeschaltet.

Weitere Informationen zur Funktionsweise der Batchverarbeitung mit Lambda-Resolvern finden Sie unter. Anwendungsfall für Fortgeschrittene: Batching

Vorlage für die Zuordnung anfordern

Wenn Batching aktiviert ist und die Vorlage für die Anforderungszuweisung nicht bereitgestellt wird, AWS AppSync sendet eine Liste von `Context` Objekten als `BatchInvoke` Vorgang direkt an Ihre Lambda-Funktion.

Vorlage für die Zuordnung von Antworten

Wenn Batching aktiviert ist und die Antwortzuordnungsvorlage nicht bereitgestellt wird, entspricht die Antwortlogik der folgenden Antwortzuordnungsvorlage:

```
#if( $context.result && $context.result.errorMessage )
      $utils.error($context.result.errorMessage, $context.result.errorType,
```

```
      $context.result.data)
#else
      $utils.toJson($context.result.data)
#end
```

Die Lambda-Funktion muss eine Ergebnisliste in derselben Reihenfolge wie die Liste der gesendeten `Context` Objekte zurückgeben. Sie können einzelne Fehler zurückgeben, indem Sie ein `errorMessage` und `errorType` für ein bestimmtes Ergebnis angeben. Jedes Ergebnis in der Liste hat das folgende Format:

```
{
   "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**
>
> Andere Felder im Ergebnisobjekt werden derzeit ignoriert.

Umgang mit Fehlern von Lambda

Sie können für alle Ergebnisse einen Fehler zurückgeben, indem Sie eine Ausnahme oder einen Fehler in Ihrer Lambda-Funktion auslösen. Wenn die Payload-Anfrage oder die Antwortgröße für Ihre Batch-Anfrage zu groß ist, gibt Lambda einen Fehler zurück. In diesem Fall sollten Sie erwägen, Ihre `maxBatchSize` oder die Größe der Antwortnutzlast zu reduzieren.

Hinweise zur Behandlung einzelner Fehler finden Sie unterRückgabe einzelner Fehler.

Beispiele für Lambda-Funktionen

Mithilfe des folgenden Schemas können Sie einen Direct Lambda Resolver für den `Post.relatedPosts` Field Resolver erstellen und die Batchverarbeitung aktivieren, indem Sie die obigen Einstellungen vornehmen: `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]
}
```

In der folgenden Abfrage wird die Lambda-Funktion mit Batches von Auflösungsanforderungen aufgerufen: `relatedPosts`

```
query getAllPosts {
  allPosts {
    id
    relatedPosts {
      id
    }
  }
}
```

Im Folgenden finden Sie eine einfache Implementierung einer Lambda-Funktion:

```
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' }
    }
  })
}
```

AWS AppSync Referenz zur Resolver-Mapping-Vorlage für EventBridge

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Die mit der EventBridge Datenquelle verwendete AWS AppSync Resolver-Mapping-Vorlage ermöglicht es Ihnen, benutzerdefinierte Ereignisse an den EventBridge Amazon-Bus zu senden.

Vorlage für die Zuordnung anfordern

Mit der Vorlage für die `PutEvents` Anforderungszuweisung können Sie mehrere benutzerdefinierte Ereignisse an einen EventBridge Event-Bus senden. Das -Zuweisungsdokument weist die folgende Struktur auf:

```
{
```

```
    "version" : "2018-05-29",
    "operation" : "PutEvents",
    "events" : [{}]
}
```

Im Folgenden finden Sie ein Beispiel für eine Vorlage zur Anforderungszuweisung für 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"
    }

    ]
}
```

Vorlage für die Zuordnung von Antworten

Wenn der `PutEvents` Vorgang erfolgreich ist, EventBridge ist die Antwort von in der folgenden Datei enthalten`$ctx.result`:

```
#if($ctx.error)
  $util.error($ctx.error.message, $ctx.error.type, $ctx.result)
#end
  $util.toJson($ctx.result)
```

Fehler, die bei der Ausführung von `PutEvents` Vorgängen wie `InternalExceptions` oder auftretenTimeouts, werden in angezeigt`$ctx.error`. Eine Liste der EventBridge häufigsten Fehler finden Sie in der Referenz zu EventBridge häufigen Fehlern.

`result`Sie werden das folgende Format haben:

```
{
    "Entries" [
        {
            "ErrorCode" : String,
            "ErrorMessage" : String,
            "EventId" : String
        }
    ],
    "FailedEntryCount" : number
}
```

- Einträge

  Die Ergebnisse des aufgenommenen Ereignisses, sowohl erfolgreich als auch erfolglos. Wenn die Aufnahme erfolgreich war, enthält der Eintrag das`EventID`. Andernfalls können Sie das `ErrorCode` und verwenden, `ErrorMessage` um das Problem mit dem Eintrag zu identifizieren.

  Für jeden Datensatz entspricht der Index des Antwortelements dem Index im Anforderungsarray.
- FailedEntryCount

  Die Anzahl der fehlgeschlagenen Einträge. Dieser Wert wird als Ganzzahl dargestellt.

Weitere Hinweise zur Antwort von `PutEvents` finden Sie unter PutEvents.

Beispiel für Antwort 1

Das folgende Beispiel zeigt einen `PutEvents` Vorgang mit zwei erfolgreichen Ereignissen:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
```

```
        }
    ],
    "FailedEntryCount" : 0
}
```

Beispiel für eine Antwort 2

Das folgende Beispiel zeigt einen `PutEvents` Vorgang mit drei Ereignissen, zwei Erfolgen und einem Fehlschlag:

```
{
    "Entries" : [
        {
            "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
        },
        {
            "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
        },
        {
            "ErrorCode" : "SampleErrorCode",
            "ErrorMessage" : "Sample Error Message"
        }
    ],
    "FailedEntryCount" : 1
}
```

`PutEvents`-Felder

`PutEvents`enthält die folgenden Felder für Zuordnungsvorlagen:

- Version

  Das `version` Feld ist allen Vorlagen für die Zuordnung von Anfragen gemeinsam und definiert die Version, die die Vorlage verwendet. Dies ist ein Pflichtfeld. Der Wert `2018-05-29` ist die einzige Version, die für die EventBridge Zuordnungsvorlagen unterstützt wird.

- Operation

  Die einzige unterstützte Operation ist`PutEvents`. Mit diesem Vorgang können Sie Ihrem Event-Bus benutzerdefinierte Ereignisse hinzufügen.

- --Ereignisse

Eine Reihe von Ereignissen, die dem Event-Bus hinzugefügt werden. Dieses Array sollte eine Zuordnung von 1 bis 10 Elementen haben.

Das `Event` Objekt ist ein gültiges JSON-Objekt mit den folgenden Feldern:

- `"source"`: Eine Zeichenfolge, die die Quelle des Ereignisses definiert.
- `"detail"`: Ein JSON-Objekt, mit dem Sie Informationen über das Ereignis anhängen können. Dieses Feld kann eine leere Map (`{ }`) sein.
- `"detailType`: Eine Zeichenfolge, die den Ereignistyp identifiziert.
- `"resources"`: Ein JSON-Array von Zeichenketten, das die an dem Ereignis beteiligten Ressourcen identifiziert. Dieses Feld kann ein leeres Array sein.
- `"time"`: Der als Zeichenfolge bereitgestellte Zeitstempel des Ereignisses. Dies sollte dem RFC3339Zeitstempelformat entsprechen.

Die folgenden Ausschnitte sind einige Beispiele für gültige Objekte: `Event`

Beispiel 1

```
{
    "source" : "source1",
    "detail" : {
        "key1" : [1,2,3,4],
        "key2" : "strval"
    },
    "detailType" : "sampleDetailType",
    "resources" : ["Resouce1", "Resource2"],
    "time" : "2022-01-10T05:00:10Z"
}
```

Beispiel 2

```
{
    "source" : "source1",
    "detail" : {},
    "detailType" : "sampleDetailType"
}
```

Beispiel 3

```
{
    "source" : "source1",
    "detail" : {
        "key1" : 1200
    },
    "detailType" : "sampleDetailType",
    "resources" : []
}
```

AWS AppSyncReferenz zur Resolver-Mapping-Vorlage für die **None** Datenquelle

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Die AWS AppSync Resolver-Mapping-Vorlage, die mit der Datenquelle vom Typ None verwendet wird, ermöglicht es Ihnen, Anfragen für lokale Operationen zu gestalten. AWS AppSync

Vorlage für die Anforderungszuordnung

Die Zuweisungsvorlage ist einfach und ermöglicht Ihnen, so viele Kontextinformationen wie möglich über das `payload`-Feld zu übermitteln.

```
{
   "version": string,
   "payload": any type
}
```

Hier finden Sie die JSON-Schema-Darstellung der Zuweisungsvorlage für Anforderungen, nachdem sie einmal aufgelöst wurde:

```
{
    "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
}
```

Hier ist ein Beispiel, bei dem die Feldargumente über die VTL-Kontexteigenschaft `$context.arguments` übergeben werden:

```
{
    "version": "2018-05-29",
    "payload": $util.toJson($context.arguments)
}
```

Der Wert des `payload`-Felds wird an die Antwortzuweisungsvorlage weitergeleitet und ist in der VTL-Kontexteigenschaft (`$context.result`) zu finden.

Dies ist ein Beispiel für den interpolierten Wert des `payload`-Felds:

```
{
    "id": "postId1"
}
```

Version

Das `version` Feld ist allen Vorlagen zur Anforderungszuweisung gemeinsam und definiert die Version, die von der Vorlage verwendet wird.

Das Feld `version` ist ein Pflichtfeld.

Beispiel:

```
"version": "2018-05-29"
```

Nutzlast

Das `payload`-Feld ist ein Container, den Sie verwenden können, um ein beliebiges gültiges JSON-Format auf die Zuweisungsvorlage für Anforderungen zu übertragen.

Das Feld `payload` ist optional.

Vorlage für die Zuordnung von Antworten

Da es keine Datenquelle gibt, wird der Wert des `payload`-Felds an die Antwortzuweisungsvorlage weitergeleitet und auf das `context`-Objekt festgelegt, das über die VTL-Eigenschaft `$context.result` verfügbar ist.

Wenn die Form Ihres `payload`-Feldwerts exakt der Form des GraphQL-Formats entspricht, können Sie die Antwort unter Verwendung der folgenden Antwortzuweisungsvorlage weiterleiten:

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

Es gibt keine erforderlichen Felder oder Formeinschränkungen, die auf die Zuweisungsvorlage für Antworten zutreffen. Allerdings ist GraphQL stark typisiert. Deshalb muss die Zuweisungsvorlage, auf die der Resolver angewendet wurde, dem erwarteten GraphQL-Format entsprechen.

AWS AppSync Referenz zur Resolver-Mapping-Vorlage für HTTP

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Mit den AWS AppSync HTTP-Resolver-Mapping-Vorlagen können Sie Anfragen von AWS AppSync einem beliebigen HTTP-Endpunkt und Antworten von Ihrem HTTP-Endpunkt zurück an diesen senden. AWS AppSync Mithilfe von Zuordnungsvorlagen können Sie AWS AppSync Hinweise zur Art des aufzurufenden Vorgangs geben. Dieser Abschnitt beschreibt die verschiedenen Zuweisungsvorlagen für unterstützte HTTP-Resolver.

Zuordnungsvorlage anfordern

```
{
    "version": "2018-05-29",
    "method": "PUT|POST|GET|DELETE|PATCH",
    "params": {
        "query": Map,
        "headers": Map,
        "body": any
    },
    "resourcePath": string
}
```

Nachdem die Zuweisungsvorlage für HTTP-Anforderungen aufgelöst wurde, sieht die JSON-Schemadarstellung der Zuweisungsvorlage für Anforderungen wie folgt aus:

```
{
    "$id": "https://aws.amazon.com/appsync/request-mapping-template.json",
    "type": "object",
    "properties": {
        "version": {
        "$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"
    ]
```

```
}
```

Es folgt ein Beispiel für eine HTTP POST-Anforderung mit einem `text/plain`-Rumpf:

```
{
    "version": "2018-05-29",
    "method": "POST",
    "params": {
        "headers":{
        "Content-Type":"text/plain"
        },
        "body":"this is an example of text body"
    },
    "resourcePath": "/"
}
```

Version

> **Note**
>
> Dies gilt nur für die Vorlage für die Anforderungszuweisung.

Definiert die Version, die von der Vorlage verwendet wird. `version` ist allen Anforderungszuweisungsvorlagen gemeinsam und erforderlich.

```
"version": "2018-05-29"
```

Methode

> **Note**
>
> Dies gilt nur für die Vorlage für die Anforderungszuweisung.

HTTP-Methode oder Verb (GET, POST, PUT, PATCH oder DELETE), das AWS AppSync an den HTTP-Endpunkt sendet.

```
"method": "PUT"
```

ResourcePath

> **Note**
>
> Dies gilt nur für die Vorlage für die Anforderungszuweisung.

Der Ressourcenpfad, auf den Sie zugreifen möchten. Zusammen mit dem Endpunkt in der HTTP-Datenquelle bildet der Ressourcenpfad die URL, an die der AWS AppSync Dienst eine Anfrage stellt.

```
"resourcePath": "/v1/users"
```

Wenn die Zuweisungsvorlage ausgewertet wird, wird dieser Pfad als Teil der HTTP-Anforderung gesendet, einschließlich des HTTP-Endpunkts. Das vorherige Beispiel kann dann beispielsweise so aussehen:

```
PUT <endpoint>/v1/users
```

Params-Felder

> **Note**
>
> Dies gilt nur für die Vorlage für die Zuordnung von Anfragen.

Wird verwendet, um anzugeben, welche Aktion die Suche durchführt, meistens indem der Abfragewert im Body festgelegt wird. Es gibt jedoch einige andere Funktionen, die konfiguriert werden können, wie z. B. die Formatierung von Antworten.

Header

Die Header-Informationen, wie beispielsweise Schlüssel-Wert-Paare. Der Schlüssel und der Wert müssen beide Strings sein.

Zum Beispiel:

```
"headers" : {
    "Content-Type" : "application/json"
}
```

Derzeit werden folgende `Content-Type`-Header unterstützt:

```
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
```

Hinweis: Sie können die folgenden HTTP-Header nicht einstellen:

```
HOST
CONNECTION
USER-AGENT
EXPECTATION
TRANSFER_ENCODING
CONTENT_LENGTH
```

query

Schlüssel-Wert-Paare, die allgemeine Optionen angeben, z. B. Codeformatierung für JSON-Antworten. Der Schlüssel und der Wert müssen beide Strings sein. Das folgende Beispiel zeigt, wie Sie eine Abfragezeichenfolge als `?type=json` senden können:

```
"query" : {
    "type" : "json"
}
```

body

Der Rumpf enthält den HTTP-Anforderungstext, den Sie festgelegen. Der Anforderungstext ist immer eine UTF-8-codierte Zeichenfolge, es sei denn, der Inhaltstyp gibt den Zeichensatz an.

```
"body":"body string"
```

Antwort

Sehen Sie ein Beispiel hier.

Von vier HTTPS-Endpunkten anerkannte AWS AppSync Zertifizierungsstellen (CA)

> **Note**
>
> Let's Encrypt wird über die Zertifikate identrust und isrgrootx1 akzeptiert. Wenn Sie Let's Encrypt verwenden, sind keine Maßnahmen Ihrerseits erforderlich.

Derzeit werden selbstsignierte Zertifikate nicht von HTTP-Resolvern unterstützt, wenn HTTPS verwendet wird. AWS AppSync erkennt bei der Auflösung von SSL/TLS-Zertifikaten für HTTPS die folgenden Zertifizierungsstellen an:

Bekannte Stammzertifikate in AWS AppSync

Name	Date (Datum)	SHA1 Fingerabdruck
`digicertassuredidrootca`	21. April 2018	`05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43`
`trustcenterclass2caii`	21. April 2018	`AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E`
`thawtepremiumserverca`	21. April 2018	`E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66`
`cia-crt-g3-02-ca`	23. November 2016	`96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09`
`swisssignplatinumg2ca`	21. April 2018	`56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66`
`swisssignsilverg2ca`	21. April 2018	`9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB`
`thawteserverca`	21. April 2018	`9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79`

Name	Date (Datum)	SHA1 Fingerabdruck
equifaxsecureebusinessca1	21. April 2018	AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
securetrustca	21. April 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
utnuserfirstclientauthemailca	21. April 2018	B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
thawtepersonalfreemailca	21. April 2018	E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
affirmtrustnetworkingca	21. April 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
entrustevca	21. April 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
utnuserfirsthardwareca	21. April 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
certumca	21. April 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
addtrustclass1ca	21. April 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
entrustrootcag2	21. April 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
equifaxsecureca	21. April 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
quovadisrootca3	21. April 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85

Name	Date (Datum)	SHA1 Fingerabdruck
quovadisrootca2	21. April 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
digicertglobalrootg2	21. April 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
digicerthighassuranceevrootca	21. April 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
secomvalicertclass1ca	21. April 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
equifaxsecureglobalebusinessca1	21. April 2018	3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36
geotrustuniversalca	21. April 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
deprecateditsecca	27. Januar 2012	12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
verisignclass3ca	21. April 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
thawteprimaryrootcag3	21. April 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
thawteprimaryrootcag2	21. April 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
deutschetelekomrootca2	21. April 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
buypassclass3ca	21. April 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57

Name	Date (Datum)	SHA1 Fingerabdruck
utnuserfirstobject ca	21. April 2018	E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B: AC:1D:81:D8:38:5E:2D:46
geotrustprimaryca	21. April 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2: 10:0D:D6:02:90:37:F0:96
buypassclass2ca	21. April 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6: C7:6B:EB:C6:0B:12:40:99
baltimorecodesigni ngca	21. April 2018	30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD: 49:27:08:7C:60:56:7B:0D
verisignclass1ca	21. April 2018	CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45: 3E:64:09:EA:E8:7D:60:F1
baltimorecybertrus tca	21. April 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88: 2C:78:DB:28:52:CA:E4:74
starfieldclass2ca	21. April 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20: 14:C3:D0:E3:37:0E:B5:8A
camerfirmachambers commerceca	21. April 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0: DB:72:2E:31:30:61:F0:B1
ttelesecglobalroot class3ca	21. April 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70: 19:9D:2A:BE:11:E3:81:D1
verisignclass3g5ca	21. April 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD: 56:BE:3D:9B:67:44:A5:E5
ttelesecglobalroot class2ca	21. April 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62: 32:17:65:CF:17:D8:94:E9
trustcenteruniversalcai	21. April 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C: CE:BB:9D:D9:4F:4E:39:F3

Name	Date (Datum)	SHA1 Fingerabdruck
verisignclass3g4ca	21. April 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
verisignclass3g3ca	21. April 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
xrampglobalca	21. April 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
amzninternalrootca	12. Dezember 2008	A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
certplusclass3pprimaryca	21. April 2018	21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
certumtrustednetworkca	21. April 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
verisignclass3g2ca	21. April 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
globalsignr3ca	21. April 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
utndatacorpsgcca	21. April 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
secomscrootca2	21. April 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
gtecybertrustgloballca	21. April 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
secomscrootca1	21. April 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7

Name	Date (Datum)	SHA1 Fingerabdruck
affirmtrustcommercialca	21. April 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
trustcenterclass4caii	21. April 2018	A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
verisignuniversalrootca	21. April 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
globalsignr2ca	21. April 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
certplusclass2primaryca	21. April 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
digicertglobalrootca	21. April 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
globalsignca	21. April 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
thawteprimaryrootca	21. April 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
starfieldrootg2ca	21. April 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
geotrustglobalca	21. April 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
soneraclass2ca	21. April 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
verisigntsaca	21. April 2018	20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01

Name	Date (Datum)	SHA1 Fingerabdruck
soneraclass1ca	21. April 2018	07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
quovadisrootca	21. April 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
affirmtrustpremiumecccca	21. April 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
starfieldservicesrootg2ca	21. April 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
valicertclass2ca	21. April 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
comodoaaaca	21. April 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
aolrootca2	21. April 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
keynectisrootca	21. April 2018	9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
addtrustqualifiedca	21. April 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
aolrootca1	21. April 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
verisignclass2g3ca	21. April 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
addtrustexternalca	21. April 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68

Name	Date (Datum)	SHA1 Fingerabdruck
verisignclass2g2ca	21. April 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
geotrustprimarycag3	21. April 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycag2	21. April 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
swisssigngoldg2ca	21. April 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
entrust2048ca	21. April 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
chunghwaepkirootca	21. April 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
camerfirmachambersignca	21. April 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
camerfirmachambersca	21. April 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
godaddyclass2ca	21. April 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
affirmtrustpremiumca	21. April 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
verisignclass1g3ca	21. April 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
secomevrootca1	21. April 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D

Name	Date (Datum)	SHA1 Fingerabdruck
verisignclass1g2ca	21. April 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
amzninternalinfoseccag3	27. Februar 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. November 2016	2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
godaddyrootg2ca	21. April 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
digicertassuredidrootca	21. April 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
microseceszignorootca2009	21. April 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
affirmtrustcommercial	21. April 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
comodoecccertificationauthority	21. April 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
cadisigrootr2	21. April 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
swisssignsilvercag2	21. April 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
securetrustca	21. April 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11

Name	Date (Datum)	SHA1 Fingerabdruck
cadisigrootr1	21. April 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
accvraiz1	21. April 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
entrustrootcertificationauthority	21. April 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
camerfirmaglobalchambersignroot	21. April 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
dstacescax6	21. April 2018	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
identrustpublicsectorrootca1	21. April 2018	BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
starfieldrootcertificateauthorityg2	21. April 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
secureglobalca	21. April 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
eecertificationcentrerootca	21. April 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
opentrustrootcag3	21. April 2018	6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6
teliasonerarootcav1	21. April 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37

Name	Date (Datum)	SHA1 Fingerabdruck
autoridaddecertificacionfirmaprofesionalcifa62634068	21. April 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
opentrustrootcag2	21. April 2018	79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
opentrustrootcag1	21. April 2018	79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E
globalsigneccrootcar5	21. April 2018	1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
globalsigneccrootcar4	21. April 2018	69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
izenpecom	21. April 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
turktrustelektroniksertifikahizmetsaglayicisih5	21. April 2018	C4:18:F6:4D:46:D1:DF:00:3D:27:30:13:72:43:A9:12:11:C6:75:FB
gdcatrustauthr5root	21. April 2018	0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
dtrustrootclass3ca22009	21. April 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
quovadisrootca3	21. April 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
quovadisrootca2	21. April 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7

Name	Date (Datum)	SHA1 Fingerabdruck
geotrustprimarycertificationauthorityg3	21. April 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycertificationauthorityg2	21. April 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
oistewisekeyglobalrootgbca	21. April 2018	0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
addtrustexternalroot	21. April 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
chambersofcommerceroot2008	21. April 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
digicertglobalrootg3	21. April 2018	7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E
comodoaaaservicesroot	21. April 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
digicertglobalrootg2	21. April 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
certinomisrootca	21. April 2018	9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C:01:B9:32:C5:34:E7:88:A8
oistewisekeyglobalrootgaca	21. April 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
dstrootcax3	21. April 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13

Name	Date (Datum)	SHA1 Fingerabdruck
certigna	21. April 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
digicerthighassuranceevrootca	21. April 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
soneraclass2rootca	21. April 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
trustcorrootcertca2	21. April 2018	B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
usertrustrsacertificationauthority	21. April 2018	2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
trustcorrootcertca1	21. April 2018	FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
geotrustuniversalca	21. April 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
certsignrootca	21. April 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
amazonrootca4	21. April 2018	F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
amazonrootca3	21. April 2018	0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
amazonrootca2	21. April 2018	5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
verisignuniversalrootcertificationauthority	21. April 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54

Name	Date (Datum)	SHA1 Fingerabdruck
amazonrootca1	21. April 2018	8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
networksolutionscertificate authority	21. April 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
thawteprimaryrootcag3	21. April 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
affirmtrustnetworking	21. April 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
thawteprimaryrootcag2	21. April 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
trustcoreca1	21. April 2018	58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
deutschetelekomrootca2	21. April 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
godaddyrootcertificateauthorityg2	21. April 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
entrustrootcertificationauthorityec1	21. April 2018	20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
szafirrootca2	21. April 2018	E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
tubitakkamusmsslkoksertifikasisurum1	21. April 2018	31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA

Name	Date (Datum)	SHA1 Fingerabdruck
buypassclass3rootca	21. April 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
comodorsacertificationauthority	21. April 2018	AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
netlockaranyclassgoldfotanusitvany	21. April 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
securitycommunicationrootca2	21. April 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
dtrustrootclass3ca2ev2009	21. April 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
starfieldclass2ca	21. April 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
pscprocert	21. April 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
actalisauthenticationrootca	21. April 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
staatdernederlandenrootcag3	21. April 2018	D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC
cfcaevroot	21. April 2018	E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
digicerttrustedrootg4	21. April 2018	DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F:C8:3A:4D:7D:77:5D:05:E4
staatdernederlandenrootcag2	21. April 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16

Name	Date (Datum)	SHA1 Fingerabdruck
securitycommunicationevrootca1	21. April 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
globalsignrootcar3	21. April 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
globalsignrootcar2	21. April 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
certumtrustednetworkca2	21. April 2018	D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
acraizfnmtrcm	21. April 2018	EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
hellenicacademicandresearchinstitutionseccrootca2015	21. April 2018	9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
certplusrootcag2	21. April 2018	4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A
twcarootcertificationauthority	21. April 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
twcaglobalrootca	21. April 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
certplusrootcag1	21. April 2018	22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
geotrustuniversalca2	21. April 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79

Name	Date (Datum)	SHA1 Fingerabdruck
baltimorecybertrustroot	21. April 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
buypassclass2rootca	21. April 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
certumtrustednetworkca	21. April 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
digicertassuredidrootg3	21. April 2018	F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
digicertassuredidrootg2	21. April 2018	A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F
isrgrootx1	21. April 2018	CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
entrustnetpremium2048secureserverca	21. April 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
certplusclass2primaryca	21. April 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
digicertglobalrootca	21. April 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
entrustrootcertificationauthorityg2	21. April 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
starfieldservicesrootcertificateauthorityg2	21. April 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
thawteprimaryrootca	21. April 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81

Name	Date (Datum)	SHA1 Fingerabdruck
atostrustedroot2011	21. April 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
geotrustglobalca	21. April 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
luxtrustglobalroot2	21. April 2018	1E:0E:56:19:0A:D1:8B:25:98:B2:04:44:FF:66:8A:04:17:99:5F:3F
etugracertificationauthority	21. April 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
visaecommerceroot	21. April 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
quovadisrootca	21. April 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
identrustcommercialrootca1	21. April 2018	DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
staatdernederlandenevrootca	21. April 2018	76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
ttelesecglobalrootclass3	21. April 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
ttelesecglobalrootclass2	21. April 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
comodocertificationauthority	21. April 2018	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
securitycommunicationrootca	21. April 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7

Name	Date (Datum)	SHA1 Fingerabdruck
quovadisrootca3g3	21. April 2018	48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
xrampglobalcaroot	21. April 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
securesignrootca11	21. April 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
affirmtrustpremium	21. April 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
globalsignrootca	21. April 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
swisssigngoldcag2	21. April 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
quovadisrootca2g3	21. April 2018	09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
affirmtrustpremiumecc	21. April 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
geotrustprimarycertificationauthority	21. April 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
quovadisrootca1g3	21. April 2018	1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67
hongkongpostrootca1	21. April 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
usertrustecccertificationauthority	21. April 2018	D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0

Name	Date (Datum)	SHA1 Fingerabdruck
cybertrustglobalroot	21. April 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
godaddyclass2ca	21. April 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
hellenicacademicandresearchinstitutionsrootca2015	21. April 2018	01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
ecacc	21. April 2018	28:90:3A:63:5B:52:80:FA:E6:77:4C:0B:6D:A7:D6:BA:A6:4A:F2:E8
hellenicacademicandresearchinstitutionsrootca2011	21. April 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
verisignclass3publicprimarycertificationauthorityg5	21. April 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
verisignclass3publicprimarycertificationauthorityg4	21. April 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
verisignclass3publicprimarycertificationauthorityg3	21. April 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6

Name	Date (Datum)	SHA1 Fingerabdruck
trustisfpsrootca	21. April 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
epkirootcertificationauthority	21. April 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
globalchambersignroot2008	21. April 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
camerfirmachambersofcommerceroot	21. April 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
mozillacert81.pem	13. März 2014	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
mozillacert99.pem	13. März 2014	F1:7F:6F:B6:31:DC:99:E3:A3:C8:7F:FE:1C:F1:81:10:88:D9:60:33
mozillacert145.pem	13. März 2014	10:1D:FA:3F:D5:0B:CB:BB:9B:B5:60:0C:19:55:A4:1A:F4:73:3A:04
mozillacert37.pem	13. März 2014	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert4.pem	13. März 2014	E3:92:51:2F:0A:CF:F5:05:DF:F6:DE:06:7F:75:37:E1:65:EA:57:4B
mozillacert70.pem	13. März 2014	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
mozillacert88.pem	13. März 2014	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
mozillacert134.pem	13. März 2014	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert26.pem	13. März 2014	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
mozillacert77.pem	13. März 2014	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert123.pem	13. März 2014	2A:B6:28:48:5E:78:FB:F3:AD:9E:79:10:DD:6B:DF:99:72:2C:96:E5
mozillacert15.pem	13. März 2014	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
mozillacert66.pem	13. März 2014	DD:E1:D2:A9:01:80:2E:1D:87:5E:84:B3:80:7E:4B:B1:FD:99:41:34
mozillacert112.pem	13. März 2014	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
mozillacert55.pem	13. März 2014	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
mozillacert101.pem	13. März 2014	99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39
mozillacert119.pem	13. März 2014	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
mozillacert44.pem	13. März 2014	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
mozillacert108.pem	13. März 2014	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
mozillacert95.pem	13. März 2014	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert141.pem	13. März 2014	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
mozillacert33.pem	13. März 2014	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
mozillacert0.pem	13. März 2014	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
mozillacert84.pem	13. März 2014	D3:C0:63:F2:19:ED:07:3E:34:AD:5D:75:0B:32:76:29:FF:D5:9A:F2
mozillacert130.pem	13. März 2014	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
mozillacert148.pem	13. März 2014	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
mozillacert22.pem	13. März 2014	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
mozillacert7.pem	13. März 2014	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
mozillacert73.pem	13. März 2014	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
mozillacert137.pem	13. März 2014	4A:65:D5:F4:1D:EF:39:B8:B8:90:4A:4A:D3:64:81:33:CF:C7:A1:D1
mozillacert11.pem	13. März 2014	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
mozillacert29.pem	13. März 2014	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert62.pem	13. März 2014	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert126.pem	13. März 2014	25:01:90:19:CF:FB:D9:99:1C:B7:68:25:74:8D:94:5F:30:93:95:42
mozillacert18.pem	13. März 2014	79:98:A3:08:E1:4D:65:85:E6:C2:1E:15:3A:71:9F:BA:5A:D3:4A:D9
mozillacert51.pem	13. März 2014	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
mozillacert69.pem	13. März 2014	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
mozillacert115.pem	13. März 2014	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
mozillacert40.pem	13. März 2014	80:25:EF:F4:6E:70:C8:D4:72:24:65:84:FE:40:3B:8A:8D:6A:DB:F5
mozillacert58.pem	13. März 2014	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
mozillacert104.pem	13. März 2014	4F:99:AA:93:FB:2B:D1:37:26:A1:99:4A:CE:7F:F0:05:F2:93:5D:1E
mozillacert91.pem	13. März 2014	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
mozillacert47.pem	13. März 2014	1B:4B:39:61:26:27:6B:64:91:A2:68:6D:D7:02:43:21:2D:1F:1D:96
mozillacert80.pem	13. März 2014	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert98.pem	13. März 2014	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
mozillacert144.pem	13. März 2014	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
mozillacert36.pem	13. März 2014	23:88:C9:D3:71:CC:9E:96:3D:FF:7D:3C:A7:CE:FC:D6:25:EC:19:0D
mozillacert3.pem	13. März 2014	87:9F:4B:EE:05:DF:98:58:3B:E3:60:D6:33:E7:0D:3F:FE:98:71:AF
mozillacert87.pem	13. März 2014	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
mozillacert133.pem	13. März 2014	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
mozillacert25.pem	13. März 2014	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
mozillacert76.pem	13. März 2014	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert122.pem	13. März 2014	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
mozillacert14.pem	13. März 2014	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
mozillacert65.pem	13. März 2014	69:BD:8C:F4:9C:D3:00:FB:59:2E:17:93:CA:55:6A:F3:EC:AA:35:FB
mozillacert111.pem	13. März 2014	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert129.pem	13. März 2014	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
mozillacert54.pem	13. März 2014	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
mozillacert100.pem	13. März 2014	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
mozillacert118.pem	13. März 2014	7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
mozillacert151.pem	13. März 2014	AC:ED:5F:65:53:FD:25:CE:01:5F:1F:7A:48:3B:6A:74:9F:61:78:C6
mozillacert43.pem	13. März 2014	F9:CD:0E:2C:DA:76:24:C1:8F:BD:F0:F0:AB:B6:45:B8:F7:FE:D5:7A
mozillacert107.pem	13. März 2014	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
mozillacert94.pem	13. März 2014	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
mozillacert140.pem	13. März 2014	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
mozillacert32.pem	13. März 2014	60:D6:89:74:B5:C2:65:9E:8A:0F:C1:88:7C:88:D2:46:69:1B:18:2C
mozillacert83.pem	13. März 2014	A0:73:E5:C5:BD:43:61:0D:86:4C:21:13:0A:85:58:57:CC:9C:EA:46
mozillacert147.pem	13. März 2014	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert21.pem	13. März 2014	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
mozillacert39.pem	13. März 2014	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
mozillacert6.pem	13. März 2014	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
mozillacert72.pem	13. März 2014	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
mozillacert136.pem	13. März 2014	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert10.pem	13. März 2014	5F:3A:FC:0A:8B:64:F6:86:67:34:74:DF:7E:A9:A2:FE:F9:FA:7A:51
mozillacert28.pem	13. März 2014	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
mozillacert61.pem	13. März 2014	E0:B4:32:2E:B2:F6:A5:68:B6:54:53:84:48:18:4A:50:36:87:43:84
mozillacert79.pem	13. März 2014	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
mozillacert125.pem	13. März 2014	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert17.pem	13. März 2014	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
mozillacert50.pem	13. März 2014	8C:96:BA:EB:DD:2B:07:07:48:EE:30:32:66:A0:F3:98:6E:7C:AE:58

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert68.pem	13. März 2014	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
mozillacert114.pem	13. März 2014	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert57.pem	13. März 2014	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
mozillacert103.pem	13. März 2014	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
mozillacert90.pem	13. März 2014	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
mozillacert46.pem	13. März 2014	40:9D:4B:D9:17:B5:5C:27:B6:9B:64:CB:98:22:44:0D:CD:09:B8:89
mozillacert97.pem	13. März 2014	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
mozillacert143.pem	13. März 2014	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
mozillacert35.pem	13. März 2014	2A:C8:D5:8B:57:CE:BF:2F:49:AF:F2:FC:76:8F:51:14:62:90:7A:41
mozillacert2.pem	13. März 2014	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
mozillacert86.pem	13. März 2014	74:2C:31:92:E6:07:E4:24:EB:45:49:54:2B:E1:BB:C5:3E:61:74:E2
mozillacert132.pem	13. März 2014	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert24.pem	13. März 2014	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
mozillacert9.pem	13. März 2014	F4:8B:11:BF:DE:AB:BE:94:54:20:71:E6:41:DE:6B:BE:88:2B:40:B9
mozillacert75.pem	13. März 2014	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
mozillacert121.pem	13. März 2014	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
mozillacert139.pem	13. März 2014	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
mozillacert13.pem	13. März 2014	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
mozillacert64.pem	13. März 2014	62:7F:8D:78:27:65:63:99:D2:7D:7F:90:44:C9:FE:B3:F3:3E:FA:9A
mozillacert110.pem	13. März 2014	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
mozillacert128.pem	13. März 2014	A9:E9:78:08:14:37:58:88:F2:05:19:B0:6D:2B:0D:2B:60:16:90:7D
mozillacert53.pem	13. März 2014	7F:8A:B0:CF:D0:51:87:6A:66:F3:36:0F:47:C8:8D:8C:D3:35:FC:74
mozillacert117.pem	13. März 2014	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
mozillacert150.pem	13. März 2014	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert42.pem	13. März 2014	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
mozillacert106.pem	13. März 2014	E7:A1:90:29:D3:D5:52:DC:0D:0F:C6:92:D3:EA:88:0D:15:2E:1A:6B
mozillacert93.pem	13. März 2014	31:F1:FD:68:22:63:20:EE:C6:3B:3F:9D:EA:4A:3E:53:7C:7C:39:17
mozillacert31.pem	13. März 2014	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
mozillacert49.pem	13. März 2014	61:57:3A:11:DF:0E:D8:7E:D5:92:65:22:EA:D0:56:D7:44:B3:23:71
mozillacert82.pem	13. März 2014	2E:14:DA:EC:28:F0:FA:1E:8E:38:9A:4E:AB:EB:26:C0:0A:D3:83:C3
mozillacert146.pem	13. März 2014	21:FC:BD:8E:7F:6C:AF:05:1B:D1:B3:43:EC:A8:E7:61:47:F2:0F:8A
mozillacert20.pem	13. März 2014	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert38.pem	13. März 2014	CB:A1:C5:F8:B0:E3:5E:B8:B9:45:12:D3:F9:34:A2:E9:06:10:D3:36
mozillacert5.pem	13. März 2014	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
mozillacert71.pem	13. März 2014	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
mozillacert89.pem	13. März 2014	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert135.pem	13. März 2014	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
mozillacert27.pem	13. März 2014	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
mozillacert60.pem	13. März 2014	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
mozillacert78.pem	13. März 2014	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
mozillacert124.pem	13. März 2014	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert16.pem	13. März 2014	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
mozillacert67.pem	13. März 2014	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
mozillacert113.pem	13. März 2014	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
mozillacert56.pem	13. März 2014	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
mozillacert102.pem	13. März 2014	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
mozillacert45.pem	13. März 2014	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert109.pem	13. März 2014	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert96.pem	13. März 2014	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
mozillacert142.pem	13. März 2014	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
mozillacert34.pem	13. März 2014	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
mozillacert1.pem	13. März 2014	23:E5:94:94:51:95:F2:41:48:03:B4:D5:64:D2:A3:A3:F5:D8:8B:8C
mozillacert85.pem	13. März 2014	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
mozillacert131.pem	13. März 2014	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
mozillacert149.pem	13. März 2014	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
mozillacert23.pem	13. März 2014	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
mozillacert8.pem	13. März 2014	3E:2B:F7:F2:03:1B:96:F3:8C:E6:C4:D8:A8:5D:3E:2D:58:47:6A:0F
mozillacert74.pem	13. März 2014	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
mozillacert120.pem	13. März 2014	DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97:FE:2F:9D:F5:B7:D1:8A:41
mozillacert138.pem	13. März 2014	E1:9F:E3:0E:8B:84:60:9E:80:9B:17:0D:72:A8:C5:BA:6E:14:09:BD

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert12.pem	13. März 2014	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
mozillacert63.pem	13. März 2014	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
mozillacert127.pem	13. März 2014	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
mozillacert19.pem	13. März 2014	B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB:B3:94:BD:63:7B:A7:82:B7
mozillacert52.pem	13. März 2014	8B:AF:4C:9B:1D:F0:2A:92:F7:DA:12:8E:B9:1B:AC:F4:98:60:4B:6F
mozillacert116.pem	13. März 2014	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
mozillacert41.pem	13. März 2014	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
mozillacert59.pem	13. März 2014	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
mozillacert105.pem	13. März 2014	77:47:4F:C6:30:E4:0F:4C:47:64:3F:84:BA:B8:C6:95:4A:8A:41:EC
mozillacert92.pem	13. März 2014	A3:F1:33:3F:E2:42:BF:CF:C5:D1:4E:8F:39:42:98:40:68:10:D1:A0
mozillacert30.pem	13. März 2014	E7:B4:F6:9D:61:EC:90:69:DB:7E:90:A7:40:1A:3C:F4:7D:4F:E8:EE
mozillacert48.pem	13. März 2014	A0:A1:AB:90:C9:FC:84:7B:3B:12:61:E8:97:7D:5F:D3:22:61:D3:CC

Name	Date (Datum)	SHA1 Fingerabdruck
verisignc4g2.pem	20. März 2014	0B:77:BE:BB:CB:7A:A2:47:05:DE:CC:0F:BD:6A:02:FC:7A:BD:9B:52
verisignc2g3.pem	20. März 2014	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
verisignc1g6.pem	31. Dezember 2014	51:7F:61:1E:29:91:6B:53:82:FB:72:E7:44:D9:8D:C3:CC:53:6D:64
verisignc2g2.pem	20. März 2014	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
verisignroot.pem	20. März 2014	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
verisignc2g1.pem	20. März 2014	67:82:AA:E0:ED:EE:E2:1A:58:39:D3:C0:CD:14:68:0A:4F:60:14:2A
verisignc3g5.pem	20. März 2014	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
verisignc1g3.pem	20. März 2014	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
verisignc3g4.pem	20. März 2014	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
verisignc1g2.pem	20. März 2014	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
verisignc3g3.pem	20. März 2014	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
verisignc1g1.pem	20. März 2014	90:AE:A2:69:85:FF:14:80:4C:43:49:52:EC:E9:60:84:77:AF:55:6F

Name	Date (Datum)	SHA1 Fingerabdruck
verisignc3g2.pem	20. März 2014	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
verisignc3g1.pem	20. März 2014	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
verisignc2g6.pem	31. Dezember 2014	40:B3:31:A0:E9:BF:E8:55:BC:39:93:CA:70:4F:4E:C2:51:D4:1D:8F
verisignc4g3.pem	20. März 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. Dezember 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. Dezember 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. Dezember 2014	27:AC:93:69:FA:F2:52:07:BB:26:27:CE:FA:CC:BE:4E:F9:C3:19:B8
dstacescax6	18. Juni 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. Juni 2018	27:AC:93:69:FA:F2:52:07:BB:26:27:CE:FA:CC:BE:4E:F9:C3:19:B8
verisignc4g3.pem	18. Juni 2018	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
swisssignplatinumg2ca	21. April 2018	56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66

Name	Date (Datum)	SHA1 Fingerabdruck
geotrustprimarycertificationauthorityg3	18. Juni 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycertificationauthorityg2	18. Juni 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
buypassclass2rootca	18. Juni 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
camerfirmachambersofcommerceroot	18. Juni 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
mozillacert20.pem	18. Juni 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert12.pem	18. Juni 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
mozillacert90.pem	18. Juni 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
mozillacert82.pem	18. Juni 2018	2E:14:DA:EC:28:F0:FA:1E:8E:38:9A:4E:AB:EB:26:C0:0A:D3:83:C3
mozillacert140.pem	18. Juni 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
mozillacert74.pem	18. Juni 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
mozillacert132.pem	18. Juni 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert66.pem	18. Juni 2018	DD:E1:D2:A9:01:80:2E:1D:87:5E:84:B3:80:7E:4B:B1:FD:99:41:34
mozillacert124.pem	18. Juni 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert58.pem	18. Juni 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
securitycommunicationrootca2	18. Juni 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
mozillacert116.pem	18. Juni 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
mozillacert108.pem	18. Juni 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
certigna	18. Juni 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert3.pem	18. Juni 2018	87:9F:4B:EE:05:DF:98:58:3B:E3:60:D6:33:E7:0D:3F:FE:98:71:AF
verisignc1g1.pem	18. Juni 2018	90:AE:A2:69:85:FF:14:80:4C:43:49:52:EC:E9:60:84:77:AF:55:6F
verisignc4g2.pem	18. Juni 2018	0B:77:BE:BB:CB:7A:A2:47:05:DE:CC:0F:BD:6A:02:FC:7A:BD:9B:52
deutschetelekomrootca2	18. Juni 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
starfieldrootg2ca	21. April 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E

Name	Date (Datum)	SHA1 Fingerabdruck
comodoecccertificationauthority	18. Juni 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
digicertglobalrootg3	18. Juni 2018	7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E
digicertglobalrootg2	18. Juni 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
mozillacert11.pem	18. Juni 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
mozillacert81.pem	18. Juni 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
mozillacert73.pem	18. Juni 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
szafirrootca2	18. Juni 2018	E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
mozillacert131.pem	18. Juni 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
ecacc	18. Juni 2018	28:90:3A:63:5B:52:80:FA:E6:77:4C:0B:6D:A7:D6:BA:A6:4A:F2:E8
mozillacert65.pem	18. Juni 2018	69:BD:8C:F4:9C:D3:00:FB:59:2E:17:93:CA:55:6A:F3:EC:AA:35:FB
turktrustelektroniksertifikahizmetsaglayicisih5	18. Juni 2018	C4:18:F6:4D:46:D1:DF:00:3D:27:30:13:72:43:A9:12:11:C6:75:FB

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert123.pem	18. Juni 2018	2A:B6:28:48:5E:78:FB:F3:AD:9E:79:10:DD:6B:DF:99:72:2C:96:E5
mozillacert57.pem	18. Juni 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
mozillacert115.pem	18. Juni 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
mozillacert49.pem	18. Juni 2018	61:57:3A:11:DF:0E:D8:7E:D5:92:65:22:EA:D0:56:D7:44:B3:23:71
mozillacert107.pem	18. Juni 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
verisignclass3g4ca	21. April 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
securetrustca	18. Juni 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
mozillacert2.pem	18. Juni 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
buypassclass2ca	21. April 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
secomscrootca2	21. April 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
secomscrootca1	21. April 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
trustisfpsrootca	18. Juni 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04

Name	Date (Datum)	SHA1 Fingerabdruck
hongkongpostrootca1	18. Juni 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
certsignrootca	18. Juni 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
geotrustprimaryca	21. April 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
twcaglobalrootca	18. Juni 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
camerfirmachambersca	21. April 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
mozillacert10.pem	18. Juni 2018	5F:3A:FC:0A:8B:64:F6:86:67:34:74:DF:7E:A9:A2:FE:F9:FA:7A:51
mozillacert80.pem	18. Juni 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
mozillacert72.pem	18. Juni 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
comodoaaaca	21. April 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert130.pem	18. Juni 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
mozillacert64.pem	18. Juni 2018	62:7F:8D:78:27:65:63:99:D2:7D:7F:90:44:C9:FE:B3:F3:3E:FA:9A
mozillacert122.pem	18. Juni 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert56.pem	18. Juni 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
equifaxsecureebusinessca1	21. April 2018	AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
camerfirmachambersignca	21. April 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
mozillacert114.pem	18. Juni 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert48.pem	18. Juni 2018	A0:A1:AB:90:C9:FC:84:7B:3B:12:61:E8:97:7D:5F:D3:22:61:D3:CC
pscprocert	18. Juni 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
mozillacert106.pem	18. Juni 2018	E7:A1:90:29:D3:D5:52:DC:0D:0F:C6:92:D3:EA:88:0D:15:2E:1A:6B
mozillacert1.pem	18. Juni 2018	23:E5:94:94:51:95:F2:41:48:03:B4:D5:64:D2:A3:A3:F5:D8:8B:8C
eecertificationcentrerootca	18. Juni 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
digicertglobalrootca	18. Juni 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
thawteprimaryrootcag3	18. Juni 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
thawteprimaryrootcag2	18. Juni 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12

Name	Date (Datum)	SHA1 Fingerabdruck
entrustrootcertificationauthorityec1	18. Juni 2018	20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
valicertclass2ca	21. April 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
globalchambersignroot2008	18. Juni 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
amazonrootca4	18. Juni 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. Juni 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
amazonrootca3	18. Juni 2018	0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
amazonrootca2	18. Juni 2018	5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
securitycommunicationrootca	18. Juni 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
amazonrootca1	18. Juni 2018	8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
acraizfnmtrcm	18. Juni 2018	EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
quovadisrootca3g3	18. Juni 2018	48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
certplusrootcag2	18. Juni 2018	4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A

Name	Date (Datum)	SHA1 Fingerabdruck
certplusrootcag1	18. Juni 2018	22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
mozillacert71.pem	18. Juni 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
mozillacert63.pem	18. Juni 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
mozillacert121.pem	18. Juni 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
ttelesecglobalrootclass3ca	21. April 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
mozillacert55.pem	18. Juni 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
mozillacert113.pem	18. Juni 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
baltimorecybertrustca	21. April 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
mozillacert47.pem	18. Juni 2018	1B:4B:39:61:26:27:6B:64:91:A2:68:6D:D7:02:43:21:2D:1F:1D:96
mozillacert105.pem	18. Juni 2018	77:47:4F:C6:30:E4:0F:4C:47:64:3F:84:BA:B8:C6:95:4A:8A:41:EC
mozillacert39.pem	18. Juni 2018	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
usertrustecccertificationauthority	18. Juni 2018	D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert0.pem	18. Juni 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
securitycommunicationevrootca1	18. Juni 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
verisignc3g5.pem	18. Juni 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
globalsignr3ca	21. April 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
trustcoreca1	18. Juni 2018	58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
equifaxsecureglobalebusinessca1	21. April 2018	3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36
geotrustuniversalca	18. Juni 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
affirmtrustpremiumca	21. April 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
staatdernederlandenrootcag3	18. Juni 2018	D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC
staatdernederlandenrootcag2	18. Juni 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
mozillacert70.pem	18. Juni 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
secomevrootca1	21. April 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D

Name	Date (Datum)	SHA1 Fingerabdruck
geotrustglobalca	18. Juni 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
mozillacert62.pem	18. Juni 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert120.pem	18. Juni 2018	DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97:FE:2F:9D:F5:B7:D1:8A:41
mozillacert54.pem	18. Juni 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
mozillacert112.pem	18. Juni 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
mozillacert46.pem	18. Juni 2018	40:9D:4B:D9:17:B5:5C:27:B6:9B:64:CB:98:22:44:0D:CD:09:B8:89
swisssigngoldcag2	18. Juni 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert104.pem	18. Juni 2018	4F:99:AA:93:FB:2B:D1:37:26:A1:99:4A:CE:7F:F0:05:F2:93:5D:1E
mozillacert38.pem	18. Juni 2018	CB:A1:C5:F8:B0:E3:5E:B8:B9:45:12:D3:F9:34:A2:E9:06:10:D3:36
certplusclass3pprimaryca	21. April 2018	21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
entrustrootcertificationauthorityg2	18. Juni 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
godaddyrootg2ca	21. April 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B

Name	Date (Datum)	SHA1 Fingerabdruck
cfcaevroot	18. Juni 2018	E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
verisignc3g4.pem	18. Juni 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
geotrustuniversalca2	18. Juni 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
starfieldservicesrootg2ca	21. April 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
digicerthighassuranceevrootca	18. Juni 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
entrustnetpremium2048secureserverca	18. Juni 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
camerfirmaglobalchambersignroot	18. Juni 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
verisignclass3g3ca	21. April 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
godaddyclass2ca	18. Juni 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
mozillacert61.pem	18. Juni 2018	E0:B4:32:2E:B2:F6:A5:68:B6:54:53:84:48:18:4A:50:36:87:43:84
mozillacert53.pem	18. Juni 2018	7F:8A:B0:CF:D0:51:87:6A:66:F3:36:0F:47:C8:8D:8C:D3:35:FC:74
atostrustedroot2011	18. Juni 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert111.pem	18. Juni 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
staatdernederlandenevrootca	18. Juni 2018	76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
mozillacert45.pem	18. Juni 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert103.pem	18. Juni 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
mozillacert37.pem	18. Juni 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert29.pem	18. Juni 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
izenpecom	18. Juni 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
comodorsacertificationauthority	18. Juni 2018	AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
mozillacert99.pem	18. Juni 2018	F1:7F:6F:B6:31:DC:99:E3:A3:C8:7F:FE:1C:F1:81:10:88:D9:60:33
mozillacert149.pem	18. Juni 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
utnuserfirstobjectca	21. April 2018	E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
verisignc3g3.pem	18. Juni 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6

Name	Date (Datum)	SHA1 Fingerabdruck
dstrootcax3	18. Juni 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
addtrustexternalroot	18. Juni 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
certumtrustednetworkca	18. Juni 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
affirmtrustpremiumecc	18. Juni 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
starfieldclass2ca	18. Juni 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
actalisauthenticationrootca	18. Juni 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
verisignclass2g3ca	21. April 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
isrgrootx1	18. Juni 2018	CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
godaddyrootcertificateauthorityg2	18. Juni 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
mozillacert60.pem	18. Juni 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
chunghwaepkirootca	21. April 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert52.pem	18. Juni 2018	8B:AF:4C:9B:1D:F0:2A:92:F7:DA:12:8E:B9:1B:AC:F4:98:60:4B:6F

Name	Date (Datum)	SHA1 Fingerabdruck
microseceszignorootca2009	18. Juni 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
securesignrootca11	18. Juni 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
mozillacert110.pem	18. Juni 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
mozillacert44.pem	18. Juni 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
mozillacert102.pem	18. Juni 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
mozillacert36.pem	18. Juni 2018	23:88:C9:D3:71:CC:9E:96:3D:FF:7D:3C:A7:CE:FC:D6:25:EC:19:0D
mozillacert28.pem	18. Juni 2018	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
baltimorecybertrustroot	18. Juni 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
amzninternalrootca	12. Dezember 2008	A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
mozillacert98.pem	18. Juni 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
mozillacert148.pem	18. Juni 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
verisignc3g2.pem	18. Juni 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F

Name	Date (Datum)	SHA1 Fingerabdruck
quovadisrootca2g3	18. Juni 2018	09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
geotrustprimarycertificationauthority	18. Juni 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
opentrustrootcag3	18. Juni 2018	6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6
opentrustrootcag2	18. Juni 2018	79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
opentrustrootcag1	18. Juni 2018	79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E
verisignclass3ca	21. April 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
globalsignca	21. April 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
ttelesecglobalrootclass2ca	21. April 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
verisignclass1g3ca	21. April 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
verisignuniversalrootca	21. April 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
soneraclass2ca	21. April 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27

Name	Date (Datum)	SHA1 Fingerabdruck
starfieldservicesrootcertificateauthorityg2	18. Juni 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
mozillacert51.pem	18. Juni 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
mozillacert43.pem	18. Juni 2018	F9:CD:0E:2C:DA:76:24:C1:8F:BD:F0:F0:AB:B6:45:B8:F7:FE:D5:7A
mozillacert101.pem	18. Juni 2018	99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39
mozillacert35.pem	18. Juni 2018	2A:C8:D5:8B:57:CE:BF:2F:49:AF:F2:FC:76:8F:51:14:62:90:7A:41
globalsignr2ca	21. April 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
mozillacert27.pem	18. Juni 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
affirmtrustpremium	18. Juni 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
mozillacert19.pem	18. Juni 2018	B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB:B3:94:BD:63:7B:A7:82:B7
mozillacert97.pem	18. Juni 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
netlockaranyclassgoldfotanusitvany	18. Juni 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
mozillacert89.pem	18. Juni 2018	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D

Name	Date (Datum)	SHA1 Fingerabdruck
verisignroot.pem	18. Juni 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
mozillacert147.pem	18. Juni 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
aolrootca2	21. April 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
cia-crt-g3-01-ca	23. November 2016	2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
aolrootca1	21. April 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
verisignc3g1.pem	18. Juni 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert139.pem	18. Juni 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
soneraclass2rootca	18. Juni 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
swisssignsilverg2ca	21. April 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
thawteprimaryrootca	18. Juni 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
gdcatrustauthr5root	18. Juni 2018	0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
trustcenterclass4caii	21. April 2018	A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50

Name	Date (Datum)	SHA1 Fingerabdruck
usertrustrsacertificationauthority	18. Juni 2018	2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
digicertassuredidrootg3	18. Juni 2018	F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
digicertassuredidrootg2	18. Juni 2018	A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F
mozillacert50.pem	18. Juni 2018	8C:96:BA:EB:DD:2B:07:07:48:EE:30:32:66:A0:F3:98:6E:7C:AE:58
mozillacert42.pem	18. Juni 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
mozillacert100.pem	18. Juni 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
mozillacert34.pem	18. Juni 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
affirmtrustcommercialca	21. April 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert26.pem	18. Juni 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
globalsigneccrootcar5	18. Juni 2018	1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
globalsigneccrootcar4	18. Juni 2018	69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
buypassclass3rootca	18. Juni 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert18.pem	18. Juni 2018	79:98:A3:08:E1:4D:65:85:E6:C2:1E:15:3A:71:9F:BA:5A:D3:4A:D9
mozillacert96.pem	18. Juni 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
verisignc2g6.pem	18. Juni 2018	40:B3:31:A0:E9:BF:E8:55:BC:39:93:CA:70:4F:4E:C2:51:D4:1D:8F
secomvalicertclass1ca	21. April 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
mozillacert88.pem	18. Juni 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
accvraiz1	18. Juni 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
mozillacert146.pem	18. Juni 2018	21:FC:BD:8E:7F:6C:AF:05:1B:D1:B3:43:EC:A8:E7:61:47:F2:0F:8A
mozillacert138.pem	18. Juni 2018	E1:9F:E3:0E:8B:84:60:9E:80:9B:17:0D:72:A8:C5:BA:6E:14:09:BD
verisignclass3g2ca	21. April 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
dtrustrootclass3ca2ev2009	18. Juni 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
xrampglobalca	21. April 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
mozillacert9.pem	18. Juni 2018	F4:8B:11:BF:DE:AB:BE:94:54:20:71:E6:41:DE:6B:BE:88:2B:40:B9

Name	Date (Datum)	SHA1 Fingerabdruck
verisignuniversalrootcertificationauthority	18. Juni 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
tubitakkamusmsslkoksertifikasisurum1	18. Juni 2018	31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA
mozillacert41.pem	18. Juni 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
mozillacert33.pem	18. Juni 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
mozillacert25.pem	18. Juni 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
mozillacert17.pem	18. Juni 2018	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
mozillacert95.pem	18. Juni 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
affirmtrustpremiumeccca	21. April 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
mozillacert87.pem	18. Juni 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
mozillacert145.pem	18. Juni 2018	10:1D:FA:3F:D5:0B:CB:BB:9B:B5:60:0C:19:55:A4:1A:F4:73:3A:04
mozillacert79.pem	18. Juni 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert137.pem	18. Juni 2018	4A:65:D5:F4:1D:EF:39:B8:B8:90:4A:4A:D3:64:81:33:CF:C7:A1:D1
digicertassuredidrootca	18. Juni 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
addtrustqualifiedca	21. April 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert129.pem	18. Juni 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
verisignclass2g2ca	21. April 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
baltimorecodesigningca	21. April 2018	30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD:49:27:08:7C:60:56:7B:0D
luxtrustglobalroot2	18. Juni 2018	1E:0E:56:19:0A:D1:8B:25:98:B2:04:44:FF:66:8A:04:17:99:5F:3F
visaecommerceroot	18. Juni 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
oistewisekeyglobalrootgbca	18. Juni 2018	0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
mozillacert8.pem	18. Juni 2018	3E:2B:F7:F2:03:1B:96:F3:8C:E6:C4:D8:A8:5D:3E:2D:58:47:6A:0F
comodocertificationauthority	18. Juni 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. November 2016	96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09

Name	Date (Datum)	SHA1 Fingerabdruck
verisignc1g6.pem	18. Juni 2018	51:7F:61:1E:29:91:6B:53:82:FB:72:E7:44:D9:8D:C3:CC:53:6D:64
trustcenterclass2caii	21. April 2018	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
quovadisrootca1g3	18. Juni 2018	1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67
mozillacert40.pem	18. Juni 2018	80:25:EF:F4:6E:70:C8:D4:72:24:65:84:FE:40:3B:8A:8D:6A:DB:F5
cadisigrootr2	18. Juni 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
cadisigrootr1	18. Juni 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
mozillacert32.pem	18. Juni 2018	60:D6:89:74:B5:C2:65:9E:8A:0F:C1:88:7C:88:D2:46:69:1B:18:2C
utndatacorpsgcca	21. April 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
mozillacert24.pem	18. Juni 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
addtrustclass1ca	21. April 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
mozillacert16.pem	18. Juni 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
affirmtrustnetworkingca	21. April 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F

Name	Date (Datum)	SHA1 Fingerabdruck
mozillacert94.pem	18. Juni 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
mozillacert86.pem	18. Juni 2018	74:2C:31:92:E6:07:E4:24:EB:45:49:54:2B:E1:BB:C5:3E:61:74:E2
mozillacert144.pem	18. Juni 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
mozillacert78.pem	18. Juni 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
mozillacert136.pem	18. Juni 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert128.pem	18. Juni 2018	A9:E9:78:08:14:37:58:88:F2:05:19:B0:6D:2B:0D:2B:60:16:90:7D
verisignclass1g2ca	21. April 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
hellenicacademicandresearchinstitutionsrootca2015	18. Juni 2018	01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
soneraclass1ca	21. April 2018	07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
hellenicacademicandresearchinstitutionsrootca2011	18. Juni 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
certumtrustednetworkca2	18. Juni 2018	D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92

Name	Date (Datum)	SHA1 Fingerabdruck
equifaxsecureca	21. April 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
thawteserverca	21. April 2018	9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
mozillacert7.pem	18. Juni 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
affirmtrustnetworking	18. Juni 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
deprecateditsecca	27. Januar 2012	12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
globalsignrootcar3	18. Juni 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
globalsignrootcar2	18. Juni 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
quovadisrootca	18. Juni 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
mozillacert31.pem	18. Juni 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
entrustrootcertificationauthority	18. Juni 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert23.pem	18. Juni 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
mozillacert15.pem	18. Juni 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB

Name	Date (Datum)	SHA1 Fingerabdruck
verisignc2g3.pem	18. Juni 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
mozillacert93.pem	18. Juni 2018	31:F1:FD:68:22:63:20:EE:C6:3B:3F:9D:EA:4A:3E:53:7C:7C:39:17
mozillacert151.pem	18. Juni 2018	AC:ED:5F:65:53:FD:25:CE:01:5F:1F:7A:48:3B:6A:74:9F:61:78:C6
mozillacert85.pem	18. Juni 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
certplusclass2primaryca	18. Juni 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
mozillacert143.pem	18. Juni 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
mozillacert77.pem	18. Juni 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert135.pem	18. Juni 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
mozillacert69.pem	18. Juni 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
mozillacert127.pem	18. Juni 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
mozillacert119.pem	18. Juni 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
geotrustprimarycag3	21. April 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD

Name	Date (Datum)	SHA1 Fingerabdruck
identrustpublicsectorrootca1	18. Juni 2018	BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
geotrustprimarycag2	21. April 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
trustcorrootcertca2	18. Juni 2018	B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
mozillacert6.pem	18. Juni 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
trustcorrootcertca1	18. Juni 2018	FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
networksolutionscertificate authority	18. Juni 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
twcarootcertificationauthority	18. Juni 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
addtrustexternalca	21. April 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
verisignclass3g5ca	21. April 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
autoridaddecertificacionfirmaprofesionalcifa62634068	18. Juni 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA

Name	Date (Datum)	SHA1 Fingerabdruck
hellenicacademicandresearchinstitutionseccrootca2015	18. Juni 2018	9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
verisigntsaca	21. April 2018	20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
utnuserfirsthardwareca	21. April 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
identrustcommercialrootca1	18. Juni 2018	DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
dtrustrootclass3ca22009	18. Juni 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
epkirootcertificationauthority	18. Juni 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert30.pem	18. Juni 2018	E7:B4:F6:9D:61:EC:90:69:DB:7E:90:A7:40:1A:3C:F4:7D:4F:E8:EE
teliasonerarootcav1	18. Juni 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
buypassclass3ca	21. April 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
mozillacert22.pem	18. Juni 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
mozillacert14.pem	18. Juni 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25

Name	Date (Datum)	SHA1 Fingerabdruck
verisignc2g2.pem	18. Juni 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
certumca	21. April 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
mozillacert92.pem	18. Juni 2018	A3:F1:33:3F:E2:42:BF:CF:C5:D1:4E:8F:39:42:98:40:68:10:D1:A0
mozillacert150.pem	18. Juni 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
mozillacert84.pem	18. Juni 2018	D3:C0:63:F2:19:ED:07:3E:34:AD:5D:75:0B:32:76:29:FF:D5:9A:F2
ttelesecglobalrootclass3	18. Juni 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
globalsignrootca	18. Juni 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
ttelesecglobalrootclass2	18. Juni 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
mozillacert142.pem	18. Juni 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
mozillacert76.pem	18. Juni 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert134.pem	18. Juni 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
mozillacert68.pem	18. Juni 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA

Name	Date (Datum)	SHA1 Fingerabdruck
etugracertificationauthority	18. Juni 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert126.pem	18. Juni 2018	25:01:90:19:CF:FB:D9:99:1C:B7:68:25:74:8D:94:5F:30:93:95:42
keynectisrootca	21. April 2018	9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
mozillacert118.pem	18. Juni 2018	7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
quovadisrootca3	18. Juni 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
quovadisrootca2	18. Juni 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
mozillacert5.pem	18. Juni 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
verisignc1g3.pem	18. Juni 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
cybertrustglobalroot	18. Juni 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
amzninternalinfosecag3	27. Februar 2015	B9:B1:CA:38:F7:BF:9C:D2:D4:95:E7:B6:5E:75:32:9B:A8:78:2E:F6
starfieldrootcertificateauthorityg2	18. Juni 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
entrust2048ca	21. April 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31

Name	Date (Datum)	SHA1 Fingerabdruck
swisssignsilvercag2	18. Juni 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
affirmtrustcommercial	18. Juni 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
certinomisrootca	18. Juni 2018	9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C:01:B9:32:C5:34:E7:88:A8
xrampglobalcaroot	18. Juni 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
secureglobalca	18. Juni 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
swisssigngoldg2ca	21. April 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert21.pem	18. Juni 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
mozillacert13.pem	18. Juni 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
verisignc2g1.pem	18. Juni 2018	67:82:AA:E0:ED:EE:E2:1A:58:39:D3:C0:CD:14:68:0A:4F:60:14:2A
mozillacert91.pem	18. Juni 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
oistewisekeyglobalrootgaca	18. Juni 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
mozillacert83.pem	18. Juni 2018	A0:73:E5:C5:BD:43:61:0D:86:4C:21:13:0A:85:58:57:CC:9C:EA:46

Name	Date (Datum)	SHA1 Fingerabdruck
entrustevca	21. April 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert141.pem	18. Juni 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
mozillacert75.pem	18. Juni 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
mozillacert133.pem	18. Juni 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
mozillacert67.pem	18. Juni 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
mozillacert125.pem	18. Juni 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert59.pem	18. Juni 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
thawtepremiumserverca	21. April 2018	E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66
mozillacert117.pem	18. Juni 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
utnuserfirstclientauthemailca	21. April 2018	B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
entrustrootcag2	21. April 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
mozillacert109.pem	18. Juni 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71

Name	Date (Datum)	SHA1 Fingerabdruck
digicerttrustedroo tg4	18. Juni 2018	DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F: C8:3A:4D:7D:77:5D:05:E4
gdroot-g2.pem	18. Juni 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62: A7:9F:45:C2:54:FD:E6:8B
comodoaaaservicesr oot	18. Juni 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2: F1:F1:60:17:64:D8:E3:49
mozillacert4.pem	18. Juni 2018	E3:92:51:2F:0A:CF:F5:05:DF:F6:DE:06: 7F:75:37:E1:65:EA:57:4B
verisignclass3publ icprimary certifica tionauthorityg5	18. Juni 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD: 56:BE:3D:9B:67:44:A5:E5
chambersofcommerce root2008	18. Juni 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50: BA:9E:A8:7E:FE:9A:CE:3C
verisignclass3publ icprimary certifica tionauthorityg4	18. Juni 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7: CF:8A:2D:64:C9:3F:6C:3A
verisignclass3publ icprimary certifica tionauthorityg3	18. Juni 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3: 39:E2:55:76:60:9B:5C:C6
thawtepersonalfree mailca	21. April 2018	E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8: E9:25:2B:45:A6:4F:B7:E2
verisignc1g2.pem	18. Juni 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B: 56:16:7F:62:F5:32:E5:47

Name	Date (Datum)	SHA1 Fingerabdruck
gtecybertrustglobalca	21. April 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
trustcenteruniversalcai	21. April 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
camerfirmachamberscommerceca	21. April 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
verisignclass1ca	21. April 2018	CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45:3E:64:09:EA:E8:7D:60:F1

AWS AppSync Changelog der Resolver-Mapping-Vorlage

> **Note**
>
> Wir unterstützen jetzt hauptsächlich die APPSYNC_JS-Laufzeit und ihre Dokumentation. Bitte erwägen Sie, die APPSYNC_JS-Laufzeit und ihre Anleitungen hier zu verwenden.

Resolver- und Funktionszuweisungsvorlagen sind versioniert. Die Version der Mapping-Vorlage, z. B.`2018-05-29`), schreibt Folgendes vor:

- Die erwartete Form der Konfiguration der Datenquellenanforderung, die von der Anforderungsvorlage bereitgestellt wird
- Das Ausführungsverhalten der Anforderungszuordnungsvorlage und der Antwortzuordnungsvorlage

Versionen werden im YYYY-MM-DD Format dargestellt, ein späteres Datum entspricht einer neueren Version. Auf dieser Seite werden die Unterschiede zwischen den Versionen der Mapping-Vorlagen aufgeführt, die derzeit in unterstützt werden AWS AppSync.

Themen

- Matrix der Verfügbarkeit von Datenquellenoperationen pro Version

- Änderung der Version in einer Unit-Resolver-Zuweisungsvorlage
- Änderung der Version in einer Funktion
- 2018-05-29
- 2017-02-28

Matrix der Verfügbarkeit von Datenquellenoperationen pro Version

Operation/Unterstützte Version	2017-02-28	2018-05-29
AWS Lambda Aufrufen	Ja	Ja
AWS Lambda BatchInvoke	Ja	Ja
Keine Datenquelle	Ja	Ja
Amazon OpenSearch GET	Ja	Ja
Amazon OpenSearch POST	Ja	Ja
Amazon OpenSearch PUT	Ja	Ja
Amazon OpenSearch LÖSCHEN	Ja	Ja
Amazon OpenSearch GET	Ja	Ja
DynamoDB GetItem	Ja	Ja
DynamoDB Scan	Ja	Ja
DynamoDB Query	Ja	Ja
DynamoDB DeleteItem	Ja	Ja
DynamoDB PutItem	Ja	Ja
DynamoDB BatchGetItem	Nein	Ja

Operation/Unterstützte Version	2017-02-28	2018-05-29
DynamoDB BatchPutItem	Nein	Ja
DynamoDB BatchDeleteItem	Nein	Ja
HTTP	Nein	Ja
Amazon RDS	Nein	Ja

Hinweis: In Funktionen wird zurzeit nur die Version 2018-05-29 unterstützt.

Änderung der Version in einer Unit-Resolver-Zuweisungsvorlage

Für Unit-Resolver wird die Version als Teil des Texts der Zuweisungsvorlage für Anforderungen angegeben. Die Version aktualisieren Sie, indem Sie einfach das Feld `version` mit der neuen Version aktualisieren.

Um beispielsweise die Version der Vorlage zu aktualisieren: AWS Lambda

```
{
    "version": "2017-02-28",
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments": $utils.toJson($context.arguments)
    }
}
```

Sie müssen das Versionsfeld wie folgt von `2017-02-28` in `2018-05-29` aktualisieren:

```
{
    "version": "2018-05-29", ## Note the version
    "operation": "Invoke",
    "payload": {
        "field": "getPost",
        "arguments": $utils.toJson($context.arguments)
    }
}
```

Änderung der Version in einer Funktion

Für Funktionen wird die Version im Feld `functionVersion` des Funktionsobjekts angegeben. Um die Version zu aktualisieren, aktualisieren Sie einfach `functionVersion`: Hinweis: Zurzeit wird für die Funktion nur `2018-05-29` unterstützt.

Im Folgenden finden Sie ein Beispiel für einen CLI-Befehl zur Aktualisierung einer vorhandenen Funktionsversion:

```
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)"
```

Hinweis: Es wird empfohlen, das Versionsfeld der Zuweisungsvorlage für Anforderungen der Funktion leer zu lassen, da es nicht berücksichtigt wird. Wenn Sie in einer Zuweisungsvorlage für Anforderungen in einer Funktion eine Version angeben, wird der Versionswert durch den Wert des Feldes `functionVersion` überschrieben.

2018-05-29

Verhaltensänderung

- Wenn das Ergebnis des Aufrufs der Datenquelle `null` ist, wird die Zuweisungsvorlage für Antworten ausgeführt.
- Wenn der Aufruf der Datenquelle zu einem Fehler führt, müssen Sie sich jetzt um die Fehlerbehandlung kümmern, da das von der Zuweisungsvorlage für Antworten ausgewertete Ergebnis immer im Block `data` der GraphQL-Antwort platziert wird.

Reasoning

- `null` als Ergebnis eines Aufrufs hat eine Bedeutung und für einige Anwendungsfälle müssen `null`-Ergebnisse daher besonders behandelt werden. Eine Anwendung prüft möglicherweise, ob ein Datensatz in einer Amazon DynamoDB-Tabelle vorhanden ist, um einige Autorisierungsprüfungen durchzuführen. In diesem Fall würde das `null`-Ergebnis eines Aufrufs

dazu führen, dass der Benutzer möglicherweise nicht autorisiert werden kann. Das Ausführen der Zuweisungsvorlage für Antworten bietet jetzt die Möglichkeit, einen Fehler wegen fehlender Autorisierung auszulösen. Dieses Verhalten bietet mehr Kontrolle für API-Designer.

Nehmen wir die folgende Zuweisungsvorlage für Antworten:

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

Mit `2017-02-28` war es bisher so, dass die Zuweisungsvorlage für Antworten nicht ausgeführt wurde, wenn `$ctx.result` mit null zurückgegeben wurde. Mit `2018-05-29` können wir dieses Szenario jetzt behandeln. Wir können z. B. wie folgt einen Autorisierungsfehler auslösen:

```
# throw an unauthorized error if the result is null
#if ( $util.isNull($ctx.result) )
    $util.unauthorized()
#end
$util.toJson($ctx.result)
```

Hinweis: Fehler, die von einer Datenquelle zurückgegeben werden, sind manchmal nicht schwerwiegend oder werden sogar erwartet, daher sollte die Zuweisungsvorlage für Antworten die Flexibilität bieten, den Aufruffehler zu behandeln und zu entscheiden, ob er ignoriert oder erneut ausgelöst wird oder ob ein anderer Fehler auslöst wird.

Nehmen wir die folgende Zuweisungsvorlage für Antworten:

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

Mit `2017-02-28` wurde im Fall eines Aufruffehlers die Zuweisungsvorlage für Antworten ausgewertet und das Ergebnis automatisch im Block `errors` der GraphQL-Antwort platziert. Mit `2018-05-29` können Sie jetzt entscheiden, wie mit dem Fehler verfahren wird, ihn erneut auslösen oder ihn beim Zurückgegeben der Daten anfügen.

Erneutes Auslösen eines Aufruffehlers

In der folgenden Antwortvorlage lösen wir erneut den Fehler aus, der von der Datenquelle zurückgegeben wurde.

```
#if ( $ctx.error )
```

```
    $util.error($ctx.error.message, $ctx.error.type)
#end
$util.toJson($ctx.result)
```

Im Fall eines Aufruffehlers (wenn z. B. `$ctx.error` vorhanden ist) sieht der Antwortblock wie folgt aus:

```
{
    "data": {
        "getPost": null
    },
    "errors": [
        {
            "path": [
                "getPost"
            ],
            "errorType": "DynamoDB:ConditionalCheckFailedException",
            "message": "Conditional check failed exception...",
            "locations": [
                {
                    "line": 5,
                    "column": 5
                }
            ]
        }
    ]
}
```

Auslösen eines anderen Fehlers

In der folgenden Antwortvorlage lösen wir einen eigenen, benutzerdefinierten Fehler aus, nachdem der von der Datenquelle zurückgegebene Fehler verarbeitet wurde.

```
#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)
```

Im Fall eines Aufruffehlers (wenn z. B. `$ctx.error` vorhanden ist) sieht der Antwortblock wie folgt aus:

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

Anfügen eines Fehlers an Rückgabedaten

In der folgenden Antwortvorlage fügen wir den Fehler an, der von der Datenquelle zurückgegeben wurde, während wird die Daten in der Antwort zurückgeben. Dies wird auch als partielle Antwort bezeichnet.

```
#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
```

Im Fall eines Aufruffehlers (wenn z. B. `$ctx.error` vorhanden ist) sieht der Antwortblock wie folgt aus:

```
{
    "data": {
        "getPost": {
            "id": "1",
            "title: "A post"
        }
    },
    "errors": [
        {
            "path": [
                "getPost"
            ],
            "errorType": "ConditionalCheckFailedException",
            "message": "Conditional check failed exception...",
            "locations": [
                {
                    "line": 5,
                    "column": 5
                }
            ]
        }
    ]
}
```

Migration von 2017-02-28 zu 2018-05-29

Die Migration von 2017-02-28 zu 2018-05-29 ist ganz einfach. Ändern Sie das Versionsfeld der Resolver-Zuweisungsvorlage für Anforderungen oder das Versionsobjekt der Funktion. Beachten Sie jedoch, dass 2018-05-29 ein anderes Ausführungsverhalten zeigt als 2017-02-28. Die Änderungen werden hier erläutert.

Beibehalten des gleichen Ausführungsverhaltens von 2017-02-28 zu 2018-05-29

In einigen Fällen ist es möglich, das Ausführungsverhalten von 2017-02-28 beizubehalten, während eine mit 2018-05-29 versionierte Vorlage ausgeführt wird.

Beispiel: DynamoDB PutItem

Angesichts der folgenden PutItem DynamoDB-Anforderungsvorlage vom 28.02.2017:

```
{
    "version" : "2017-02-28",
    "operation" : "PutItem",
    "key": {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "attributeValues" : {
        "baz" : ... typed value
    },
    "condition" : {
       ...
    }
}
```

Und die folgende Antwortvorlage:

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

Bei der Migration zu 2018-05-29 werden diese Vorlagen wie folgt verändert:

```
{
    "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" : {
       ...
    }
}
```

Und die Antwortvorlage wird wie folgt verändert:

```
## 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)
```

Da es jetzt in Ihrer Verantwortung liegt, die Fehler zu behandeln, lösen wird den von DynamoDB zurückgegebenen Fehler mithilfe von `$util.error()` noch einmal aus. Sie können dieses Snippet anpassen, um Ihre Zuweisungsvorlage in 2018-05-29 zu konvertieren. Beachten Sie, dass Sie bei einer anderen Antwortvorlage die Änderungen des Ausführungsverhaltens berücksichtigen müssen.

Beispiel: DynamoDB GetItem

Angesichts der folgenden GetItem DynamoDB-Anforderungsvorlage vom 28.02.2017:

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "consistentRead" : true
}
```

Und die folgende Antwortvorlage:

```
## map table attribute postId to field Post.id
$util.qr($ctx.result.put("id", $ctx.result.get("postId")))

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

Bei der Migration zu 2018-05-29 werden diese Vorlagen wie folgt verändert:

```
{
    "version" : "2018-05-29", ## Note the new 2018-05-29 version
    "operation" : "GetItem",
    "key" : {
```

```
        "foo" : ... typed value,
        "bar" : ... typed value
    },
    "consistentRead" : true
}
```

Und die Antwortvorlage wird wie folgt verändert:

```
## 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)
```

In Version 2017-02-28 wurde die Zuweisungsvorlage für Antworten nicht ausgeführt, wenn der Aufruf der Datenquelle `null` ergeben hatte, was der Fall ist, wenn kein Element in der DynamoDB-Tabelle mit unserem Schlüssel übereinstimmt. Das kann in den meisten Fällen kein Problem sein, wenn `$ctx.result` jedoch nicht `null` sein darf, müssen Sie dieses Szenario jetzt behandeln.

2017-02-28

Merkmale

- Wenn das Ergebnis des Aufrufs der Datenquelle `null` ist, wird die Zuweisungsvorlage für Antworten nicht ausgeführt.
- Wenn der Aufruf der Datenquelle zu einem Fehler führt, wird die Zuweisungsvorlage für Antworten ausgeführt und das ausgewertete Ergebnis wird im Block `errors.data` der GraphQL-Antwort platziert.

GraphQL-Typreferenz

Skalare Typen in GraphQL repräsentieren primitive Blattwerte in einem GraphQL-Schema. Dies sind die grundlegendsten Datentypen, die zu einem einzigen Wert aufgelöst werden. Im Gegensatz zu Objekttypen können Skalartypen keine Unterfelder haben. GraphQL enthält eine Reihe von standardmäßigen Skalartypen:

- Int: Eine 32-Bit-Ganzzahl mit Vorzeichen
- Float: Ein Gleitkommawert mit doppelter Genauigkeit mit Vorzeichen
- Zeichenfolge: Eine UTF-8-Zeichenfolge
- Boolean: Ein wahrer oder falscher Wert
- ID: Eine eindeutige Kennung, die häufig zum erneuten Abrufen eines Objekts oder als Schlüssel für einen Cache verwendet wird

Diese skalaren Typen dienen als Bausteine für komplexere Typen in Ihrem Schema. Sie werden verwendet, um Felder zu definieren, die einfache, singuläre Werte enthalten. Zusätzlich zu diesen integrierten Skalaren stehen Ihnen zusätzliche Skalare für verschiedene Anwendungsfälle zur AWS AppSync Verfügung.

Schnittstellen und Unions in GraphQL sind abstrakte Typen, die ein flexibles und erweiterbares Schemadesign ermöglichen. Sie bieten Mechanismen zum Gruppieren verwandter Typen und ermöglichen polymorphe Abfragen. Eine Schnittstelle in GraphQL ist ein abstrakter Typ, der eine Reihe von Feldern definiert, die ein Typ enthalten muss, um die Schnittstelle zu implementieren. Es dient als Vertrag für Objekte, indem es einen gemeinsamen Satz von Feldern spezifiziert, über den implementierende Typen verfügen müssen. Schnittstellen sind nützlich, wenn Sie ein Objekt oder ein Feld zurückgeben möchten, das mehrere verschiedene Typen haben kann, aber dennoch einige garantierte Felder hat. Im Gegensatz dazu stellt eine Union in GraphQL einen Typ dar, der einer von mehreren Objekttypen sein könnte, definiert aber keine gemeinsamen Felder zwischen diesen Typen. Unions sind hilfreich, wenn Sie ein Feld zurückgeben müssen, das mehrere Typen haben kann, und diese Typen haben nicht unbedingt gemeinsame Felder. Sowohl Interfaces als auch Unions sind besonders nützlich in Szenarien, in denen ein Feld möglicherweise unterschiedliche Datentypen zurückgibt, sodass Clients auf der Grundlage des zurückgegebenen Typs nach bestimmten Feldern abfragen können.

Dieser Abschnitt dient als Referenz für Schematypen.

Topics

- Skalare Typen in GraphQL
- Schnittstellen und Unions in GraphQL

Skalare Typen in GraphQL

Ein GraphQL-Objekttyp hat einen Namen und Felder, und diese Felder können Unterfelder haben. Letztlich müssen die Felder eines Objekttyps in skalare Typen aufgelöst werden, die die Blätter der Abfrage darstellen. Weitere Informationen zu Objekttypen und Skalaren finden Sie unter Schemas and types auf der GraphQL-Website.

Zusätzlich zum Standardsatz von GraphQL-Skalaren können Sie damit AWS AppSync auch die dienstdefinierten Skalare verwenden, die mit dem Präfix beginnen. AWS AWS AppSync unterstützt nicht die Erstellung von benutzerdefinierten (benutzerdefinierten) Skalaren. Sie müssen entweder den Standard oder AWSSkalare verwenden.

Sie können es nicht AWSals Präfix für benutzerdefinierte Objekttypen verwenden.

Der folgende Abschnitt ist eine Referenz für die Schematypisierung.

Standard-Skalare

GraphQL definiert die folgenden Standardskalare:

Liste der Standard-Skalare

`ID`

Ein eindeutiger Bezeichner für ein Objekt. Dieser Skalar ist wie ein serialisiert, soll `String` aber nicht für Menschen lesbar sein.

`String`

Eine UTF-8-Zeichenfolge.

`Int`

Ein ganzzahliger Wert zwischen - (2^{31}) und 2^{31} -1.

`Float`

Ein IEEE 754-Gleitkommawert.

`Boolean`

Ein boolescher Wert, entweder `true` oder `false`.

AWS AppSync Skalare

AWS AppSync definiert die folgenden Skalare:

AWS AppSync Liste der Skalare

`AWSDate`

Eine erweiterte ISO 8601-Datumszeichenfolge im Format. `YYYY-MM-DD`

`AWSTime`

Eine erweiterte ISO 8601-Zeitzeichenfolge im Format. `hh:mm:ss.sss`

`AWSDateTime`

Eine erweiterte ISO 8601-Datums- und Uhrzeitzeichenfolge im Format. `YYYY-MM-DDThh:mm:ss.sssZ`

> **Note**
>
> Die `AWSDateTime` Skalare `AWSDateAWSTime`, und können optional einen Zeitzonen-Offset enthalten. Zum Beispiel `1970-01-01+05:30` sind die Werte `1970-01-01Z1970-01-01-07:00`, und alle gültig für`AWSDate`. Der Zeitzonen-Offset muss entweder `Z` (UTC) oder ein Offset in Stunden und Minuten (und optional in Sekunden) sein. Beispiel, `±hh:mm:ss`. Das Sekundenfeld im Zeitzonen-Offset wird als gültig angesehen, obwohl es nicht Teil des ISO 8601-Standards ist.

`AWSTimestamp`

Ein ganzzahliger Wert, der die Anzahl der Sekunden davor oder danach `1970-01-01-T00:00Z` angibt.

`AWSEmail`

Eine E-Mail-Adresse in dem in RFC 822 definierten Format`local-part@domain-part`.

`AWSJSON`

Eine JSON-Zeichenfolge. Jedes gültige JSON-Konstrukt wird automatisch analysiert und als Zuordnungen, Listen oder Skalarwerte und nicht als wörtliche Eingabezeichenfolgen in den Resolver-Code geladen. Zeichenketten ohne Anführungszeichen oder anderweitig ungültiges JSON führen zu einem GraphQL-Validierungsfehler.

`AWSPhone`

Eine Telefonnummer. Dieser Wert wird als Zeichenfolge gespeichert. Telefonnummern können entweder Leerzeichen oder Bindestriche enthalten, um Zifferngruppen voneinander zu trennen. Bei Telefonnummern ohne Landesvorwahl wird davon ausgegangen, dass es sich um US-/nordamerikanische Nummern handelt, die dem North American Numbering Plan (NANP) entsprechen.

`AWSURL`

Eine URL gemäß der Definition in RFC 1738. Zum Beispiel oder. `https://www.amazon.com/dp/B000NZW3KC/` `mailto:example@example.com` URLsmuss ein Schema (`http`,`mailto`) enthalten und darf nicht zwei Schrägstriche (`//`) im Pfadteil enthalten.

`AWSIPAddress`

Eine gültige IPv4 IPv6 Oder-Adresse. IPv4 Adressen werden in Vierpunkt-Notation () erwartet. `123.12.34.56` IPv6 Adressen werden in einem Format ohne Klammern und durch Doppelpunkte getrennt erwartet (). `1a2b:3c4b::1234:4567` Sie können ein optionales CIDR-Suffix (`123.45.67.89/16`) angeben, um die Subnetzmaske anzugeben.

Beispiel für die Verwendung eines Schemas

Das folgende GraphQL-Beispielschema verwendet alle benutzerdefinierten Skalare als „Objekt“ und zeigt die Resolver-Anforderungs- und Antwortvorlagen für grundlegende Put-, Get- und List-Operationen. Schließlich zeigt das Beispiel, wie Sie dies beim Ausführen von Abfragen und Mutationen verwenden können.

```
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
}
```

So `putObject` könnte eine Anforderungsvorlage für aussehen. A `putObject` verwendet eine `PutItem` Operation, um ein Element in Ihrer Amazon DynamoDB-Tabelle zu erstellen oder zu aktualisieren. Beachten Sie, dass dieser Codeausschnitt keine konfigurierte Amazon DynamoDB-Tabelle als Datenquelle hat. Dies wird nur als Beispiel verwendet:

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

```
}
```

Die Antwortvorlage für `putObject` gibt die Ergebnisse zurück:

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

So `getObject` könnte eine Anforderungsvorlage für aussehen. A `getObject` verwendet eine `GetItem` Operation, um eine Reihe von Attributen für das Element zurückzugeben, wenn der Primärschlüssel angegeben ist. Beachten Sie, dass dieser Codeausschnitt keine konfigurierte Amazon DynamoDB-Tabelle als Datenquelle hat. Dies wird nur als Beispiel verwendet:

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

Die Antwortvorlage für `getObject` gibt die Ergebnisse zurück:

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

So `listObjects` könnte eine Anforderungsvorlage für aussehen. A `listObjects` verwendet eine `Scan` Operation, um ein oder mehrere Elemente und Attribute zurückzugeben. Beachten Sie, dass dieser Codeausschnitt keine konfigurierte Amazon DynamoDB-Tabelle als Datenquelle hat. Dies wird nur als Beispiel verwendet:

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

Die Antwortvorlage für `listObjects` gibt die Ergebnisse zurück:

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

Im Folgenden finden Sie einige Beispiele für die Verwendung dieses Schemas mit GraphQL-Abfragen:

```
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
    }
}
```

Schnittstellen und Unions in GraphQL

Das GraphQL-Typsystem unterstützt Schnittstellen. Eine Schnittstelle stellt eine bestimmte Gruppe von Feldern dar, die ein Typ umfassen muss, um die Schnittstelle zu implementieren.

Das GraphQL-Typsystem unterstützt auch Unions. Vereinigungen sind weitgehend identisch mit Schnittstellen, definieren aber keinen gemeinsamen Satz von Feldern. Vereinigungen werden in der Regel gegenüber Schnittstellen bevorzugt, wenn die möglichen Typen keine gemeinsame logische Hierarchie besitzen.

Der folgende Abschnitt ist eine Referenz für die Schematypisierung.

Beispiele für Benutzeroberflächen

Wir könnten eine `Event` Schnittstelle darstellen, die jede Art von Aktivität oder Zusammenkunft von Menschen darstellt. Einige mögliche Ereignistypen sind `ConcertConference`,, und`Festival`. All diese Typen haben gemeinsame Merkmale: Name, Veranstaltungsort sowie Start- und Enddatum. Auch bei diesen Typen gibt es Unterschiede. Bei einer `Conference` handelt es sich um eine Liste von Rednern und Workshops, bei `Concert` einer tritt eine Band auf.

In der Schema Definition Language (SDL) ist die `Event` Schnittstelle wie folgt definiert:

```
interface Event {
        id: ID!
        name : String!
        startsAt: String
        endsAt: String
        venue: Venue
        minAgeRestriction: Int
}
```

Und jeder der Typen implementiert die `Event` Schnittstelle wie folgt:

```
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]
}
```

Schnittstellen sind nützlich zur Darstellung von Elementen, die in Form unterschiedlicher Typen auftreten können. Zum Beispiel könnten wir nach allen Ereignissen an einem bestimmten Veranstaltungsort suchen. Fügen wir nun folgendermaßen ein `findEventsByVenue`-Feld zum Schema hinzu:

```
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]
}
```

Das `findEventsByVenue` gibt eine Liste von zurück`Event`. Da GraphQL-Schnittstellenfelder von allen Implementierungstypen verwendet werden, können Sie jedes beliebige Feld der `Event`-

Schnittstelle auswählen (`id`, `name`, `startsAt`, `endsAt`, `venue` und `minAgeRestriction`). Alternativ können Sie mit GraphQL-Fragmenten auf die Felder jedes Implementierungstyps zugreifen, sofern Sie den Typ angeben.

Schauen wir uns ein Beispiel für eine GraphQL-Abfrage an, die die Schnittstelle verwendet.

```
query {
  findEventsAtVenue(venueId: "Madison Square Garden") {
    id
    name
    minAgeRestriction
    startsAt

    ... on Festival {
      performers
    }

    ... on Concert {
      performingBand
    }

    ... on Conference {
      speakers
      workshops
    }
  }
}
```

Die vorherige Abfrage gibt eine einzige Ergebnisliste aus und der Server kann die Ereignisse standardmäßig nach Startdatum sortieren.

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

Da Ergebnisse als eine einzelne Sammlung von Ereignissen zurückgegeben werden, ist die Verwendung von Schnittstellen zur Darstellung gemeinsamer Merkmale für die Sortierung der Ergebnisse sehr hilfreich.

Beispiele für Vereinigungen

Wie bereits erwähnt, definieren Gewerkschaften keine gemeinsamen Feldgruppen. Ein Suchergebnis kann viele verschiedene Typen repräsentieren. Mit dem `Event`-Schema können Sie eine `SearchResult`-Vereinigung definieren. Gehen Sie wie folgt vor:

```
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
```

In diesem Fall müssen Sie Fragmente verwenden, um ein beliebiges Feld in unserer `SearchResult` Union abzufragen:

```
query {
  search(query: "Madison") {
    ... on Venue {
      id
      name
      address
    }

    ... on Festival {
      id
      name
      performers
    }

    ... on Concert {
      id
      name
      performingBand
    }

    ... on Conference {
      speakers
      workshops
    }
  }
}
```

Geben Sie Auflösung ein AWS AppSync

Die Typenauflösung ist der Mechanismus, durch den die GraphQL-Engine einen aufgelösten Wert als bestimmten Objekttyp identifiziert.

Zurück zum Beispiel für die Union-Suche: Vorausgesetzt, dass unsere Abfrage Ergebnisse liefert, muss jedes Element in der Ergebnisliste als einer der möglichen Typen dargestellt werden, die von der `SearchResult` Union definiert wurden (d. h.`Conference`,, `FestivalConcert`, oder`Venue`).

Da die Logik zum Unterscheiden zwischen `Festival`, `Venue` und `Conference` von den Anwendungsanforderungen abhängt, muss die GraphQL-Engine Informationen erhalten, um unsere möglichen Typen in den Raw-Ergebnissen zu identifizieren.

Mit AWS AppSync wird dieser Hinweis durch ein Metafeld mit dem Namen dargestellt`__typename`, dessen Wert dem Namen des identifizierten Objekttyps entspricht. `__typename`ist für Rückgabetypen erforderlich, bei denen es sich um Interfaces oder Unions handelt.

Beispiel für die Typauflösung

Verwenden wir erneut das vorherige Schema. Sie können dies reproduzieren, indem Sie zur Konsole gehen und Folgendes unter der Seite Schema hinzufügen:

```
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
}
```

Hängen wir nun einen Resolver an das `Query.search`-Feld an. Wählen Sie in dem `Resolvers` Abschnitt Anhängen aus, erstellen Sie eine neue Datenquelle vom Typ NONE, und geben Sie ihr dann einen Namen StubDataSource. Für dieses Beispiel nehmen wir an, wir haben die Ergebnisse aus einer externen Quelle abgerufen und codieren die abgerufenen Ergebnisse fest in unserer Zuweisungsvorlage für Anforderungen.

Geben Sie im Bereich der Anforderungszuweisungsvorlage Folgendes ein:

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

Wenn die Anwendung den Typnamen als Teil des `id` Felds zurückgibt, muss die Typauflösungslogik das `id` Feld analysieren, um den Typnamen zu extrahieren, und dann das `__typename` Feld zu jedem der Ergebnisse hinzufügen. Sie können die Logik in der Antwortzuweisungsvorlage folgendermaßen umsetzen:

> **Note**
>
> Sie können diese Aufgabe auch als Teil Ihrer Lambda-Funktion ausführen, wenn Sie die Lambda-Datenquelle verwenden.

```
#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)
```

Führen Sie die folgende Abfrage aus:

```
query {
  search(query: "Madison") {
    ... on Venue {
      id
      name
      address
    }

    ... on Festival {
        id
      name
      performers
    }

    ... on Concert {
      id
      name
      performingBand
    }

    ... on Conference {
      speakers
      workshops
    }
  }
}
```

Die Abfrage liefert die folgenden Ergebnisse:

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

Die Typenauflösungslogik variiert je nach Anwendung. Sie können beispielsweise eine andere Identifizierungslogik verwenden, die prüft, ob bestimmte Felder oder auch Kombinationen von Feldern vorhanden sind. Sie können also feststellen, ob das `performers`-Feld vorhanden ist, um ein `Festival` zu identifizieren, oder die Kombination der Felder `speakers` und `workshops`, um eine `Conference` zu identifizieren. Letztlich liegt es an Ihnen, die Logik zu definieren, die Sie verwenden möchten.

Problembehebung und häufige Fehler in AWS AppSync

In diesem Abschnitt wird auf einige häufige Fehler eingegangen und erläutert, wie diese möglicherweise behoben werden können.

Falsche DynamoDB-Schlüsselzuweisung

Wenn Ihr GraphQL-Vorgang die folgende Fehlermeldung zurückgibt, kann das daran liegen, dass Ihre Vorlagenstruktur für die Anforderungszuweisung nicht mit der Amazon DynamoDB DynamoDB-Schlüsselstruktur übereinstimmt:

```
The provided key element does not match the schema (Service: AmazonDynamoDBv2; Status
 Code: 400; Error Code
```

Wenn Ihre DynamoDB-Tabelle beispielsweise einen Hash-Schlüssel hat `"id"` und Ihre Vorlage sagt`"PostID"`, wie im folgenden Beispiel, führt dies zu dem vorherigen Fehler, weil `"id"` nicht übereinstimmt. `"PostID"`

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

Fehlender Resolver

Wenn Sie eine GraphQL-Operation ausführen, wie z. B. eine Abfrage, und eine Null-Antwort erhalten, kann dies daran liegen, dass Sie keinen Resolver konfiguriert haben.

Wenn Sie beispielsweise ein Schema importieren, das ein `getCustomer(userId: ID!):-` Feld definiert, und Sie für dieses Feld keinen Resolver konfiguriert haben, erhalten Sie z. B. für die Abfrage `getCustomer(userId:"ID123"){...}` eine Antwort, die ungefähr der nachstehenden entspricht:

```
{
```

```
    "data": {
    "getCustomer": null
    }
}
```

Fehler bei der Zuordnungsvorlage

Wenn Ihre Zuweisungsvorlage nicht korrekt konfiguriert ist, erhalten Sie eine GraphQL-Antwort, deren `errorType` `MappingTemplate` ist. Das `message`-Feld sollte darauf hinweisen, wo das Problem in Ihrer Zuweisungsvorlage liegt.

Wenn Sie beispielsweise in Ihrer Zuweisungsvorlage über kein `operation`-Feld verfügen, oder der `operation`-Feldname nicht korrekt angegeben wurde, erhalten Sie in etwa folgende Antwort:

```
{
    "data": {
        "searchPosts": null
    },
    "errors": [
        {
        "path": [
            "searchPosts"
        ],
        "errorType": "MappingTemplate",
        "locations": [
            {
            "line": 2,
            "column": 3
            }
        ],
        "message": "Value for field '$[operation]' not found."
        }
    ]
}
```

Falsche Rückgabetypen

Der Rückgabetyp aus der Datenquelle muss dem definierten Typ eines Objekts in Ihrem Schema entsprechen, andernfalls wird Ihnen möglicherweise ein GraphQL-Fehler angezeigt, der dem folgenden Beispiel entspricht:

```
"errors": [
    {
    "path": [
        "posts"
    ],
    "locations": null,
    "message": "Can't resolve value (/posts) : type mismatch error, expected type LIST,
 got OBJECT"
    }
]
```

Dies könnte beispielsweise bei der folgenden Abfragedefinition auftreten:

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

Hierbei wird eine LISTE mit `[Posts]`-Objekten erwartet. Wenn Sie beispielsweise eine Lambda-Funktion in Node.JS mit ungefähr folgendem Inhalt hätten:

```
const result = { data: data.Items.map(item => { return item ; }) };
callback(err, result);
```

Dies würde einen Fehler zurückgeben, da es sich bei `result` um ein Objekt handelt. In diesem Fall müssten Sie entweder die Callback-Funktion zu `result.data` ändern oder Ihr Schema so anpassen, dass keine LISTE zurückgegeben wird.

Ungültige Anfragen werden verarbeitet

Wenn AWS AppSync eine Anfrage nicht verarbeitet und nicht an den Field Resolver gesendet werden kann (aufgrund falscher Daten, z. B. einer ungültigen Syntax), gibt die Antwortnutzlast die Felddaten mit den auf gesetzten Werten `null` und allen relevanten Fehlern zurück.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.