

Guia do Desenvolvedor

AWS AppSync GraphQL



AWS AppSync GraphQL: Guia do Desenvolvedor

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

As marcas comerciais e imagens de marcas da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestige a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

O que é AWS AppSync	1
AWS AppSync features	1
Você é um AWS AppSync usuário iniciante?	2
Serviços relacionados	2
Preços para AWS AppSync	3
GraphQL e arquitetura AWS AppSync	4
O que é uma API	5
Clientes	5
Recursos	5
O que é REST	6
Interface uniforme	6
Ausência de estado	7
Sistema em camadas	7
Capacidade de armazenamento em cache	7
O que é uma RESTful API?	7
Como RESTful APIs funciona?	8
O que é GraphQL	8
Comparação entre REST e GraphQL	8
Por que usar GraphQL em vez de REST	11
Componentes de uma API GraphQL	13
Esquemas do GraphQL	13
Fontes de dados	32
Resolvedores	49
Propriedades adicionais do GraphQL	59
Declarativo	59
Hierárquico	60
Introspectivo	61
Tipificação forte	62
Conceitos básicos do : como criar sua primeira API do GraphQL	63
Como iniciar um esquema	64
Fazendo um tour pelo AWS AppSync console	68
Designer do esquema	68
Fontes de dados	69
Consultas	70

Configurações	70
Uso de mutações do GraphQL para adicionar dados a uma tabela do DynamoDB	71
Uso de consultas do GraphQL para recuperar dados de uma tabela do DynamoDB	76
Seções complementares	79
Integração	79
Leitura complementar	80
Projetando o GraphQL APIs	81
Estruturação de uma API GraphQL (em branco ou importada) APIs	81
Como projetar seu esquema GraphQL	82
Como anexar uma fonte de dados	110
Configurando resolvedores AWS AppSync	122
Usando APIs com o CDK	179
Usando assinaturas para aplicativos de dados em tempo real	197
Diretivas de assinatura do esquema do GraphQL	197
Usar argumentos de assinatura	200
Criação de genéricos pub/sub APIs com tecnologia sem servidor WebSockets	204
Definição de filtros de assinatura avançados	208
Cancelando a assinatura de WebSocket conexões usando filtros	219
Construindo um WebSocket cliente em tempo real	223
Mesclando APIs	245
Fusão APIs e federação	247
Resolução de conflitos de API mesclada	249
Configurar esquemas	257
Configurar modos de autorização	257
Configurar perfis de execução	258
Configurando contas cruzadas mescladas usando APIs AWS RAM	260
Mesclar	262
Suporte adicional para Merged APIs	263
Limitações de APIs mescladas	264
Criando mesclado APIs	264
Construindo o GraphQL APIs com introspecção do RDS	266
Usar o recurso de introspecção (console)	267
Usar o recurso de introspecção (API)	271
Como criar um aplicativo do cliente	274
JavaScript tutoriais de resolvedores	277

Criação de um aplicativo de postagem simples usando resolvedores do DynamoDB

JavaScript	277
Criação da API GraphQL	278
Definição de uma API Post básica	278
Configuração de sua tabela do Amazon DynamoDB	279
Configurando um resolvedor AddPost (Amazon DynamoDB) PutItem	280
Configuração do resolvedor GetPost (Amazon DynamoDB) GetItem	283
Crie uma mutação UpdatePost (Amazon DynamoDB) UpdateItem	286
Crie mutações de votos (Amazon DynamoDB) UpdateItem	291
Configurando um resolvedor DeletePost (Amazon DynamoDB) DeleteItem	294
Configurar o resolvedor allPost (Scan do Amazon DynamoDB)	300
Configurando um resolvedor de allPostsBy autor (Amazon DynamoDB Query)	304
Uso de conjuntos	310
Conclusão	317
Usando AWS Lambda resolvedores	317
Criar uma função do Lambda	317
Configurar a fonte de dados para o Lambda	319
Criar um esquema do GraphQL	320
Configurar resolvedores	124
Teste da sua API GraphQL	322
Retornar erros	323
Caso de uso avançado: agrupamento em lotes	326
Uso de resolvedores locais	336
Criação do aplicativo pub/sub	336
Enviar e assinar mensagens	337
Combinação de resolvedores do GraphQL	338
Esquema de exemplo	338
Alteração de dados por meio de resolvedores	340
DynamoDB e serviços OpenSearch	340
Usando resolvedores OpenSearch de serviço	343
Criar um novo domínio OpenSearch de serviço	343
Configurar uma fonte de dados para o OpenSearch serviço	344
Conexão de um resolvedor	346
Modificação das pesquisas	348
Adicionando dados ao OpenSearch serviço	349
Recuperação de um único documento	350

Executar consultas e mutações	351
Práticas recomendadas	352
Execução de transações do DynamoDB	352
Permissões	352
Fonte de dados	353
Transações	355
Uso de operações em lote do DynamoDB	362
Lotes de tabela única	363
Lote de várias tabelas	368
Tratamento de erros	376
Uso de resolvedores HTTP	382
Criar uma API REST	382
Criação da API GraphQL	383
Criar um esquema do GraphQL	383
Configurar a fonte de dados HTTP	384
Configuração de resolvedores do	156
Invocando serviços AWS	388
Uso do Aurora PostgreSQL com a API de dados	389
Configurar seu banco de dados Aurora PostgreSQL	390
Criar o banco de dados e uma tabela	391
Criar um esquema do GraphQL	392
Resolvedores para RDS	394
Excluir o cluster	402
Tutoriais do resolvedor VTL	404
Criação de uma aplicação de postagem simples usando resolvedores do DynamoDB	405
Configuração de tabelas do DynamoDB	405
Criação da API GraphQL	383
Definição de uma API Post básica	407
Configuração da fonte de dados para as tabelas do DynamoDB	408
Configurando o resolvedor AddPost (DynamoDB) PutItem	409
Configurando o GetPost Resolver (DynamoDB) GetItem	414
Criar uma mutação UpdatePost (DynamoDB) UpdateItem	417
Modificando o UpdatePost Resolver (DynamoDB) UpdateItem	421
Crie mutações UpVotePost e DownVotePost (DynamoDB) UpdateItem	427
Configurando o DeletePost Resolver (DynamoDB) DeleteItem	431
Configurar o resolvedor allPost (Scan do DynamoDB)	438

Configurando o allPostsBy Author Resolver (DynamoDB Query)	442
Uso de conjuntos	310
Uso de listas e mapas	456
Conclusão	459
Usando AWS Lambda resolvedores	460
Criar uma função do Lambda	460
Configurar a fonte de dados para o Lambda	462
Criar um esquema do GraphQL	383
Configurar resolvedores	156
Teste da sua API GraphQL	466
Retornar erros	468
Caso de uso avançado: agrupamento em lotes	470
Usando resolvedores OpenSearch de serviço	481
Configuração com um clique	481
Criar um novo domínio OpenSearch de serviço	482
Configurar fonte de dados para OpenSearch serviço	482
Conexão de um resolvedor	484
Modificação das pesquisas	486
Adicionando dados ao OpenSearch serviço	487
Recuperação de um único documento	488
Executar consultas e mutações	489
Práticas recomendadas	490
Uso de resolvedores locais	490
Criar o aplicativo de paginação	490
Enviar e assinar páginas	492
Combinação de resolvedores do GraphQL	492
Esquema de exemplo	493
Alterar os dados por meio de resolvedores	494
DynamoDB e serviços OpenSearch	495
Uso de operações em lote do DynamoDB	499
Permissões	499
Fonte de dados	500
Lote de tabela única	501
Lote de várias tabelas	505
Tratamento de erros	513
Execução de transações do DynamoDB	519

Permissões	499
Fonte de dados	500
Transações	521
Uso de resolvedores HTTP	531
Configuração com um clique	481
Criar uma API REST	382
Criação da API GraphQL	383
Criar um esquema do GraphQL	383
Configurar a fonte de dados HTTP	384
Configurar resolvedores	156
Invocando serviços AWS	537
Usando resolvedores Aurora Serverless v2	539
Configurando seu cluster de banco de dados	540
Ativar API de dados	541
Criar banco de dados e tabela	542
Esquema do GraphQL	542
Conecte sua API às operações do banco de dados	156
Modifique seus dados por meio da API	551
Recupere seus dados	553
Proteja seu acesso aos dados	554
Uso de resolvedores de pipeline	556
Configuração com um clique	481
Configuração manual	557
Teste da API GraphQL	466
Uso de operações de sincronização delta em fontes de dados versionadas	571
Configuração com um clique	481
Schema	572
Mutações	575
Consultas de sincronização	575
Exemplo	575
Configuração e definições	582
Como configurar o cache do lado do servidor e a compactação da carga útil da API	583
Tipos de instância	583
Comportamento de armazenamento em cache	584
Criptografia de cache	587
Remoção de cache	587

Remover uma entrada de cache	587
Remover uma entrada de cache com base na identidade	589
Compactar respostas da API	590
Configurando nomes de domínio personalizados para GraphQL e em tempo real APIs	591
Registrar e configurar um nome de domínio	592
Criando um nome de domínio personalizado no AWS AppSync	592
Nomes de domínio personalizados curinga em AWS AppSync	594
Versionamento, detecção de conflitos e operações de sincronização para DynamoDB	594
Versionamento de fontes de dados do DynamoDB	595
Deteção e resolução de conflitos	599
Uso de operações de sincronização do DynamoDB em fontes de dados versionadas	610
Usando CloudWatch para monitorar e registrar dados da API GraphQL	610
Definição e configuração	611
CloudWatch métricas	613
CloudWatch troncos	624
Referência de tipo de log	629
Analisando seus registros com o CloudWatch Logs Insights	632
Análise seus registros com o OpenSearch Service	633
Migração do formato de log	634
Rastreando solicitações em AWS X-Ray	634
Definição e configuração	611
Rastreamento da sua API com o X-Ray	635
Registro de chamadas de API do AWS CloudTrail com	637
AWS AppSync informações em CloudTrail	637
AWS AppSync eventos de dados em CloudTrail	638
Entendendo as entradas do arquivo de AWS AppSync log	640
Usando o AWS AppSync Private APIs	648
Criação de AWS AppSync dados privados APIs	650
Criar um endpoint da interface para o AWS AppSync	651
Exemplos avançados	652
Usar políticas do IAM para limitar a criação de API pública	656
Compartilhando o GraphQL APIs	657
Pré-requisitos para compartilhar o GraphQL AWS AppSync APIs	657
Compartilhe AWS AppSync GraphQL APIs	658
Pare de compartilhar o AWS AppSync GraphQL APIs	660
Eventos entre contas	661

Como configurar a complexidade de execução, a profundidade das consultas e a introspecção do GraphQL	661
Usar o recurso de introspecção	661
Configurar limites de profundidade de consultas	663
Configurar limites de contagem de resolvedores	664
Usar variáveis de ambiente do	666
Configurar variáveis de ambiente (console)	667
Configurar variáveis de ambiente (API)	668
Configuração de variáveis de ambiente (CFN)	669
variáveis de ambiente e mescladas APIs	669
Recuperar variáveis de ambiente	670
Autorizando e autenticando o GraphQL APIs	672
Tipos de autorização	672
API_KEY autorização	673
AWS_LAMBDA autorização	675
Como contornar as limitações de autorização de tokens do SigV4 e OIDC	681
AWS_IAM autorização	682
OPENID_CONNECT autorização	684
Autorização do AMAZON_COGNITO_USER_POOLS	686
Usar modos de autorização adicionais	687
Controle de acesso refinado	690
Filtrar informações	692
Acesso à fonte de dados	693
Casos de uso de controle de acesso para proteger solicitações e respostas	694
Visão geral	694
Leitura de dados	695
Gravação de dados	699
Registros públicos e privados	702
Dados em tempo real	703
Usando AWS WAF para proteger APIs	706
Integre uma AppSync API com AWS WAF	707
Criar regras para uma ACL da Web	709
Segurança	713
Proteção de dados	714
Criptografia em movimento	715
Validação de conformidade	715

Segurança da infraestrutura	716
Resiliência	717
Gerenciamento de identidade e acesso	717
Público	718
Autenticação com identidades	719
Gerenciar o acesso usando políticas	722
Como AWS AppSync funciona com o IAM	725
Políticas baseadas em identidade	732
Solução de problemas	744
Registrando chamadas de AWS AppSync API com AWS CloudTrail	747
AWS AppSync informações em CloudTrail	748
Entendendo as entradas do arquivo de AWS AppSync log	749
Práticas recomendadas	490
Entenda os métodos de autenticação	751
Entenda como as alterações na configuração da API se propagam	752
Usar o TLS para resolvedores HTTP	752
Usar perfis com o mínimo de permissões possível	752
Práticas recomendadas das políticas do IAM	752
Referência do resolvedor (JavaScript)	754
JavaScript visão geral dos resolvedores	754
Atributos compatíveis de runtime	755
Resolvedores de unidade	755
Anatomia de um resolvedor de JavaScript pipeline	755
Tópicos complementares	760
Exemplo de resolvedor de pipeline com o Amazon DynamoDB	760
Configuração de utilitários para o runtime do APPSYNC_JS	763
Agrupamento e mapas TypeScript de origem para o tempo de execução APPSYNC_JS	766
Como testar seu resolvedor e manipuladores de função	773
Migrando da VTL para JavaScript	775
Escolha entre acesso direto à fonte de dados e proxy por meio de uma fonte de dados do Lambda	779
JavaScript referência de objeto de contexto do resolvedor	781
Acesso ao context	781
JavaScript recursos de tempo de execução para resolvedores e funções	792
Atributos compatíveis de runtime	793
Utilitários integrados	800

Módulos integrados	803
Utilitários Runtime	827
Auxiliares de tempo de util.time	828
Auxiliares do DynamoDB em util.dynamodb	830
Auxiliares HTTP em util.http	836
Auxiliares de transformação em util.transform	838
Auxiliares de string em util.str	852
Extensões	852
Auxiliares XML em util.xml	856
JavaScript referência da função de resolução para o DynamoDB	858
GetItem	860
PutItem	861
UpdateItem	865
DeleteItem	869
Consulta	872
Verificar	877
Sincronização	881
BatchGetItem	884
BatchDeleteItem	887
BatchPutItem	890
TransactGetItems	892
TransactWriteItems	895
Sistema de tipo (mapeamento da solicitação)	902
Sistema de tipo (mapeamento da resposta)	907
Filtros	911
Expressões de condição	913
Expressões de condição da transação	926
Projeções	929
JavaScript referência da função resolvidor para OpenSearch	930
Solicitação	931
Resposta	931
operation field	932
path field	932
params field	933
Envio de variáveis	935
JavaScript referência da função de resolução para Lambda	936

Objeto de solicitação	936
Objeto da resposta	940
Resposta em lote da função do Lambda	940
JavaScript referência da função de resolução para fonte EventBridge de dados	940
Solicitação	931
Resposta	941
Campos de PutEvents	943
JavaScript referência da função de resolução para fonte None de dados	945
Solicitação	931
Carga útil	938
Resposta	941
JavaScript referência da função resolvidor para HTTP	946
Solicitação	931
Método	947
ResourcePath	947
Campo Params	947
Resposta	941
JavaScript referência da função de resolução para Amazon RDS	949
Modelo marcado com SQL	950
Criar declarações	951
Recuperação de dados	951
Funções do utilitário	952
Conversão	960
JavaScript referência de função de resolução para Amazon Bedrock	962
Objeto de solicitação	962
Objeto da resposta	969
Invocações de longa duração	969
Referência para tipos	970
Referência de modelo de mapeamento do resolvidor (VTL)	977
Visão geral do modelo de mapeamento do resolvidor	978
Resolvedores de unidade	978
Resolvedores de pipeline	175
Exemplo de modelo	984
Regras de desserialização do modelo de mapeamento avaliado	986
Guia de programação do modelo de mapeamento do resolvidor	988
Configuração	989

Variáveis	990
Métodos de chamada	992
Strings	993
Loops	994
Matrizes	995
Verificações condicionais	996
Operadores	997
Contexto	999
Filtrar	999
Referência de contexto do modelo de mapeamento do resolvidor	1004
Acesso ao \$context	1004
Limpar entradas	1014
Referência do utilitário de modelo de mapeamento do resolvidor	1015
Utilitários auxiliares em \$util	1017
AWS AppSync diretivas	1029
Auxiliares de tempo do \$util.time	1030
Auxiliares de lista em \$util.list	1033
Auxiliares de mapa em \$util.map	1034
Auxiliares do DynamoDB em \$util.dynamodb	1034
Auxiliares do Amazon RDS em \$util.rds	1044
Auxiliares HTTP em \$util.http	1047
Auxiliares XML em \$util.xml	1050
Auxiliares de transformação em \$util.transform	1052
Auxiliares de matemática em \$util.math	1066
Auxiliares de string em \$util.str	1067
Extensões	1068
Referência do modelo de mapeamento do resolvidor para DynamoDB	1082
GetItem	1084
PutItem	1086
UpdateItem	1090
DeleteItem	1096
Consulta	1099
Verificar	1104
Sincronização	1108
BatchGetItem	1111
BatchDeleteItem	1116

BatchPutItem	1119
TransactGetItems	1123
TransactWriteItems	1127
Sistema de tipo (mapeamento da solicitação)	1135
Sistema de tipo (mapeamento da resposta)	1140
Filtros	1144
Expressões de condição	1146
Expressões de condição da transação	1159
Projeções	1162
Referência de modelo de mapeamento do resolvidor para RDS	1164
Modelo de mapeamento de solicitações	1164
Versão	1166
Declarações e VariableMap	1166
VariableTypeHintMap	1166
Referência do modelo de mapeamento do resolvidor para OpenSearch	1167
Modelo de mapeamento de solicitações	1164
Modelo de mapeamento de respostas	931
operation field	932
path field	932
params field	933
Envio de variáveis	935
Referência do modelo de mapeamento do resolvidor para Lambda	1172
Modelo de mapeamento de solicitações	1164
Modelo de mapeamento de respostas	931
Resposta em lote da função do Lambda	1178
Resolvedores diretos do Lambda	1178
Referência do modelo de mapeamento do resolvidor para EventBridge	1185
Modelo de mapeamento de solicitações	1164
Modelo de mapeamento de respostas	931
Campos de PutEvents	943
Referência do modelo de mapeamento do resolvidor para fonte de dados None	1189
Modelo de mapeamento de solicitações	1164
Versão	1166
Carga útil	1176
Modelo de mapeamento de respostas	931
Referência do modelo de mapeamento do resolvidor para HTTP	1192

Modelo de mapeamento de solicitações	1164
Versão	1166
Método	1195
ResourcePath	1195
Campo Params	933
Resposta	1197
Autoridades de certificação (CA) reconhecidas AWS AppSync por quatro endpoints	
HTTPS	1197
Registro de alterações do modelo de mapeamento do resolvidor	1276
Disponibilidade de operação da fonte de dados por matriz de versão	1277
Alterar a versão em um modelo de mapeamento do resolvidor de unidade	1278
Alterar a versão em uma função	1279
2018-05-29	1279
2017-02-28	1287
Referência de tipos do GraphQL	1288
Tipos escalares no GraphQL	1289
Escalares padrão	1289
AWS AppSync escalares	1290
Exemplo de uso de esquema	1291
Interfaces e uniões no GraphQL	1295
Exemplos de interface	1295
Exemplos de uniões	1299
Resolução de texto em AWS AppSync	1300
Exemplo de resolução de tipo	1301
Solução de problemas e erros comuns	1306
Mapeamento de chave do DynamoDB incorreto	1306
Resolvidor ausente	1306
Erros do modelo de mapeamento	1307
Tipos de retorno incorretos	1307
Processamento de solicitações inválidas	1308
.....	mcccix

O que é AWS AppSync?

AWS AppSync permite que os desenvolvedores conectem seus aplicativos e serviços a dados e eventos com GraphQL e Pub/Sub seguros, sem servidor e de alto desempenho. APIs Você pode fazer o seguinte com AWS AppSync:

- Acesse dados de uma ou mais fontes de dados a partir de um único endpoint da API GraphQL.
- Combine várias fontes do GraphQL APIs em uma única API do GraphQL mesclada.
- Publique atualizações de dados em tempo real para seus aplicativos.
- Aproveite a segurança, o monitoramento, o registro e o rastreamento integrados, com cache opcional para baixa latência.
- Pague apenas pelas solicitações de API e pelas mensagens em tempo real que forem entregues.

Important

A partir de 13 de março de 2025, você pode criar uma PubSub API em tempo real baseada no WebSockets uso de AWS AppSync Eventos. Para obter mais informações, consulte [Publicar eventos WebSocket](#) no Guia do desenvolvedor de AWS AppSync eventos.

Tópicos

- [AWS AppSync features](#)
- [Você é um AWS AppSync usuário iniciante?](#)
- [Serviços relacionados](#)
- [Preços para AWS AppSync](#)

AWS AppSync features

- Consulta e acesso a dados simplificados, com tecnologia GraphQL
- Sem servidor para assinaturas do WebSockets GraphQL e canais pub/sub
- Cache no servidor para disponibilizar dados em caches na memória de alta velocidade para baixa latência

- JavaScript e TypeScript suporte para escrever lógica de negócios
- Segurança corporativa com Private APIs para restringir o acesso à API e a integração com AWS WAF
- Controles de autorização integrados, com suporte para chaves de API, IAM, Amazon Cognito, provedores OpenID Connect e autorização do Lambda para lógica personalizada.
- Mesclado APIs para dar suporte a casos de uso federados

Para obter mais detalhes sobre cada um desses recursos, consulte [AWS AppSync recursos](#).

Você é um AWS AppSync usuário iniciante?

Recomendamos que AWS AppSync os usuários iniciantes comecem lendo as seguintes seções:

- Se você não estiver familiarizado com o GraphQL, consulte o [Conceitos básicos do : como criar sua primeira API do GraphQL](#).
- Se você estiver criando aplicativos que consomem GraphQL APIs, consulte e. [Como criar um aplicativo do cliente the section called “Usando assinaturas para aplicativos de dados em tempo real”](#)
- Se você estiver procurando informações do resolvidor do GraphQL, consulte o seguinte:

JavaScript/TypeScript

- [Tutoriais do Resolver \(\) JavaScript](#)
- [Referência do resolvidor \(JavaScript\)](#)

VTL

- [Tutoriais do resolvidor \(VTL\)](#)
- [Referência de modelo de mapeamento do resolvidor \(VTL\)](#)
- Se você estiver procurando por AWS AppSync exemplos de projetos, atualizações e muito mais, consulte o [AppSyncblog](#).

Serviços relacionados

Se você estiver criando um aplicativo web ou para dispositivos móveis do zero, considere usar [AWS Amplify](#). O Amplify usa AWS AppSync e outros AWS serviços para ajudá-lo a criar aplicativos móveis e web mais robustos e poderosos com menos trabalho.

Preços para AWS AppSync

AWS AppSync tem um preço baseado em milhões de solicitações e atualizações. O armazenamento em cache custa uma taxa adicional. Para obter mais informações, consulte [Preços do AWS AppSync](#).

A seguir estão listadas as exceções aos AWS AppSync preços gerais:

- As solicitações não são cobradas por falhas de autorização e autenticação.
- Chamadas para métodos que exigem chaves de API não são cobrados quando as chaves de API estão ausentes ou são inválidas.

GraphQL e arquitetura AWS AppSync

Note

Este guia pressupõe que o usuário tenha um conhecimento prático do estilo de arquitetura REST. Recomendamos revisar este e outros tópicos de front-end antes de trabalhar com o GraphQL e AWS AppSync.

GraphQL é uma linguagem de consulta e manipulação para APIs. O GraphQL fornece uma sintaxe flexível e intuitiva para descrever os requisitos e interações de dados. Ele permite que os desenvolvedores solicitem exatamente o que é necessário e obtenham resultados previsíveis. Ele também possibilita acessar várias fontes em uma única solicitação, reduzindo o número de chamadas de rede e os requisitos de largura de banda, economizando a vida útil da bateria e os ciclos de CPU consumidos pelos aplicativos.

Fazer atualizações nos dados é simples com as mutações, permitindo que os desenvolvedores descrevam como os dados devem mudar. O GraphQL também facilita a configuração rápida de soluções em tempo real por meio de assinaturas. Todos esses atributos combinados, juntamente com poderosas ferramentas para desenvolvedores, tornam o GraphQL essencial para gerenciar dados de aplicativos.

O GraphQL é uma alternativa ao REST. RESTful a arquitetura é atualmente uma das soluções mais populares para comunicação cliente-servidor. Ele se concentra no conceito de seus recursos (dados) serem expostos por um URL. Eles URLs podem ser usados para acessar e manipular os dados por meio de operações CRUD (criar, ler, atualizar, excluir) na forma de métodos HTTP como GET, POST, e DELETE. A vantagem do REST é que ele é relativamente simples de aprender e implementar. Você pode se configurar rapidamente RESTful APIs para ligar para uma ampla variedade de serviços.

No entanto, a tecnologia está ficando mais complicada. À medida que aplicativos, ferramentas e serviços começam a se expandir para um público mundial, a necessidade de arquiteturas rápidas e escaláveis é de suma importância. O REST tem muitas obstáculos ao lidar com operações escaláveis. Veja este [caso de uso](#) de exemplo.

Nas seções a seguir, analisaremos alguns dos conceitos relacionados RESTful APIs. Em seguida, apresentaremos o GraphQL e como ele funciona.

Para obter mais informações sobre o GraphQL e os benefícios de migrar para AWS, consulte o Guia de [decisão para implementações do GraphQL](#).

Tópicos

- [O que é uma API?](#)
- [O que é REST?](#)
- [O que é GraphQL?](#)
- [Comparação entre REST e GraphQL](#)
- [Por que usar o GraphQL em vez do REST?](#)
- [Componentes de uma API GraphQL](#)
- [Propriedades adicionais do GraphQL](#)

O que é uma API?

Uma interface de programação de aplicações (API) define as regras que você deve seguir para se comunicar com outros sistemas de software. Os desenvolvedores expõem ou criam APIs para que outros aplicativos possam se comunicar com seus aplicativos programaticamente. Por exemplo, o aplicativo de planilha de horas expõe uma API que solicita o nome completo de um funcionário e um intervalo de datas. Quando recebe essas informações, ele processa internamente a planilha de horas do funcionário e retorna o número de horas trabalhadas nesse intervalo de datas.

Você pode pensar em uma API web como um gateway entre clientes e recursos na web.

Clientes

Clientes são usuários que desejam acessar informações da web. O cliente pode ser uma pessoa ou um sistema de software que usa a API. Por exemplo, os desenvolvedores podem escrever programas que acessam dados meteorológicos de um sistema meteorológico. Ou você pode acessar os mesmos dados no seu navegador ao visitar diretamente o site meteorológico.

Recursos

Recursos são as informações que diferentes aplicativos fornecem a seus clientes. Os recursos podem ser imagens, vídeos, texto, números ou qualquer tipo de dado. A máquina que fornece o recurso ao cliente também é chamada de servidor. As organizações usam APIs para compartilhar

recursos e fornecer serviços web, mantendo a segurança, o controle e a autenticação. Além disso, APIs ajude-os a determinar quais clientes têm acesso a recursos internos específicos.

O que é REST?

Em um alto nível, a transferência de estado representacional (REST) é uma arquitetura de software que impõe condições sobre como uma API deve funcionar. O REST foi criado inicialmente como uma diretriz para gerenciar a comunicação em uma rede complexa como a Internet. Você pode usar a arquitetura baseada em REST para oferecer suporte à comunicação confiável e de alto desempenho em grande escala. Você pode facilmente implementá-la e modificá-la, trazendo visibilidade e portabilidade multiplataforma para qualquer sistema de API.

Os desenvolvedores de API podem projetar APIs usando várias arquiteturas diferentes. APIs que seguem o estilo arquitetônico REST são chamados de REST APIs. Os serviços da Web que implementam a arquitetura REST são chamados de serviços RESTful da Web. O termo RESTful API geralmente se refere à RESTful web APIs. No entanto, você pode usar os termos API REST e RESTful API de forma intercambiável.

Veja a seguir alguns dos princípios do estilo de arquitetura REST:

Interface uniforme

A interface uniforme é fundamental para o design de qualquer RESTful serviço web. Isso indica que o servidor transfere informações em um formato padrão. O recurso formatado é chamado de representação em REST. Esse formato pode ser diferente da representação interna do recurso no aplicativo do servidor. Por exemplo, o servidor pode armazenar dados como texto, mas os envia em um formato de representação HTML.

A interface uniforme impõe quatro restrições de arquitetura:

1. As solicitações devem identificar os recursos. Elas fazem isso usando um identificador de recurso uniforme.
2. Os clientes têm informações suficientes na representação do recurso para modificar ou excluir o atributo, se quiserem. O servidor atende a essa condição enviando metadados que descrevem melhor o atributo.
3. Os clientes recebem informações sobre como processar ainda mais a representação. O servidor consegue isso enviando mensagens autodescritivas que contêm metadados sobre como o cliente pode melhor usá-las.

4. Os clientes recebem informações sobre todos os outros recursos relacionados de que precisam para concluir uma tarefa. O servidor consegue isso enviando hiperlinks na representação para que os clientes possam descobrir dinamicamente mais recursos.

Ausência de estado

Na arquitetura REST, a ausência de estado refere-se a um método de comunicação no qual o servidor conclui todas as solicitações do cliente, independentemente de todas as solicitações anteriores. Os clientes podem solicitar recursos em qualquer ordem, e cada solicitação é sem estado ou isolada de outras solicitações. Essa restrição de design da API REST implica que o servidor possa sempre entender e atender completamente à solicitação.

Sistema em camadas

Em uma arquitetura de sistema em camadas, o cliente pode se conectar a outros intermediários autorizados entre o cliente e o servidor e ainda receberá respostas do servidor. Os servidores também podem transmitir solicitações para outros servidores. Você pode projetar seu serviço RESTful web para ser executado em vários servidores com várias camadas, como segurança, aplicação e lógica de negócios, trabalhando em conjunto para atender às solicitações dos clientes. Essas camadas permanecem invisíveis para o cliente.

Capacidade de armazenamento em cache

RESTful os serviços web suportam cache, que é o processo de armazenar algumas respostas no cliente ou em um intermediário para melhorar o tempo de resposta do servidor. Por exemplo, suponha que você acesse um site que tenha imagens de cabeçalho e rodapé comuns em todas as páginas. Toda vez que você acessa uma nova página do site, o servidor precisa reenviar as mesmas imagens. Para evitar isso, o cliente armazena ou armazena essas imagens após a primeira resposta e, em seguida, usa as imagens diretamente do cache. RESTful os serviços da web controlam o armazenamento em cache usando respostas de API que se definem como armazenáveis em cache ou não.

O que é uma RESTful API?

RESTful A API é uma interface que dois sistemas de computador usam para trocar informações com segurança pela Internet. A maioria dos aplicativos empresariais precisa se comunicar com outros aplicativos internos e de terceiros para realizar várias tarefas. Por exemplo, para gerar recibos mensais, seu sistema interno de contas precisa compartilhar dados com o sistema bancário do

cliente para automatizar o faturamento e se comunicar com um aplicativo interno de quadro de horários. RESTful APIs apoiam essa troca de informações porque seguem padrões de comunicação de software seguros, confiáveis e eficientes.

Como RESTful APIs funciona?

A função básica de uma RESTful API é a mesma de navegar na Internet. O cliente entra em contato com o servidor usando a API quando precisa de um recurso. Os desenvolvedores de API explicam como o cliente deve usar a API REST na documentação da API do aplicativo de servidor. Estas são as etapas gerais para qualquer chamada da API REST:

1. O cliente envia a seguinte solicitação ao servidor. O cliente segue a documentação da API para formatar a solicitação de uma forma que o servidor entenda.
2. O servidor autentica o cliente e confirma que o cliente tem o direito de fazer essa solicitação.
3. O servidor recebe a solicitação e a processa internamente.
4. O servidor retorna uma resposta para o cliente. A resposta contém informações que informam ao cliente se a solicitação foi bem-sucedida. A resposta também inclui todas as informações solicitadas pelo cliente.

Os detalhes da solicitação e da resposta da API REST variam um pouco, dependendo de como os desenvolvedores da API projetam a API.

O que é GraphQL?

O GraphQL é uma linguagem de consulta APIs e um tempo de execução para executar essas consultas. O GraphQL permite que os clientes solicitem exatamente os dados de que precisam, fornecendo uma alternativa mais flexível e eficiente ao REST em muitos cenários. Ao contrário do REST, que depende de endpoints predefinidos, o GraphQL usa um único endpoint em que os clientes podem especificar seus requisitos de dados na forma de consultas e mutações.

Consulte [Componentes de uma API do GraphQL](#) para obter mais informações sobre como o GraphQL é estruturado. APIs

Comparação entre REST e GraphQL

APIs (Interfaces de programação de aplicativos) desempenham um papel crucial na facilitação da troca de dados entre aplicativos. Conforme mencionado anteriormente, APIs surgiram duas

abordagens proeminentes para design: GraphQL e REST. Embora ambas atendam ao propósito fundamental de permitir a comunicação cliente-servidor, elas diferem significativamente em seus casos de implementação e uso.

As abordagens GraphQL e REST compartilham várias características importantes:

1. Modelo cliente-servidor: ambas usam uma arquitetura cliente-servidor para troca de dados.
2. Ausência de estado: nenhuma das duas mantém as informações da sessão do cliente entre as solicitações.
3. Baseada em HTTP: ambas normalmente usam HTTP como o protocolo de comunicação subjacente.
4. Design orientado a recursos: ambas projetam seu intercâmbio de dados em torno de recursos, que se referem a quaisquer dados ou objetos que o cliente possa acessar e manipular por meio da API.
5. Flexibilidade de formato de dados: JSON é o formato de troca de dados mais usado em ambas, embora outros formatos, como XML e HTML, também sejam compatíveis.
6. Independente de linguagem e banco de dados: ambas podem funcionar com qualquer linguagem de programação ou estrutura de banco de dados, tornando-as altamente interoperáveis.
7. Suporte ao armazenamento em cache: ambas oferecem suporte ao armazenamento em cache, permitindo que clientes e servidores armazenem dados acessados com frequência para melhorar o desempenho.

Embora compartilhem alguns princípios fundamentais, GraphQL e REST diferem significativamente na abordagem para design de API e busca de dados:

1. Estrutura da solicitação e busca de dados

REST usa diferentes métodos HTTP (GET, POST, PUT, DELETE) para executar operações em recursos. Isso geralmente requer vários endpoints para diferentes recursos, o que pode levar a ineficiências na recuperação de dados. Por exemplo, executar uma operação GET para recuperar dados de um usuário pode resultar em busca excessiva ou insuficiente dos dados. Para obter os dados corretos, é possível chamar operações de truncamento ou várias operações.

GraphQL usa um único endpoint para todas as operações. Ele se baseia em consultas para buscar dados e em mutações para modificá-los. Os clientes podem usar consultas para buscar exatamente os dados de que precisam em uma única solicitação, o que reduz a sobrecarga da rede ao minimizar a transferência de dados.

2. Esquema do lado do servidor

REST não exige um esquema do lado do servidor, embora seja possível definir um opcionalmente para projeto e documentação eficientes da API.

GraphQL usa um esquema do lado do servidor fortemente tipado para definir dados e serviços de dados. O esquema, escrito na linguagem de definição de esquema (SDL) do GraphQL, inclui tipos de objetos e campos para cada objeto e funções de resolvidor do lado do servidor que definem operações para cada campo.

3. Versionamento

REST geralmente inclui versionamento no URL, o que pode levar à manutenção de várias versões da API simultaneamente. O versionamento não é obrigatório, mas pode ajudar a evitar alterações significativas.

GraphQL promove uma evolução contínua da API sem versionamento explícito, exigindo compatibilidade com versões anteriores. Os campos excluídos retornam mensagens de erro, enquanto as tags de depreciação eliminam gradualmente os campos antigos e retornam mensagens de aviso.

4. Tratamento de erros

REST tem uma tipagem fraca, exigindo que o tratamento de erros seja incorporado ao código ao redor. Isso pode não identificar automaticamente erros relacionados ao tipo (por exemplo, analisar um número como texto).

Por outro lado, GraphQL é fortemente tipado e requer uma definição de esquema abrangente. Isso permite que seu serviço identifique automaticamente muitos erros de solicitação com um alto nível de detalhes.

5. Casos de uso

REST é mais adequado para:

- Aplicações menores com requisitos de dados menos complexos.
- Cenários em que dados e operações são usados de forma semelhante por todos os clientes.
- Aplicações sem necessidades de consultas complexas de dados.

GraphQL é mais adequado para:

- Cenários com largura de banda limitada, em que é fundamental minimizar solicitações e respostas.

- Aplicações com várias fontes de dados que precisam ser combinados em um único endpoint.
- Casos em que as solicitações dos clientes variam significativamente e esperam estruturas de resposta diferentes.

Observe que é possível usar o GraphQL e o REST APIs em um único aplicativo para diferentes áreas de funcionalidade. Além disso, você pode atualizar uma RESTful API para incluir recursos do GraphQL sem uma reescrita completa. Consulte [Como criar resolvedores GraphQL para fontes de AWS dados para](#) ver um exemplo.

Por que usar o GraphQL em vez do REST?

REST é um dos estilos arquitetônicos fundamentais da web. APIs No entanto, à medida que o mundo se torna mais interconectado, a necessidade de desenvolver aplicativos robustos e escaláveis se tornará uma questão mais urgente. Embora o REST seja frequentemente usado para criar a web APIs, existem várias desvantagens recorrentes nas RESTful implementações que foram identificadas:

1. Solicitações de dados: usando RESTful APIs, você normalmente solicitaria os dados de que precisa por meio de endpoints. O problema surge quando há dados que podem não estar tão bem empacotados. Os dados de que você precisa podem estar por trás de várias camadas de abstração, e a única maneira de buscar os dados é usando vários endpoints, o que significa fazer várias solicitações para extrair todos os dados.
2. Busca excessiva e busca insuficiente: piorando ainda mais a questão de várias solicitações, os dados de cada endpoint são estritamente definidos, o que significa que você retornará qualquer dado que seja definido para essa API, mesmo que tecnicamente não tenha interesse nele.

Isso pode resultar em busca excessiva, o que significa que nossas solicitações retornam dados supérfluos. Por exemplo, digamos que você esteja solicitando dados pessoais da empresa e queira saber os nomes dos funcionários de um certo departamento. O endpoint que retorna os dados contém os nomes, mas também pode conter outros dados, como cargo ou data de nascimento. Como a API é fixa, você não pode simplesmente solicitar os nomes individualmente; o resto dos dados vem com eles.

A situação oposta, na qual não retornamos dados suficientes, é chamada de busca insuficiente. Para obter todos os dados solicitados, talvez seja necessário fazer várias solicitações ao serviço. Dependendo de como os dados foram estruturados, você pode se deparar com consultas ineficientes, resultando em problemas como o temido problema $n+1$.

3. **Iterações de desenvolvimento lentas:** muitos desenvolvedores se adaptam RESTful APIs para se adequarem ao fluxo de seus aplicativos. No entanto, conforme seus aplicativos crescem, tanto o front-end quanto o back-end podem exigir grandes mudanças. Como resultado, eles APIs podem não se ajustar mais ao formato dos dados de uma forma eficiente ou impactante. Isso resulta em iterações de produto mais lentas devido à necessidade de modificações na API.
4. **Desempenho em grande escala:** devido a esse acúmulo de problemas, há muitas áreas em que a escalabilidade será afetada. O desempenho no lado do aplicativo pode ser afetado porque suas solicitações retornarão dados de mais ou de menos (resultando em mais solicitações). Ambas as situações causam sobrecarga desnecessária na rede, resultando em baixo desempenho. Do lado do desenvolvedor, a velocidade de desenvolvimento pode ser reduzida porque você APIs é fixo e não se ajusta mais aos dados que eles estão solicitando.

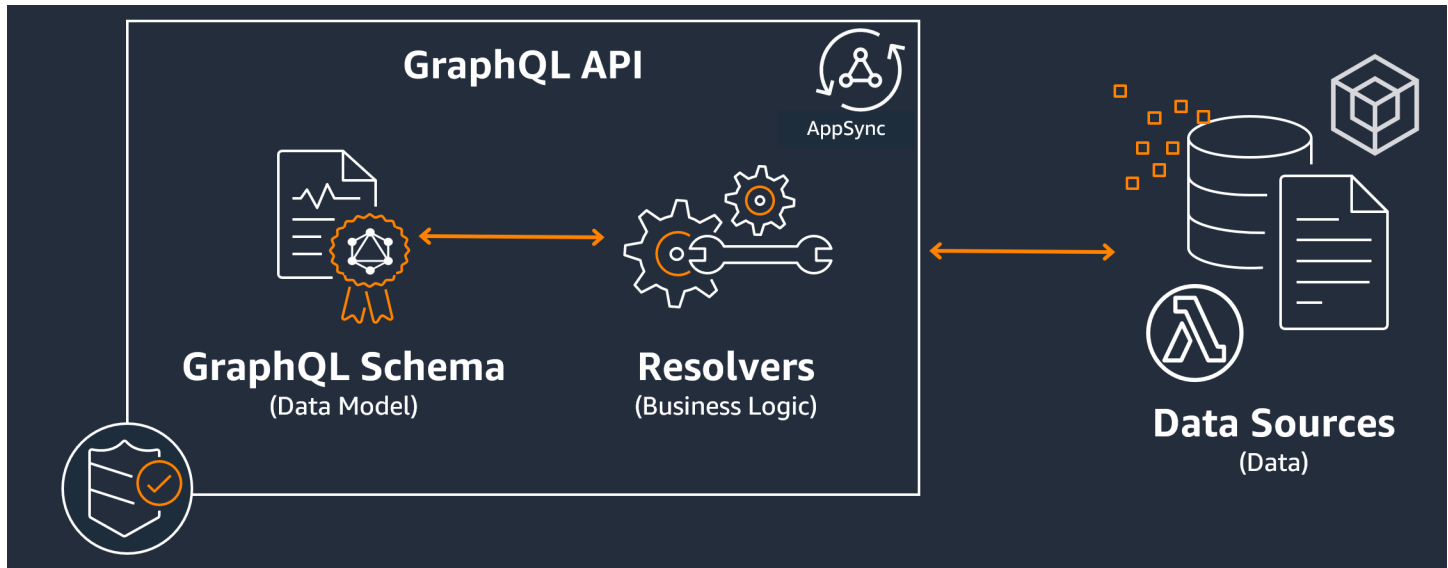
O ponto de venda do GraphQL é superar as desvantagens do REST. Aqui estão algumas das principais soluções que o GraphQL oferece aos desenvolvedores:

1. **Endpoints únicos:** o GraphQL usa um único endpoint para consultar dados. Não há necessidade de criar vários APIs para se adequar ao formato dos dados. Isso resulta em menos solicitações passando pela rede.
2. **Busca:** o GraphQL resolve os problemas inerentes da busca excessiva e insuficiente simplesmente definindo os dados de que você precisa. O GraphQL permite que você modele os dados de acordo com suas necessidades, para que você receba apenas o que solicitou.
3. **Abstração:** O APIs GraphQL contém alguns componentes e sistemas que descrevem os dados usando um padrão independente de linguagem. Em outras palavras, a forma e a estrutura dos dados são padronizadas para que tanto o front-end quanto o back-end saibam como eles serão enviados pela rede. Isso permite que os desenvolvedores de ambos os lados trabalhem com os sistemas do GraphQL e não em torno deles.
4. **Iterações rápidas:** devido à padronização dos dados, as mudanças em uma extremidade do desenvolvimento podem não ser necessárias na outra. Por exemplo, alterações na apresentação do front-end podem não resultar em alterações extensivas no back-end porque o GraphQL permite que a especificação dos dados seja modificada prontamente. Você pode simplesmente definir ou modificar a forma dos dados para atender às necessidades do aplicativo à medida que ele cresce. Isso resulta em menos trabalho potencial de desenvolvimento.

Esses são apenas alguns dos benefícios do GraphQL. Nas próximas seções, você aprenderá como o GraphQL é estruturado e as propriedades que o tornam uma alternativa exclusiva ao REST.

Componentes de uma API GraphQL

Uma API GraphQL padrão é composta por um único esquema que manipula a forma dos dados que serão consultados. Seu esquema está vinculado a uma ou mais de suas fontes de dados, como um banco de dados ou uma função do Lambda. Entre os dois, há um ou mais resolvedores que lidam com a lógica de negócios de suas solicitações. Cada componente desempenha um papel importante na implementação do GraphQL. As seções a seguir apresentarão esses três componentes e o papel que eles desempenham no serviço GraphQL.



Tópicos

- [Esquemas do GraphQL](#)
- [Fontes de dados](#)
- [Resolvedores](#)

Esquemas do GraphQL

O esquema do GraphQL é a base de uma API GraphQL. Ele serve como o esquema que define a forma dos dados. Também se trata de um contrato entre seu cliente e servidor que define como seus dados serão recuperados e/ou modificados.

Os esquemas do GraphQL são escritos em Schema Definition Language (SDL). O SDL é composto por tipos e campos com uma estrutura estabelecida:

- **Tipos:** os tipos são como o GraphQL define a forma e o comportamento dos dados. O GraphQL é compatível com uma infinidade de tipos que serão explicados ainda nesta seção. Cada um dos tipos definidos em seu esquema terá um escopo próprio. Dentro do escopo, um ou mais campos vão apresentar um valor ou lógica que será usada em seu serviço do GraphQL. Os tipos têm muitas funções diferentes, sendo as mais comuns objetos ou escalares (tipos de valores primitivos).
- **Campos:** os campos existem dentro do escopo de um tipo e contêm o valor solicitado do serviço do GraphQL. Eles são muito semelhantes às variáveis de outras linguagens de programação. A forma dos dados que você define em seus campos determinará como os dados são estruturados em uma operação de solicitação/resposta. Isso permite que os desenvolvedores prevejam o que será retornado sem saber como o back-end do serviço é implementado.

Para visualizar a aparência de um esquema, vamos revisar o conteúdo de um esquema simples do GraphQL. No código de produção, seu esquema normalmente ficará em um arquivo chamado `schema.graphql` ou `schema.json`. Vamos supor que estamos examinando um projeto que implementa um serviço GraphQL. Este projeto está armazenando dados de funcionários da empresa, e o arquivo `schema.graphql` está sendo usado para recuperar dados de funcionários e adicionar novos funcionários a um banco de dados. O código pode ser semelhante a:

`schema.graphql`

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

Podemos ver que há três tipos definidos no esquema: `Person`, `Query` e `Mutation`. Observando `Person`, podemos supor que esse é o esquema de uma instância de um funcionário da empresa, o que faria desse tipo um objeto. Dentro de seu escopo, vemos `id`, `name` e `age`. Esses são os campos que definem as propriedades de uma `Person`. Isso significa que nossa fonte de dados armazena

cada `name` de `Person` como um tipo `String` escalar (primitivo) e `age` como um tipo `Int` escalar (primitivo). O `id` atua como um identificador especial e exclusivo para cada `Person`. Ele também é um valor obrigatório, conforme indicado pelo símbolo `!`.

Os próximos dois tipos de objetos se comportam de forma diferente. O GraphQL reserva algumas palavras-chave para tipos de objetos especiais que definem como os dados serão preenchidos no esquema. Um tipo `Query` recupera dados da fonte. Em nosso exemplo, nossa consulta pode recuperar objetos `Person` de um banco de dados. Isso pode lembrá-lo das GET operações na RESTful terminologia. Uma `Mutation` modifica os dados. Em nosso exemplo, nossa mutação pode adicionar mais objetos `Person` ao banco de dados. Isso pode soar familiar devido às operações de alteração de estado, como PUT ou POST. Os comportamentos de todos os tipos de objetos especiais serão explicados posteriormente nesta seção.

Vamos supor que `Query` em nosso exemplo recupere algo do banco de dados. Se olharmos para os campos de `Query`, veremos um campo chamado `people`. Seu valor de campo é `[Person]`. Isso significa que queremos recuperar alguma instância de `Person` no banco de dados. No entanto, a adição de colchetes significa que queremos retornar uma lista de todas as instâncias de `Person` e não apenas uma específica.

O tipo `Mutation` é responsável por realizar operações de mudança de estado, como modificação de dados. Uma mutação é responsável por realizar alguma operação de alteração de estado na fonte de dados. Em nosso exemplo, nossa mutação contém uma operação chamada `addPerson` que adiciona um novo objeto `Person` ao banco de dados. A mutação usa a `Person` e espera uma entrada para os campos `id`, `name` e `age`.

Neste ponto, você pode estar se perguntando como funcionam operações como `addPerson` sem uma implementação de código, já que ela supostamente executa algum comportamento e se parece muito com uma função, com um nome e parâmetros de função. Neste momento, isso não funcionaria porque um esquema serve apenas como declaração. Para implementar o comportamento de `addPerson`, teríamos que adicionar um resolvedor a ele. Um resolvedor é uma unidade de código que é executada sempre que seu campo associado (nesse caso, a operação `addPerson`) é chamado. Se você quiser usar uma operação, precisará adicionar a implementação do resolvedor em algum momento. De certa forma, você pode pensar na operação do esquema como a declaração da função e no resolvedor como a definição. Os resolvedores serão explicados em uma seção diferente.

Este exemplo mostra apenas as formas mais simples de um esquema manipular dados. Você cria aplicativos complexos, robustos e escaláveis aproveitando os atributos do GraphQL e AWS

AppSync. Na próxima seção, definiremos todos os diferentes tipos e comportamentos de campo que você pode utilizar em seu esquema.

Tipos no GraphQL

O GraphQL é compatível com muitos tipos diferentes. Como você viu na seção anterior, os tipos definem a forma ou o comportamento dos seus dados. Eles são os blocos de construção fundamentais de um esquema do GraphQL.

Os tipos podem ser categorizados em entradas e saídas. As entradas são tipos que podem ser transmitidos como argumento para os tipos de objetos especiais (Query, Mutation, etc.), enquanto os tipos de saída são usados estritamente para armazenar e retornar dados. Uma lista de tipos e suas categorizações estão listadas abaixo:

- **Objetos:** um objeto contém campos que descrevem uma entidade. Por exemplo, um objeto pode ser algo como um book com campos descrevendo suas características, como `authorName`, `publishingYear`, etc. Eles são estritamente tipos de saída.
- **Escalares:** esses são tipos primitivos como `int`, `string`, etc. Normalmente, eles são atribuídos a campos. Usando o campo `authorName` como exemplo, ele pode ser atribuído ao escalar `String` para armazenar um nome como “John Smith”. Escalares podem ser do tipo de entrada e saída.
- **Entradas:** as entradas permitem que você transmita um grupo de campos como argumento. Elas são estruturadas de forma muito semelhante aos objetos, mas podem ser passadas como argumentos para objetos especiais. As entradas permitem que você defina escalares, enums e outras entradas em seu escopo. As entradas só podem ser tipos de entrada.
- **Objetos especiais:** objetos especiais realizam operações de mudança de estado e fazem a maior parte do trabalho pesado do serviço. Há três tipos de objetos especiais: consulta, mutação e assinatura. As consultas normalmente buscam dados; as mutações manipulam dados; as assinaturas abrem e mantêm uma conexão bidirecional entre clientes e servidores para comunicação constante. Objetos especiais não são de entrada nem saída, dada sua funcionalidade.
- **Enums:** enums são listas predefinidas de valores legais. Se você chamar um enum, seus valores só poderão ser definidos em seu escopo. Por exemplo, se você tivesse uma enumeração chamada `trafficLights` representando uma lista de sinais de trânsito, ela poderia ter valores como `redLight` e `greenLight`, mas não `purpleLight`. Um semáforo real terá um limite de sinais, então você pode usar o enum para defini-los e forçá-los a serem os únicos valores legais ao fazer referência a `trafficLight`. Escalares podem ser do tipo de entrada e saída.

- **Unões/interfaces:** as uniões permitem que você retorne uma ou mais coisas em uma solicitação, dependendo dos dados solicitados pelo cliente. Por exemplo, se você tivesse um tipo `Book` com um `title` campo e um tipo `Author` com um campo `name`, você poderia criar uma união entre os dois tipos. Se seu cliente quisesse consultar um banco de dados para a frase “Júlio César”, a união poderia retornar Júlio César (a peça de William Shakespeare) do `title Book` e Júlio César (o autor de *Commentarii de Bello Gallico*) do `name Author`. As uniões só podem ser tipos de saída.

As interfaces são conjuntos de campos que os objetos devem implementar. Isso é um pouco semelhante às interfaces em linguagens de programação como Java, nas quais você precisa implementar os campos definidos na interface. Por exemplo, digamos que você tenha criado uma interface chamada `Book` que continha um campo `title`. Digamos que você tenha criado posteriormente um tipo chamado `Novel` que implementou `Book`. Seu `Novel` teria que incluir um campo `title`. No entanto, seu `Novel` também pode incluir outros campos que não estão na interface, como `pageCount` ou `ISBN`. As interfaces só podem ser tipos de saída.

As seções a seguir explicarão como cada tipo funciona no GraphQL.

Objetos

Os objetos do GraphQL são o tipo principal que você verá no código de produção. No GraphQL, você pode pensar em um objeto como um agrupamento de campos diferentes (semelhantes às variáveis em outras linguagens), e cada campo é definido por um tipo (normalmente um escalar ou outro objeto) que pode conter um valor. Os objetos representam uma unidade de dados que pode ser recuperada/manipulada a partir da implementação do serviço.

Os tipos de objetos são declarados usando a palavra-chave `Type`. Vamos modificar um pouco nosso exemplo de esquema:

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

Os tipos de objeto aqui são `Person` e `Occupation`. Cada objeto tem seus próprios campos com seus próprios tipos. Um atributo do GraphQL é a capacidade de definir campos como outros tipos. Você pode ver que o campo `occupation` em `Person` contém um tipo de objeto `Occupation`. Podemos fazer essa associação porque o GraphQL está apenas descrevendo os dados e não a implementação do serviço.

Escalares

Os escalares são tipos essencialmente primitivos que contêm valores. Em AWS AppSync, há dois tipos de escalares: os escalares e AWS AppSync escalares padrão do GraphQL. Os escalares costumam ser usados para armazenar valores de campo em tipos de objetos. Os tipos padrão do GraphQL incluem `Int`, `Float`, `String`, `Boolean` e `ID`. Vamos usar o exemplo anterior novamente:

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

Destacando os campos `name` e `title` e, ambos possuem um escalar `String`, e `Name` poderia retornar um valor de string como "John Smith" e o título poderia retornar algo como "firefighter". Algumas implementações do GraphQL também oferecem suporte a escalares personalizados usando a palavra-chave `Scalar` e implementando o comportamento do tipo. No entanto, atualmente o AWS AppSync não oferece suporte a escalares personalizados. Para obter uma lista de escalares, consulte [Tipos de escalares](#) no AWS AppSync.

Entradas

Devido ao conceito de tipos de entrada e saída, existem certas restrições ao transmitir argumentos. Os tipos que normalmente precisam ser transmitidos, principalmente objetos, são restritos. Você pode usar o tipo de entrada para ignorar essa regra. As entradas são tipos que contêm escalares, enums e outros tipos de entrada.

As entradas são definidas usando a palavra-chave `input`:

```
type Person {
```

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

type Occupation {
  title: String
}

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

input occupationInput {
  title: String
}
```

Como você pode ver, podemos ter entradas separadas que imitam o tipo original. Essas entradas geralmente serão usadas em suas operações de campo da seguinte forma:

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

type Occupation {
  title: String
}

input occupationInput {
  title: String
}

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

Observe como ainda estamos transmitindo `occupationInput` em vez de `Occupation` para criar uma `Person`.

Este é apenas um dos cenários para entradas. Elas não precisam copiar objetos 1:1 e, no código de produção, você provavelmente não as usará dessa forma. É uma prática recomendada aproveitar os esquemas do GraphQL definindo somente o que você precisa inserir como argumentos.

Além disso, as mesmas entradas podem ser usadas em várias operações, mas não recomendamos fazer isso. Preferencialmente, cada operação deve conter sua própria cópia exclusiva das entradas, caso os requisitos do esquema mudem.

Objetos especiais

O GraphQL reserva algumas palavras-chave para objetos especiais que definem parte da lógica de negócios de como seu esquema recuperará/manipulará dados. No máximo, pode haver uma de cada uma dessas palavras-chave em um esquema. Eles atuam como pontos de entrada para todos os dados solicitados que seus clientes executam no seu serviço GraphQL.

Objetos especiais também são definidos usando a palavra-chave `type`. Embora sejam usados de forma diferente dos tipos de objetos comuns, sua implementação é muito semelhante.

Queries

As consultas são muito semelhantes às operações `GET`, pois realizam uma busca somente para leitura para obter dados da sua fonte. No GraphQL, a `Query` define todos os pontos de entrada para clientes que fazem solicitações em seu servidor. Sempre haverá um `Query` em sua implementação do GraphQL.

Aqui estão os tipos `Query` de objetos modificados que usamos em nosso exemplo de esquema anterior:

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



```
  people: [Person]
}
```

Nosso Query contém um campo chamado `people` que retorna uma lista de instâncias `Person` da fonte de dados. Digamos que precisemos mudar o comportamento do nosso aplicativo e agora precisamos retornar uma lista somente das instâncias de `Occupation` para algum propósito separado. Poderíamos simplesmente adicioná-lo à consulta:

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

No GraphQL, podemos tratar nossa consulta como a única fonte de solicitações. Como você pode ver, isso é potencialmente muito mais simples do que RESTful implementações que podem usar endpoints diferentes para alcançar a mesma coisa (`.../api/1/people`). `.../api/1/occupations`

Supondo que tenhamos uma implementação de resolvedor para essa consulta, agora podemos realizar uma consulta real. Embora o tipo `Query` exista, precisamos chamá-lo explicitamente para que ele seja executado no código do aplicativo. Isso pode ser feito usando a palavra-chave `query`:

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

Como você pode ver, essa consulta é chamada `getItems` e retorna `people` (uma lista de objetos `Person`) e `occupations` (uma lista de objetos `Occupation`). Em `people`, estamos retornando somente o campo `name` de cada `Person`, enquanto retornamos o campo `title` de cada `Occupation`. A resposta pode ser semelhante a:

```
{
  "data": {
    "people": [
```

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

O exemplo de resposta mostra como os dados seguem a forma da consulta. Cada entrada recuperada é listada dentro do escopo do campo. `people` e `occupations` estão retornando itens como listas separadas. Embora isso seja útil, talvez seja mais conveniente modificar a consulta para retornar uma lista dos nomes e ocupações das pessoas:

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

Essa é uma modificação legal porque nosso tipo `Person` contém um campo `occupation` de tipo `Occupation`. Quando listados no escopo de `people`, retornamos cada um `name` de `Person` junto com os `Occupation` associados por `title`. A resposta pode ser semelhante a:

```
}
```

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

Mutations

As mutações são semelhantes às operações de alteração de estado, como ou PUT POST. Eles realizam uma operação de gravação para modificar os dados na fonte e, em seguida, buscam a resposta. Eles definem seus pontos de entrada para solicitações de modificação de dados. Diferentemente das consultas, uma mutação pode ou não ser incluída no esquema, dependendo das necessidades do projeto. Veja a mutação do exemplo do esquema:

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

O campo `addPerson` representa um ponto de entrada que adiciona uma `Person` à fonte de dados. `addPerson` é o nome do campo; `id`, `name` e `age` são os parâmetros; e `Person` é o tipo de retorno. Relembrando o tipo `Person`:

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

```
}
```

Adicionamos o campo `occupation`. No entanto, não podemos definir esse campo como `Occupation` diretamente porque os objetos não podem ser passados como argumentos; eles são estritamente tipos de saída. Em vez disso, devemos passar uma entrada com os mesmos campos de um argumento:

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

Também podemos atualizar facilmente nosso `addPerson` para incluir isso como um parâmetro ao criar novas instâncias de `Person`:

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

Veja o esquema atualizado:

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

Observe que `occupation` passará o campo `title` de `occupationInput` para concluir a criação do objeto `Person` em vez do `Occupation` original. Supondo que tenhamos uma

implementação de resolvedor para `addPerson`, agora podemos realizar uma mutação real. Embora o tipo `Mutation` exista, precisamos chamá-lo explicitamente para que ele seja executado no código do aplicativo. Isso pode ser feito usando a palavra-chave `mutation`:

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

Essa mutação é chamada de `createPerson`, e `addPerson` é a operação. Para criar uma nova `Person`, podemos inserir os argumentos para `id`, `name`, `age` e `occupation`. No escopo de `addPerson`, também podemos ver outros campos como `name`, `age`, etc. Esta é sua resposta; esses são os campos que serão retornados após a conclusão da operação `addPerson`. Aqui está a parte final do exemplo:

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

Usando essa mutação, o resultado pode ser semelhante a:

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

```
    }  
  }  
}  
}
```

Como você pode ver, a resposta retornou os valores que solicitamos no mesmo formato definido em nossa mutação. A prática recomendada é retornar todos os valores que foram modificados para reduzir a confusão e a necessidade de mais consultas no futuro. As mutações permitem que você inclua várias operações em seu escopo. Eles serão executados sequencialmente na ordem listada na mutação. Por exemplo, se criarmos outra operação chamada `addOccupation` que adiciona cargos à fonte de dados, podemos chamar isso na mutação depois de `addPerson`. `addPerson` será tratado primeiro, seguido por `addOccupation`.

Subscriptions

As assinaturas são usadas [WebSockets](#) para abrir uma conexão bidirecional duradoura entre o servidor e seus clientes. Normalmente, um cliente assina ou escuta o servidor. Sempre que o servidor fizer uma alteração no servidor ou realizar um evento, o cliente assinante receberá as atualizações. Esse tipo de protocolo é útil quando vários clientes são assinantes e precisam ser notificados sobre alterações que estão acontecendo no servidor ou em outros clientes. Por exemplo, as assinaturas podem ser usadas para atualizar feeds de mídia social. Pode haver dois usuários, o usuário A e o usuário B, que são assinantes das atualizações automáticas de notificação sempre que recebem mensagens diretas. O usuário A no cliente A poderia enviar uma mensagem direta para o usuário B no cliente B. O cliente do usuário A enviaria a mensagem direta, que seria processada pelo servidor. O servidor então enviaria a mensagem direta para a conta do Usuário B enquanto enviaria uma notificação automática para o Cliente B.

Aqui está um exemplo de uma `Subscription` que poderíamos adicionar ao exemplo do esquema:

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

O campo `personAdded` enviará uma mensagem aos clientes inscritos sempre que uma nova `Person` for adicionada à fonte de dados. Supondo que tenhamos uma implementação de resolvidor para `personAdded`, agora podemos usar a assinatura. Embora o tipo `Subscription` exista, precisamos chamá-lo explicitamente para que ele seja executado no código do aplicativo. Isso pode ser feito usando a palavra-chave `subscription`:

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

A assinatura é chamada de `personAddedOperation` e a operação é `personAdded`. `personAdded` retornará os campos `id` e `name` de novas instâncias de `Person`. Analisando o exemplo de mutação, adicionamos uma `Person` usando esta operação:

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

Se nossos clientes tiverem assinatura para receber atualizações da `Person` recém-adicionada, eles poderão ver isso após as execuções de `addPerson`:

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

Veja abaixo um resumo do que as assinaturas oferecem:

As assinaturas são canais bidirecionais que permitem que o cliente e o servidor recebam atualizações rápidas, mas constantes. Eles normalmente usam o WebSocket protocolo, que cria conexões padronizadas e seguras.

As assinaturas são ágeis, pois reduzem a sobrecarga de configuração da conexão. Depois de fazer uma assinatura, o cliente pode continuar executando essa assinatura por longos períodos. Eles geralmente usam recursos de computação de forma eficiente, permitindo que os desenvolvedores personalizem a vida útil da assinatura e configurem quais informações serão solicitadas.

No geral, o cliente pode fazer várias assinaturas ao mesmo tempo. No que diz respeito AWS AppSync, as assinaturas são usadas apenas para receber atualizações em tempo real do serviço. AWS AppSync Elas não podem ser usadas para realizar consultas ou mutações.

A principal alternativa às assinaturas é a sondagem, que envia consultas em intervalos definidos para solicitar dados. Esse processo geralmente é menos eficiente do que as assinaturas e sobrecarrega muito o cliente e o back-end.

Algo que não foi mencionado em nosso exemplo de esquema é o fato de que seus tipos de objetos especiais também devem ser definidos em um schema raiz. Então, quando você exporta um esquema AWS AppSync, ele pode ter a seguinte aparência:

schema.graphql

```
schema {  
  query: Query  
  mutation: Mutation  
  subscription: Subscription  
}  
  
.  
.  
.  
  
type Query {  
  # code goes here  
}  
type Mutation {  
  # code goes here  
}  
type Subscription {  
  # code goes here  
}
```

Enumerações

Enumerações, ou enums, são escalares especiais que limitam os argumentos legais que um tipo ou campo pode ter. Isso significa que sempre que uma enum for definida no esquema, seu tipo ou campo associado será limitado aos valores da enum. As enums são serializadas como escalares de string. Observe que diferentes linguagens de programação podem lidar com enumerações do GraphQL de forma diferente. Por exemplo, não JavaScript tem suporte nativo para enum, portanto, os valores enum podem ser mapeados para valores int em vez disso.

As enums são definidas usando a palavra-chave enum. Veja um exemplo abaixo:


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

Ao chamar o `trafficLights` enum, os argumentos só podem ser `solidRed`, `solidYellow`, `solidGreen`, etc. É comum usar enums para descrever coisas que têm um número distinto, mas limitado, de opções.

Unões/Interfaces

Consulte [Interfaces e uniões](#) no GraphQL.

Campos do GraphQL

Os campos existem dentro do escopo de um tipo e contêm o valor solicitado do serviço do GraphQL. Eles são muito semelhantes às variáveis de outras linguagens de programação. Por exemplo, veja um tipo de objeto `Person`:

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

Nesse caso, os campos são `name` e `age` e contêm um valor `String` e `Int` respectivamente. Campos de objetos como os mostrados acima podem ser usados como entradas nos campos (operações) de suas consultas e mutações. Por exemplo, veja a Query a seguir:

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

O campo `people` está solicitando todas as instâncias de `Person` da fonte de dados. Ao adicionar ou recuperar uma `Person` em seu servidor GraphQL, você pode esperar que os dados sigam o formato dos seus tipos e campos, ou seja, a estrutura dos seus dados no esquema determina como eles serão estruturados na sua resposta:

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

Os campos desempenham um papel importante na estruturação dos dados. Há algumas propriedades adicionais explicadas abaixo que podem ser aplicadas aos campos para maior personalização.

Listas

As listas retornam todos os itens de um tipo especificado. Uma lista pode ser adicionada ao tipo de um campo usando colchetes []:

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

Na Query, os colchetes ao redor de Person indicam que você deseja retornar todas as instâncias de Person da fonte de dados como uma matriz. Na resposta, os valores name e age de cada Person serão retornados como uma única lista delimitada:

```
}
  "data": {
```

```

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

```

Não é necessário usar apenas tipos de objetos especiais. Você também pode usar listas nos campos de tipos de objetos regulares.

Não nulos

Os valores não nulos indicam um campo que não pode ser nulo na resposta. Você pode definir um campo como não nulo usando o símbolo !:

```

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

```

O campo name não pode ser explicitamente nulo. Se você consultasse a fonte de dados e fornecesse uma entrada nula para esse campo, um erro seria gerado.

Você pode combinar listas e não nulos. Compare estas consultas:

```

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

.

```

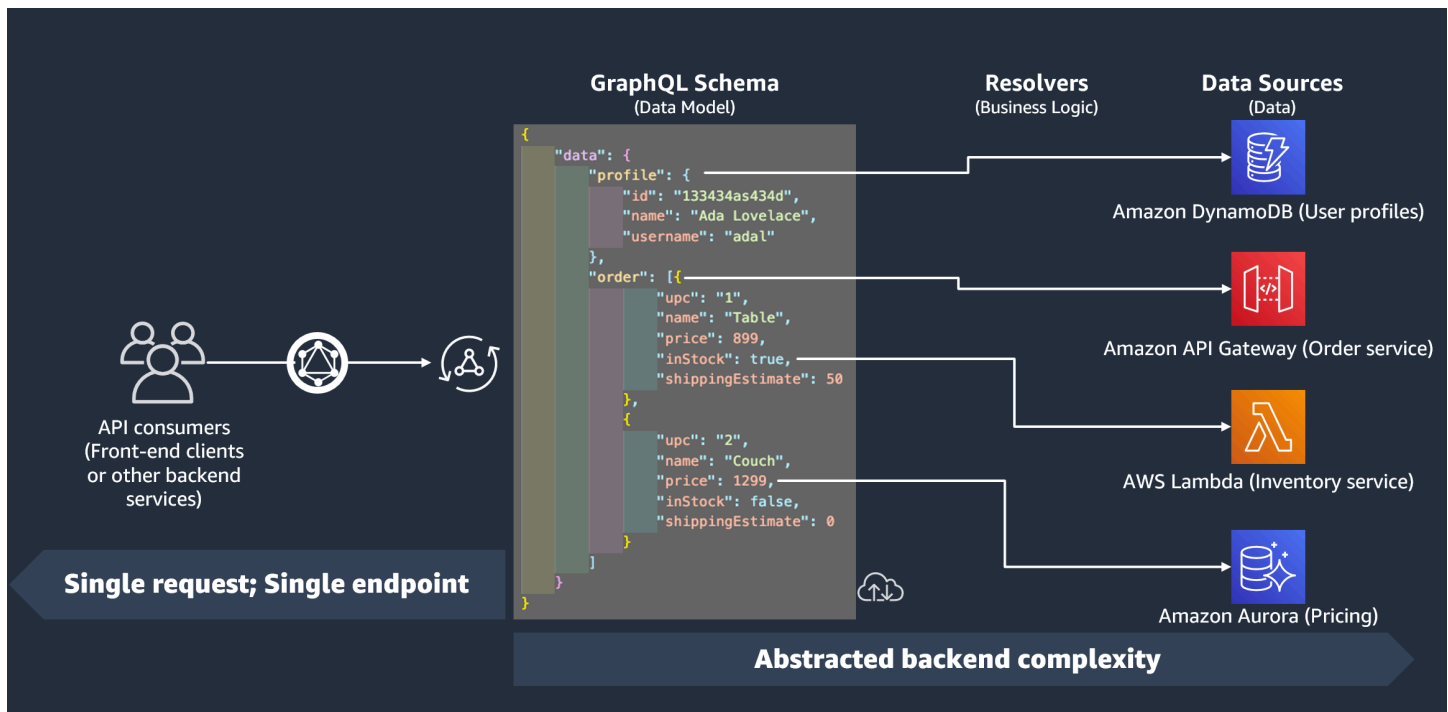
```
.  
.   
  
type Query {  
  people: [Person!]!    # Use case 2  
}  
  
.   
.   
.   
  
type Query {  
  people: [Person!]!    # Use case 3  
}
```

No caso de uso 1, a lista não pode conter itens nulos. No caso de uso 2, a lista em si não pode ser definida como nula. No caso de uso 3, a lista e seus itens não podem ser nulos. No entanto, em qualquer um dos casos, você ainda pode retornar listas vazias.

Como você pode ver, há muitos componentes móveis no GraphQL. Nesta seção, mostramos a estrutura de um esquema simples e os diferentes tipos e campos que um esquema permite. Na seção a seguir, você descobrirá os outros componentes de uma API GraphQL e como eles funcionam com o esquema.

Fontes de dados

Na seção anterior, aprendemos que um esquema define a forma dos seus dados. No entanto, não explicamos de onde vieram esses dados. Em projetos reais, seu esquema é como um gateway que manipula todas as solicitações feitas ao servidor. Quando uma solicitação é feita, o esquema atua como o único endpoint que interage com o cliente. O esquema acessa, processa e retransmite dados da fonte de dados para o cliente. Veja o infográfico abaixo:



AWS AppSync e o GraphQL implementam de forma excelente as soluções Backend For Frontend (BFF). Eles trabalham em conjunto para reduzir a complexidade em grande escala ao abstrair o back-end. Caso seu serviço use fontes de dados e/ou microsserviços diferentes, você pode basicamente abstrair parte da complexidade definindo a forma dos dados de cada fonte (subgráfico) em um único esquema (supergráfico). Isso significa que sua API GraphQL não precisa usar uma única fonte de dados. Você pode associar qualquer número de fontes de dados à sua API GraphQL e especificar no seu código como elas interagirão com o serviço.

Como você pode ver no infográfico, o esquema do GraphQL contém todas as informações que os clientes precisam para solicitar dados. Isso significa que tudo pode ser processado em uma única solicitação, em vez de várias solicitações, como é o caso do REST. Essas solicitações passam pelo esquema, que é o único endpoint do serviço. Quando as solicitações são processadas, um resolvidor (explicado na próxima seção) executa seu código para processar os dados da fonte de dados relevante. Quando a resposta for retornada, o subgráfico vinculado à fonte de dados será preenchido com os dados no esquema.

AWS AppSync oferece suporte a vários tipos diferentes de fontes de dados. Na tabela abaixo, descreveremos cada tipo, listaremos alguns dos benefícios de cada um e forneceremos links úteis que trazem mais contexto.

Fonte de dados	Descrição	Benefícios	Informações complementares
Amazon DynamoDB	“O Amazon DynamoDB é um serviço de banco de dados NoSQL totalmente gerenciado que fornece uma performance rápida e previsível com escalabilidade integrada. O DynamoDB permite que você transfira os encargos administrativos de operação e escalabilidade de um banco de dados distribuído. Assim, você não precisa se preocupar com provisionamento, instalação e configuração de hardware, replicação, correção de software nem escalabilidade de cluster. Além disso, o DynamoDB oferece criptografia em repouso, o que elimina a carga e a complexidade operacionais envolvidas na	- Desempenho em grande escala: o DynamoDB foi projetado pensando no desempenho consistente em qualquer escala. Isso é possível graças ao uso de partições. O DynamoDB particionará automaticamente suas tabelas em várias alocações que serão armazenadas em várias em SSDs vários nós. Isso geralmente aumenta o throughput da rede e reduz a latência. - Capacidade em grande escala: o DynamoDB monitora seu tráfego e permite que você escale automaticamente seu throughput se a rede permanecer	Documentação oficial do DynamoDB Partições Ajuste de escala automático Tolerância a falhas Monitoramento Segurança GraphQL e DynamoDB Operações de resolutor para o DynamoDB Modelo de definição de preços

Fonte de dados	Descrição	Benefícios	Informações complementares
	proteção de dados confidenciais.”	sobrecarregada por longos períodos. - Disponibilidade e tolerância a falhas: o DynamoDB é compatível com várias regiões fisicamente isoladas, cada uma contendo várias zonas de disponibilidade fisicamente isoladas. O DynamoDB alterna automaticamente para uma zona de backup em caso de interrupção do serviço. Você também pode fazer o backup e a replicação dos dados manualmente para garantir a segurança dos dados. - Logs e monitoramento: o DynamoDB fornece várias ferramentas analíticas para suas tabelas. Você pode monitorar	

Fonte de dados	Descrição	Benefícios	Informações complementares
		o desempenho da sua tabela e criar alarmes que notificam sobre mudanças drásticas no serviço. • Segurança: o DynamoDB segue protocolos rígidos para garantir que seus dados estejam em conformidade com os requisitos de segurança da sua organização. • Integração com AWS AppSync: O DynamoDB está perfeitamente integrado ao nosso serviço. Você pode criar novas tabelas do DynamoDB e gerar automaticamente um esquema a partir delas para agilizar seu processo de desenvolvimento. Também fornecemos uma coleção completa de operações para	

Fonte de dados	Descrição	Benefícios	Informações complementares
		solicitar facilmente dados de tabelas existentes do DynamoDB em sua conta em seu resolvedor.	

Fonte de dados	Descrição	Benefícios	Informações complementares
AWS Lambda	“AWS Lambda é um serviço de computação que permite executar código sem provisionar ou gerenciar servidores. O Lambda executa seu código em uma infraestrutura de computação de alta disponibilidade e executa toda a administração dos recursos computacionais, incluindo manutenção do servidor e do sistema operacional, provisionamento e escalabilidade automática da capacidade e registro em log do código. Com o Lambda, tudo o que você precisa fazer é fornecer seu código em um dos runtimes de linguagens compatíveis com o Lambda.”	• Pay-as-you-use modelo: o Lambda cobra apenas quando você usa seus recursos. Ele também permite que você escale a quantidade de recursos usados de acordo com as necessidades do seu aplicativo. • Escalabilidade automática: às vezes, seu aplicativo pode exigir poder computacional extra para um processo específico. O Lambda permite que você escale automaticamente os recursos de computação para atender às necessidades do seu aplicativo. • Tempos de implantação mais rápidos: você pode simplificar seu processo de desenvolvimento	• Documentação oficial • Escalabilidade • implantação • runtimes • Tutorial de resolvedores do Lambda • Modelo de definição de preços

Fonte de dados	Descrição	Benefícios	Informações complementares
		por meio de um pacote de implantação. Use um pacote para carregar seu código de função no serviço Lambda. Em seguida, você pode usar seus ambientes de runtime para testar e executar suas funções. • Versatilidade: o Lambda pode ser usado em vários casos de uso. Você pode integrar perfeitamente o Lambda com serviços da AWS e serviços de terceiros. Alguns exemplos incluem pipelines de CI/CD e serviços de envio em massa. • Integração com AWS AppSync: Você pode facilmente invocar suas funções do Lambda em seu resolvedo	

Fonte de dados	Descrição	Benefícios	Informações complementares
		Para lidar com solicitações. Nosso serviço fornece uma operação de solicitação simplificada para realizar chamadas do Lambda. Permitimos chamadas individuais e em lote.	

Fonte de dados	Descrição	Benefícios	Informações complementares
OpenSearch	“O Amazon OpenSearch Service é um serviço gerenciado que facilita a implantação, a operação e a escalabilidade de OpenSearch clusters na AWS nuvem. O Amazon OpenSearch Service suporta OpenSearch e lega o Elasticsearch OSS (até a versão 7.10, a versão final de código aberto do software). Ao criar um cluster, você tem a opção de escolher qual mecanismo de pesquisa deseja usar. OpenSearch é um mecanismo de pesquisa e análise totalmente de código aberto para casos de uso como análise de registros, monitoramento de aplicativos em tempo real e análise de fluxo de cliques. Para obter mais informações,	• Escalabilidade: você pode escalar facilmente o serviço para atender às suas necessidades de serviço por meio do OpenSearch Serverless. • Ingestão de dados: você pode usar a OpenSearch ingestão para importar, processar e analisar dados. Há muitos aplicativos para ingestão de dados, que você pode encontrar aqui. • Segurança: OpenSearch pode gerenciar sua configuração AWS de segurança, incluindo IAM, CloudTrail, VPCs, autenticação, etc. • Disponibilidade: OpenSearch também oferece suporte a diferentes regiões e zonas de	• Documentação oficial • Sem servidor • Modelo de definição de preços

Fonte de dados	Descrição	Benefícios	Informações complementares
	consulte a documentação do OpenSearch. O Amazon OpenSearch Service provisiona todos os recursos do seu OpenSearch cluster e o executa. Ele também detecta e substitui automaticamente os nós de OpenSearch serviço com falha, reduzindo a sobrecarga associada às infraestruturas autogerenciadas. Você pode escalar seu cluster com uma única chamada de API ou alguns cliques no console.”	disponibilidade em seu serviço. • Integração com AWS AppSync: Em AWS AppSync, você pode usar o GraphQL APIs para armazenar e recuperar dados de domínios de OpenSearch serviço existentes em sua conta.	

Fonte de dados	Descrição	Benefícios	Informações complementares
Endpoints de HTTP	Você pode usar endpoints HTTP como fontes de dados. AWS AppSync pode enviar solicitações aos endpoints com as informações relevantes, como parâmetros e carga útil. A resposta HTTP será exposta ao resolvedor, que retornará a resposta final após concluir suas operações.	Isso é útil para aplicativos simples que não são tão integrados a serviços como o Lambda.	Referência do resolvedor

Fonte de dados	Descrição	Benefícios	Informações complementares
Amazon EventBridge	“EventBridge é um serviço sem servidor que usa eventos para conectar componentes do aplicativo, facilitando a criação de aplicativos escaláveis orientados por eventos. Use-o para rotear eventos de fontes como aplicativos, AWS serviços e software de terceiros desenvolvidos internamente para aplicativos de consumo em toda a sua organização. EventBridge fornece uma maneira simples e consistente de ingerir, filtrar, transformar e entregar eventos para que você possa criar novos aplicativos rapidamente.”	- Arquitetura orientada a eventos: você pode tirar proveito da arquitetura orientada a eventos. - Agendamento: você pode usar o EventBridge Agendador para automatizar suas tarefas e regras usando expressões cron ou definir intervalos de tempo como alternativa aos padrões de eventos. - Pipes: usando EventBridge Pipes, você pode substituir o barramento de eventos por um canal que inclui padrões adicionais de eventos de filtragem e enriquecimento por meio de transformações de dados antes de enviar o evento ao destino.	Documentação oficial Pipes Scheduler Referência do resolvidor Modelo de definição de preços

Fonte de dados	Descrição	Benefícios	Informações complementares
		Integração com AWS AppSync: AWS AppSync permite que você envie eventos para barramentos de eventos usando seu resolvedor.	

Fonte de dados	Descrição	Benefícios	Informações complementares
Bancos de dados relacionais	“O Amazon Relational Database Service (Amazon RDS) é um serviço web que facilita a configuração, a operação e a escalabilidade de um banco de dados relacional na nuvem. AWS Ele fornece capacidade e econômica e redimensionável para um banco de dados relacional padrão do setor e gerencia tarefas comuns de administração de banco de dados.	- Gerenciamento facilitado: periodicamente, o RDS realiza a manutenção de seus recursos. Geralmente, a manutenção envolve atualizações do hardware subjacente da instância do banco de dados, do sistema operacional (SO) subjacente ou da versão do mecanismo de banco de dados. Em circunstâncias normais, você pode decidir quando realizar as atualizações (as exceções incluem patches de segurança). - Recomendações: o atributo de recomendação do RDS fornece sugestões automatizadas para corrigir possíveis	Documentação oficial Atributos Manutenção Recomendações Opções de armazenamento Disponibilidade Segurança Modelo de definição de preços

Fonte de dados	Descrição	Benefícios	Informações complementares
		problemas em sua instância. • Disponibilidade: o RDS está disponível em diferentes regiões físicas em todo o mundo. Você pode distribuir facilmente suas necessidades de banco de dados em diferentes nós para fornecer um melhor serviço aos seus clientes. • Personalização: o RDS é adaptado para atender aos requisitos de grandes corporações. O RDS oferece várias opções para computação, implantação rápida, escalabilidade e armazenamento. • Segurança: o RDS é integrado a várias ferramentas e serviços para manter a segurança do banco de dados nos níveis de	

Fonte de dados	Descrição	Benefícios	Informações complementares
		usuário, banco de dados e rede. Integração com AWS AppSync: Se você estiver procurando por uma solução de back-end madura, AWS AppSync permite enviar, processar, armazenar e retornar dados usando sua instância como fonte de dados.	
Fonte de dados none	Se você não pretende usar uma fonte de dados, pode defini-la como none. Uma fonte de dados none, embora ainda seja explicitamente categorizada como fonte de dados, não é um meio de armazenamento. Apesar disso, ela ainda é útil em algumas instâncias para manipulação e passagem de dados.	- Pode ser útil também para atividades como conversão de dados - Útil ao resolver algo localmente	Referência do resolvidor

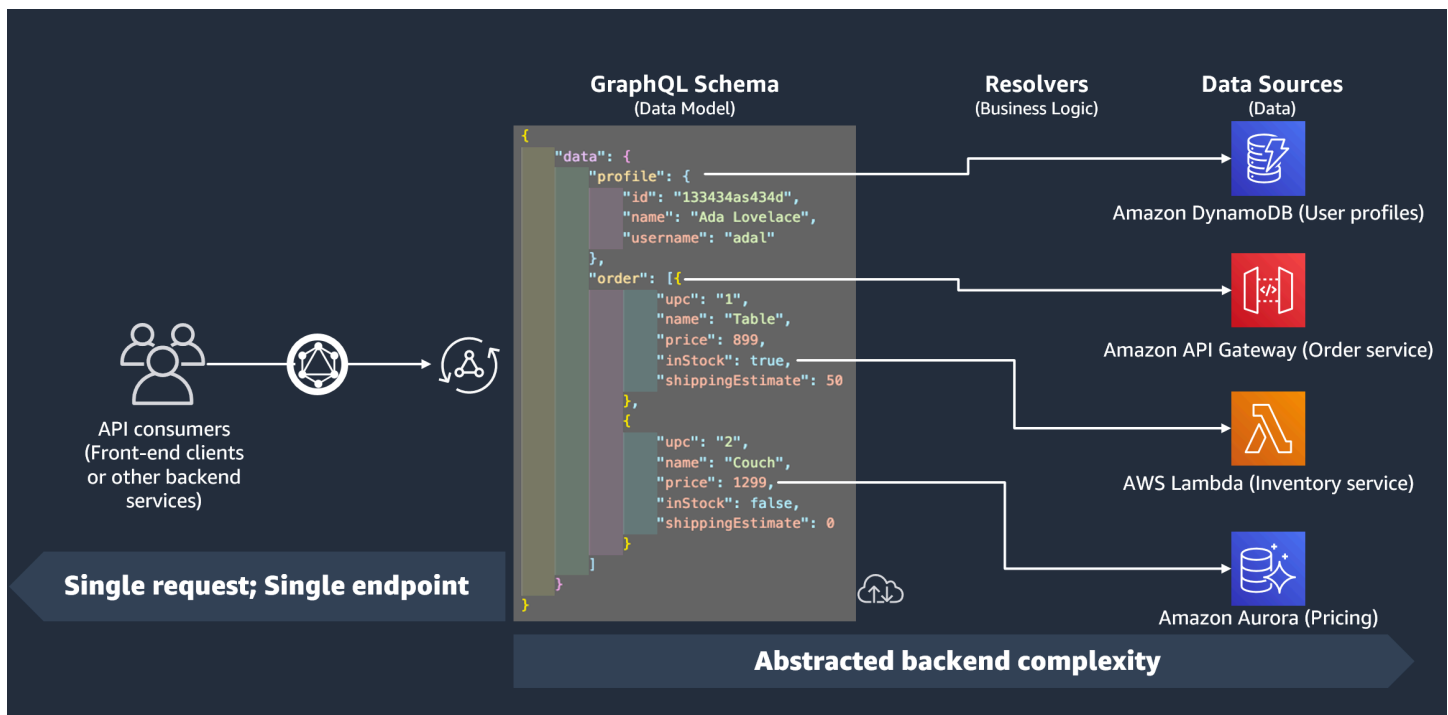
Tip

Para obter mais informações sobre como as fontes de dados interagem AWS AppSync, consulte [Anexando uma fonte de dados](#).

Resolvedores

Nas seções anteriores, você aprendeu sobre os componentes do esquema e da fonte de dados. Agora, precisamos abordar como o esquema e as fontes de dados interagem. Tudo começa com o resolvedor.

Um resolvedor é uma unidade de código que controla como os dados desse campo serão resolvidos quando uma solicitação for feita ao serviço. Os resolvedores são anexados a campos específicos dentro dos seus tipos em seu esquema. Eles costumam ser usados para implementar as operações de mudança de estado para suas operações de campo de consulta, mutação e assinatura. O resolvedor processará a solicitação de um cliente e retornará o resultado, que pode ser um grupo de tipos de saída, como objetos ou escalares:



Runtime do resolvedor

Em AWS AppSync, você deve primeiro especificar um tempo de execução para seu resolvedor. Um runtime do resolvedor indica o ambiente no qual um resolvedor é executado. Também determina o

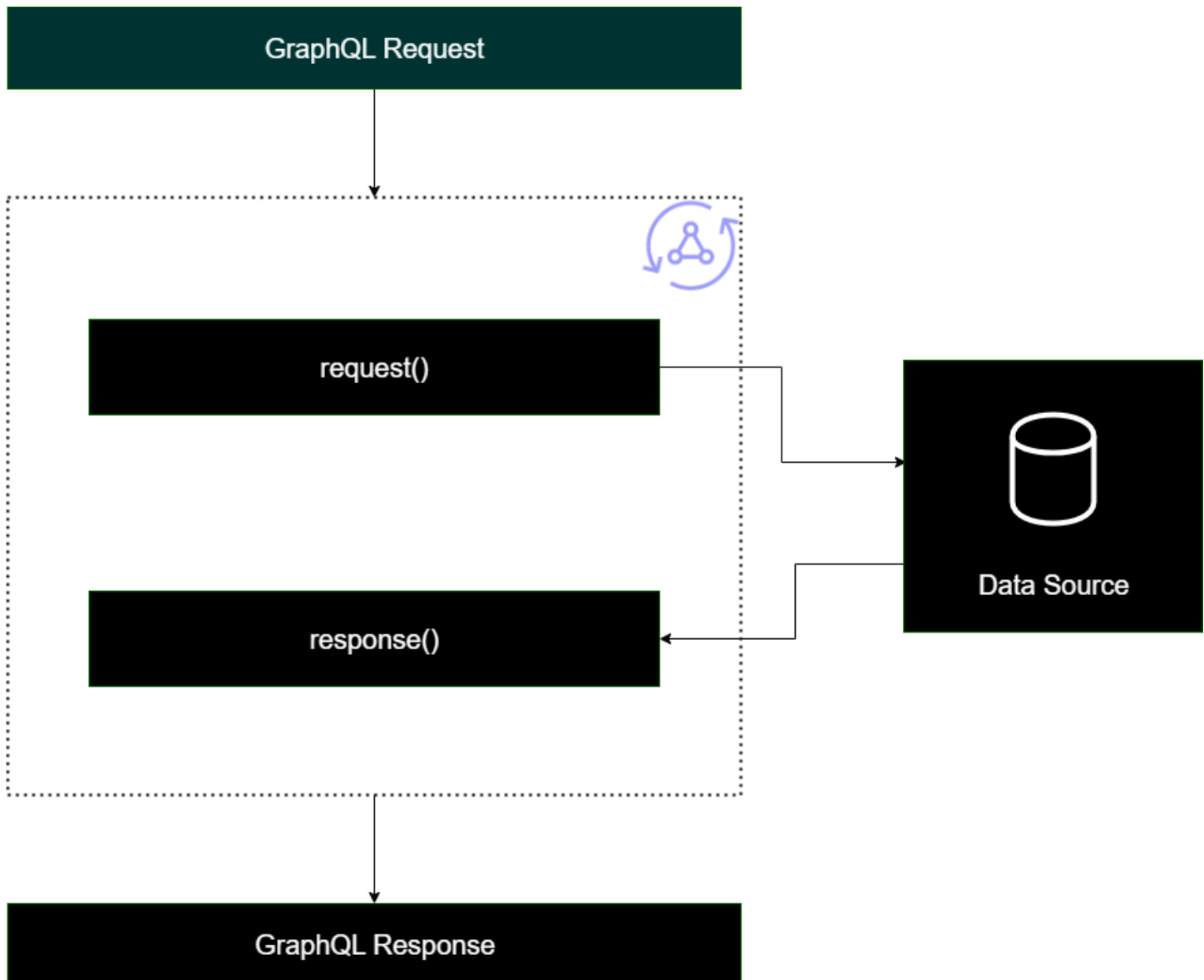
idioma em que seus resolvedores serão escritos. AWS AppSync atualmente suporta APPSYNC_JS JavaScript e Velocity Template Language (VTL). Consulte os [recursos JavaScript de tempo de execução para resolvedores e funções](#) JavaScript ou a [referência do utilitário de modelo de mapeamento Resolver](#) para VTL.

Estrutura do resolvedor

Em termos de código, os resolvedores podem ser estruturados de duas maneiras. Existem resolvedores de unidades e de pipeline.

Resolvedores de unidade

Um resolvedor de unidade é composto de código que define um único manipulador de solicitação e resposta que é executado em uma fonte de dados. O manipulador da solicitação usa um objeto de contexto como argumento e retorna a payload da solicitação usada para chamar sua fonte de dados. O manipulador de respostas recebe uma payload da fonte de dados com o resultado da solicitação executada. O manipulador de respostas transforma a payload em uma resposta do GraphQL para resolver o campo do GraphQL.



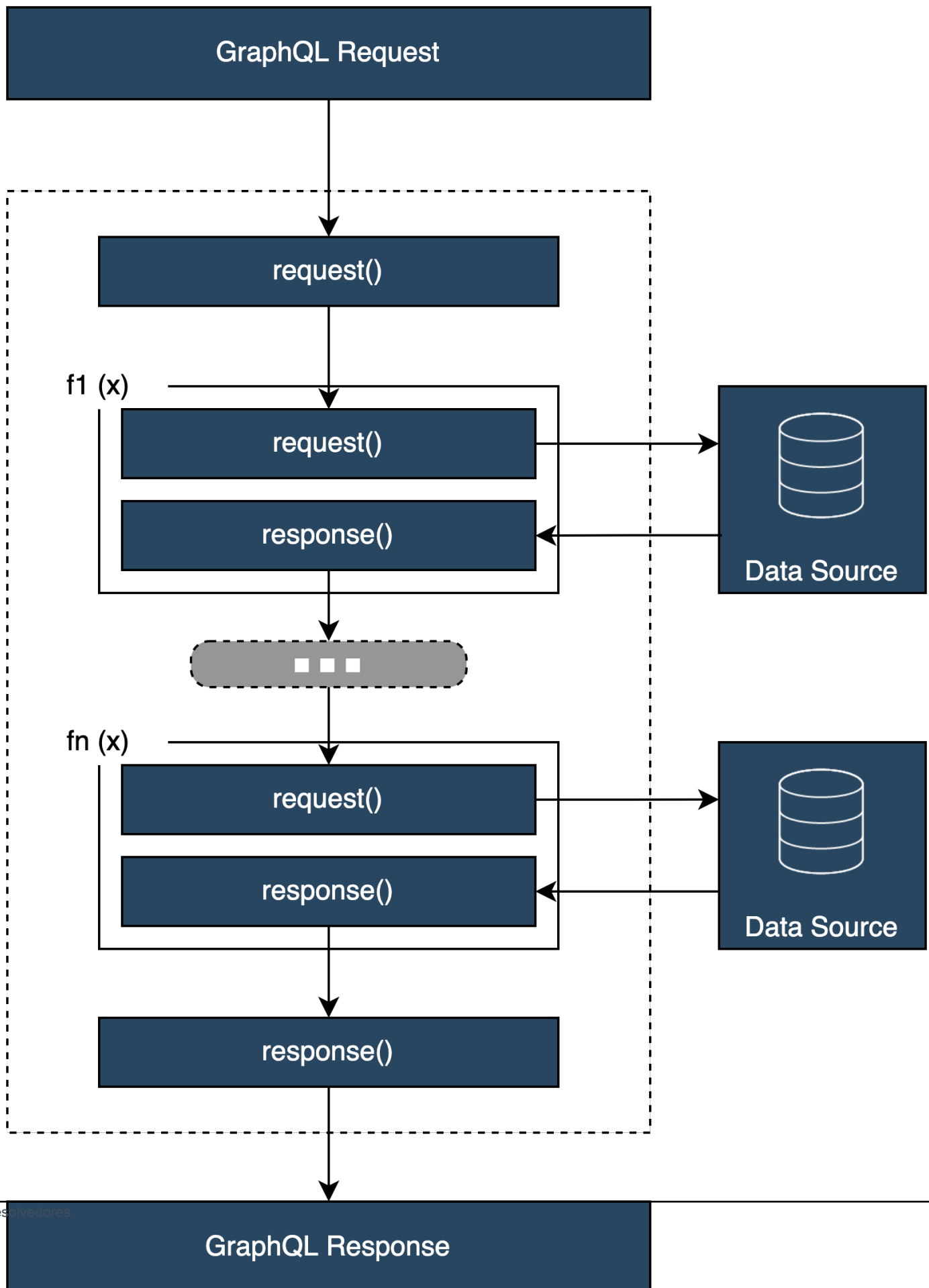
Resolvedores de pipeline

Ao implementar resolvedores, eles seguem uma estrutura geral:

- **Etapa Anterior:** quando uma solicitação é feita pelo cliente, os resolvedores dos campos do esquema que estão sendo usados (normalmente consultas, mutações e assinaturas) recebem os dados da solicitação. O resolvedor começará a processar os dados da solicitação com um manipulador de etapas anteriores, o que permite que algumas operações de pré-processamento sejam executadas antes que os dados passem pelo resolvedor.
- **Função(ões):** Após a execução da etapa anterior, a solicitação é passada para a lista de funções. A primeira função na lista será executada na fonte de dados. Uma função é um subconjunto

do código do resolvedor contendo seu próprio manipulador de solicitações e respostas. Um manipulador de solicitações pegará os dados da solicitação e executará operações na fonte de dados. O manipulador de respostas processará a resposta da fonte de dados antes de passá-la de volta para a lista. Se houver mais de uma função, os dados da solicitação serão enviados para a próxima função a ser executada na lista. As funções na lista serão executadas na ordem definida pelo desenvolvedor. Depois que todas as funções forem executadas, o resultado final será passado para a etapa posterior.

- Etapa Posterior: a etapa Posterior é uma função do manipulador que permite realizar algumas operações finais na resposta da função final antes de passá-la para a resposta do GraphQL.



Estrutura do manipulador do resolvidor

Os manipuladores são normalmente funções chamadas `Request` e `Response`:

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

Em um resolvidor de unidades, haverá apenas um conjunto dessas funções. Em um resolvidor de pipeline, haverá um conjunto dessas etapas para a etapa Anterior e etapa Posterior e um conjunto adicional por função. Para visualizar esse caso, vamos analisar um tipo Query simples:

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

Essa é uma consulta simples com um campo chamado `helloWorld` do tipo `String`. Vamos supor que sempre queremos que esse campo retorne a string “Hello World”. Para implementar esse comportamento, precisamos adicionar o resolvidor a esse campo. Em um resolvidor de unidades, poderíamos adicionar algo assim:

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

A `request` pode ser deixada em branco porque não estamos solicitando ou processando dados. Também podemos supor que nossa fonte de dados seja `None`, indicando que esse código não precisa realizar nenhuma invocação. A resposta simplesmente retorna “Hello World”. Para testar esse resolvidor, precisamos fazer uma solicitação usando o tipo de consulta:

```
query helloWorldTest {
```

```
helloWorld
}
```

Essa é uma consulta chamada `helloWorldTest` que retorna o campo `helloWorld`. Quando executado, o resolvidor do campo `helloWorld` também executa e retorna a resposta:

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

Retornar constantes como essa é a coisa mais simples que você pode fazer. Na realidade, você retornará entradas, listas e muito mais. O exemplo a seguir é mais complicado:

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

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

Aqui estamos retornando uma lista de `Books`. Vamos supor que estejamos usando uma tabela do `DynamoDB` para armazenar dados do livro. Os manipuladores podem ser semelhantes a estes:

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

Nossa solicitação usou uma operação de verificação integrada para pesquisar todas as entradas na tabela, armazenar as descobertas no contexto e depois passá-las para a resposta. A resposta pegou os itens do resultado e os retornou na resposta:

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

Contexto do resolvedor

Em um resolvedor, cada etapa na cadeia de manipuladores deve estar ciente do estado dos dados das etapas anteriores. O resultado de um manipulador pode ser armazenado e passado para outro como argumento. O GraphQL define quatro argumentos básicos do resolvedor:

Argumentos básicos do resolvedor	Descrição
obj, root, parent etc.	O resultado do pai.
args	Os argumentos fornecidos ao campo na consulta do GraphQL.
context	Um valor que é fornecido a cada resolvedor e contém informações contextuais importantes

Argumentos básicos do resolvidor	Descrição
	es, como o usuário atualmente conectado ou o acesso a um banco de dados.
info	Um valor que contém informações específicas do campo relevantes para a consulta atual, bem como os detalhes do esquema.

Em AWS AppSync, o argumento [context](#) (ctx) pode conter todos os dados mencionados acima. Trata-se um objeto criado por solicitação e contém dados como credenciais de autorização, dados de resultados, erros, metadados de solicitações etc. O contexto é uma maneira fácil para os programadores manipularem dados provenientes de outras partes da solicitação. Considere este trecho novamente:

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

A solicitação recebe o contexto (ctx) como argumento; esse é o estado da solicitação. Ele executa uma varredura de todos os itens em uma tabela e, em seguida, armazena o resultado no contexto em `result`. O contexto é então passado para o argumento de resposta, que acessa o `result` e retorna seu conteúdo.

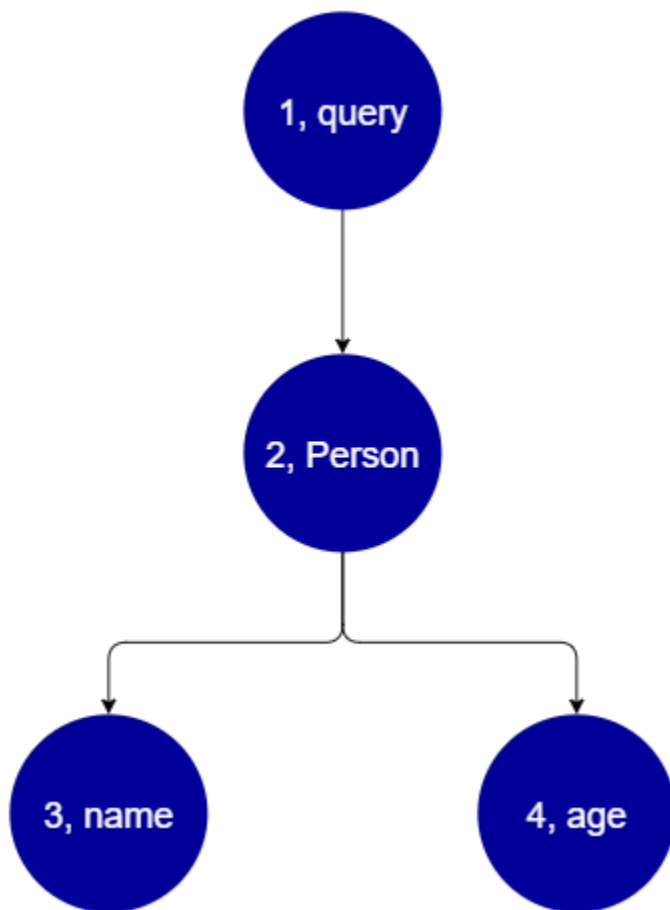
Solicitações e análises

Quando você faz uma consulta ao seu serviço do GraphQL, ela deve passar por um processo de análise e validação antes de ser executada. Sua solicitação será analisada e traduzida em uma árvore de sintaxe abstrata. O conteúdo da árvore é validado por meio da execução de vários

algoritmos de validação em relação ao seu esquema. Após a etapa de validação, os nós da árvore são percorridos e processados. Os resolvedores são invocados, os resultados são armazenados no contexto e a resposta é retornada. Por exemplo, considere esta consulta:

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

Estamos retornando Person com campos name e age. Ao executar essa consulta, a árvore será semelhante a esta:



Na árvore, parece que essa solicitação pesquisará a raiz de Query no esquema. Dentro da consulta, o campo Person será resolvido. Com base em exemplos anteriores, sabemos que isso pode ser uma entrada do usuário, uma lista de valores, etc. A Person provavelmente está vinculada a um tipo de objeto que contém os campos de que precisamos (name e age). Depois que esses dois campos secundários são encontrados, eles são resolvidos na ordem indicada (name seguido por

age). Depois que a árvore for completamente resolvida, a solicitação será concluída e enviada de volta ao cliente.

Propriedades adicionais do GraphQL

O GraphQL consiste em vários princípios de design para manter a simplicidade e a robustez em grande escala.

Declarativo

O GraphQL é declarativo, o que significa que o usuário descreverá (moldará) os dados declarando apenas os campos que deseja consultar. A resposta retornará somente os dados dessas propriedades. Por exemplo, aqui está uma operação que recupera um Book objeto em uma tabela do DynamoDB com o valor ISBN 13 de: id **9780199536061**

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

A resposta retornará os campos na payload (name, year e author) e nada mais:

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

Por causa desse princípio de design, o GraphQL elimina os problemas perenes de busca excessiva e insuficiente que o REST APIs enfrenta em sistemas complexos. Isso resulta em coleta de dados mais eficiente e melhor desempenho da rede.

Hierárquico

O GraphQL é flexível, pois os dados solicitados podem ser moldados pelo usuário para atender às necessidades do aplicativo. Os dados solicitados sempre seguem os tipos e a sintaxe das propriedades definidas na sua API GraphQL. Por exemplo, o trecho a seguir mostra a `getBook` operação com um novo escopo de campo chamado `quotes` que retorna todas as cadeias de aspas armazenadas e páginas vinculadas ao: Book **9780199536061**

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

Ao executar esta consulta, retornaremos o seguinte resultado:

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

```



```

    "page": 251
  }
]
}
}
}

```

Como você pode ver, os campos quotes vinculados ao livro solicitado foram retornados como uma matriz no mesmo formato descrito em nossa consulta. Embora não tenha sido mostrado aqui, o GraphQL tem a vantagem adicional de não ser restrito sobre a localização dos dados que está recuperando. Books e quotes podem ser armazenados separadamente, mas o GraphQL ainda recuperará as informações enquanto a associação existir. Isso significa que sua consulta pode recuperar vários dados autônomos em uma única solicitação.

Introspectivo

O GraphQL é autodocumentado ou introspectivo. Ele oferece suporte a várias operações integradas que permitem que os usuários visualizem os tipos e campos subjacentes no esquema. Por exemplo, aqui está um tipo Foo com um campo date e description:

```

type Foo {
  date: String
  description: String
}

```

Poderíamos usar a operação `__type` para encontrar os metadados de digitação abaixo do esquema:

```

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

```

Isso retornará uma resposta:

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

Esse atributo pode ser usado para descobrir quais tipos e campos um determinado esquema do GraphQL aceita. O GraphQL é compatível com uma ampla variedade dessas operações introspectivas. Para obter mais informações, consulte [Introspecção](#).

Tipificação forte

O GraphQL oferece suporte a tipificação forte por meio de seu sistema de tipos e campos. Quando você define algo em seu esquema, ele deve ter um tipo que possa ser validado antes do runtime. Ele também deve seguir a especificação de sintaxe do GraphQL. Esse conceito não é diferente da programação em outras linguagens. Por exemplo, este é o tipo Foo anterior:

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

Podemos ver que esse Foo é o objeto que será criado. Dentro de uma instância de Foo, haverá um campo `date` e `description`, ambos do tipo primitivo `String` (escalar). Sintaticamente, vemos que Foo foi declarado e seus campos existem dentro de seu escopo. Essa combinação de verificação de tipo e sintaxe lógica garante que sua API GraphQL seja concisa e evidente. A especificação de tipificação e sintaxe do GraphQL pode ser encontrada [aqui](#).

Começando: Criando sua primeira API GraphQL em AWS AppSync

Você pode usar o AWS AppSync console para configurar e iniciar uma API GraphQL. O GraphQL APIs geralmente requer três componentes:

1. **Esquema do GraphQL:** trata-se do esquema da API. Ele define os tipos e campos que você pode solicitar quando uma operação é executada. Para preencher o esquema com dados, você deve conectar as fontes de dados à API do GraphQL. Neste guia de início rápido, criaremos um esquema usando um modelo predefinido.
2. **Fontes de dados:** são os recursos que contêm os dados para preencher a API do GraphQL. Isso pode ser uma tabela do DynamoDB, uma função Lambda, etc AWS AppSync . Suporta uma infinidade de fontes de dados para criar um GraphQL robusto e escalável. APIs As fontes de dados estão vinculadas aos campos no esquema. Sempre que uma solicitação é executada em um campo, os dados da fonte preenchem o campo. Esse mecanismo é controlado pelo resolvidor. Neste guia de início rápido, criaremos uma fonte de dados usando um modelo predefinido juntamente com o esquema.
3. **Resolvedores:** os resolvedores são responsáveis por vincular o campo do esquema à fonte de dados. Eles recuperam os dados da fonte e retornam o resultado com base no que foi definido pelo campo. AWS AppSync suporta ambos JavaScript e VTL para escrever resolvedores para seu GraphQL. APIs Neste guia de início rápido, os resolvedores serão gerados automaticamente com base no esquema e na fonte de dados. Não vamos nos aprofundar nisso nesta seção.

AWS AppSync suporta a criação e configuração de todos os componentes do GraphQL. Ao abrir o console, você pode usar os seguintes métodos para criar a API:

1. **Projetar uma API do GraphQL personalizada** gerando-a por meio de um modelo predefinido e configurando uma nova tabela do DynamoDB (fonte de dados) para suportá-la.
2. **Projetando uma API do GraphQL com um esquema em branco e sem fontes de dados ou resolvedores.**
3. **Usando uma tabela do DynamoDB para importar dados e gerar os tipos e campos do esquema.**
4. **Usando AWS AppSync os WebSocket recursos e a arquitetura Pub/Sub para desenvolver em tempo real. APIs**
5. **Usando o GraphQL existente APIs (fonte APIs) para vincular a uma API mesclada.**

 Note

Recomendamos ler a seção [Criar um esquema](#) antes de trabalhar com ferramentas mais avançadas. Esses guias explicarão exemplos mais simples que você pode usar conceitualmente para criar aplicativos mais complexos. AWS AppSync

AWS AppSync também oferece suporte a várias opções que não são de console para criar o GraphQL APIs. Isso inclui:

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

O exemplo a seguir mostrará como criar os componentes básicos de uma API do GraphQL usando modelos predefinidos e o DynamoDB.

Tópicos

- [Iniciando um esquema no console AWS AppSync](#)
- [Fazendo um tour pelo AWS AppSync console](#)
- [Usando mutações do GraphQL para adicionar dados a uma tabela do DynamoDB no console AWS AppSync](#)
- [Usando consultas do GraphQL para recuperar dados de uma tabela do DynamoDB no console AWS AppSync](#)
- [Seções complementares para o console AWS AppSync](#)

Iniciando um esquema no console AWS AppSync

Neste exemplo, você criará uma Todo API que permite aos usuários criar Todo itens para lembretes de tarefas diárias, como *Finish task* ou *Pick up groceries*. Essa API demonstrará como usar operações do GraphQL em que o estado é mantido em uma tabela do DynamoDB.

Conceitualmente, há três etapas principais para criar sua primeira API do GraphQL. Você deve definir o esquema (tipos e campos), anexar as fontes de dados aos campos e, em seguida, gravar

o resolvedor que manipula a lógica de negócios. No entanto, a experiência do console muda essa ordem. Começaremos definindo como queremos que a fonte de dados interaja com o esquema e, em seguida, definiremos o esquema e o resolvedor posteriormente.

Para criar a API do GraphQL

1. Faça login no AWS Management Console e abra o [AppSync console](#).
2. No Painel, escolha Criar API.
3. Enquanto o GraphQL APIs estiver selecionado, escolha Design do zero. Em seguida, escolha Avançar.
4. Em Nome da API, altere o nome pré-preenchido para **Todo API** e escolha Avançar.

 Note

Também há outras opções presentes aqui, mas não as usaremos neste exemplo.

5. Na seção Especificar recursos do GraphQL, faça o seguinte:
 - a. Escolha Criar tipo com apoio de uma tabela do DynamoDB agora.

 Note

Isso significa que vamos criar uma tabela do DynamoDB para anexar como fonte de dados.

- b. No campo Nome do modelo, insira **Todo**.

 Note

Nosso primeiro requisito é definir o esquema. O Nome do modelo será o nome do tipo. Sendo assim, você está criando uma type chamada Todo que existirá no esquema:

```
type Todo {}
```

- c. Em Campos, faça o seguinte:
 - i. Crie um campo chamado **id**, com o tipo ID e obrigatório definido como Yes.

 Note

Esses são os campos que estarão dentro do escopo do tipo `Todo`. O nome do campo aqui será chamado de `id` com o tipo `ID!`:

```
type Todo {  
  id: ID!  
}
```

AWS AppSync suporta vários valores escalares para diferentes casos de uso.

- ii. Em Adicionar novo campo, crie quatro campos adicionais com os valores de Name definidos como **name**, **when**, **where** e **description**. Os valores de Type serão `String`, e os valores de Array e Required serão `No`. Será algo semelhante a:

Model information

Model name

A model is a type with preconfigured queries, mutations, and subscriptions.

The model name must have 1 to 50 characters. Valid characters: A-Z, a-z, 0-9, and _

Fields

Models have fields. Fields have a name and a type.

Name	Type	Array	Required	
id	ID	No	Yes	Remove
name	String	No	No	Remove
when	String	No	No	Remove
where	String	No	No	Remove
description	String	No	No	Remove
Add new field				

 Note

O tipo completo e seus campos serão semelhantes a:

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

Como estamos criando um esquema usando o modelo predefinido, ele também será preenchido com várias mutações padronizadas com base no tipo, como `create`, `delete` e `update` para ajudar você a preencher sua fonte de dados com facilidade.

- d. Em configurar tabela de modelo, insira um nome de tabela, como **TodoAPITable**. Defina a Chave primária como `id`.

 Note

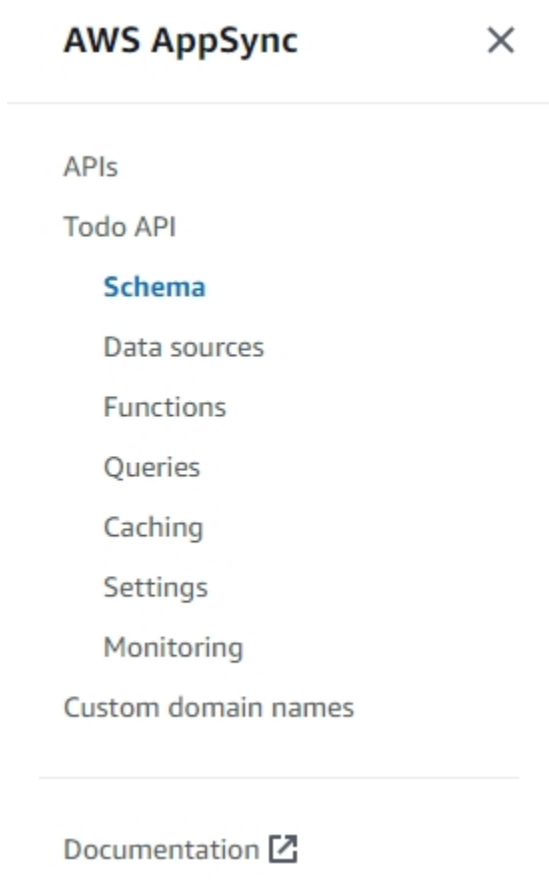
Basicamente, estamos criando uma nova tabela do DynamoDB *TodoAPITable* chamada que será anexada à API como nossa principal fonte de dados. Nossa chave primária é definida como o campo `id` obrigatório que definimos antes disso. Observe que essa nova tabela está em branco e não contém dados, exceto a chave de partição.

- e. Escolha Avançar.
6. Verifique suas alterações e selecione Criar API. Espere um momento para que o AWS AppSync serviço termine de criar sua API.

Você criou com sucesso uma API do GraphQL com o esquema e a fonte de dados do DynamoDB. Para resumir as etapas acima, optamos por criar uma API do GraphQL completamente nova. Definimos o nome da API e, em seguida, adicionamos nossa definição de esquema adicionando o primeiro tipo. Definimos o tipo e os respectivos campos e, em seguida, optamos por anexar uma fonte de dados a um dos campos criando uma tabela do DynamoDB sem dados.

Fazendo um tour pelo AWS AppSync console

Antes de adicionarmos dados à nossa tabela do DynamoDB, devemos analisar os recursos básicos da experiência do AWS AppSync console. A guia do AWS AppSync console no lado esquerdo da página permite que os usuários naveguem facilmente até qualquer um dos principais componentes ou opções de configuração que AWS AppSync fornecem:



Designer do esquema

Escolha Esquema para visualizar o esquema que você acabou de criar. Se você revisar o conteúdo do esquema, notará que ele já foi carregado com várias operações auxiliares para simplificar o processo de desenvolvimento. No editor Esquema, se você percorrer o código, acabará alcançando o modelo definido na seção anterior:

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



```
description: String
}
```

Seu modelo passou a ser o tipo básico usado em todo o esquema. Começaremos adicionando dados à nossa fonte de dados usando mutações que foram geradas automaticamente a partir desse tipo.

Veja abaixo algumas dicas e fatos adicionais sobre o editor Esquema:

1. O editor de código tem recursos de verificação de erros e lint que podem ser usados ao gravar seus próprios aplicativos.
2. O lado direito do console mostra os tipos do GraphQL que foram criados e os resolvedores em diferentes tipos de nível superior, como consultas.
3. Ao adicionar novos tipos a um esquema (por exemplo, `type User { ... }`), você pode AWS AppSync provisionar recursos do DynamoDB para você. Eles incluem a chave primária, a chave de classificação e o design de índice adequados para melhor corresponder ao padrão de acesso aos dados do GraphQL. Se escolher Criar recursos na parte superior e selecionar um desses tipos definidos pelo usuário no menu, você poderá escolher opções de campo diferentes no design do esquema. Abordaremos isso na seção [Criar um esquema](#).

Configuração do resolvedor

No designer do esquema, a seção Resolvedores contém todos os tipos e campos do esquema. Se você percorrer a lista de campos, notará que pode anexar resolvedores a determinados campos escolhendo Anexar. Isso abrirá um editor de código no qual você poderá escrever seu código de resolução. AWS AppSync suporta VTL e JavaScript tempos de execução, que podem ser alterados na parte superior da página escolhendo Ações e, em seguida, Atualizar Tempo de Execução. Na parte inferior da página, você também pode criar funções que executarão várias operações em uma sequência. No entanto, resolvedores são um tópico avançado, e não o abordaremos nesta seção.

Fontes de dados

Escolha Fontes de dados para visualizar a tabela do DynamoDB. Ao escolher a opção Resource (se disponível), você poderá visualizar a configuração da fonte de dados. No exemplo, isso leva ao console do DynamoDB. Nele, você pode editar seus dados. Também é possível editar diretamente alguns dos dados. Para isso, escolha a fonte de dados e, em seguida, Editar. Se precisar excluir a fonte de dados, você poderá escolher a fonte de dados e selecionar Excluir. Por fim, você pode criar

fontes de dados escolhendo Criar fonte de dados e configurando o nome e o tipo. Observe que essa opção serve para vincular o AWS AppSync serviço a um recurso existente. Você ainda precisa criar o recurso em sua conta usando o serviço relevante antes de AWS AppSync reconhecê-lo.

Consultas

Escolha Consultas para ver suas consultas e mutações. Quando criamos nossa API GraphQL usando nosso modelo, geramos AWS AppSync automaticamente algumas mutações e consultas auxiliares para fins de teste. No editor de consultas, o lado esquerdo contém o Explorador. Trata-se de uma lista que mostra todas as mutações e consultas. Você pode habilitar facilmente as operações e os campos que deseja usar aqui clicando nos valores de nome. Isso fará com que o código apareça automaticamente na parte central do editor. Aqui, você pode editar as mutações e consultas modificando os valores. Na parte inferior do editor, há o editor Variáveis de consulta, que permite inserir os valores dos campos para as variáveis de entrada de suas operações. Ao escolher Executar na parte superior do editor, será aberta uma lista suspensa para selecionar a consulta/mutação a ser executada. A saída para essa execução aparecerá no lado direito da página. De volta à seção Explorador, na parte superior, você pode escolher uma operação (Consulta, Mutação, Assinatura) e, em seguida, escolher o símbolo + para adicionar uma nova instância da operação em questão. Na parte superior da página, haverá outra lista suspensa que contém o modo de autorização para a execução de sua consulta. No entanto, não abordaremos esse atributo nesta seção. Para obter mais informações, consulte [Segurança](#).

Configurações

Escolha Configurações para ver algumas opções de configuração para a API do GraphQL. Aqui, você pode ativar algumas opções, como log, rastreamento e funcionalidade de firewall de aplicativos da web. Você também pode adicionar novos modos de autorização para proteger os dados contra vazamentos indesejados para o público. No entanto, essas opções são mais avançadas e não serão abordadas nesta seção.

Note

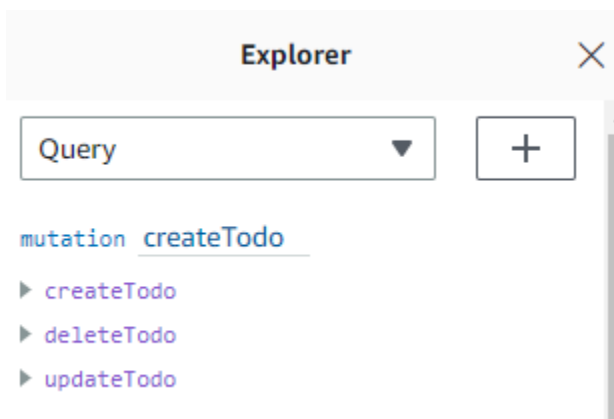
O modo de autorização padrão, API_KEY, usa uma chave da API para testar o aplicativo. Essa é a autorização básica concedida a todo o APIs GraphQL recém-criado. Recomendamos usar outro método de produção. Para fins de exemplo nesta seção, usaremos somente a chave de API. Para obter mais informações sobre os métodos de autorização aceitos, consulte [Segurança](#).

Usando mutações do GraphQL para adicionar dados a uma tabela do DynamoDB no console AWS AppSync

Sua próxima etapa é adicionar dados à tabela em branco do DynamoDB usando uma mutação do GraphQL. As mutações são um dos tipos de operação fundamentais no GraphQL. Elas são definidas no esquema e permitem que você manipule dados na fonte de dados. Em termos de REST APIs, elas são muito semelhantes a operações como PUT ou POST.

Para adicionar a fonte de dados

1. Se você ainda não tiver feito isso, entre no AWS Management Console e abra o [AppSync console](#).
2. Escolha sua API na tabela.
3. Na guia à esquerda, escolha Consultas.
4. Na guia Explorador à esquerda da tabela, você pode ver várias mutações e consultas já definidas no editor de consultas:



Note

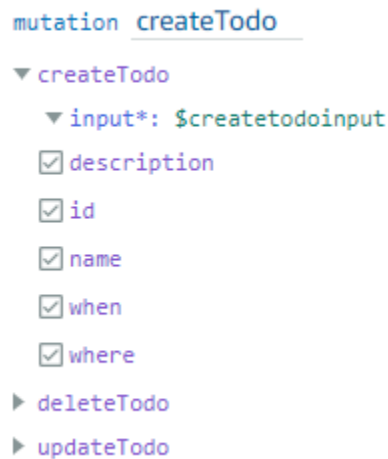
Na verdade, essa mutação está no esquema como o tipo do Mutation. Ela tem o código:

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

Como você pode ver, as operações aqui são semelhantes às aquelas do editor de consultas.

AWS AppSync os gerou automaticamente a partir do modelo que definimos anteriormente. Este exemplo usará a `createTodo` mutação para adicionar entradas à nossa *TodoAPITable* tabela.

- Escolha a operação do `createTodo` expandindo-a sob a mutação do `createTodo`:



```
mutation createTodo
  ▼ createTodo
    ▼ input*: $createtodoinput
      ☒ description
      ☒ id
      ☒ name
      ☒ when
      ☒ where
    ▶ deleteTodo
    ▶ updateTodo
```

Ative as caixas de seleção de todos os campos, como na imagem acima.

Note

Os atributos exibidos aqui são os diferentes elementos modificáveis da mutação. Seu `input` pode ser considerado como parâmetro de `createTodo`. As várias opções com caixas de seleção são os campos que serão retornados na resposta quando uma operação for executada.

- No editor de código no centro da tela, você notará que a operação aparece abaixo da mutação do `createTodo`:

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

```
    id
    description
  }
}
```

Note

Para explicar esse trecho corretamente, também precisamos examinar o código do esquema. A declaração `mutation createTodo($createtodoinput: CreateTodoInput!){}` é a mutação com uma de suas operações, `createTodo`. A mutação completa está localizada no esquema:

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

Voltando à declaração de mutação do editor, o parâmetro é um objeto chamado `$createtodoinput` com um tipo de entrada obrigatório de `CreateTodoInput`. Observe que `CreateTodoInput` (e todas as entradas na mutação) também são definidas no esquema. Por exemplo, aqui está o código clichê para `CreateTodoInput`:

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

Ele contém os campos que definimos em nosso modelo, chamados `name`, `when`, `where` e `description`.

Voltando ao código do editor, em `createTodo(input: $createtodoinput) {}`, declaramos a entrada como `$createtodoinput`, que também foi usada na declaração de mutação. Fazemos isso porque isso permite que o GraphQL valide nossas entradas em relação aos tipos fornecidos e garanta que elas estejam sendo usadas com as entradas corretas.

A parte final do código do editor mostra os campos que serão retornados na resposta após a execução de uma operação:

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

Na guia Variáveis de consulta abaixo do editor, haverá um objeto do `createtodoinput` que pode ter os seguintes dados:

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

Note

É aqui que alocamos os valores para a entrada mencionada anteriormente:

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

Altere o `createtodoinput` adicionando as informações que queremos colocar na tabela do DynamoDB. Nesse caso, queríamos criar alguns itens do `Todo` como lembretes:

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

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

7. Escolha Executar na parte superior do editor. Escolha createTodo na lista suspensa. No lado direito do editor, você verá a resposta. Essa lista pode ser semelhante a:

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

Se você navegar até o serviço do DynamoDB, agora verá uma entrada na fonte de dados com as seguintes informações:

TodoAPITable

► **Scan or query items**
Expand to query or scan items.

✔ Completed. Read capacity units consumed: 2

Items returned (1)

☐	id	description	name	when	where
☐		I need to buy ...	Shopping List	Friday	Home

Para resumir a operação, o mecanismo GraphQL analisou o registro, e um resolvedor o inseriu na tabela do Amazon DynamoDB. Você pode verificar isso no console. Observe que não é preciso enviar um valor de `id`. Um `id` é gerado e retornado nos resultados. Isso ocorre porque o exemplo usou uma função de `autoId()` em um resolvedor do GraphQL para a chave de partição definida nos recursos do DynamoDB. Abordaremos como você pode criar resolvedores em uma seção diferente. Anote o valor do `id` retornado; você o usará na próxima seção para recuperar dados com uma consulta do GraphQL.

Usando consultas do GraphQL para recuperar dados de uma tabela do DynamoDB no console AWS AppSync

Agora que há um registro no banco de dados, você obterá resultados ao executar uma consulta. Uma consulta é uma das outras operações fundamentais do GraphQL. Ela é usada para analisar e recuperar informações da fonte de dados. Em termos de REST APIs, isso é semelhante à GET operação. A principal vantagem das consultas do GraphQL é a capacidade de especificar os requisitos de dados exatos do aplicativo para que você busque os dados relevantes no momento certo.

Para consultar a fonte de dados

1. Se você ainda não tiver feito isso, entre no AWS Management Console e abra o [AppSync console](#).
2. Escolha sua API na tabela.
3. Na guia à esquerda, escolha Consultas.
4. Na guia Explorador à esquerda da tabela, abaixo de `querylistTodos`, expanda a operação do `getTodo`:

```
query listTodos
{
  getTodo {
    id*
    description
    id
    name
    when
    where
  }
  listTodos
}
```


5. No editor de código, você deverá ver o código da operação:

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

Em (id: ""), preencha o valor que você salvou no resultado da operação de mutação. Neste exemplo, o resultado seria .

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

6. Escolha Executar e, em seguida, listTodos. O resultado aparecerá à direita do editor. Nosso exemplo ficou semelhante a:

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

 Note

As consultas retornam apenas os campos especificados. Você pode desmarcar os campos desnecessários excluindo-os do campo de retorno:

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

Você também pode desmarcar a caixa na guia Explorador ao lado do campo que você deseja excluir.

7. Você também pode tentar a operação do `listTodos` repetindo as etapas para criar uma entrada na fonte de dados e, em seguida, repetindo as etapas de consulta com a operação do `listTodos`. Veja um exemplo em que adicionamos uma segunda tarefa:

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

Ao chamar a operação do `listTodos`, ela retornou as entradas antigas e novas:

```
{
  "data": {
    "listTodos": {
      "items": [
        {
          "id": "abcdefgh-1234-1234-1234-abcdefghijkl",
          "name": "Shopping List",
          "when": "Friday",
          "where": "Home",

```

```
      "description": "I need to buy eggs"
    },
    {
      "id": "aaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
      "name": "Second Task",
      "when": "Monday",
      "where": "Home",
      "description": "I need to mow the lawn"
    }
  ]
}
```

Seções complementares para o console AWS AppSync

Essas seções são uma referência para AWS AppSync tópicos mais avançados. Recomendamos seguir a seção Leitura suplementar antes de realizar qualquer outra ação.

Integração

Na guia do console, se você escolher o nome da API, a página Integração será exibida:

AWS AppSync

APIs

Todo API

Schema

Data sources

Functions

Queries

Caching

Settings

Monitoring

Custom domain names

Ela resume as etapas para configurar a API e descreve as próximas etapas para criar um aplicativo cliente. A seção Integrar com seu aplicativo fornece detalhes sobre como usar o conjunto de [ferramentas do AWS Amplify](#) para automatizar o processo de conexão da sua API com iOS, Android e JavaScript aplicativos por meio de configuração e geração de código. A cadeia de ferramentas do Amplify oferece suporte total à criação de projetos a partir da estação de trabalho local, incluindo o provisionamento do GraphQL e fluxos de trabalho para CI/CD.

A seção Amostras de clientes também lista exemplos de aplicativos cliente (por exemplo JavaScript, iOS, Android) para testar uma end-to-end experiência. É possível clonar e fazer download das amostras e do arquivo de configuração que contém as informações necessárias (como o URL do endpoint) de que precisa para começar a usar. Siga as instruções na página [Cadeia de ferramentas do AWS Amplify](#) para executar o aplicativo.

Leitura complementar

- [Projetando o GraphQL APIs](#) - Este é um guia completo para criar seu GraphQL usando um esquema em branco sem fontes de dados ou resolvedores.

Projetando o GraphQL APIs com AWS AppSync

AWS AppSync permite que você crie o GraphQL APIs usando a experiência do console. Abordamos isso rapidamente na seção [Lançar um esquema de amostra](#). No entanto, esse guia não mostrou todo o catálogo de opções e configurações que você poderia aproveitar. AWS AppSync

Quando você escolhe criar uma API GraphQL no console, há várias opções a serem exploradas. Se você seguiu as orientações do guia [Launching a sample schema](#), mostramos como criar uma API usando um modelo predefinido. Nas seções a seguir, guiaremos você pelo restante das opções e configurações para criar o APIs AWS AppSync GraphQL no.

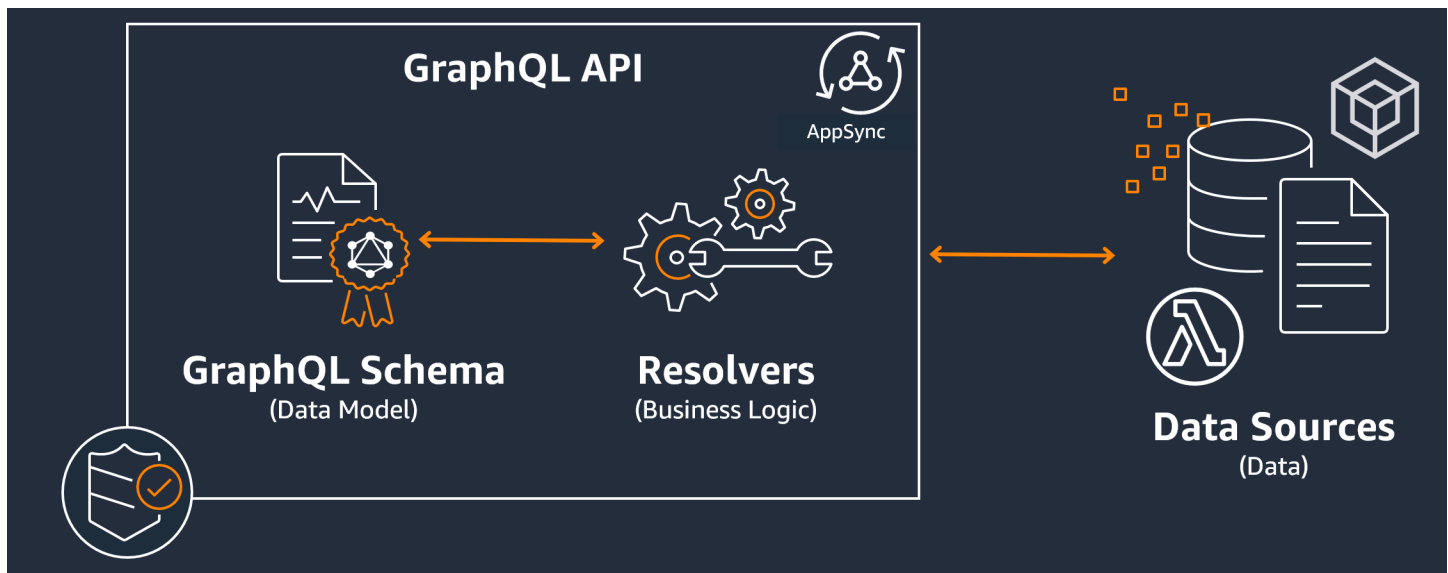
Nesta seção, você analisa os seguintes conceitos:

1. [Blank APIs or imports](#): este guia aborda todo o processo de criação de uma API GraphQL. Você vai aprender a criar uma API GraphQL a partir de um modelo em branco sem modelo, configurar fontes de dados para seu esquema e adicionar seu primeiro resolvidor a um campo.
2. [Real-time data](#): este guia mostrará as possíveis opções para criar uma API usando AWS AppSync o WebSocket mecanismo.
3. [Merged APIs](#): Este guia mostrará como criar um novo GraphQL APIs associando e mesclando dados de vários GraphQL existentes. APIs
4. [the section called “Construindo o GraphQL APIs com introspecção do RDS”](#): este guia mostrará como integrar as tabelas do Amazon RDS usando uma API de dados.

Estruturação de uma API GraphQL (em branco ou importada) APIs

Antes de criar sua API GraphQL a partir de um modelo em branco, leia os conceitos sobre o GraphQL. Há três componentes fundamentais de uma API GraphQL:

1. O esquema é o arquivo que contém a forma e a definição dos seus dados. Quando uma solicitação é feita por um cliente ao seu serviço do GraphQL, os dados retornados seguem a especificação do esquema. Para obter mais informações, consulte [Esquemas do GraphQL](#).
2. A fonte de dados é anexada ao seu esquema. Quando uma solicitação é feita, os dados são recuperados e modificados. Para obter mais informações, consulte [Data sources](#).
3. O resolvidor fica entre o esquema e a fonte de dados. Quando uma solicitação é feita, o resolvidor executa a operação nos dados da fonte e retorna o resultado como resposta. Para obter mais informações, consulte [Resolvers](#).



AWS AppSync gerencia você, APIs permitindo que você crie, edite e armazene o código para seus esquemas e resolvedores. Suas fontes de dados serão provenientes de repositórios externos, como bancos de dados, tabelas do DynamoDB e funções do Lambda. Se você estiver usando um AWS serviço para armazenar seus dados ou estiver planejando fazer isso, AWS AppSync oferece uma experiência quase perfeita ao associar dados de suas contas AWS ao seu GraphQL. APIs

Na próxima seção, você aprenderá a criar cada um desses componentes usando o AWS AppSync serviço.

Tópicos

- [Como projetar seu esquema GraphQL](#)
- [Anexando uma fonte de dados em AWS AppSync](#)
- [Configurando resolvedores em AWS AppSync](#)
- [Usando uma AWS AppSync API com o AWS CDK](#)

Como projetar seu esquema GraphQL

O esquema GraphQL é a base de qualquer implementação de servidor de GraphQL. Cada API GraphQL é definida por um único esquema que contém tipos e campos que descrevem como os dados das solicitações serão preenchidos. Os dados que fluem pela sua API e as operações realizadas devem ser validados com base no esquema.

O [sistema do tipo GraphQL](#) descreve os recursos de um servidor de GraphQL e é usado para determinar se uma consulta é válida. Um sistema de tipo de servidor geralmente é chamado de

esquema e pode consistir em diferentes tipos de objetos, escalares, entradas, entre outros. O GraphQL é declarativo e fortemente tipificado, o que significa que os tipos serão bem definidos no runtime e retornarão somente o que foi especificado.

AWS AppSync permite que você defina e configure esquemas do GraphQL. A seção a seguir descreve como criar esquemas do GraphQL do zero usando AWS AppSync os serviços da.

Estruturar um esquema do GraphQL



Tip

Consulte a seção [Schemas](#) antes de continuar.

O GraphQL é uma ferramenta poderosa para implementar serviços de API. A descrição (em tradução livre) no [site do GraphQL](#) é a seguinte:

“O GraphQL é uma linguagem de consulta APIs e um tempo de execução para atender essas consultas com seus dados existentes. O GraphQL fornece uma descrição completa e compreensível dos dados em sua API, dá aos clientes o poder de pedir exatamente o que precisam e nada mais, facilita a evolução ao APIs longo do tempo e habilita ferramentas poderosas para desenvolvedores. “

Esta seção aborda a primeira parte da sua implementação do GraphQL, o esquema. De acordo com a citação acima, um esquema “fornece uma descrição completa e clara dos dados em sua API”. Em outras palavras, um esquema do GraphQL é uma representação textual dos dados, das operações e das relações entre eles referentes ao seu serviço. O esquema é considerado o principal ponto de entrada para a implementação do serviço do GraphQL. Não é de surpreender que muitas vezes seja uma das primeiras coisas que você faz em seu projeto. Consulte a seção [Schemas](#) antes de continuar.

Citando a seção [Schemas](#), os esquemas do GraphQL são escritos em Schema Definition Language (SDL). O SDL é composto por tipos e campos com uma estrutura estabelecida:

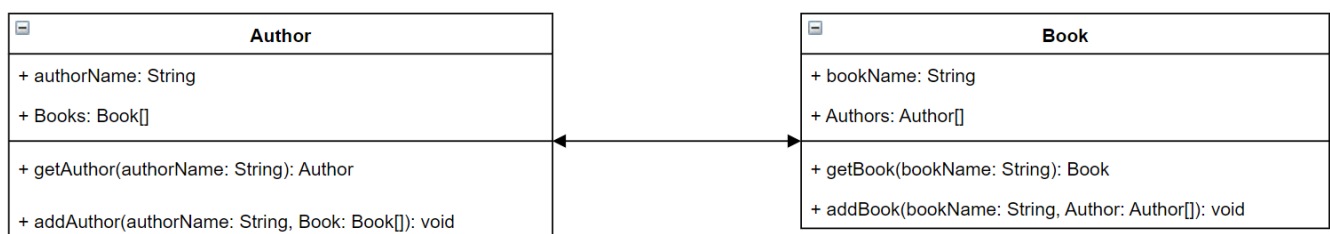
- Tipos: os tipos são como o GraphQL define a forma e o comportamento dos dados. O GraphQL é compatível com uma infinidade de tipos que serão explicados ainda nesta seção. Cada um dos tipos definidos em seu esquema terá um escopo próprio. Dentro do escopo, um ou mais campos vão apresentar um valor ou lógica que será usada em seu serviço do GraphQL. Os tipos têm muitas funções diferentes, sendo as mais comuns objetos ou escalares (tipos de valores primitivos).

- **Campos:** os campos existem dentro do escopo de um tipo e contêm o valor solicitado do serviço do GraphQL. Eles são muito semelhantes às variáveis de outras linguagens de programação. A forma dos dados que você define em seus campos determinará como os dados são estruturados em uma operação de solicitação/resposta. Isso permite que os desenvolvedores prevejam o que será retornado sem saber como o back-end do serviço é implementado.

Os esquemas mais simples conterão três categorias de dados:

1. **Raízes do esquema:** as raízes definem os pontos de entrada do seu esquema. Elas apontam para os campos que realizarão alguma operação nos dados, como adicionar, excluir ou modificar algo.
2. **Tipos:** esses são tipos básicos usados para representar o formato dos dados. Eles são muito semelhantes a objetos ou representações abstratas de algo com características definidas. Por exemplo, você pode criar um objeto `Person` que represente uma pessoa em um banco de dados. As características de cada pessoa serão definidas dentro de `Person` como campos. Eles podem ser qualquer dado da pessoa, como o nome, a idade, o emprego, o endereço etc.
3. **Tipos de objetos especiais:** esses são os tipos que definem o comportamento das operações em seu esquema. Cada tipo de objeto especial é definido uma vez por esquema. Eles são colocados primeiro na raiz do esquema e, em seguida, definidos no corpo do esquema. Cada campo em um tipo de objeto especial define uma única operação a ser implementada pelo seu solucionador.

Para entender melhor, imagine que você esteja criando um serviço que armazena dados de autores de livros e suas obras. Cada autor tem um nome e uma série de livros escritos. Cada livro tem um nome e uma lista de autores associados. Além disso, queremos adicionar ou recuperar livros e autores. Uma representação UML simples desse relacionamento pode ser como esta:



No GraphQL, as entidades `Author` e `Book` representam dois tipos diferentes de objetos em seu esquema:


```
type Author {  
}  
  
type Book {  
}
```

Author contém **authorName** e **Books**, enquanto **Book** contém **bookName** e **Authors**. Eles podem ser representados como os campos dentro do escopo de seus tipos:

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

Como você pode ver, as representações de tipo estão muito próximas do diagrama. No entanto, os métodos são os pontos mais complexos. Eles serão colocados em um dos poucos tipos de objetos especiais como um campo. A categorização como objeto especial depende do comportamento dele. O GraphQL contém três tipos fundamentais de objetos especiais: consultas, mutações e assinaturas. Para obter mais informações, consulte [Special objects](#).

Como **getAuthor** e **getBook** estão solicitando dados, eles serão colocados em um tipo de objeto especial de **Query**:

```
type Author {  
  authorName: String  
  Books: [Book]  
}  
  
type Book {  
  bookName: String  
  Authors: [Author]  
}  
  
type Query {  
  getAuthor(authorName: String): Author  
  getBook(bookName: String): Book
```

```
}
```

As operações são vinculadas à consulta, que por sua vez está vinculada ao esquema. Adicionar uma raiz de esquema definirá o tipo de objeto especial (neste caso, `Query`) como um dos seus pontos de entrada. Isso pode ser feito usando a palavra-chave `schema`:

```
schema {  
  query: Query  
}  
  
type Author {  
  authorName: String  
  Books: [Book]  
}  
  
type Book {  
  bookName: String  
  Authors: [Author]  
}  
  
type Query {  
  getAuthor(authorName: String): Author  
  getBook(bookName: String): Book  
}
```

Analisando os últimos dois métodos, `addAuthor` e `addBook` estão adicionando informações ao seu banco de dados para que sejam definidos em um tipo de objeto especial `Mutation`. No entanto, na página [Tipos](#), não é permitido usar entradas que fazem referência direta a objetos, porque são tipos de saída. Nesse caso, não podemos usar `Author` nem `Book`, então precisamos criar um tipo de entrada com os mesmos campos. Neste exemplo, adicionamos `AuthorInput` e `BookInput`, que aceitam os mesmos campos de seus respectivos tipos. Em seguida, criamos nossa mutação usando as entradas como nossos parâmetros:

```
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!
}
```

Vamos analisar o que acabamos de fazer:

1. Criamos um esquema com os tipos `Author` e `Book` para representar nossas entidades.
2. Adicionamos os campos contendo as características de nossas entidades.
3. Adicionamos uma consulta para recuperar essas informações do banco de dados.
4. Adicionamos uma mutação para manipular dados no banco de dados.
5. Adicionamos tipos de entrada para substituir nossos parâmetros de objeto na mutação para cumprir as regras do GraphQL.
6. Adicionamos a consulta e a mutação ao nosso esquema raiz para que a implementação do GraphQL identifique a localização do tipo raiz.

Como você pode notar, o processo de criação de um esquema usa muitos conceitos da modelagem de dados (especialmente da modelagem de banco de dados) em geral. Podemos dizer que

o esquema se ajusta à forma dos dados da fonte. Ele também serve como o modelo que o solucionador implementará. Nas seções a seguir, você aprenderá como criar um esquema usando várias ferramentas e AWS serviços suportados.

Note

Os exemplos nas seções a seguir não devem ser executados em uma aplicação real. Eles apenas mostram os comandos para que você crie suas próprias aplicações.

Criar esquemas

Seu esquema estará em um arquivo chamado `schema.graphql`. AWS AppSync permite que os usuários criem novos esquemas para seu APIs GraphQL usando vários métodos. Neste exemplo, criaremos uma API em branco junto com um esquema em branco.

Console

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
 - a. No Painel, escolha Criar API.
 - b. Em Opções de API, escolha GraphQL APIs, Design from scratch e Next.
 - i. Para o nome da API, troque o nome pré-preenchido algo que seja necessário para a aplicação.
 - ii. Para obter detalhes de contato, você pode inserir um ponto de contato para identificar um gerente para a API. Esse é um campo opcional.
 - iii. Em Configuração da API privada, é possível habilitar os atributos da API privada. Uma API privada só pode ser acessada de um endpoint da VPC (VPCE) configurado. Para obter mais informações, consulte [Privado APIs](#).

Não recomendamos habilitar esse atributo para este exemplo. Após analisar suas entradas, selecione Próximo.

- c. Em Criar um tipo de GraphQL, você pode criar uma tabela do DynamoDB para usar como fonte de dados ou ignorar e fazer isso depois.

Para este exemplo, escolha Criar recursos do GraphQL mais tarde. Vamos criar um recurso em uma seção separada.

- d. Revise suas entradas e selecione Criar API.
2. Você estará no painel da sua API específica. É possível identificar o nome da API na parte superior do painel. Se esse não for o caso, você pode selecionar APIs na barra lateral e escolher sua API no APIs painel.
 - Na barra lateral abaixo do nome da sua API, escolha Esquema.
3. Você pode configurar seu arquivo `schema.graphql` no editor de esquemas. Ele pode estar vazio ou preenchido com tipos gerados a partir de um modelo. À direita, você tem a seção Solucionadores para anexar solucionadores aos campos do esquema. Não examinaremos os solucionadores nesta seção.

CLI

Note

Ao usar a CLI, verifique se você tem as permissões certas para acessar e criar recursos no serviço. Considere definir políticas de [privilegio mínimo](#) para usuários não administradores que precisam acessar o serviço. Para obter mais informações sobre AWS AppSync políticas, consulte [Gerenciamento de identidade e acesso para AWS AppSync](#).

Além disso, recomendamos primeiro ler a versão do console.

1. Caso ainda não tenha feito isso, [instale](#) e configure o AWS CLI e adicione sua [configuração](#).
2. Crie um objeto da API GraphQL executando o comando [create-graphql-api](#).

Você precisará digitar dois parâmetros para esse comando específico:

1. O nome da sua API.
2. O `authentication-type`, ou o tipo de credencial usado para acessar a API (IAM, OIDC etc.).

Note

Outros parâmetros, como `Region` devem ser configurados, mas geralmente usam como padrão os valores de configuração da CLI.

Veja um exemplo de comando:

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

Uma saída será retornada na CLI. Veja um exemplo abaixo:

```
{
  "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

Esse é um comando opcional que carrega um esquema existente no serviço do AWS AppSync usando um blob Base64. Não usaremos esse comando para preservar este exemplo.

Execute o comando [start-schema-creation](#).

Você precisará digitar dois parâmetros para esse comando específico:

1. Seu `api-id` da etapa anterior.
2. O esquema `definition` é um blob binário codificado com Base64.

Veja um exemplo de comando:

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

Uma saída será retornada:

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

Esse comando não retornará a saída final depois do processamento. Você deve usar um comando separado, [get-schema-creation-status](#), para ver o resultado. Esses dois comandos são assíncronos, portanto, você pode verificar o status da saída mesmo durante a criação do esquema.

CDK

Tip

Antes de usar o CDK, recomendamos revisar a documentação oficial do CDK junto com AWS AppSync a referência do CDK.

As etapas listadas abaixo mostram apenas um exemplo geral do trecho usado para adicionar um recurso específico. Isso não é uma solução funcional para seu código de produção. Também presumimos que você já tenha uma aplicação em funcionamento.

1. O ponto de partida do CDK é um pouco diferente. O ideal é que seu arquivo `schema.graphql` já tenha sido criado. Você só precisa criar um novo arquivo com a extensão `.graphql`. Ele pode ser um arquivo vazio.
2. Em geral, talvez seja necessário adicionar a diretiva de importação ao serviço que você está usando. Por exemplo, estas são as possíveis formas:

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

Para adicionar uma API GraphQL, seu arquivo de pilha precisa importar o serviço: AWS AppSync

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

Note

Isso significa que estamos importando todo o serviço com a palavra-chave `appsync`. Para usar isso em seu aplicativo, suas AWS AppSync construções usarão o formato `appsync.construct_name`. Por exemplo, se quiséssemos criar uma API GraphQL, teríamos um `new appsync.GraphqlApi(args_go_here)`. A etapa a seguir mostra isso.

3. A API mais básica do GraphQL incluirá um name para a API e o caminho do schema.

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

Note

Vamos analisar o que esse trecho faz. Dentro do escopo da `api`, estamos criando uma nova API GraphQL chamando o `appsync.GraphqlApi(scope: Construct, id: string, props: GraphqlApiProps)`. O escopo é `this`, que se refere ao objeto atual. O `id` é `API_ID`, que será o nome do recurso da sua API GraphQL AWS CloudFormation quando ela for criada. O `GraphqlApiProps` contém o nome da sua API GraphQL e o `schema`. `schema` isso gerará um esquema (`SchemaFile.fromAsset`) pesquisando o caminho absoluto (`__dirname`) do `.graphql` arquivo (`schema_name.graphql`). Em um cenário real, seu arquivo de esquema provavelmente estará na aplicação do CDK. Para usar as alterações feitas na sua API GraphQL, você precisará reimplantar a aplicação.

Adicionar tipos aos esquemas

Agora que você adicionou seu esquema, pode começar a adicionar os tipos de entrada e saída. Os tipos aqui não devem ser usados em código real; eles são apenas exemplos para ajudar você a entender o processo.

Primeiro, vamos criar um tipo de objeto. No código real, não é necessário começar com esses tipos. Você pode criar o tipo que preferir a qualquer momento, desde que siga as regras e a sintaxe do GraphQL.

Note

As próximas seções usarão o editor de esquemas, portanto, mantenha-o aberto.

Console

- Você pode criar um tipo de objeto usando a palavra-chave `type` junto com o nome do tipo:

```
type Type_Name_Goes_Here {}
```

Dentro do escopo do tipo, você pode adicionar campos que representam as características do objeto:

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

Veja um exemplo abaixo:

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

 Note

Nesta etapa, adicionamos um tipo de objeto genérico com um campo obrigatório `id` armazenado como `ID`, um campo `title` armazenado como `String`, e um campo `date` armazenado como `AWSDateTime`. Para visualizar uma lista de tipos e campos e o que eles fazem, consulte [Schemas](#). Para visualizar uma lista de escalares e o que eles fazem, consulte a [Type reference](#).

CLI

 Note

Recomendamos que seja feita a leitura da versão do console primeiro.

- Você pode criar um tipo de objeto executando o comando [create-type](#).

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. A `definition`, ou o conteúdo do seu tipo. No exemplo do console, tínhamos:

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

3. O `format` da sua entrada. Neste exemplo, usamos o `SDL`.

Veja um exemplo de comando:

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

Uma saída será retornada na CLI. Veja um exemplo abaixo:

```
{
  "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

Nesta etapa, adicionamos um tipo de objeto genérico com um campo obrigatório `id` armazenado como `ID`, um campo `title` armazenado como `String`, e um campo `date` armazenado como `AWSDateTime`. Para visualizar uma lista de tipos e campos e o que eles fazem, consulte [Schemas](#). Para visualizar uma lista de escalares e o que eles fazem, consulte [Type reference](#).

Além disso, inserir a definição diretamente funciona para tipos menores, mas é inviável para adicionar tipos maiores ou vários tipos. Você pode adicionar tudo em um arquivo `.graphql` e depois [transmiti-lo como a entrada](#).

CDK

Tip

[Antes de usar o CDK, recomendamos revisar a documentação oficial do CDK junto com AWS AppSync a referência do CDK.](#)

As etapas listadas abaixo mostram apenas um exemplo geral do trecho usado para adicionar um recurso específico. Isso não é uma solução funcional para seu código de produção. Também presumimos que você já tenha uma aplicação em funcionamento.

Para adicionar um tipo, você precisa adicioná-lo ao seu arquivo `.graphql`. Por exemplo, o exemplo do console foi:

```
type Obj_Type_1 {
```

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

Você pode adicionar seus tipos diretamente ao esquema, como qualquer outro arquivo.

 Note

Para usar as alterações feitas na sua API GraphQL, você precisará reimplantar a aplicação.

O [tipo de objeto](#) tem campos que são [tipos escalares](#), como cadeias de caracteres e números inteiros. AWS AppSync também permite que você use tipos escalares aprimorados, como `AWSDateTime` adição aos escalares básicos do GraphQL. Qualquer campo que termine com um ponto de exclamação é um campo obrigatório.

O tipo de escalar `ID` é um identificador exclusivo que pode ser `String` ou `Int`. É possível controlar isso nos modelos de mapeamento do solucionador para atribuição automática.

Há semelhanças entre tipos de objetos especiais de `Query` e tipos de objetos “comuns”, como no exemplo acima, pois ambos usam a palavra-chave `type` e são considerados objetos. No entanto, para os tipos de objetos especiais (`Query`, `Mutation` e `Subscription`), o comportamento deles é muito diferente porque eles são expostos como pontos de entrada da sua API. Eles também priorizam a modelagem de operações em vez dos dados. Para obter mais informações, consulte [The query and mutation types](#).

No tópico dos tipos de objetos especiais, a próxima etapa pode ser adicionar um ou mais deles para realizar operações nos dados modelados. Em um cenário real, todo esquema do GraphQL deve ter pelo menos um tipo de consulta-raiz para solicitar dados. Você pode pensar na consulta como um dos pontos de entrada (ou endpoints) do seu servidor do GraphQL. Vamos adicionar uma consulta como exemplo.

Console

- Para criar uma consulta, basta adicioná-la ao arquivo do esquema como qualquer outro tipo. Uma consulta exigiria um tipo de `Query` e uma entrada na raiz como esta:

```
schema {
```

```
query: Name_of_Query
}

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

Observe que *Name_of_Query* em um ambiente de produção simplesmente será chamado Query na maioria dos casos. Recomendamos manter esse valor baixo. Dentro do tipo de consulta, você pode adicionar campos. Cada campo executará uma operação na solicitação. Como resultado, a maioria desses campos, se não todos, serão anexados a um solucionador. No entanto, não vamos abordar esse assunto nesta seção. Em relação ao formato da operação do campo, ela pode ser como esta:

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

Veja um exemplo abaixo:

```
schema {
  query: Query
}

type Query {
  getObj: [Obj_Type_1]
}

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

Note

Nesta etapa, adicionamos um tipo de Query e o definimos na raiz do nosso schema. Nosso tipo de Query definiu um campo *getObj* que retorna uma lista de objetos *Obj_Type_1*. Esse *Obj_Type_1* é o objeto da etapa anterior. No código de produção, suas operações de campo normalmente trabalharão com dados moldados por objetos como *Obj_Type_1*. Além disso, campos como *getObj* normalmente

terão um solucionados para executar a lógica de negócios. Isso será abordado em outra seção.

Como observação adicional, adiciona AWS AppSync automaticamente uma raiz do esquema durante as exportações, então, tecnicamente, você não precisa adicioná-la diretamente ao esquema. Nosso serviço processará esquemas duplicados de maneira automática. Essa é uma prática recomendada.

CLI

Note

Recomendamos que seja feita a leitura da versão do console primeiro.

1. Crie um schema raiz com uma definição da query executando o comando [create-type](#).

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. A `definition`, ou o conteúdo do seu tipo. No exemplo do console, tínhamos:

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

3. O `format` da sua entrada. Neste exemplo, usamos o SDL.

Veja um exemplo de comando:

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

Uma saída será retornada na CLI. Veja um exemplo abaixo:

```
{  
  "type": {  
    "definition": "schema {query: Query}",  
    "name": "schema",
```

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

Note

Se você tiver cometido um erro no comando `create-type`, poderá atualizar a raiz do esquema (ou qualquer tipo no esquema) executando o comando [update-type](#). Neste exemplo, alteraremos temporariamente a raiz do esquema para conter uma definição de `subscription`.

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. O `type-name` do seu tipo. No exemplo do console, tínhamos `schema`.
3. A `definition`, ou o conteúdo do seu tipo. No exemplo do console, tínhamos:

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

O esquema após a adição de um `subscription` será semelhante a este:

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

4. O `format` da sua entrada. Neste exemplo, usamos o `SDL`.

Veja um exemplo de comando:

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

Uma saída será retornada na CLI. Veja um exemplo abaixo:

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

Adicionar arquivos pré-formatados ainda funcionará neste exemplo.

2. Crie uma Query executando o comando [create-type](#).

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. A `definition`, ou o conteúdo do seu tipo. No exemplo do console, tínhamos:

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

3. O `format` da sua entrada. Neste exemplo, usamos o SDL.

Veja um exemplo de comando:

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

Uma saída será retornada na CLI. Veja um exemplo abaixo:

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



```
}
```

Note

Nesta etapa, adicionamos um tipo de Query e o definimos na raiz do seu schema. Nosso tipo de Query definiu um campo getObject que retornou uma lista de objetos Obj_Type_1.

Na query: Query do código-raiz do schema, a parte query: indica que uma consulta foi definida em seu esquema, enquanto a parte Query representa o nome real do objeto especial.

CDK

Tip

[Antes de usar o CDK, recomendamos revisar a documentação oficial do CDK junto com AWS AppSync a referência do CDK.](#)

As etapas listadas abaixo mostram apenas um exemplo geral do trecho usado para adicionar um recurso específico. Isso não é uma solução funcional para seu código de produção. Também presumimos que você já tenha uma aplicação em funcionamento.

Você precisará adicionar sua consulta e a raiz do esquema ao arquivo do .graphql. Nosso exemplo se parece com o exemplo abaixo, mas considere substituí-lo pelo código do seu esquema real:

```
schema {  
  query: Query  
}  
  
type Query {  
  getObject: [Obj_Type_1]  
}  
  
type Obj_Type_1 {  
  id: ID!  
  title: String  
  date: AWSDateTime  
}
```

```
}
```

Você pode adicionar seus tipos diretamente ao esquema, como qualquer outro arquivo.

 Note

A atualização da raiz do esquema é opcional. Nós a adicionamos a esse exemplo como uma prática recomendada.

Para usar as alterações feitas na sua API GraphQL, você precisará reimplantar a aplicação.

Mostramos um exemplo de criação de objetos comuns e objetos especiais (consultas). Também abordamos como eles se interconectam para descrever dados e operações. Pode haver esquemas apenas com a descrição dos dados e uma ou mais consultas. No entanto, gostaríamos de adicionar outra operação para incluir dados na fonte de dados. Adicionaremos outro tipo de objeto especial chamado `Mutation` que modifica os dados.

Console

- Uma mutação será chamada de `Mutation`. Como a `Query`, as operações de campo em `Mutation` descrevem uma operação e serão anexadas a um solucionador. Além disso, precisamos defini-lo na raiz do `schema` porque é um tipo de objeto especial. Veja um exemplo de mutação abaixo:

```
schema {  
  mutation: Name_of_Mutation  
}  
  
type Name_of_Mutation {  
  # Add field operation here  
}
```

Uma mutação típica será listada na raiz como uma consulta. A mutação é definida usando a `type` palavra-chave junto com o nome. `Name_of_Mutation` normalmente serão chamados `Mutation`, por isso recomendamos que continue assim. Cada campo também executará uma operação. Em relação ao formato da operação do campo, ela pode ser como esta:

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

Veja um exemplo abaixo:

```
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

Nesta etapa, adicionamos um tipo de `Mutation` com um campo `addObj`. Vamos resumir o que esse campo faz:

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

O `addObj` usa o objeto `Obj_Type_1` para realizar uma operação. Isso é evidente ao observar os campos, mas a sintaxe prova isso no tipo de retorno : `Obj_Type_1`. Em `addObj`, os campos `id`, `title` e `date` do objeto `Obj_Type_1` como parâmetros. Como podemos notar, ela se parece muito com uma declaração de método. No entanto, ainda não descrevemos o comportamento do nosso método. Conforme mencionado anteriormente, o esquema só existe para definir quais serão os dados e as operações e não a forma como operam. A implementação da lógica de negócios real será abordada quando criarmos nossos primeiros solucionadores.

Depois de concluir o esquema, há uma opção para exportá-lo como um arquivo `schema.graphql`. No Editor de esquemas, você pode escolher Exportar esquema para fazer download do arquivo em um formato compatível.

Como observação adicional, adiciona AWS AppSync automaticamente uma raiz do esquema durante as exportações, então, tecnicamente, você não precisa adicioná-la diretamente ao esquema. Nosso serviço processará esquemas duplicados de maneira automática. Essa é uma prática recomendada.

CLI

Note

Recomendamos que seja feita a leitura da versão do console primeiro.

1. Atualize seu esquema-raiz executando o comando [update-type](#).

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. O `type-name` do seu tipo. No exemplo do console, tínhamos `schema`.
3. A `definition`, ou o conteúdo do seu tipo. No exemplo do console, tínhamos:

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

4. O `format` da sua entrada. Neste exemplo, usamos o SDL.

Veja um exemplo de comando:

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

Uma saída será retornada na CLI. Veja um exemplo abaixo:

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

2. Crie uma Mutation executando o comando [create-type](#).

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. A `definition`, ou o conteúdo do seu tipo. No exemplo do console, tínhamos

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

3. O `format` da sua entrada. Neste exemplo, usamos o `SDL`.

Veja um exemplo de comando:

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

Uma saída será retornada na CLI. Veja um exemplo abaixo:

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

CDK

 Tip

Antes de usar o CDK, recomendamos revisar a documentação oficial do CDK junto com AWS AppSync a referência do CDK.

As etapas listadas abaixo mostram apenas um exemplo geral do trecho usado para adicionar um recurso específico. Isso não é uma solução funcional para seu código de produção. Também presumimos que você já tenha uma aplicação em funcionamento.

Você precisará adicionar sua consulta e a raiz do esquema ao arquivo do `.graphql`. Nosso exemplo se parece com o exemplo abaixo, mas considere substituí-lo pelo código do seu esquema real:

```
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

A atualização da raiz do esquema é opcional. Nós a adicionamos a esse exemplo como uma prática recomendada.

Para usar as alterações feitas na sua API GraphQL, você precisará reimplantar a aplicação.

Considerações opcionais: usar enumerados como status

Neste ponto, você já sabe como criar um esquema básico. No entanto, há muitos elementos que você pode adicionar para aumentar a funcionalidade do esquema. Uma coisa comum encontrada nas aplicações é o uso de enumerados como status. Você pode usar um enumerado para aplicar um valor específico de um conjunto de valores a ser escolhido quando chamado. Isso é bom para elementos que não mudarão drasticamente por longos períodos. Hipoteticamente falando, poderíamos adicionar um enumerado que retorna o código de status ou string na resposta.

Como exemplo, vamos supor que estamos criando uma aplicação de mídia social que armazena os dados de postagem de um usuário no back-end. Nosso esquema contém um tipo de Post que representa os dados de uma postagem individual:

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

Nossa Post conterá um único id, um title, uma date de postagem e um enumerado chamado de PostStatus, que representa o estado da postagem conforme ela é processada pela aplicação. Para nossas operações, teremos uma consulta que retornará todos os dados da postagem:

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

Também teremos uma mutação que adiciona postagens à fonte de dados:

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

Analisando nosso esquema, o enumerado `PostStatus` pode ter vários status. É possível termos os três estados básicos chamados de `success` (postagem processada com sucesso), `pending` (postagem sendo processada) e `error` (postagem que não foi processada). Para adicionar o enumerado, podemos fazer o seguinte:

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

O esquema completo pode ser semelhante a este:

```
schema {  
  query: Query  
  mutation: Mutation  
}  
  
type Post {  
  id: ID!  
  title: String  
  date: AWSDateTime  
  poststatus: PostStatus  
}  
  
type Mutation {  
  addPost(id: ID!, title: String, date: AWSDateTime, poststatus: PostStatus): Post  
}  
  
type Query {  
  getPosts: [Post]  
}  
  
enum PostStatus {  
  success  
  pending  
  error  
}
```

Se um usuário adicionar uma `Post` à aplicação, a operação `addPost` será chamada para processar esses dados. À medida que o solucionador anexado à `addPost` processa os dados, ele vai atualizar o `poststatus` continuamente com o status da operação. Quando consultado, a `Post` vai mostrar o

status final dos dados. Aqui só descrevemos como queremos que os dados funcionem no esquema. Fazemos muitas suposições sobre a implementação de nossos solucionadores, que implementarão a lógica comercial real para lidar com os dados e atender à solicitação.

Considerações opcionais: assinaturas

As assinaturas em AWS AppSync são invocadas como resposta a uma mutação. Isso pode ser configurado com um tipo `Subscription` e diretiva `@aws_subscribe()` no esquema para indicar quais mutações invocam uma ou mais assinaturas. Consulte [Real-time data](#) para obter mais informações sobre como configurar assinaturas.

Considerações opcionais: relações e paginação

Suponha que você tenha um milhão de `Posts` armazenadas em uma tabela do DynamoDB e queira retornar alguns desses dados. No entanto, o exemplo de consulta fornecido acima retorna todas as postagens. Não é fácil buscar tudo isso sempre que fizer uma solicitação. O melhor é [paginar](#) todas elas. Faça as seguintes alterações ao esquema:

- No campo `getPosts`, adicione dois argumentos de entrada: `nextToken` (iterador) e `limit` (limite de iteração).
- Adicione um novo tipo de `PostIterator` contendo `Posts` (isso recupera a lista de objetos `Post`) e campos `nextToken` (iterador).
- Altere `getPosts` para que retorne `PostIterator`, não uma lista de objetos `Post`.

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

```
type Query {  
  getPosts(limit: Int, nextToken: String): PostIterator  
}  
  
enum PostStatus {  
  success  
  pending  
  error  
}  
  
type PostIterator {  
  posts: [Post]  
  nextToken: String  
}
```

O tipo `PostIterator` permite que você retorne parte da lista de objetos de `Post` e um `nextToken` para obter a próxima parte. Em `PostIterator`, há uma lista de itens da `Post` (`[Post]`) que é retornada com um token de paginação (`nextToken`). Em AWS AppSync, isso seria conectado ao Amazon DynamoDB por meio de um resolvidor e gerado automaticamente como um token criptografado. Isso converte o valor do argumento `limit` no parâmetro `maxResults` e o argumento `nextToken` no parâmetro `exclusiveStartKey`. Para ver exemplos e exemplos de modelos integrados no AWS AppSync console, consulte [Referência do Resolver \(JavaScript\)](#).

Anexando uma fonte de dados em AWS AppSync

As fontes de dados são recursos em sua AWS conta com os quais o GraphQL APIs pode interagir. AWS AppSync oferece suporte a várias fontes de dados AWS Lambda, como Amazon DynamoDB, bancos de dados relacionais (Amazon Aurora Serverless), Amazon Service e endpoints HTTP. OpenSearch Uma AWS AppSync API pode ser configurada para interagir com várias fontes de dados, permitindo que você agregue dados em um único local. AWS AppSync pode usar AWS recursos existentes da sua conta ou provisionar tabelas do DynamoDB em seu nome a partir de uma definição de esquema.

A seção a seguir mostrará como anexar uma fonte de dados à sua API GraphQL.

Tipos de fontes de dados

Agora que você criou um esquema no AWS AppSync console, você pode anexar uma fonte de dados a ele. Quando você começa a criar uma API, há a opção de provisionar uma tabela do Amazon DynamoDB durante a criação do esquema predefinido. No entanto, não abordaremos essa opção nesta seção. Você pode ver um exemplo disso na seção [Iniciar um esquema](#).

Em vez disso, analisaremos todos os AWS AppSync suportes de fontes de dados. Há muitos fatores que influenciam a escolha da solução certa para seu aplicativo. As seções abaixo fornecerão contexto adicional para cada fonte de dados. Para obter informações gerais sobre fontes de dados, consulte [Data sources](#).

Amazon DynamoDB

O Amazon DynamoDB é uma das principais soluções AWS de armazenamento para aplicativos escaláveis. O componente principal do DynamoDB é a tabela, que é simplesmente um conjunto de dados. Normalmente, você cria tabelas com base em entidades como `Book` ou `Author`. As informações de entrada da tabela são armazenadas como itens, que são grupos de campos exclusivos para cada entrada. Um item completo representa um row/record no banco de dados. Por exemplo, um item para uma entrada da `Book` pode incluir `title` e `author` com seus valores. Os campos individuais, como `title` e `author` são chamados de atributos, que são semelhantes aos valores das colunas em bancos de dados relacionais.

Como você pode imaginar, as tabelas serão usadas para armazenar dados do seu aplicativo. AWS AppSync permite que você conecte suas tabelas do DynamoDB à sua API do GraphQL para manipular dados. Veja este [caso de uso](#) no blog Front-end web and mobile. Esta aplicação permite que os usuários se inscrevam em um aplicativo de mídia social. Os usuários podem participar de grupos e fazer upload de postagens que são transmitidas para outros usuários inscritos no grupo. A aplicação armazena informações de usuários, publicações e grupos de usuários no DynamoDB. A API GraphQL (gerenciada por AWS AppSync) faz interface com a tabela do DynamoDB. Quando um usuário faz uma alteração no sistema que será refletida no front-end, a API GraphQL recupera essas alterações e as transmite para outros usuários em tempo real.

AWS Lambda

O Lambda é um serviço orientado por eventos que cria automaticamente os recursos necessários para executar códigos como resposta a um evento. O Lambda usa funções, que são declarações de grupo contendo o código, as dependências e as configurações para executar um recurso. As funções são executadas automaticamente quando detectam um gatilho, um grupo de atividades que invocam sua função. Um gatilho pode ser qualquer coisa como um aplicativo fazendo uma chamada de API, um AWS serviço em sua conta gerando um recurso etc. Quando acionadas, as funções processarão eventos, que são documentos JSON com os dados a serem modificados.

O Lambda é bom para executar códigos sem precisar provisionar os recursos para executá-lo. Veja este [caso de uso](#) no blog Front-end web and mobile. Esse caso de uso é um pouco semelhante ao apresentado na seção do DynamoDB. Nessa aplicação, a API GraphQL é responsável por

definir as operações para coisas como adicionar postagens (mutações) e buscar esses dados (consultas). Para implementar a funcionalidade de suas operações (por exemplo, `getPost ( id: String ! ) : Post` e `getPostsByAuthor ( author: String ! ) : [ Post ]`), eles usam funções do Lambda para processar solicitações de entrada. Na Opção 2: AWS AppSync com o resolvidor Lambda, eles usam o AWS AppSync serviço para manter seu esquema e vincular uma fonte de dados Lambda a uma das operações. Quando a operação é chamada, o Lambda interage com o Amazon RDS Proxy para executar a lógica de negócios no banco de dados.

Amazon RDS

O Amazon RDS permite que você crie e configure rapidamente bancos de dados relacionais. No Amazon RDS, você criará uma instância de banco de dados genérica que servirá como ambiente de banco de dados isolado na nuvem. Nesta instância, você usará um mecanismo de banco de dados, que é o software RDBMS real (PostgreSQL, MySQL etc.). O serviço elimina grande parte do trabalho de back-end ao fornecer escalabilidade usando AWS infraestrutura, serviços de segurança, como patches e criptografia, e reduz os custos administrativos das implantações.

Veja o mesmo [caso de uso](#) da seção do Lambda. Na Opção 3: AWS AppSync com o resolvidor do Amazon RDS, outra opção apresentada é vincular a API do GraphQL diretamente ao AWS AppSync Amazon RDS. Com uma [API de dados](#), eles associam o banco de dados à API GraphQL. Um resolvidor é anexado a um campo (geralmente uma consulta, mutação ou assinatura) e implementa as instruções SQL necessárias para acessar o banco de dados. Quando uma solicitação de chamada do campo é feita pelo cliente, o resolvidor executa as instruções e retorna a resposta.

Amazon EventBridge

Em EventBridge, você criará barramentos de eventos, que são pipelines que recebem eventos de serviços ou aplicativos anexados (a origem do evento) e os processam com base em um conjunto de regras. Um evento é uma mudança de estado em um ambiente de execução, enquanto uma regra é um conjunto de filtros para eventos. Uma regra segue um padrão de evento ou metadados da mudança de estado de um evento (id, região, número da conta, ARN(s) etc.). Quando um evento corresponde ao padrão do evento, EventBridge enviará o evento pelo pipeline até o serviço de destino (destino) e acionará a ação especificada na regra.

EventBridge é bom para rotear operações de mudança de estado para algum outro serviço. Veja este [caso de uso](#) no blog Front-end web and mobile. O exemplo mostra uma solução de comércio eletrônico que tem várias equipes mantendo serviços diferentes. Um desses serviços fornece atualizações de pedidos ao cliente em cada etapa da entrega (pedido feito, em andamento, enviado, entregue etc.) no front-end. No entanto, a equipe de front-end que gerencia esse serviço não tem

acesso direto aos dados do sistema de pedidos, pois eles são mantidos por uma equipe de back-end separada. O sistema de pedidos da equipe de back-end também é descrito como uma caixa preta, por isso, é difícil coletar informações sobre a forma como eles estruturam os dados. No entanto, a equipe de back-end configurou um sistema que publicou dados de pedidos por meio de um barramento de eventos gerenciado por EventBridge. Para acessar os dados provenientes do barramento de eventos e encaminhá-los para o front-end, a equipe de front-end criou um novo alvo apontando para a API GraphQL instalada. AWS AppSync Eles também criaram uma regra para enviar apenas dados relevantes para a atualização do pedido. Quando uma atualização é feita, os dados do barramento de eventos são enviados para a API GraphQL. O esquema na API processa os dados e os transfere para o front-end.

Nenhuma fonte de dados

Se você não planeja usar uma fonte de dados, pode defini-la como `none`. Uma fonte de dados `none`, embora ainda seja explicitamente categorizada como fonte de dados, não é um meio de armazenamento. Normalmente, um resolvidor invoca uma ou mais fontes de dados em algum momento para processar a solicitação. No entanto, há situações em que talvez você não precise manipular uma fonte de dados. Definir a fonte de dados como `none` vai executar a solicitação, ignorar a etapa de invocação de dados e executar a resposta.

Veja o mesmo [caso de uso](#) da EventBridge seção. No esquema, a mutação processa a atualização de status e a envia aos assinantes. Semelhante ao funcionamento dos resolvidores, geralmente há pelo menos uma invocação de fonte de dados. No entanto, os dados nesse cenário já foram enviados automaticamente pelo barramento de eventos. Isso significa que não é necessário passar pela mutação para realizar uma invocação da fonte de dados; o status do pedido pode ser tratado localmente. A mutação é definida como `none`, o que funciona como um valor de passagem sem invocação da fonte de dados. O esquema é preenchido com os dados, que são enviados aos assinantes.

OpenSearch

O Amazon OpenSearch Service é um conjunto de ferramentas para implementar pesquisa de texto completo, visualização de dados e registro. Você pode usar esse serviço para consultar os dados estruturados que enviou.

Nesse serviço, você criará instâncias de OpenSearch. Eles são chamados de nós. Em um nó, você adicionará pelo menos um índice. Conceitualmente, os índices são um pouco como tabelas em bancos de dados relacionais. (No entanto, OpenSearch não é compatível com ACID, então não deve ser usado dessa forma). Você preencherá seu índice com os dados que você envia para o

OpenSearch serviço. Quando seus dados forem carregados, eles serão indexados em um ou mais fragmentos existentes no índice. Um fragmento é como uma partição do seu índice que contém alguns dos seus dados e pode ser consultado separadamente de outros fragmentos. Depois de carregados, seus dados serão estruturados como arquivos JSON chamados documentos. Em seguida, você pode consultar o nó em busca de dados no documento.

Endpoints de HTTP

Você pode usar endpoints HTTP como fontes de dados. AWS AppSync pode enviar solicitações aos endpoints com as informações relevantes, como parâmetros e carga útil. A resposta HTTP será exposta ao resolvidor, que retornará a resposta final após concluir suas operações.

Adicionar uma fonte de dados

Se você criou uma fonte de dados, pode vinculá-la ao AWS AppSync serviço e, mais especificamente, à API.

Console

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
 - a. Escolha sua API no Painel.
 - b. Na barra lateral, escolha Fontes de dados.
2. Escolha Criar fonte de dados.
 - a. Dê um nome à sua fonte de dados. Você também pode incluir uma descrição, mas isso é opcional.
 - b. Selecione o tipo de fonte de dados.
 - c. Para o DynamoDB, você precisará escolher sua Região e, em seguida, a tabela na Região. Você pode definir regras de interação com sua tabela criando um novo perfil genérico ou importando um perfil existente. Você pode habilitar o [versionamento](#), que pode criar automaticamente versões de dados para cada solicitação quando vários clientes estão tentando atualizar os dados ao mesmo tempo. O versionamento é usado para manter diversas variantes de dados para fins de detecção e resolução de conflitos. Você também pode ativar a geração automática de esquemas, que usa sua fonte de dados e gera parte do CRUD, List e Query das operações necessárias para acessá-la em seu esquema.

Para isso OpenSearch, você terá que escolher sua região e, em seguida, o domínio (cluster) na região. Você pode definir regras de interação com seu domínio criando uma nova função genérica ou importando uma função existente.

Para o Lambda, você terá que escolher sua Região e, em seguida, o ARN da função do Lambda na Região. Você pode definir regras de interação com sua função do Lambda criando um novo perfil da tabela genérica ou importando um perfil existente.

Para HTTP, você precisará inserir seu endpoint HTTP.

Pois EventBridge, você terá que escolher sua região e, em seguida, o ônibus do evento na região. Você pode definir regras de interação com seu barramento de eventos, criando uma nova função genérica ou importando uma função existente.

Para o RDS, você precisará escolher sua Região, depois o armazenamento secreto (nome de usuário e senha), nome do banco de dados e esquema.

Para nenhum deles, você adicionará uma fonte de dados sem uma fonte de dados real. Isso serve para lidar com resolvedores localmente, e não por meio de uma fonte de dados real.

 Note

Se você estiver importando funções existentes, elas precisarão de uma política de confiança. Para obter mais informações sobre a política de confiança, consulte [política de confiança do IAM](#).

3. Escolha Criar.

 Note

Como alternativa, se você estiver criando uma fonte de dados do DynamoDB, acesse a página Esquema no console, escolha Criar recursos na parte superior da página e preencha um modelo predefinido para converter em uma tabela. Nessa opção, você vai preencher ou importar o tipo de base, configurar os dados básicos da tabela, incluindo a chave de partição, além de analisar as alterações do esquema.

CLI

- Crie um objeto da fonte de dados executando o comando [create-data-source](#).

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. O nome da tabela.
3. O `type` da fonte de dados. Dependendo do tipo de fonte de dados escolhido, talvez seja necessário inserir um `-config` e uma tag `service-role-arn`.

Veja um exemplo de comando:

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

CDK

Tip

[Antes de usar o CDK, recomendamos revisar a documentação oficial do CDK junto com AWS AppSync a referência do CDK.](#)

As etapas listadas abaixo mostram apenas um exemplo geral do trecho usado para adicionar um recurso específico. Isso não é uma solução funcional para seu código de produção. Também presumimos que você já tenha uma aplicação em funcionamento.

Para adicionar sua fonte de dados específica, você precisará incluir a estrutura ao seu arquivo de pilha. Uma lista dos tipos de fontes de dados pode ser encontrada aqui:

- [DynamoDbDataSource](#)
- [EventBridgeDataSource](#)
- [HttpDataSource](#)
- [LambdaDataSource](#)
- [NoneDataSource](#)

- [OpenSearchDataSource](#)
- [RdsDataSource](#)

1. Em geral, talvez seja necessário adicionar a diretiva de importação ao serviço que você está usando. Por exemplo, estas são as possíveis formas:

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

Por exemplo, veja como você pode importar os serviços AWS AppSync e do DynamoDB:

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

2. Alguns serviços, como o RDS, exigem alguma configuração adicional no arquivo de pilha antes de criar a fonte de dados (por exemplo, criação de VPC, funções e credenciais de acesso). Consulte os exemplos nas páginas relevantes do CDK para obter mais informações.
3. Para a maioria das fontes de dados, especialmente AWS serviços, você criará uma nova instância da fonte de dados em seu arquivo de pilha. Normalmente, isso será exibido da seguinte forma:

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

Por exemplo, aqui está um exemplo de tabela do Amazon DynamoDB:

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

Note

A maioria das fontes de dados terá pelo menos um suporte obrigatório (será indicado sem um símbolo ?). Consulte a documentação do CDK para ver quais propriedades são necessárias.

4. Em seguida, você precisa vincular a fonte de dados à API GraphQL. O método recomendado é adicioná-la ao criar uma função para o resolvidor de pipeline. Por exemplo, o trecho abaixo é uma função que verifica todos os elementos em uma tabela do DynamoDB:

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

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

Nas propriedades de `dataSource`, você pode chamar a API GraphQL (`add_api`) e usar um de seus métodos integrados (`addDynamoDbDataSource`) para fazer a associação entre a tabela e a API GraphQL. Os argumentos são o nome desse link que existirá no AWS AppSync console (`data_source_name_in_console` neste exemplo) e o método da tabela (`add_ddb_table`). Mais informações sobre esse tópico serão reveladas na próxima seção, quando você começar a criar resolvidores.

Existem métodos alternativos para vincular uma fonte de dados. Tecnicamente, você pode adicionar itens da `api` à lista de propriedades na função de tabela. Por exemplo, aqui está o trecho da etapa 3, mas com propriedades da `api` contendo uma API GraphQL:

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

```
...
});

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

...

  api: add_api
});
```

Como alternativa, você pode chamar a estrutura do `GraphQLApi` separadamente:

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

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

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

Recomendamos criar a associação somente nas propriedades da função. Caso contrário, você precisará vincular sua função de resolução à fonte de dados manualmente no AWS AppSync console (se quiser continuar usando o valor do `consoledata_source_name_in_console`) ou criar uma associação separada na função com outro nome, como `data_source_name_in_console_2`. Isso se deve às limitações na forma como as propriedades processam as informações.

 Note

Você precisará reimplantar a aplicação para conferir as alterações.

Política de confiança do IAM

Se você estiver usando uma função do IAM existente para sua fonte de dados, precisará conceder a essa função as permissões apropriadas para realizar operações em seu AWS recurso, como `PutItem` em uma tabela do Amazon DynamoDB. Você também precisa modificar a política de

confiança dessa função para permitir seu uso AWS AppSync para acesso a recursos, conforme mostrado no exemplo de política a seguir:

JSON

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

Você também pode adicionar condições à sua política de confiança para limitar o acesso à fonte de dados, conforme desejado. Atualmente, as chaves `SourceArn` e `SourceAccount` podem ser usadas nessas condições. Por exemplo, a política a seguir limita o acesso à sua fonte de dados na conta 123456789012:

JSON

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

```
]
}
```

Como alternativa, é possível restringir o acesso de uma API específica a uma fonte de dados, por exemplo, abcdefghijklmnopq, usando a seguinte política:

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"
        }
      }
    }
  ]
}
```

Você pode limitar o acesso a todos AWS AppSync APIs de uma região específica, por exemplo us-east-1, usando a seguinte política:

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/*"
      }
    }
  }
}
```

Na próxima seção ([Configurar os resolvedores](#)), vamos adicionar nossa lógica de negócios do resolvedor e anexá-la aos campos em nosso esquema para processar os dados em nossa fonte de dados.

Para obter mais informações, consulte [Modificando um perfil](#) no Guia do usuário do IAM.

Para obter mais informações sobre o acesso entre contas de AWS Lambda resolvedores para AWS AppSync, consulte [Criação de resolvedores entre contas AWS Lambda](#) para AWS AppSync.

Configurando resolvedores em AWS AppSync

Nas seções anteriores, você aprendeu a criar o esquema e a fonte de dados do GraphQL e, em seguida, vinculá-los no serviço. AWS AppSync No esquema, você pode ter estabelecido um ou mais campos (operações) na consulta e na mutação. Embora o esquema descrevesse os tipos de dado que as operações solicitariam da fonte de dados, ele nunca implementou o modo como essas operações se comportariam em relação aos dados.

O comportamento de uma operação é sempre implementado no resolvedor, que será vinculado ao campo que executa a operação. Para obter mais informações sobre como os resolvedores funcionam em geral, consulte a página [Resolvedores](#).

Em AWS AppSync, seu resolvedor está vinculado a um tempo de execução, que é o ambiente no qual seu resolvedor é executado. Os runtimes determinam a linguagem na qual o resolvedor será gravado. Atualmente, há dois tempos de execução suportados: APPSYNC_JS (JavaScript) e Apache Velocity Template Language (VTL).

Ao implementar resolvedores, eles seguem uma estrutura geral:

- Etapa Anterior: quando uma solicitação é feita pelo cliente, os resolvedores dos campos do esquema que estão sendo usados (normalmente consultas, mutações e assinaturas) recebem

os dados da solicitação. O resolvidor começará a processar os dados da solicitação com um manipulador de etapas anteriores, o que permite que algumas operações de pré-processamento sejam executadas antes que os dados passem pelo resolvidor.

- **Função(ões):** Após a execução da etapa anterior, a solicitação é passada para a lista de funções. A primeira função na lista será executada na fonte de dados. Uma função é um subconjunto do código do resolvidor contendo seu próprio manipulador de solicitações e respostas. Um manipulador de solicitações pegará os dados da solicitação e executará operações na fonte de dados. O manipulador de respostas processará a resposta da fonte de dados antes de passá-la de volta para a lista. Se houver mais de uma função, os dados da solicitação serão enviados para a próxima função a ser executada na lista. As funções na lista serão executadas na ordem definida pelo desenvolvedor. Depois que todas as funções forem executadas, o resultado final será passado para a etapa posterior.
- **Etapa Posterior:** a etapa Posterior é uma função do manipulador que permite realizar algumas operações finais na resposta da função final antes de passá-la para a resposta do GraphQL.

Esse fluxo é um exemplo de um resolvidor de pipeline. Os resolvidores de pipeline são compatíveis em ambos os runtimes. No entanto, essa é uma explicação simplificada do que os resolvidores de pipeline podem fazer. Além disso, estamos descrevendo apenas uma configuração possível do resolvidor. Para obter mais informações sobre as configurações de resolvidor suportadas, consulte a visão geral dos [JavaScript resolvidores para APPSYNC_JS](#) ou a [visão geral do](#) modelo de mapeamento do [Resolver](#) para VTL.

Como você pode ver, os resolvidores são modulares. Para que os componentes do resolvidor funcionem corretamente, eles devem ser capazes de examinar o estado da execução por meio de outros componentes. Na seção [Resolvidores](#), você sabe que cada componente no resolvidor pode receber informações essenciais sobre o estado da execução como um conjunto de argumentos (`args`, `context` etc.). Em AWS AppSync, isso é tratado estritamente pelo `context`. Trata-se de um contêiner para as informações sobre o campo que está sendo resolvido. Isso pode incluir tudo, desde argumentos passados, resultados, dados de autorização, dados de cabeçalho etc. Para obter mais informações sobre o contexto, consulte [Referência do objeto de contexto do resolvidor](#) para APPSYNC_JS ou a [Referência de contexto do modelo de mapeamento do resolvidor](#) para VTL.

O contexto não é a única ferramenta que você pode usar para implementar seu resolvidor. AWS AppSync suporta uma ampla variedade de utilitários para geração de valor, tratamento de erros, análise, conversão, etc. Você pode ver uma lista de utilitários [aqui](#) para APPSYNC_JS ou [aqui](#) para VTL.

Nas seções a seguir, você vai aprender a configurar resolvedores na API do GraphQL.

Tópicos

- [Criação de consultas básicas \(\) JavaScript](#)
- [Criação de consultas básicas \(VTL\)](#)

Criação de consultas básicas () JavaScript

Os resolvedores do GraphQL conectam os campos em um esquema de tipo a uma fonte de dados. Os resolvedores são o mecanismo pelo qual as solicitações são atendidas.

Resolvedores em AWS AppSync uso JavaScript para converter uma expressão GraphQL em um formato que a fonte de dados possa usar. Como alternativa, os modelos de mapeamento podem ser escritos em [Apache VTL \(Velocity Template Language\)](#) para converter uma expressão de GraphQL em um formato que a fonte de dados possa usar.

Esta seção descreve como configurar resolvedores usando o JavaScript. A seção [Tutoriais do Resolver \(JavaScript\)](#) fornece tutoriais detalhados sobre como implementar resolvedores usando JavaScript. A seção [Referência do Resolvedor \(JavaScript\)](#) fornece uma explicação das operações do utilitário que podem ser usadas com JavaScript resolvedores.

Recomendamos seguir este guia antes de tentar usar qualquer um dos tutoriais mencionados acima.

Nesta seção, mostraremos como criar e configurar resolvedores de consultas e mutações.

Note

Este guia pressupõe que você tenha criado seu esquema e tenha pelo menos uma consulta ou mutação. Se você estiver procurando por assinaturas (dados em tempo real), consulte [este guia](#).

Nesta seção, abordaremos algumas etapas gerais para configurar resolvedores e mostraremos um exemplo com o esquema abaixo:

```
// 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]
}
```

Criação de resolvedores de consultas básicos

Esta seção mostrará como criar um resolvedor de consultas básico.

Console

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, escolha Esquema.
2. Insira os detalhes do esquema e da fonte de dados. Consulte as seções [Criar seu esquema](#) e [Anexar uma fonte de dados](#) para obter mais informações.
3. Ao lado do editor de esquemas, há uma janela chamada Resolvedores. Essa caixa contém uma lista dos tipos e campos conforme definido na janela Esquema. É possível anexar resolvedores aos campos. Você provavelmente estará anexando resolvedores às suas operações de campo. Nesta seção, veremos configurações de consultas simples. Em Tipo de consulta, escolha Anexar ao lado do campo da sua consulta.
4. Na página Anexar resolvedor, em Tipo de resolvedor, você pode escolher entre resolvedores de pipeline ou de unidade. Para obter mais informações sobre esses tipos de regra, consulte [Resolvedores](#). Este guia fará uso de pipeline resolvers.

Tip

Ao criar resolvedores de pipeline, suas fontes de dados serão anexadas às funções do pipeline. As funções são geradas depois que você cria o próprio resolvedor de pipeline, e é por isso que não há opção de configurá-lo nesta página. Se você estiver usando um resolvedor de unidades, a fonte de dados estará vinculada diretamente ao resolvedor, portanto, você o definiria nesta página.

Para o tempo de execução do Resolver, escolha APPSYNC_JS ativar o JavaScript tempo de execução.

5. Você pode ativar o [armazenamento em cache](#) dessa API. Recomendamos desativar esse atributo por enquanto. Selecione Criar.
6. Na página Editar resolvedor, há um editor de código chamado Código do resolvedor que permite implementar a lógica para o manipulador e a resposta do resolvedor (etapas anterior e posterior). Para obter mais informações, consulte a [visão geral dos JavaScript resolvedores](#).

Note

Em nosso exemplo, vamos deixar a solicitação em branco e a resposta definida para retornar o resultado da última fonte de dados do [contexto](#):

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

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

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

Abaixo dessa seção, há uma tabela chamada Funções. As funções permitem que você implemente códigos que possam ser reutilizados em vários resolvedores. Em vez de

reescrever ou copiar sempre o código, você pode armazenar o código-fonte como uma função a ser adicionada a um resolvedor sempre que precisar.

As funções compõem a maior parte da lista de operações de um pipeline. Ao usar várias funções em um resolvedor, você define a ordem das funções, e elas serão executadas nessa ordem sequencialmente. Elas são executadas depois da função de solicitação e antes do início da função de resposta.

Para adicionar uma nova função, em Funções, escolha Adicionar função e, em seguida, Criar nova função. Como alternativa, pode haver um botão Criar função para escolher.

- a. Escolha uma fonte de dados. Essa será a fonte de dados em que o resolvedor vai atuar.

 Note

Em nosso exemplo, estamos anexando um resolvedor para `getPost`, que recupera um objeto `Post` pelo `id`. Vamos supor que já tenhamos configurado uma tabela do DynamoDB para esse esquema. Sua chave de partição está vazia e definida como `id`.

- b. Insira um `Function name`.
- c. Em Código da função, você precisará implementar o comportamento da função. Isso pode ser confuso, mas cada função terá seu próprio manipulador local de solicitações e respostas. A solicitação é executada e, em seguida, a invocação da fonte de dados é feita para lidar com a solicitação e, em seguida, a resposta da fonte de dados é processada pelo manipulador. O resultado é armazenado no objeto do [contexto](#). Depois disso, a próxima função na lista será executada ou transmitida para o manipulador de resposta da etapa posterior, se for a última.

 Note

Em nosso exemplo, estamos anexando um resolvedor ao `getPost`, que obtém uma lista de objetos `Post` da fonte de dados. Nossa função de solicitação solicitará os dados da nossa tabela, a tabela passará sua resposta para o contexto (`ctx`) e, em seguida, a resposta retornará o resultado no contexto. AWS AppSync A força da está em sua interconexão com outros serviços. AWS Como estamos usando o DynamoDB, temos [um conjunto de operações](#) para simplificar

processos como esses. Também temos alguns exemplos padronizados para outros tipos de fontes de dados.

Nosso código será semelhante a este:

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

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

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

Nesta etapa, adicionamos duas funções:

- **request**: o manipulador de solicitações executa a operação de recuperação na fonte de dados. O argumento contém o objeto de contexto (`ctx`) ou alguns dados que estão disponíveis para todos os resolvedores que executam uma operação específica. Por exemplo, ele pode conter dados de autorização, os nomes dos campos que estão sendo resolvidos etc. A instrução `return` executa uma operação de [Scan](#) (confira exemplos [aqui](#)). Como estamos trabalhando com o DynamoDB, podemos usar algumas das operações desse serviço. A verificação executa uma busca básica de todos os itens em nossa tabela. O resultado dessa operação é armazenado no objeto de contexto como um contêiner de `result` antes de ser transmitido ao manipulador de respostas. A `request` é executada antes da resposta no pipeline.
- **response**: o manipulador de respostas que retorna o resultado da `request`. O argumento é o objeto de contexto atualizado e a instrução de retorno é `ctx.prev.result`. Neste ponto do guia, é possível que você não conheça esse valor. `ctx` refere-se ao objeto de contexto, e `prev` refere-se à operação anterior no pipeline, que era nossa `request`. O `result` contém os resultados do resolvedor à medida que ele se move pelo pipeline. Se você juntar tudo

isso, `ctx.prev.result` está retornando o resultado da última operação realizada, que foi o manipulador da solicitação.

- d. Depois de concluir, escolha Criar.
7. De volta à tela do resolvedor, em Funções, escolha o menu suspenso Adicionar função e inclua sua função na sua lista.
8. Escolha Salvar para atualizar o resolvedor.

CLI

Para adicionar sua função

- Crie uma função para seu resolvedor de pipeline usando o comando [create-function](#).

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. O nome da função no AWS AppSync console.
3. O `data-source-name`, ou o nome da fonte de dados que a função usará. Ele já deve ter sido criado e vinculado à sua API do GraphQL no serviço do AWS AppSync .
4. `Oruntime`, ou ambiente e idioma da função. Para JavaScript, o nome deve ser `APPSYNC_JS`, e o tempo de execução, `1.0.0`.
5. O `code`, ou manipuladores de solicitações e respostas de sua função. Embora você possa digitá-lo manualmente, é muito mais fácil adicioná-lo a um arquivo `.txt` (ou formato similar) e depois transmiti-lo como argumento.

Note

Nosso código de consulta estará em um arquivo transmitido como argumento:

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

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

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

Veja um exemplo de comando:

```
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}
```

Uma saída será retornada na CLI. Veja um exemplo abaixo:

```
{
  "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

Certifique-se de gravar o `functionId` em algum lugar, pois isso será usado para anexar a função ao solucionados.

Criar seu resolvidor

- Crie uma função de pipeline para Query executando o comando [create-resolver](#).

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. O `type-name`, ou o tipo de objeto especial em seu esquema (consulta, mutação, assinatura).
3. O `field-name` ou a operação do campo no tipo de objeto especial a ser anexado ao resolvidor.
4. O `kind`, que especifica um resolvidor de unidade ou pipeline. Defina como `PIPELINE` para ativar as funções do pipeline.
5. A `pipeline-config`, ou as funções a serem anexadas ao resolvidor. É preciso saber os valores de `functionId` de suas funções. A ordem da listagem é importante.
6. O `runtime`, que foi `APPSYNC_JS` (JavaScript). Atualmente o `runtimeVersion` é `1.0.0`.
7. O `code`, que contém os manipuladores das etapas anterior e posterior.

Note

Nosso código de consulta estará em um arquivo transmitido como argumento:

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

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

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

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

Veja um exemplo de comando:

```
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}
```

Uma saída será retornada na CLI. Veja um exemplo abaixo:

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


CDK

Tip

Antes de usar o CDK, recomendamos revisar a documentação oficial do CDK junto com AWS AppSync a referência do CDK.

As etapas listadas abaixo mostram apenas um exemplo geral do trecho usado para adicionar um recurso específico. Isso não é uma solução funcional para seu código de produção. Também presumimos que você já tenha uma aplicação em funcionamento.

Uma aplicação básica precisará do seguinte:

1. Diretivas de importação de serviços
2. Código do esquema
3. Gerador de fontes de dados
4. Código da função
5. Código do resolvedor

Nas seções [Projetar seu esquema](#) e [Anexar uma fonte de dados](#), sabemos que o arquivo de pilha incluirá as diretivas de importação do formulário:

```
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

Nas seções anteriores, declaramos apenas como importar AWS AppSync construções. Em código real, você precisará importar mais serviços apenas para executar a aplicação. Em nosso exemplo, se criássemos um aplicativo CDK muito simples, importaríamos pelo menos o AWS AppSync serviço junto com nossa fonte de dados, que era uma tabela do DynamoDB. Também precisaríamos importar algumas estruturas adicionais para implantar a aplicação:

```
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';
```

Para resumir cada uma delas:

- `import * as cdk from 'aws-cdk-lib';` :: isso permite que você defina a aplicação e as estruturas do CDK, como a pilha. Ela também contém algumas funções utilitárias relevantes para nosso aplicativo, como a manipulação de metadados. Se você já conhece essa diretiva de importação, mas não sabe por que a biblioteca principal do CDK não está sendo usada aqui, consulte a página de [migração](#).
- `import * as appsync from 'aws-cdk-lib/aws-appsync';` :: isso importa o [serviço do AWS AppSync](#).
- `import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';` :: isso importa o [serviço do DynamoDB](#).
- `import { Construct } from 'constructs';` :: precisamos disso para definir a [estrutura-raiz](#).

O tipo de importação depende dos serviços que você está chamando. Recomendamos consultar a documentação do CDK para ver exemplos. O esquema na parte superior da página será um arquivo separado na sua aplicação do CDK como um arquivo `.graphql`. No arquivo de pilha, podemos associá-lo a um novo GraphQL usando este formato:

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

Note

No escopo do `add_api`, vamos adicionar uma nova API GraphQL usando a `new` palavra-chave seguida por `appsync.GraphqlApi(scope: Construct, id: string, props: GraphqlApiProps)`. Nosso escopo é `this`, o ID do CFN é `graphql-example`, e nossas propriedades são `my-first-api` (nome da API no console) e `schema.graphql` (o caminho absoluto para o arquivo do esquema).

Para incluir uma fonte de dados, primeiro você precisa adicionar sua fonte de dados à pilha. Em seguida, associá-la à API GraphQL usando o método específico da fonte. A associação acontecerá quando seu resolvidor funcionar. Enquanto isso, vamos usar um exemplo criando a tabela do DynamoDB usando `dynamodb.Table`:

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

 Note

Se usássemos isso em nosso exemplo, adicionaríamos uma nova tabela do DynamoDB com o ID do CFN e uma chave de partição `posts-table` do `id` (S).

Em seguida, precisamos implementar nosso resolvidor no arquivo de pilha. Veja a seguir um exemplo de uma consulta simples que verifica todos os itens em uma tabela do DynamoDB:

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

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

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

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

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

Note

Primeiro, criamos uma função chamada `add_func`. Essa ordem de criação pode parecer um pouco contraintuitiva, mas você precisa criar as funções em seu resolvidor de pipeline antes de criar o próprio resolvidor. Uma função segue o formato:

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

Nosso escopo era `this`, nosso ID de CFN, `func-get-posts`, e nossas propriedades continham os detalhes reais da função. Dentro das propriedades, incluímos:

- A name da função que estará presente no AWS AppSync console (`get_posts_func_1`).
- A API GraphQL que criamos anteriormente (`add_api`).
- A fonte de dados; esse é o ponto em que vinculamos a fonte de dados ao valor da API GraphQL e a anexamos à função. Pegamos a tabela que criamos (`add_ddb_table`) e a anexamos à API GraphQL (`add_api`) usando um dos métodos `GraphQLApi` ([addDynamoDbDataSource](#)). O valor do ID (`table-for-posts`) é o nome da fonte de dados no console do AWS AppSync. Para obter a lista dos métodos específicos da fonte, consulte as seguintes páginas:
 - [DynamoDbDataSource](#)
 - [EventBridgeDataSource](#)
 - [HttpDataSource](#)
 - [LambdaDataSource](#)
 - [NoneDataSource](#)

- [OpenSearchDataSource](#)
- [RdsDataSource](#)
- O código contém os manipuladores de solicitações e respostas da nossa função, que são uma simples digitalização e retorno.
- O runtime especifica que queremos usar a versão 1.0.0 do runtime do APPSYNC_JS. Observe que atualmente essa é a única versão disponível do APPSYNC_JS.

Em seguida, precisamos anexar a função ao resolvidor de pipeline. Criamos nosso resolvidor usando o formulário:

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

Nosso escopo era `this`, nosso ID de CFN, `pipeline-resolver-get-posts`, e nossas propriedades continham os detalhes reais da função. Dentro das propriedades, incluímos:

- A API GraphQL que criamos anteriormente (`add_api`).
- O nome do tipo de objeto especial; essa é uma operação de consulta, então só adicionamos o valor da `Query`.
- O nome do campo (`getPost`) é aquele no esquema abaixo do tipo de `Query`.
- O código contém seus manipuladores de antes e depois. Nosso exemplo só vão retornar os resultados que estavam no contexto depois que a função tiver executado a operação.
- O runtime especifica que queremos usar a versão 1.0.0 do runtime do APPSYNC_JS. Observe que atualmente essa é a única versão disponível do APPSYNC_JS.
- A configuração do pipeline contém a referência à função que criamos (`add_func`).

Para resumir o que aconteceu neste exemplo, você viu uma AWS AppSync função que implementou um manipulador de solicitações e respostas. A função foi responsável por interagir com sua fonte de dados. O manipulador da solicitação enviou uma `Scan` operação para AWS AppSync, instruindo-a sobre qual operação realizar na sua fonte de dados do DynamoDB. O manipulador de respostas retornou a lista de itens (`ctx.result.items`). A lista de itens foi então mapeada automaticamente para o tipo `Post GraphQL`.

Criar resolvedores básicos de mutação

Esta seção mostrará como criar um resolvedor de mutação básico.

Console

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, escolha Esquema.
2. Na seção Resolvedores e no tipo de mutação, escolha Anexar ao lado do seu campo.

Note

Em nosso exemplo, estamos anexando um resolvedor para `createPost`, que recupera um objeto `Post` para nossa tabela. Vamos supor que estamos usando a mesma tabela do DynamoDB da última seção. Sua chave de partição está vazia e definida como `id`.

3. Na página Anexar resolvedor, em Tipo de resolvedor, escolha `pipeline resolvers`. Como lembrete, você pode encontrar mais informações sobre resolvedores [aqui](#). Para o tempo de execução do Resolver, escolha APPSYNC_JS ativar o JavaScript tempo de execução.
4. Você pode ativar o [armazenamento em cache](#) dessa API. Recomendamos desativar esse atributo por enquanto. Selecione Criar.
5. Escolha Adicionar função e, em seguida, Criar nova função. Como alternativa, pode haver um botão Criar função para escolher.
 - a. Selecione sua fonte de dados. Essa deve ser a fonte cujos dados você manipulará com a mutação.
 - b. Insira um `Function name`.
 - c. Em Código da função, você precisará implementar o comportamento da função. Isso é uma mutação, portanto, o ideal é que a solicitação realize alguma operação de mudança de estado na fonte de dados invocada. O resultado será processado pela função de resposta.

 Note

O `createPost` adiciona ou “insere” um novo Post na tabela com nossos parâmetros como dados. Isso pode ser semelhante a:

```
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;
}
```

Nesta etapa, adicionamos as funções `request` e `response`:

- `request`: o manipulador da solicitação aceita o contexto como argumento. A instrução `return` do manipulador de solicitações executa um comando [PutItem](#), que é uma operação integrada do DynamoDB (confira alguns exemplos [neste link](#) ou [aqui](#)). O comando `PutItem` adiciona um objeto Post à nossa tabela do DynamoDB considerando o valor `key` da partição (gerado automaticamente por `util.autoId()`) e `attributes` a partir da entrada do argumento de contexto (esses são os valores que vamos transmitir em nossa solicitação). A `key` é o `id`, e os `attributes` são os argumentos de campo `date` e `title`. Ambos são pré-formatados por meio do auxiliar [util.dynamodb.toMapValues](#) para trabalhar com a tabela do DynamoDB.

- **response:** a resposta aceita o contexto atualizado e retorna o resultado do manipulador da solicitação.

- d. Depois de concluir, escolha Criar.
6. De volta à tela do resolvedor, em Funções, escolha o menu suspenso Adicionar função e inclua sua função na sua lista.
 7. Escolha Salvar para atualizar o resolvedor.

CLI

Para adicionar sua função

- Crie uma função para seu resolvedor de pipeline usando o comando [create-function](#).

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. O nome da função no AWS AppSync console.
3. O `data-source-name`, ou o nome da fonte de dados que a função usará. Ele já deve ter sido criado e vinculado à sua API do GraphQL no serviço do AWS AppSync .
4. `Oruntime`, ou ambiente e idioma da função. Para JavaScript, o nome deve ser `APPSYNC_JS`, e o tempo de execução, `1.0.0`.
5. O `code`, ou manipuladores de solicitações e respostas de sua função. Embora você possa digitá-lo manualmente, é muito mais fácil adicioná-lo a um arquivo `.txt` (ou formato similar) e depois transmiti-lo como argumento.

Note

Nosso código de consulta estará em um arquivo transmitido como argumento:

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

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



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

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

Veja um exemplo de comando:

```
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}
```

Uma saída será retornada na CLI. Veja um exemplo abaixo:

```
{
  "functionConfiguration": {
    "functionId": "vulcmbfcxffiram63psb4ddua",
    "functionArn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/functions/vulcmbfcxffiram63psb4ddua",
    "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

Certifique-se de gravar o `functionId` em algum lugar, pois isso será usado para anexar a função ao solucionados.

Criar seu resolvidor

- Crie uma função de pipeline para `Mutation` executando o comando [create-resolver](#).

Você precisará inserir alguns parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. O `type-name`, ou o tipo de objeto especial em seu esquema (consulta, mutação, assinatura).
3. O `field-name` ou a operação do campo no tipo de objeto especial a ser anexado ao resolvidor.
4. O `kind`, que especifica um resolvidor de unidade ou pipeline. Defina como `PIPELINE` para ativar as funções do pipeline.
5. A `pipeline-config`, ou as funções a serem anexadas ao resolvidor. É preciso saber os valores de `functionId` de suas funções. A ordem da listagem é importante.
6. O `runtime`, que foi `APPSYNC_JS` (JavaScript). Atualmente o `runtimeVersion` é `1.0.0`.
7. O `code`, que contém os manipuladores das etapas anterior e posterior.

 Note

Nosso código de consulta estará em um arquivo transmitido como argumento:

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

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

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

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

Veja um exemplo de comando:

```
aws appsync create-resolver \
--api-id abcdefghijklmnopqrstuvwxyz \
--type-name Mutation \
--field-name createPost \
--kind PIPELINE \
--pipeline-config functions=vulcmbfcxffiram63psb4ddua \
--runtime name=APPSYNC_JS,runtimeVersion=1.0.0 \
--code file:///path/to/file/{filename}.{fileType}
```

Uma saída será retornada na CLI. Veja um exemplo abaixo:

```
{
  "resolver": {
    "typeName": "Mutation",
    "fieldName": "createPost",
    "resolverArn": "arn:aws:appsync:us-west-2:107289374856:apis/
abcdefghijklmnopqrstuvwxyz/types/Mutation/resolvers/createPost",
    "kind": "PIPELINE",
    "pipelineConfig": {
      "functions": [
        "vulcmbfcxffiram63psb4ddua"
      ]
    },
    "maxBatchSize": 0,
    "runtime": {
      "name": "APPSYNC_JS",
      "runtimeVersion": "1.0.0"
    },
  },
}
```

```

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

```

CDK

Tip

Antes de usar o CDK, recomendamos revisar a documentação oficial do CDK junto com AWS AppSync a referência do CDK.

As etapas listadas abaixo mostram apenas um exemplo geral do trecho usado para adicionar um recurso específico. Isso não é uma solução funcional para seu código de produção. Também presumimos que você já tenha uma aplicação em funcionamento.

- Para fazer uma mutação, supondo que esteja no mesmo projeto, você pode adicioná-la ao arquivo de pilha como na consulta. Aqui está uma função e um resolvedor modificados para uma mutação que adiciona uma nova Post à tabela:

```

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

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

new appsync.Resolver(this, 'pipeline-resolver-create-posts', {

```

```
add_api,  
typeName: 'Mutation',  
fieldName: 'createPost',  
  code: appsync.Code.fromInline(`  
    export function request(ctx) {  
      return {};  
    }  
  
    export function response(ctx) {  
      return ctx.prev.result;  
    }  
  `),  
runtime: appsync.FunctionRuntime.JS_1_0_0,  
pipelineConfig: [add_func_2],  
});
```

Note

Como essa mutação e a consulta são estruturadas de forma semelhante, explicaremos apenas as alterações que implementamos para fazer a mutação. Na função, alteramos o ID da CFN func-add-post e o nome para add_posts_func_1 para refletir o fato de que estamos adicionando Posts à tabela. Na fonte de dados, fizemos uma nova associação à nossa tabela (add_ddb_table) no AWS AppSync console table-for-posts-2 porque o addDynamoDbDataSource método exige isso. Lembre-se de que essa nova associação ainda está usando a mesma tabela que criamos anteriormente, mas agora temos duas conexões com ela no AWS AppSync console: uma para a consulta as table-for-posts e outra para a mutação a stable-for-posts-2. O código foi alterado para adicionar uma Post gerando o valor do id automaticamente e aceitando a entrada de um cliente para o resto dos campos.

No resolvidor, mudamos o valor do ID para pipeline-resolver-create-posts a fim de refletir o fato de que estamos adicionando Posts à tabela. Para refletir a mutação no esquema, o nome do tipo mudou para Mutation, e o nome para createPost. A configuração do pipeline foi definida para nossa nova função de mutação add_func_2.

Para resumir o que está acontecendo neste exemplo, converte AWS AppSync automaticamente os argumentos definidos no createPost campo do seu esquema do GraphQL em operações

do DynamoDB. O exemplo armazena registros no DynamoDB usando uma chave do `id`, que é criada automaticamente usando nosso auxiliar `util.autoId()`. Todos os outros campos que você passar para os argumentos de contexto (`ctx.args.input`) de solicitações feitas no AWS AppSync console ou de outra forma serão armazenados como atributos da tabela. Tanto a chave quanto os atributos são mapeados automaticamente para um formato compatível do DynamoDB usando o auxiliar `util.dynamodb.toMapValues(values)`.

AWS AppSync também oferece suporte a fluxos de trabalho de teste e depuração para edição de resolvedores. Use um objeto `context` de simulação para ver o valor do modelo transformado antes da invocá-lo. Uma alternativa é visualizar a execução de solicitação completa de uma fonte de dados de forma interativa ao executar uma consulta. Para obter mais informações, consulte [Resolvers de teste e depuração \(JavaScript\)](#) e [Monitoramento e registro](#).

Resolvedores avançados

Se você estiver seguindo a seção opcional de paginação em [Projetar seu esquema](#), ainda precisará adicionar seu resolvedor à sua solicitação para usar a paginação. Nosso exemplo usou uma paginação de consulta chamada `getPosts` para retornar somente uma parte das coisas solicitadas por vez. O código do nosso resolvedor nesse campo pode ter a seguinte aparência:

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

Na solicitação, transmitimos no contexto dela. Nosso `limit` é **20**, o que significa que retornamos até 20 Posts na primeira consulta. Nosso cursor do `nextToken` está fixo na primeira entrada da Post na fonte de dados. Eles são transmitidos para os argumentos. Em seguida, a solicitação executa uma varredura desde a primeira Post até o número limite de varredura. A fonte de dados

armazena o resultado no contexto, que é transmitido para a resposta. A resposta retorna as Posts que foram recuperadas e, em seguida, define `nextToken` como a entrada Post logo após o limite. A próxima solicitação é enviada para fazer exatamente a mesma coisa, mas começando pelo deslocamento logo após a primeira consulta. Lembre-se de que esses tipos de solicitações são feitos sequencialmente e não de maneira simultânea.

Testando e depurando resolvedores em () AWS AppSync JavaScript

AWS AppSync executa resolvedores em um campo GraphQL em relação a uma fonte de dados. Ao trabalhar com resolvedores de pipeline, as funções interagem com suas fontes de dados. Conforme descrito na [visão geral dos JavaScript resolvedores](#), as funções se comunicam com as fontes de dados usando manipuladores de solicitação e resposta gravados JavaScript e executados no APPSYNC_JS tempo de execução. Isso permite fornecer lógica e condições personalizadas antes e depois da comunicação com a fonte de dados.

Para ajudar os desenvolvedores a escrever, testar e depurar esses resolvedores, o AWS AppSync console também fornece ferramentas para criar uma solicitação e uma resposta do GraphQL com dados simulados até o resolvedor de campo individual. Além disso, você pode realizar consultas, mutações e assinaturas no AWS AppSync console e ver um fluxo de log detalhado de toda a solicitação da Amazon. CloudWatch Isso inclui resultados da fonte de dados.

Testes com dados simulados

Quando um resolvedor do GraphQL é invocado, ele contém um objeto `context` que contém informações relevantes sobre a solicitação. Isso inclui argumentos de um cliente, informações de identidade e dados do campo pai do GraphQL. Ele também armazena os resultados da fonte de dados, que podem ser usados no manipulador de respostas. Para obter mais informações sobre essa estrutura e os utilitários auxiliares disponíveis para usar durante a programação, consulte a [Referência do objeto de contexto do resolvedor](#).

Ao escrever ou editar uma função de resolução, você pode passar um objeto de contexto simulado ou de teste para o editor do console. Isso permite que você veja como os manipuladores de solicitação e de resposta são avaliados sem realmente serem executados em uma fonte de dados. Por exemplo, você pode enviar um argumento `firstname: Shaggy` de teste e ver como ele avalia ao usar `ctx.args.firstname` no código do modelo. Você também pode testar a avaliação de qualquer utilitário auxiliar, como `util.autoId()` ou `util.time.nowISO8601()`.

Teste de resolvedores

Este exemplo usará o AWS AppSync console para testar os resolvedores.

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na Barra lateral, escolha Funções.
2. Escolha uma função existente.
3. Na parte superior da página Atualizar função, escolha Selecionar contexto de teste e, em seguida, escolha Criar novo contexto.
4. Selecione um objeto de contexto de amostra ou preencha o JSON manualmente na janela Configurar contexto de teste.
5. Insira um nome de contexto de texto.
6. Clique no botão Salvar.
7. Para avaliar o resolvedor usando esse objeto de contexto simulado, escolha Executar teste.

Como exemplo mais prático, digamos que você tenha um aplicativo que armazena um tipo do GraphQL Dog que usa a geração automática de ID para objetos e os armazena no Amazon DynamoDB. Você também deseja gravar alguns valores dos argumentos de uma mutação do GraphQL e permitir que apenas usuários específicos vejam uma resposta. O trecho a seguir mostra a aparência do esquema:

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

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

Você pode escrever uma AWS AppSync função e adicioná-la ao seu addDog resolvedor para lidar com a mutação. Para testar sua AWS AppSync função, você pode preencher um objeto de contexto, como no exemplo a seguir. Ele tem argumentos do cliente de name e age, e um username preenchido no objeto identity:

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



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

Você pode testar sua AWS AppSync função usando o seguinte código:

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

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

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

O manipulador de solicitação e resposta avaliado possui os dados do objeto de contexto de teste e o valor gerado de `util.autoId()`. Além disso, se você alterasse o `username` para um valor diferente de `Nadia`, os resultados não seriam retornados pois a verificação de autorização falharia. Para obter mais informações sobre o controle de acesso refinado, consulte [Casos de uso de autorização](#).

Testando manipuladores de solicitações e respostas com's AWS AppSync APIs

Você pode usar o comando API do `EvaluateCode` para testar remotamente seu código com dados simulados. Para começar a usar o comando, certifique-se de ter adicionado a permissão `appsync:evaluateMappingCode` à sua política. Por exemplo:

JSON

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

Você pode aproveitar o comando usando o [AWS CLI](#) ou [AWS SDKs](#). Por exemplo, veja o Dog esquema e seus manipuladores de solicitação e resposta de AWS AppSync função da seção anterior. Usando a CLI em sua estação local, salve o código em um arquivo chamado `code.js` e salve o objeto `context` em um arquivo chamado `context.json`. No seu shell, execute o seguinte comando:

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

A resposta tem um `evaluationResult` contendo a carga retornada pelo seu manipulador. Também contém um objeto `logs` que contém a lista de logs que foram gerados pelo seu manipulador durante a avaliação. Isso facilita a depuração da execução do código e a visualização de informações sobre sua avaliação para ajudar na solução de problemas. Por exemplo:

```
{
  "evaluationResult": "{\"breed\":\"Miniature Schnauzer\",\"color\":\"black_grey\"}",
  "logs": [
```

```
      "INFO - code.js:13:5: \"This request is allowed\""
    ]
  }
}
```

O `evaluationResult` pode ser analisado como JSON, que fornece:

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

Usando o SDK, você pode incorporar facilmente testes do seu conjunto de testes favorito para validar o comportamento dos seus manipuladores. Recomendamos a criação de testes usando o [Estrutura de trabalho de teste Jest](#), mas qualquer conjunto de testes funciona. O trecho a seguir mostra uma execução de validação hipotética. Esperamos que a resposta de avaliação seja um JSON válido, por isso, usamos `JSON.parse` para recuperar o JSON da resposta da string:

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

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

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

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

Isso produz o seguinte resultado:

```
Ran all test suites.
> jest

PASS ./index.test.js
# request correctly calls DynamoDB (543 ms)
```

```
Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 1.511 s, estimated 2 s
```

Depuração de uma consulta atual

Não há substituto para um end-to-end teste e um registro para depurar um aplicativo de produção. AWS AppSync permite que você registre erros e detalhes completos da solicitação usando a Amazon CloudWatch. Além disso, você pode usar o AWS AppSync console para testar consultas, mutações e assinaturas do GraphQL e transmitir ao vivo os dados de log de cada solicitação de volta ao editor de consultas para depuração em tempo real. Para assinaturas, os logs exibem as informações do tempo de conexão.

Para fazer isso, você precisa ter CloudWatch os registros da Amazon habilitados com antecedência, conforme descrito em [Monitoramento e registro](#). Em seguida, no AWS AppSync console, escolha a guia Consultas e insira uma consulta GraphQL válida. Na seção inferior direita, clique e arraste a janela Registros em log para abrir a visualização de registros. No topo da página, escolha o ícone de seta de reprodução para executar a consulta do GraphQL. Em alguns instantes, os logs completos da solicitação e da resposta para a operação serão transmitidos para essa seção e você poderá visualizá-los no console.

Configurando e usando resolvedores de pipeline em AWS AppSync () JavaScript

AWS AppSync executa resolvedores em um campo GraphQL. Em alguns casos, os aplicativos requerem a execução de várias operações para resolver um único campo do GraphQL. Com os resolvedores de pipeline, os desenvolvedores agora podem elaborar operações chamadas Funções e executá-las em sequência. Os resolvedores de pipeline são úteis para aplicativos que, por exemplo, exigem a execução de uma verificação de autorização antes de obter dados para um campo.

Para obter mais informações sobre a arquitetura de um resolvedor de JavaScript pipeline, consulte a [visão geral dos JavaScript resolvedores](#).

Etapa 1: criar um resolvedor de pipeline

No AWS AppSync console, acesse a página Esquema.

Salve o seguinte esquema:

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

```
mutation: Mutation
}

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

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

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

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

Vamos conectar um resolvedor de pipeline ao campo `signUp` no tipo `Mutação`. No tipo `Mutação` no lado direito, escolha `Anexar` ao lado do campo de mutação `signUp`. Defina o resolvedor como `pipeline resolver` e o runtime `APPSYNC_JS` e, em seguida, crie o resolvedor.

Nosso resolvedor de pipeline cadastra um usuário validando primeiro a entrada do endereço de e-mail e salvando o usuário no sistema. Vamos encapsular a validação de e-mail dentro de uma função `validateEmail` e salvar o usuário dentro de uma função `saveUser`. A função `validateEmail` é executada primeiro e, se o e-mail for válido, a função `saveUser` será executada.

O fluxo de execução será da seguinte forma:

1. Manipulador de solicitações do resolvedor `Mutation.signUp`
2. Função `validateEmail`
3. Função `saveUser`
4. Manipulador de respostas do resolvedor `Mutation.signUp`

Provavelmente reutilizaremos a função `validateEmail` em outros resolvedores em nossa API. Sendo assim, queremos evitar o acesso a `ctx.args`, já que eles mudarão de um campo do GraphQL para

outro. Em vez disso, podemos usar o `ctx.stash` para armazenar o atributo de e-mail a partir do argumento de campo de entrada `signUp(input: Signup)`.

Atualize seu código de resolutor substituindo as funções de solicitação e resposta:

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

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

Escolha Criar ou Salvar para atualizar o resolutor.

Etapa 2: criar uma função

Na página do resolutor de pipeline, na seção Funções, clique em Adicionar função e em Criar função. Também é possível criar funções sem passar pela página do resolutor; para fazer isso, no AWS AppSync console, acesse a página Funções. Selecione o botão Criar função. Vamos criar uma função que verifique se um e-mail é válido e proveniente de um domínio específico. Se o e-mail não for válido, a função gerará um erro. Caso contrário, ele encaminha qualquer entrada fornecida.

Crie uma fonte de dados do tipo NONE. Escolha a fonte de dados na lista Nome da fonte de dados. Em nome da função, insira `validateEmail`. Na área código da função, substitua tudo por este trecho:

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

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

  return { payload: { email } };
}
```

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

Revise suas entradas e selecione Criar. Acabamos de criar nossa função `validateEmail`. Repita essas etapas para criar a função `saveUser` com o código a seguir (para simplificar, usamos a fonte de dados `NONE` e fingimos que o usuário foi salvo no sistema após a execução da função.):

```
import { util } from '@aws-appsync/utils';  
  
export function request(ctx) {  
  return ctx.prev.result;  
}  
  
export function response(ctx) {  
  ctx.result.id = util.autoId();  
  return ctx.result;  
}
```

Acabamos de criar nossa função `saveUser`.

Etapa 3: adicionar uma função a um resolvidor de pipeline

Nossas funções devem ter sido adicionadas automaticamente ao resolvidor de pipeline que acabamos de criar. Se não foi esse o caso ou se você criou as funções por meio da página Funções, é possível clicar em Adicionar função na página do `signUp` resolvidor para anexá-las. Adicione as funções `validateEmail` e `saveUser` ao resolvidor. A função `validateEmail` deve ser colocada antes da função `saveUser`. À medida que você adiciona mais funções, pode usar as setas para cima e para baixo para reorganizar a ordem de execução das funções. Revise suas alterações e escolha Salvar.

Etapa 4: executar uma consulta

No AWS AppSync console, acesse a página Consultas. No explorador, verifique se você está usando a mutação. Se não estiver, escolha `Mutation` na lista suspensa e escolha `+`. Digite a consulta a seguir:

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

```
}
```

Ela deve retornar algo semelhante a:

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

Cadastramos com sucesso nosso usuário e validamos o e-mail de entrada usando um resolvedor de pipeline.

Criação de consultas básicas (VTL)

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Os resolvedores do GraphQL conectam os campos em um esquema de tipo a uma fonte de dados. Os resolvedores são o mecanismo pelo qual as solicitações são atendidas. AWS AppSync pode criar e conectar automaticamente resolvedores a partir de um esquema ou criar um esquema e conectar resolvedores de uma tabela existente sem precisar escrever nenhum código.

Resolvedores em AWS AppSync uso JavaScript para converter uma expressão GraphQL em um formato que a fonte de dados possa usar. Como alternativa, os modelos de mapeamento podem ser escritos em [Apache VTL \(Velocity Template Language\)](#) para converter uma expressão de GraphQL em um formato que a fonte de dados possa usar.

Esta seção mostrará como configurar resolvedores de VTL. [Um guia introdutório de programação em estilo tutorial para escrever resolvedores pode ser encontrado no guia de programação do modelo de mapeamento do Resolver e os utilitários auxiliares disponíveis para uso durante a programação podem ser encontrados na referência de contexto do modelo de mapeamento do Resolver.](#) AWS AppSync também tem fluxos de teste e depuração integrados que você pode usar ao editar ou criar do zero. Para obter mais informações, consulte [Resolvedores de teste e depuração](#).

Recomendamos seguir este guia antes de tentar usar qualquer um dos tutoriais mencionados acima.

Nesta seção, vamos criar um resolvedor, adicionar um resolvedor para mutações e usar as configurações avançadas.

Criar seu primeiro resolvedor

Seguindo os exemplos das seções anteriores, a primeira etapa é criar um resolvedor para seu tipo de Query.

Console

1. Faça login no AWS Management Console e abra o [AppSync console](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, escolha Esquema.
2. No lado direito da página, há uma janela chamada Resolvedores. Essa caixa contém uma lista dos tipos e campos conforme definido na janela Esquema no lado esquerdo da página. Você pode anexar resolvedores aos campos. Por exemplo, no tipo de consulta, escolha Anexar ao lado do campo getTodos.
3. Na página Criar resolvedor, escolha a fonte de dados que você criou no guia [Anexar uma fonte de dados](#). Na janela Configurar modelos de mapeamento, você pode escolher os modelos genéricos de mapeamento de solicitação e resposta na lista suspensa à direita ou escrever suas próprias opções.

Note

A combinação de um modelo de mapeamento de solicitação com um modelo de mapeamento de resposta é chamado de resolvedor de unidades. Os resolvedores de unidades normalmente são destinados a realizar operações rotineiras, e recomendamos usá-los somente para operações individuais com um pequeno número de fontes de dados. Para operações mais complexas, recomendamos o uso de resolvedores de pipeline, que podem executar diversas operações com várias fontes de dados sequencialmente.

Para obter mais informações sobre a diferença entre os modelos de mapeamento de solicitação e resposta, consulte [Resolvedores de unidades](#).

Para obter mais informações sobre o uso de resolvedores de pipeline, consulte [Resolvedores de pipeline](#).

4. Para casos de uso comuns, o AWS AppSync console tem modelos integrados que você pode usar para obter itens de fontes de dados (por exemplo, todas as consultas de itens, pesquisas individuais etc.). Por exemplo, na versão simples do esquema de [Projetar seu esquema](#), onde `getTodos` não tinha paginação, o modelo de mapeamento para listar os itens é o seguinte:

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

5. Sempre é necessário ter um modelo de mapeamento da resposta para acompanhar a solicitação. O console fornece um padrão com o seguinte valor de passagem para listas:

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

Neste exemplo, o objeto context (com o alias de `$ctx`) para listas de itens tem o formato `$context.result.items`. Se a operação do GraphQL retorna um único item, ele seria `$context.result`. O AWS AppSync oferece funções auxiliares para operações comuns, como a função `$util.toJson` listada anteriormente, para formatar respostas corretamente. Para obter uma lista completa das funções, consulte [Referência do utilitário do modelo de mapeamento do resolvidor](#).

6. Escolha Salvar resolvidor.

API

1. Crie um objeto resolvidor chamando a API [CreateResolver](#).
2. Você pode modificar os campos do seu resolvidor chamando a API [UpdateResolver](#).

CLI

1. Crie um resolvidor executando o comando [create-resolver](#).

Você precisará digitar 6 parâmetros para este comando específico:

1. O `api-id` da sua API.

2. O `type-name` do tipo que você deseja modificar em seu esquema. No exemplo do console, tínhamos `Query`.
3. O `field-name` do tipo que você deseja modificar em seu tipo. No exemplo do console, tínhamos `getTodos`.
4. A fonte de dados `data-source-name` que você criou no guia [Anexar uma fonte de dados](#).
5. O `request-mapping-template`, que é o corpo da solicitação. No exemplo do console, tínhamos:

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

6. O `response-mapping-template`, que é o corpo da resposta. No exemplo do console, tínhamos:

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

Veja um exemplo de comando:

```
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)\""
```

Uma saída será retornada na CLI. Veja um exemplo abaixo:

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

```
}  
}
```

2. Para modificar os modelos de and/or mapeamento de campos de um resolvedor, execute o [update-resolver](#) comando.

Com exceção do parâmetro de `api-id`, os parâmetros usados no comando `create-resolver` serão substituídos pelos novos valores do comando `update-resolver`.

Adicionar um resolvedor para mutações

A próxima etapa é criar um resolvedor para seu tipo de `Mutation`.

Console

1. Faça login no AWS Management Console e abra o [AppSync console](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, escolha Esquema.
2. No tipo Mutação, escolha Anexar ao lado do seu campo `addTodo`.
3. Na página Criar resolvedor, escolha a fonte de dados que você criou no guia [Anexar uma fonte de dados](#).
4. Na janela Configurar modelos de mapeamento, você precisará modificar o modelo de solicitação porque essa é uma mutação em que você está adicionando um novo item ao DynamoDB. Use o seguinte modelo de mapeamento da solicitação:

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

5. AWS AppSync converte automaticamente os argumentos definidos no `addTodo` campo do seu esquema do GraphQL em operações do DynamoDB. O exemplo anterior armazena registros no DynamoDB usando uma chave do `id` que é transmitida a partir do argumento da mutação como `$ctx.args.id`. Todos os outros campos

transmitidos são mapeados automaticamente para atributos do DynamoDB com `$util.dynamodb.toMapValuesJson($ctx.args)`.

Para esse resolvedor, use o seguinte modelo de mapeamento da resposta:

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

AWS AppSync também oferece suporte a fluxos de trabalho de teste e depuração para edição de resolvedores. Use um objeto context de simulação para ver o valor transformado do modelo antes de invocar. Opcionalmente, você pode visualizar a execução de solicitação completa para uma fonte de dados de forma interativa ao executar uma consulta. Para obter mais informações, consulte [Resolvedores de teste e depuração](#) e o [Monitoramento e registro em log](#).

6. Escolha Salvar resolvedor.

API

Você também pode fazer isso APIs utilizando os comandos na seção [Criar seu primeiro resolvedor](#) e os detalhes dos parâmetros desta seção.

CLI

Além disso, é possível fazer isso no CLI utilizando os comandos na seção [Criar seu primeiro resolvedor](#) e os detalhes dos parâmetros desta seção.

Neste momento, se não estiver usando os resolvedores avançados você pode começar a usar a API GraphQL conforme descrito em [Uso da API](#).

Resolvedores avançados

Se estiver seguindo a seção Avançado e estiver criando um esquema de exemplo em [Projetar seu esquema](#) para fazer uma verificação paginada, use o seguinte modelo de solicitação para o campo `getTodos`:

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

```
}
```

Para esse caso de uso de paginação, o mapeamento da resposta é mais do que apenas uma passagem porque ele deve conter o cursor (para que o cliente saiba em qual página começar) e o conjunto de resultados. O modelo de mapeamento é conforme mostrado a seguir:

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

Os campos no modelo de mapeamento da resposta anterior devem corresponder aos campos definidos no tipo `TodoConnection`.

Se houver relações em que há uma tabela de `Comments`, e você estiver resolvendo o campo dos comentários no tipo `Todo` (que retorna um tipo de `[Comment]`), use um modelo de mapeamento que execute uma consulta mediante a segunda tabela. Para fazer isso, é necessário já ter criado uma fonte de dados para a tabela `Comments`, conforme descrito em [Associar uma fonte de dados](#).

Note

Estamos usando uma operação de consulta mediante uma segunda tabela somente para fins ilustrativos. Você pode usar outra operação mediante o DynamoDB no lugar. Além disso, você pode extrair os dados de outra fonte de dados, como AWS Lambda o Amazon OpenSearch Service, porque a relação é controlada pelo esquema do GraphQL.

Console

1. Faça login no AWS Management Console e abra o [AppSync console](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, escolha Esquema.
2. No tipo Tarefas, escolha Anexar ao lado do seu campo `comments`.
3. Na página Criar resolvedor, escolha sua fonte de dados da tabela de comentários. O nome padrão da tabela Comentários nos guias de início rápido é `AppSyncCommentTable`, mas pode variar dependendo do nome que você atribuiu a ela.
4. Adicione o seguinte trecho ao seu modelo de mapeamento da solicitação:

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

5. O `context.source` faz referência ao objeto pai do campo atual que está sendo resolvido. Neste exemplo, `source.id` se refere ao objeto `Todo` individual que é, então, usado para a expressão de consulta.

Você pode usar o modelo de mapeamento da resposta de passagem da seguinte forma:

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

6. Escolha `Salvar resolvedor`.
7. Por fim, de volta à página `Esquema` no console, anexe um resolvedor ao campo `addComment` e especifique a fonte de dados da tabela `Comments`. Neste caso, o modelo de mapeamento da solicitação é um simples `PutItem` com o `todoid` específico que está comentado como um argumento, mas use o utilitário `$utils.autoId()` para criar uma chave de classificação única para o comentário da seguinte forma:

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

Use um modelo da resposta de passagem da seguinte forma:

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

API

Você também pode fazer isso APIs utilizando os comandos na seção [Criar seu primeiro resolvidor](#) e os detalhes dos parâmetros desta seção.

CLI

Além disso, é possível fazer isso no CLI utilizando os comandos na seção [Criar seu primeiro resolvidor](#) e os detalhes dos parâmetros desta seção.

Como desabilitar modelos de mapeamento VTL com resolvidores diretos do Lambda (VTL)

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Com resolvidores Lambda diretos, você pode contornar o uso de modelos de mapeamento VTL ao usar fontes de dados. AWS Lambda AWS AppSync pode fornecer uma carga padrão para sua função Lambda, bem como uma tradução padrão da resposta de uma função Lambda para um tipo GraphQL. Você pode optar por fornecer um modelo de solicitação, um modelo de resposta ou nenhum dos dois e AWS AppSync tratará isso adequadamente.

Para saber mais sobre a carga útil padrão da solicitação e a tradução de respostas que AWS AppSync fornece, consulte a referência do resolvidor do [Direct Lambda](#). Para obter mais informações sobre como configurar uma fonte de AWS Lambda dados e configurar uma política de confiança do IAM, consulte [Anexar uma fonte de dados](#).

Configurar resolvidores diretos do Lambda

As seções a seguir mostrarão como anexar fontes de dados do Lambda e adicionar resolvidores do Lambda aos seus campos.

Adicionar uma fonte de dados do Lambda

Antes de ativar os resolvedores diretos do Lambda, você deve adicionar uma fonte de dados do Lambda.

Console

1. Faça login no AWS Management Console e abra o [AppSynconsole](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, selecione Fontes de dados.
2. Escolha Criar fonte de dados.
 - a. Para Nome da fonte de dados, digite um nome para sua fonte de dados, como **myFunction**.
 - b. Para Tipo de fonte de dados, escolha a opção Função AWS Lambda .
 - c. Para Região, escolha a região apropriada.
 - d. Para Função ARN, escolha a função do Lambda na lista suspensa. Você pode pesquisar o nome da função ou inserir manualmente o ARN da função que deseja usar.
 - e. Crie um perfil do IAM (recomendado) ou escolha uma função existente que tenha permissão `lambda:invokeFunction` do IAM. Os perfis existentes precisam de uma política de confiança, conforme explicado na seção [Anexar uma fonte de dados](#).

Veja a seguir um exemplo de política do IAM que tem as permissões necessárias para executar as operações no recurso:

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. Selecione o botão Criar.

CLI

1. Crie um objeto da fonte de dados executando o comando [create-data-source](#).

Você precisará digitar 4 parâmetros para esse comando específico:

1. O `api-id` da sua API.
2. O nome da sua fonte de dados. No exemplo do console, esse é o Nome da fonte de dados.
3. O `type` da fonte de dados. No exemplo do console, isso é função AWS Lambda .
4. O `lambda-config`, que é o ARN da função no exemplo do console.

Note

Existem outros parâmetros, como `Region`, que devem ser configurados, mas geralmente usam como padrão os valores de configuração da CLI.

Veja um exemplo de comando:

```
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
```

Uma saída será retornada na CLI. Veja um exemplo abaixo:

```
{
  "dataSource": {
    "dataSourceArn": "arn:aws:appsync:us-west-2:102847592837:apis/
abcdefghijklmnopqrstuvwxyz/datasources/myFunction",
    "type": "AWS_LAMBDA",
```

```
    "name": "myFunction",
    "lambdaConfig": {
      "lambdaFunctionArn": "arn:aws:lambda:us-
west-2:102847592837:function:appsync-lambda-example"
    }
  }
}
```

2. Para modificar os atributos de uma fonte de dados, execute o comando [update-data-source](#).

Com exceção do `api-id` parâmetro, os parâmetros usados no comando `create-data-source` serão substituídos pelos novos valores do comando `update-data-source`.

Ativar resolvedores diretos do Lambda

Depois de criar uma fonte de dados Lambda e configurar a função apropriada do IAM AWS AppSync para permitir a invocação da função, você pode vinculá-la a uma função de resolução ou pipeline.

Console

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, escolha Esquema.
2. Na janela Resolvedores, selecione um campo ou operação e selecione o botão Anexar.
3. Na página Criar novo resolvedor, escolha a função do Lambda na lista suspensa.
4. Para aproveitar os resolvedores diretos do Lambda, confirme se os modelos de mapeamento de solicitação e resposta estão desativados na seção Configurar modelos de mapeamento.
5. Selecione o botão Salvar resolvedor.

CLI

- Crie um resolvedor executando o comando [create-resolver](#).

Você precisará digitar 6 parâmetros para este comando específico:

1. O `api-id` da sua API.
2. O `type-name` do tipo no seu esquema.

3. O `field-name` do campo no seu esquema.
4. O `data-source-name`, ou o nome da sua função do Lambda.
5. O `request-mapping-template`, que é o corpo da solicitação. No exemplo do console, desabilitamos o seguinte:

```
" "
```

6. O `response-mapping-template`, que é o corpo da resposta. No exemplo do console, o seguinte também foi desabilitado:

```
" "
```

Veja um exemplo de comando:

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

Uma saída será retornada na CLI. Veja um exemplo abaixo:

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

Quando você desativa seus modelos de mapeamento, há vários comportamentos adicionais que ocorrerão no AWS AppSync:

- Ao desativar um modelo de mapeamento, você está sinalizando AWS AppSync que aceita as traduções de dados padrão especificadas na referência do resolvidor do Direct [Lambda](#).

- Ao desativar o modelo de mapeamento de solicitações, sua fonte de dados do Lambda receberá uma payload que consiste em todo o objeto [Contexto](#).
- Ao desativar o modelo de mapeamento de resposta, o resultado da sua invocação do Lambda será traduzido de acordo com a versão do modelo de mapeamento da solicitação ou se o modelo de mapeamento da solicitação também estiver desativado.

Testando e depurando resolvedores em (VTL) AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync executa resolvedores em um campo GraphQL em relação a uma fonte de dados. Conforme descrito na [Visão geral do modelo de mapeamento do resolvedor](#), os resolvedores se comunicam com as fontes de dados usando uma linguagem de modelos. Isso permite personalizar o comportamento e aplicar lógica e condições antes e depois de se comunicar com a fonte de dados. Para obter um guia de programação introdutório no estilo tutorial para programar resolvedores, consulte o [Guia de programação do modelo de mapeamento do resolvedor](#).

Para ajudar os desenvolvedores a escrever, testar e depurar esses resolvedores, o AWS AppSync console também fornece ferramentas para criar uma solicitação e uma resposta do GraphQL com dados simulados até o resolvedor de campo individual. Além disso, você pode realizar consultas, mutações e assinaturas no AWS AppSync console e ver um stream de log detalhado da Amazon CloudWatch de toda a solicitação. Isso inclui os resultados de uma fonte de dados.

Testes com dados simulados

Quando um resolvedor do GraphQL é invocado, ele contém um objeto `context` que contém informações sobre a solicitação. Isso inclui argumentos de um cliente, informações de identidade e dados do campo pai do GraphQL. Ele também contém os resultados da fonte de dados, que podem ser usados no modelo da resposta. Para obter mais informações sobre essa estrutura e os utilitários auxiliares disponíveis para o uso ao programar, consulte a [Referência de contexto do modelo de mapeamento do resolvedor](#).

Ao escrever ou editar um resolvedor, você pode passar um objeto de contexto simulado ou de teste para o editor do console. Isso permite ver como os modelos de solicitação e resposta

avaliar, sem realmente executar segundo uma fonte de dados. Por exemplo, você pode enviar um argumento `firstname: Shaggy` de teste e ver como ele avalia ao usar `$ctx.args.firstname` no código do modelo. Você também pode testar a avaliação de qualquer utilitário auxiliar, como `$util.autoId()` ou `util.time.nowISO8601()`.

Teste de resolvedores

Este exemplo usará o AWS AppSync console para testar os resolvedores.

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, escolha Esquema.
2. Se ainda não tiver feito isso, no tipo e ao lado do campo, escolha Anexar para adicionar seu resolvedor.

Para obter mais informações sobre como construir um resolvedor completo, consulte [Configuração de resolvedores](#).

Caso contrário, selecione o resolvedor que já está no campo.

3. Na parte superior da página Editar resolvedor, escolha Selecionar contexto de teste e escolha Criar novo contexto.
4. Selecione um objeto de contexto de amostra ou preencha o JSON manualmente na janela Contexto de execução.
5. Insira um Nome de contexto de texto.
6. Clique no botão Salvar.
7. Na parte superior da página Editar resolvedor, escolha Executar teste.

Como exemplo mais prático, digamos que você tenha um aplicativo que armazena um tipo do GraphQL Dog que usa a geração automática de ID para objetos e os armazena no Amazon DynamoDB. Você também deseja gravar alguns valores dos argumentos de uma mutação do GraphQL e permitir que apenas usuários específicos vejam uma resposta. Veja a seguir a possível aparência do esquema:

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

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

Ao adicionar um resolvidor para a mutação `addDog`, preencha um objeto de contexto como o exemplo a seguir. Ele tem argumentos do cliente de `name` e `age`, e um `username` preenchido no objeto `identity`:

```
{  
  "arguments" : {  
    "firstname": "Shaggy",  
    "age": 4  
  },  
  "source" : {},  
  "result" : {  
    "breed" : "Miniature Schnauzer",  
    "color" : "black_grey"  
  },  
  "identity": {  
    "sub" : "uuid",  
    "issuer" : " https://cognito-idp.{region}.amazonaws.com/{userPoolId}",  
    "username" : "Nadia",  
    "claims" : { },  
    "sourceIp" : [ "x.x.x.x" ],  
    "defaultAuthStrategy" : "ALLOW"  
  }  
}
```

Você pode testar isso usando os seguintes modelos de mapeamento da solicitação e da resposta:

Modelo de solicitação

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

Modelo da resposta

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

O modelo avaliado tem os dados do objeto de contexto de teste e o valor gerado de `$util.autoId()`. Além disso, se você alterasse o `username` para um valor diferente de `Nadia`, os resultados não seriam retornados pois a verificação de autorização falharia. Para obter mais informações sobre o controle de acesso refinado, consulte [Casos de uso de autorização](#).

Testando modelos de mapeamento com AWS AppSync APIs

Você pode usar o comando da API `EvaluateMappingTemplate` para testar remotamente seus modelos de mapeamento com dados simulados. Para começar a usar o comando, certifique-se de ter adicionado a permissão `appsync:evaluateMappingTemplate` à sua política. Por exemplo:

JSON

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

Você pode aproveitar o comando usando o [AWS CLI](#) ou [AWS SDKs](#). Por exemplo, veja o Dog esquema e seus modelos de request/response mapeamento da seção anterior. Usando a CLI em sua estação local, salve o modelo de solicitação em um arquivo chamado `request.vtl` e salve o objeto `context` em um arquivo chamado `context.json`. No seu shell, execute o seguinte comando:


```
aws appsync evaluate-mapping-template --template file://request.vtl --context file://context.json
```

O comando retorna a seguinte resposta:

```
{
  "evaluationResult": "{\n  \"version\" : \"2017-02-28\",\n  \"operation\" : \"PutItem\",\n  \"key\" : {\n    \"id\" : { \"S\" :\n  \"afcb4c85-49f8-40de-8f2b-248949176456\" }\n  },\n  \"attributeValues\" :\n  {\"firstname\":{\"S\":\"Shaggy\"},\"age\":{\"N\":\"4\"}}\n}\n"
```

O `evaluationResult` contém os resultados do teste do modelo fornecido com o `context` fornecido. Você também pode testar seus modelos usando AWS SDKs. Aqui está um exemplo usando o AWS SDK para JavaScript V2:

```
const AWS = require('aws-sdk')
const client = new AWS.AppSync({ region: 'us-east-2' })

const template = fs.readFileSync('./request.vtl', 'utf8')
const context = fs.readFileSync('./context.json', 'utf8')

client
  .evaluateMappingTemplate({ template, context })
  .promise()
  .then((data) => console.log(data))
```

Usando o SDK, você pode incorporar facilmente testes do seu conjunto de testes favorito para validar o comportamento do seu modelo. Recomendamos a criação de testes usando o [Estrutura de trabalho de teste Jest](#), mas qualquer conjunto de testes funciona. O trecho a seguir mostra uma execução de validação hipotética. Esperamos que a resposta de avaliação seja um JSON válido, por isso, usamos `JSON.parse` para recuperar o JSON da resposta da string:

```
const AWS = require('aws-sdk')
const fs = require('fs')
const client = new AWS.AppSync({ region: 'us-east-2' })

test('request correctly calls DynamoDB', async () => {
  const template = fs.readFileSync('./request.vtl', 'utf8')
  const context = fs.readFileSync('./context.json', 'utf8')
```

```
const contextJSON = JSON.parse(context)

const response = await client.evaluateMappingTemplate({ template,
context }).promise()
const result = JSON.parse(response.evaluationResult)

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

Isso produz o seguinte resultado:

```
Ran all test suites.
> jest

PASS ./index.test.js
# request correctly calls DynamoDB (543 ms)

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 1.511 s, estimated 2 s
```

Depuração de uma consulta atual

Não há substituto para um end-to-end teste e um registro para depurar um aplicativo de produção. AWS AppSync permite que você registre erros e detalhes completos da solicitação usando a Amazon CloudWatch. Além disso, você pode usar o AWS AppSync console para testar consultas, mutações e assinaturas do GraphQL e transmitir ao vivo os dados de log de cada solicitação de volta ao editor de consultas para depuração em tempo real. Para assinaturas, os logs exibem as informações do tempo de conexão.

Para fazer isso, você precisa ter CloudWatch os registros da Amazon habilitados com antecedência, conforme descrito em [Monitoramento e registro](#). Em seguida, no AWS AppSync console, escolha a guia Consultas e insira uma consulta GraphQL válida. Na seção inferior direita, clique e arraste a janela Registros em log para abrir a visualização de registros. No topo da página, escolha o ícone de seta de reprodução para executar a consulta do GraphQL. Em alguns instantes, os logs completos da solicitação e da resposta para a operação serão transmitidos para essa seção e você poderá visualizá-los no console.

Configurando e usando resolvedores de pipeline em AWS AppSync (VTL)

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync executa resolvedores em um campo GraphQL. Em alguns casos, os aplicativos requerem a execução de várias operações para resolver um único campo do GraphQL. Com os resolvedores de pipeline, os desenvolvedores agora podem elaborar operações chamadas Funções e executá-las em sequência. Os resolvedores de pipeline são úteis para aplicativos que, por exemplo, exigem a execução de uma verificação de autorização antes de obter dados para um campo.

Um resolvedor de pipeline é composto de um modelo de mapeamento Anterior, um modelo de mapeamento Posterior e uma lista de Funções. Cada função possui um modelo de mapeamento de solicitação e resposta que é executado mediante uma fonte de dados. Como um resolvedor de pipeline delega a execução a uma lista de funções, ele não está vinculado a nenhuma fonte de dados. Os resolvedores de unidade e funções que executam a operação mediante fontes de dados são primitivos. Consulte [Visão geral do modelo de mapeamento do resolvedor](#) para obter mais informações.

Etapa 1: criar um resolvedor de pipeline

No AWS AppSync console, acesse a página Esquema.

Salve o seguinte esquema:

```
schema {  
  query: Query  
  mutation: Mutation  
}  
  
type Mutation {  
  signUp(input: Signup): User  
}  
  
type Query {  
  getUser(id: ID!): User  
}
```

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

Vamos conectar um resolvidor de pipeline ao campo `signUp` no tipo `Mutação`. No tipo `Mutação` no lado direito, escolha `Anexar` ao lado do campo de mutação `signUp`. Na página de criação do resolvidor, clique em `Ações` e depois em `Atualizar runtime`. Escolha `Pipeline Resolver` e `VTL` e selecione `Atualizar`. Agora, a página deve mostrar 3 seções, uma área de texto `Modelo de mapeamento anterior`, uma seção `Funções` e uma área de texto `Modelo de mapeamento posterior`.

Nosso resolvidor de pipeline cadastra um usuário validando primeiro a entrada do endereço de e-mail e salvando o usuário no sistema. Vamos encapsular a validação de e-mail dentro de uma função `validateEmail` e salvar o usuário dentro de uma função `saveUser`. A função `validateEmail` é executada primeiro e, se o e-mail for válido, a função `saveUser` será executada.

O fluxo de execução será da seguinte forma:

1. Modelo de mapeamento de solicitação do resolvidor `Mutation.signUp`
2. Função `validateEmail`
3. Função `saveUser`
4. Modelo de mapeamento de resposta do resolvidor `Mutation.signUp`

Provavelmente reutilizaremos a função `validateEmail` em outros resolvidores em nossa API. Sendo assim, queremos evitar o acesso a `$ctx.args`, já que eles mudarão de um campo do GraphQL para outro. Em vez disso, podemos usar o `$ctx.stash` para armazenar o atributo de e-mail a partir do argumento de campo de entrada `signUp(input: Signup)`.

Modelo de mapeamento ANTERIOR:

```
## store email input field into a generic email key  
$util.qr($ctx.stash.put("email", $ctx.args.input.email))
```

```
{}
```

O console fornece um modelo de mapeamento POSTERIOR de passagem padrão que usaremos:

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

Escolha Criar ou Salvar para atualizar o resolvedor.

Etapa 2: criar uma função

Na página do resolvedor de pipeline, na seção Funções, clique em Adicionar função e em Criar função. Também é possível criar funções sem passar pela página do resolvedor; para fazer isso, no AWS AppSync console, acesse a página Funções. Selecione o botão Criar função. Vamos criar uma função que verifique se um e-mail é válido e proveniente de um domínio específico. Se o e-mail não for válido, a função gerará um erro. Caso contrário, ele encaminha qualquer entrada fornecida.

Na página da nova função, escolha Ações e, em seguida, Atualizar runtime. Escolha VTL, depois, Atualizar. Crie uma fonte de dados do tipo NONE. Escolha a fonte de dados na lista Nome da fonte de dados. Em nome da função, insira validateEmail. Na área código da função, substitua tudo por este trecho:

```
#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)} }
}
```

Cole isso no modelo de mapeamento de resposta:

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

Verifique suas escolhas e selecione Criar. Acabamos de criar nossa função validateEmail. Repita essas etapas para criar a função saveUser com o código a seguir (a fim de simplificar, usamos a fonte de dados NONE e fingimos que o usuário foi salvo no sistema após a execução da função):

Modelo de mapeamento de solicitação:

```
## $ctx.prev.result contains the signup input values. We could have also
## used $ctx.args.input.
{
  "payload": $util.toJson($ctx.prev.result)
}
```

Modelo de mapeamento da resposta:

```
## an id is required so let's add a unique random identifier to the output
$util.qr($ctx.result.put("id", $util.autoId()))
$util.toJson($ctx.result)
```

Acabamos de criar nossa função `saveUser`.

Etapa 3: adicionar uma função a um resolvedor de pipeline

Nossas funções devem ter sido adicionadas automaticamente ao resolvedor de pipeline que acabamos de criar. Se não foi esse o caso ou se você criou as funções por meio da página Funções, é possível clicar em Adicionar função na página de resolvedor para anexá-las. Adicione as funções `validateEmail` e `saveUser` ao resolvedor. A função `validateEmail` deve ser colocada antes da função `saveUser`. À medida que você adiciona mais funções, pode usar as setas para cima e para baixo para reorganizar a ordem de execução das funções. Revise suas alterações e escolha Salvar.

Etapa 4: executar uma consulta

No AWS AppSync console, acesse a página Consultas. No explorador, verifique se você está usando a mutação. Se não estiver, escolha `Mutation` na lista suspensa e escolha `+`. Digite a consulta a seguir:

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

Ela deve retornar algo semelhante a:

```
{
  "data": {
    "signup": {
      "id": "256b6cc2-4694-46f4-a55e-8cb14cc5d7fc",
      "email": "nadia@myvaliddomain.com"
    }
  }
}
```

Cadastramos com sucesso nosso usuário e validamos o e-mail de entrada usando um resolvidor de pipeline. Para seguir um tutorial mais completo com foco em resolvedores de pipeline, você pode acessar [Tutorial: Resolvedores de pipeline](#)

Usando uma AWS AppSync API com o AWS CDK

Tip

[Antes de usar o CDK, recomendamos revisar a documentação oficial do CDK junto com AWS AppSync a referência do CDK.](#)

Também recomendamos garantir que suas instalações de [AWS CLI](#) e [NPM](#) estejam funcionando em seu sistema.

Nesta seção, criaremos uma aplicação do CDK simples que pode adicionar e buscar itens de uma tabela do DynamoDB. Esse é um exemplo de início rápido usando parte do código das seções [Projetando seu esquema](#), [Anexando uma fonte de dados](#) e [Configurando resolvers](#) (). JavaScript

Configuração de um projeto de CDK

Warning

Essas etapas podem não ser totalmente precisas, dependendo do seu ambiente. Presumimos que seu sistema tenha os utilitários necessários instalados, uma forma de interagir com AWS os serviços e as configurações adequadas.

A primeira etapa é instalar o AWS CDK. Na sua CLI, você pode inserir o seguinte comando:

```
npm install -g aws-cdk
```

Depois, você precisa criar um diretório de projeto e navegar até ele. Um exemplo de conjunto de comandos para criar e navegar até um diretório é:

```
mkdir example-cdk-app
cd example-cdk-app
```

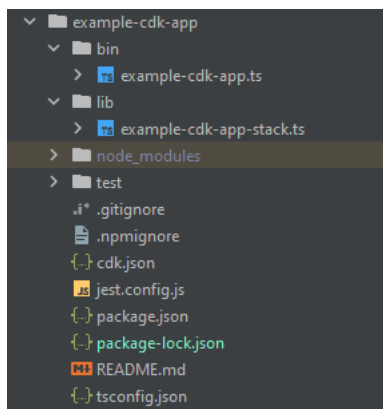
Em seguida, você precisa criar um aplicativo. Nosso serviço usa principalmente TypeScript. No diretório do seu projeto, digite o seguinte comando:

```
cdk init app --language typescript
```

Ao fazer isso, um aplicativo CDK junto com seus arquivos de inicialização será instalado:

```
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!
```

A estrutura do seu projeto pode ser semelhante a esta:



Você notará que temos vários diretórios importantes:

- **bin**: o arquivo bin inicial criará o aplicativo. Não abordaremos esse tema neste guia.
- **lib**: o diretório lib contém seus arquivos de pilha. Você pode pensar nos arquivos de pilha como unidades individuais de execução. As estruturas estarão dentro de nossos arquivos de pilha.

Basicamente, esses são recursos para um serviço que será ativado AWS CloudFormation quando o aplicativo for implantado. É aqui que a maior parte da nossa codificação acontecerá.

- `node_modules`: esse diretório é criado pelo NPM e contém todas as dependências de pacotes que você instalou usando o comando do npm.

Nosso arquivo de pilha inicial pode conter algo assim:

```
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)
    // });
  }
}
```

Esse é o código clichê para criar uma pilha em nosso aplicativo. A maior parte do nosso código neste exemplo estará dentro do escopo dessa classe.

Para verificar se seu arquivo de pilha está no aplicativo, no diretório do seu aplicativo, execute o seguinte comando no terminal:

```
cdk ls
```

Uma lista de suas pilhas deve aparecer. Caso contrário, talvez seja necessário executar as etapas novamente ou verificar a documentação oficial para obter ajuda.

Se quiser criar suas alterações de código antes da implantação, você sempre pode executar o seguinte comando no terminal:

```
npm run build
```

E para ver as mudanças antes da implantação:

```
cdk diff
```

Antes de adicionarmos nosso código ao arquivo de pilha, vamos realizar um bootstrap. O bootstrapping nos permite provisionar recursos para o CDK antes da implantação do aplicativo. Mais informações sobre esse processo podem ser encontradas [aqui](#). Para criar um bootstrap, o comando é:

```
cdk bootstrap aws://ACCOUNT-NUMBER/REGION
```

Tip

Essa etapa exige várias permissões do IAM em sua conta. Seu bootstrap será negado se você não as tiver. Se isso acontecer, talvez seja necessário excluir recursos incompletos causados pelo bootstrap, como o bucket S3 que ele gera.

O bootstrap criará vários recursos. A mensagem final terá a aparência a seguir:

```
✘ Bootstrapping environment [REDACTED]
Trusted accounts for deployment: (none)
Trusted accounts for lookup: (none)
Using default execution policy of 'arn:aws:iam::aws:policy/AdministratorAccess'. Pass '--cloudformation-execution-policies' to customize.
CDKToolkit: creating CloudFormation changeset...
✔ Environment [REDACTED] bootstrapped.
```

Como isso é feito uma vez por conta por região, você não precisará fazer com frequência. Os principais recursos do bootstrap são a AWS CloudFormation pilha e o bucket Amazon S3.

O bucket do Amazon S3 é usado para armazenar arquivos e perfis do IAM que concedem as permissões necessárias para realizar implantações. Os recursos necessários são definidos em uma AWS CloudFormation pilha, chamada pilha de bootstrap, que geralmente é nomeada. CDKToolkit. Como qualquer AWS CloudFormation pilha, ela aparece no AWS CloudFormation console depois de implantada:

Stacks (10)				
Filter by stack name			Filter status	
			Active	☒ View nested
Stack name	Status	Created time	Description	
 CDKToolkit	 CREATE_COMPLETE	2023-07-30 21:20:19 UTC-0700	This stack includes resources needed to deploy AWS CDK apps into this environment	

Isso também se aplica ao bucket:

Name	AWS Region	Access	Creation date
cdk-l-...-assets-...-us-west-2	US West (Oregon) us-west-2	Bucket and objects not public	July 30, 2023, 21:20:29 (UTC-07:00)

Para importar os serviços que precisamos em nosso arquivo de pilha, podemos usar o seguinte comando:

```
npm install aws-cdk-lib # V2 command
```

Tip

Se você estiver tendo problemas com a V2, poderá instalar as bibliotecas individuais usando os comandos da V1:

```
npm install @aws-cdk/aws-appsync @aws-cdk/aws-dynamodb
```

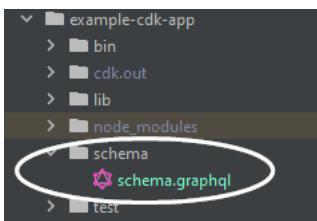
Não recomendamos isso porque a V1 foi descontinuada.

Implementação de um projeto de CDK - Esquema

Agora podemos começar a implementar nosso código. Primeiro, precisamos criar nosso esquema. Você pode simplesmente criar um arquivo `.graphql` no seu aplicativo:

```
mkdir schema
touch schema.graphql
```

Em nosso exemplo, incluímos um diretório de nível superior chamado `schema` que contém nosso `schema.graphql`:



Dentro do nosso esquema, vamos incluir um exemplo simples:

```
input CreatePostInput {
```

```

    title: String
    content: String
  }

  type Post {
    id: ID!
    title: String
    content: String
  }

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

  type Query {
    getPost: [Post]
  }

```

De volta ao nosso arquivo de pilha, precisamos garantir que as seguintes diretivas de importação estejam definidas:

```

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

```

Dentro da classe, adicionaremos código para criar nossa API GraphQL e conectá-la ao nosso arquivo `schema.graphql`:

```

export class ExampleCdkAppStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // makes a GraphQL API
    const api = new appsync.GraphqlApi(this, 'post-apis', {
      name: 'api-to-process-posts',
      schema: appsync.SchemaFile.fromAsset('schema/schema.graphql'),
    });
  }
}

```

Também adicionaremos alguns códigos para imprimir a URL, a chave de API e a região do GraphQL:

```
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
    });
  }
}
```

Neste momento, usaremos a implantação do nosso aplicativo novamente:

```
cdk deploy
```

Este é o resultado:

```
ExampleCdkAppStack: deploying... [1/1]
ExampleCdkAppStack: creating CloudFormation changeset...

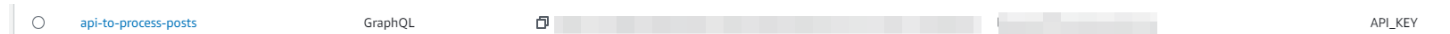
✅ ExampleCdkAppStack

🌟 Deployment time: 16.13s

Outputs:
ExampleCdkAppStack.GraphQLAPIKey = [REDACTED]
ExampleCdkAppStack.GraphQLAPIURL = https://[REDACTED]/graphql
ExampleCdkAppStack.StackRegion = us-west-2
Stack ARN:
arn:aws:cloudformation:[REDACTED]

🌟 Total time: 22s
```

Parece que nosso exemplo foi bem-sucedido, mas vamos verificar o AWS AppSync console apenas para confirmar:



Parece que nossa API foi criada. Agora, verificaremos o esquema anexado à API:

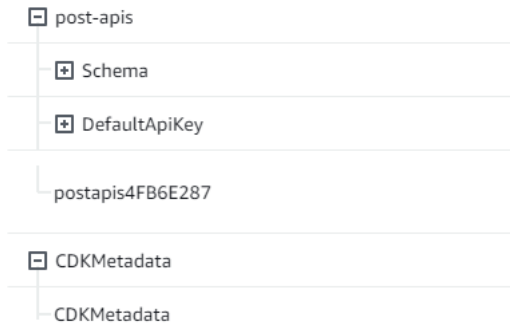
```
Schema

1 input CreatePostInput {
2   title: String
3   date: AWSDateTime
4 }
5
6 type Post {
7   id: ID!
8   title: String
9   date: AWSDateTime
10 }
11
12 type Mutation {
13   createPost(input: CreatePostInput!): Post
14 }
15
16 type Query {
17   getPost: [Post]
18 }
```

Como parece corresponder ao nosso código de esquema, então foi bem-sucedido. Outra forma de confirmar isso do ponto de vista dos metadados é examinar a pilha: AWS CloudFormation

○ ExampleCdkAppStack	🟢 UPDATE_COMPLETE	2023-07-30 22:13:31 UTC-0700	-
○ CDKToolkit	🟢 CREATE_COMPLETE	2023-07-30 21:20:19 UTC-0700	This stack includes resources needed to deploy AWS CDK apps into this environment

Quando implantamos nosso aplicativo CDK, ele usa recursos como o bootstrap. AWS CloudFormation Cada pilha em nosso aplicativo mapeia 1:1 com uma AWS CloudFormation pilha. Se você voltar ao código da pilha, o nome da pilha foi retirado do nome da classe `ExampleCdkAppStack`. Você pode ver os recursos que ele criou, que também correspondem às nossas convenções de nomenclatura em nossa estrutura da API GraphQL:



Implementação de um projeto CDK - Fonte de dados

Depois, precisamos adicionar nossa fonte de dados. Nosso exemplo usará uma tabela do DynamoDB. Dentro da classe de pilha, adicionaremos códigos para criar uma tabela:

```
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
});
}
}
```

Neste momento, vamos implantar novamente:

```
cdk deploy
```

Devemos verificar nossa nova tabela no console do DynamoDB:

☐	ExampleCdkAppStack-poststable	Active	id (S)	-	0		Provisioned (S)	Provisioned (S)	0 bytes	Standard

O nome da nossa pilha está correto e o nome da tabela corresponde ao nosso código. Se verificarmos nossa AWS CloudFormation pilha novamente, agora veremos a nova tabela:

Logical ID
post-apis
posts-table
poststable6CB5A2E6
CDKMetadata

Implementação de um projeto de CDK - Resolvedor

Este exemplo usará dois resolvedores: um para consultar a tabela e outro para adicioná-la. Como estamos usando resolvedores de pipeline, precisaremos declarar dois resolvedores de pipeline com uma função em cada um. Na consulta, adicionaremos o seguinte código:

```
export class ExampleCdkAppStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Makes a GraphQL API construct
```



```
const api = new appsync.GraphqlApi(this, 'post-apis', {
  name: 'api-to-process-posts',
  schema: appsync.SchemaFile.fromAsset('schema/schema.graphql'),
});
```

//creates a DDB table

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

// Creates a function for query

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

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

// Creates a function for mutation

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

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

// Adds a pipeline resolver with the get function
new appsync.Resolver(this, 'pipeline-resolver-get-posts', {
    api,
    typeName: 'Query',
    fieldName: 'getPost',
    code: appsync.Code.fromInline(`
        export function request(ctx) {
            return {};
        }

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

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

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

// Prints out URL
```

```
new cdk.CfnOutput(this, "GraphQLAPIURL", {
    value: api.graphqlUrl
});

// Prints out the AppSync GraphQL API key to the terminal
new cdk.CfnOutput(this, "GraphQLAPIKey", {
    value: api.apiKey || ''
});

// Prints out the stack region to the terminal
new cdk.CfnOutput(this, "Stack Region", {
    value: this.region
});
}
```

Neste trecho, adicionamos um resolvedor de pipeline chamado `pipeline-resolver-create-posts` com uma função chamada `func-add-post` anexada a ele. Esse é o código que adicionará Posts à tabela. O outro resolvedor de pipeline foi chamado `pipeline-resolver-get-posts` com uma função chamada `func-get-post` que recupera Posts adicionado à tabela.

Vamos implantar isso para adicioná-lo ao AWS AppSync serviço:

```
cdk deploy
```

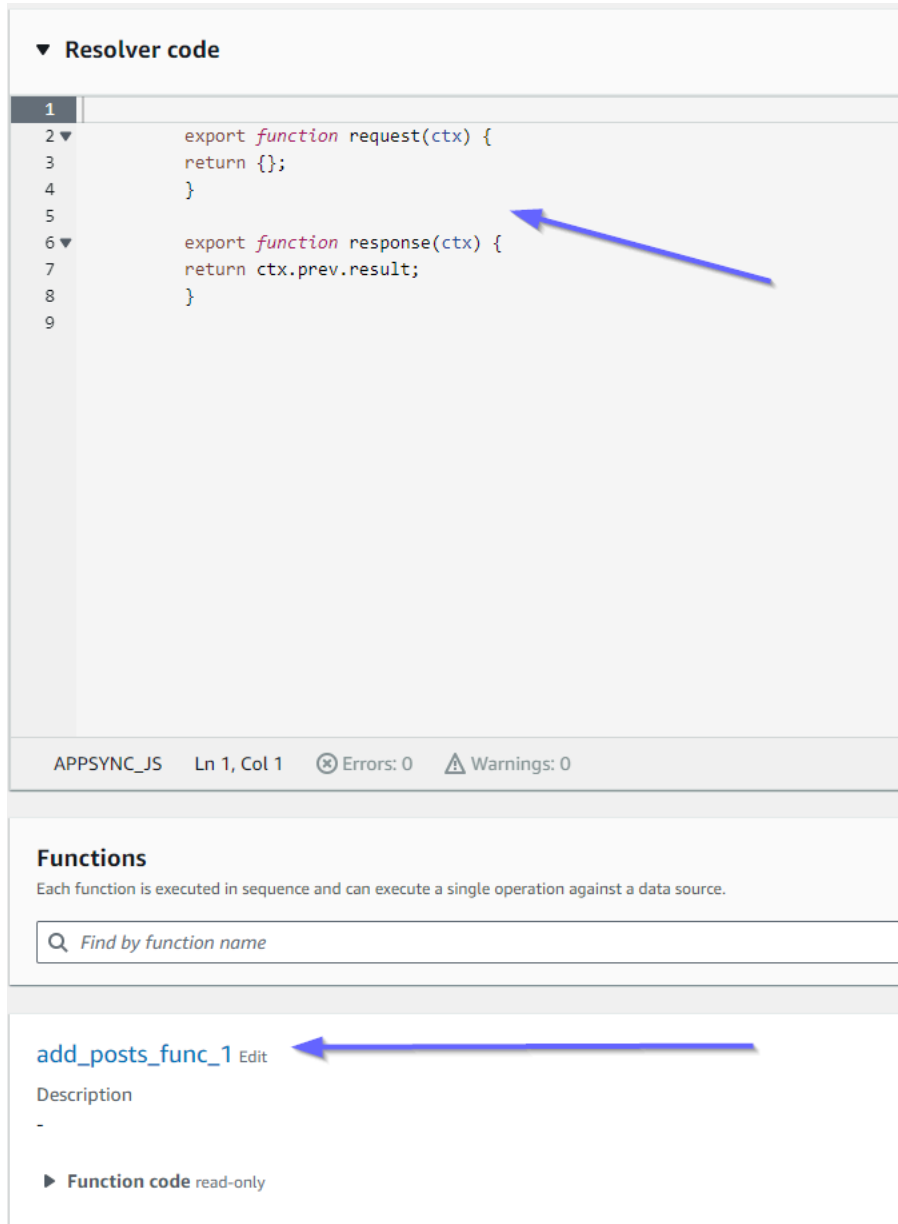
Vamos verificar o AWS AppSync console para ver se eles estavam conectados à nossa API GraphQL:

Mutation	
Field	Resolver
createPost(...): Post	 Pipeline
Query	
Field	Resolver
getPost: [Post]	 Pipeline

Parece estar correto. No código, esses dois resolvedores foram anexados à API GraphQL que criamos (indicada pelo valor de props de `api` presente nos resolvedores e nas funções). Na API

GraphQL, os campos aos quais anexamos nossos resolvedores também foram especificados nas props (definidas pelas props `typename` e `fieldName` em cada resolvedor).

Vamos ver se o conteúdo dos resolvedores está correto começando com `pipeline-resolver-get-posts`:



The screenshot displays the AWS AppSync console interface. The top section, titled "Resolver code", shows the code for the `pipeline-resolver-get-posts` resolver. The code defines two functions: `request` and `response`. The `request` function returns an empty object, and the `response` function returns `ctx.prev.result`. A blue arrow points to the `response` function. Below the code editor, the "Functions" section is visible, showing a list of functions. The function `add_posts_func_1` is highlighted, and a blue arrow points to it. The function description is empty, and the "Function code" is marked as read-only.

```
1  
2 export function request(ctx) {  
3   return {};  
4 }  
5  
6 export function response(ctx) {  
7   return ctx.prev.result;  
8 }  
9
```

APPSYNC_JS Ln 1, Col 1 Errors: 0 Warnings: 0

Functions

Each function is executed in sequence and can execute a single operation against a data source.

Find by function name

add_posts_func_1 Edit

Description

-

► Function code read-only

Os manipuladores de antes e depois correspondem ao valor de nossos props `code`. Também podemos ver que uma função chamada `add_posts_func_1` corresponde ao nome da função que anexamos no resolvedor.

Vamos dar uma olhada no conteúdo do código dessa função:

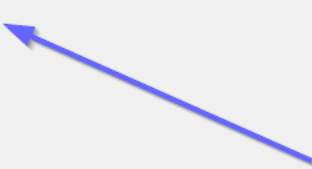
add_posts_func_1 Edit

Description

-

▼ **Function code** read-only

```
1
2   export function request(ctx) {
3     return {
4       operation: 'PutItem',
5       key: util.dynamodb.toMapValues({id: util.autoId()}),
6       attributeValues: util.dynamodb.toMapValues(ctx.args.input),
7     };
8   }
9
10  export function response(ctx) {
11    return ctx.result;
12  }
13
```



Isso corresponde aos props code da função `add_posts_func_1`. Como nossa consulta foi enviada com sucesso, vamos verificar a consulta:

▼ Resolver code

```
1
2 export function request(ctx) {
3   return {};
4 }
5
6 export function response(ctx) {
7   return ctx.prev.result;
8 }
9
```

APPSYNC_JS Ln 1, Col 1  Errors: 0  Warnings: 0

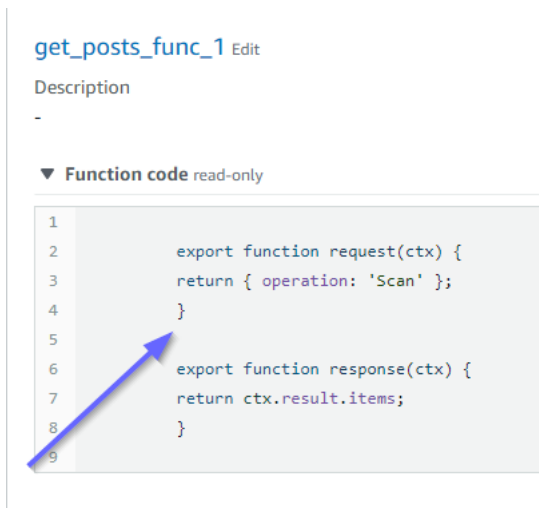
Functions
Each function is executed in sequence and can execute a single operation against a data source.

[get_posts_func_1](#) Edit

Description
-

► **Function code** read-only

Eles também correspondem ao código. Se observarmos `get_posts_func_1`:



Tudo parece estar no lugar certo. Para confirmar isso do ponto de vista dos metadados, podemos verificar nossa pilha no AWS CloudFormation novamente:

Logical ID
+ post-apis
+ posts-table
+ func-get-post
+ func-add-post
+ pipeline-resolver-get-posts
+ pipeline-resolver-create-posts
+ CDKMetadata

Agora, precisamos testar esse código executando algumas solicitações.

Implementação de um projeto de CDK - Solicitações

Para testar nosso aplicativo no AWS AppSync console, fizemos uma consulta e uma mutação:

```

1 query MyQuery {
2   getPost {
3     id
4     date
5     title
6   }
7 }
8
9 mutation MyMutation {
10  createPost(input: {date: "1970-01-01T12:30:00.000Z", title: "first post"}) {
11    date
12    id
13    title
14  }
15 }
16

```

MyMutation contém uma operação createPost com os argumentos 1970-01-01T12:30:00.000Z e first post. Retorna date e title que passamos, bem como o valor id gerado automaticamente. Executar a mutação produz o seguinte resultado:

```

{
  "data": {
    "createPost": {
      "date": "1970-01-01T12:30:00.000Z",
      "id": "4dc1c2dd-0aa3-4055-9eca-7c140062ada2",
      "title": "first post"
    }
  }
}

```

Se verificarmos a tabela do DynamoDB rapidamente, poderemos ver nossa entrada na tabela quando a verificarmos:

☐	id (String)	date	title
☐	9f62c4dd-49d5-48d5-b835-143284c72fe0	1970-01-01T12:30:00.000Z	first post

De volta ao AWS AppSync console, se executarmos a consulta para recuperar issoPost, obteremos o seguinte resultado:

```

{
  "data": {
    "getPost": [
      {
        "id": "9f62c4dd-49d5-48d5-b835-143284c72fe0",
        "date": "1970-01-01T12:30:00.000Z",

```



```
      "title": "first post"
    }
  ]
}
}
```

Usando assinaturas para aplicativos de dados em tempo real no AWS AppSync

Important

A partir de 13 de março de 2025, você pode criar uma PubSub API em tempo real baseada no WebSockets uso de AWS AppSync Eventos. Para obter mais informações, consulte [Publicar eventos WebSocket](#) no Guia do desenvolvedor de AWS AppSync eventos.

AWS AppSync permite que você utilize assinaturas para implementar atualizações de aplicativos ao vivo, notificações push etc. Quando os clientes invocam as operações de assinatura do GraphQL, uma conexão WebSocket segura é automaticamente estabelecida e mantida pelo. AWS AppSync Os aplicativos podem então distribuir dados em tempo real de uma fonte de dados para os assinantes, enquanto gerenciam AWS AppSync continuamente os requisitos de conexão e escalabilidade do aplicativo. As seções a seguir mostrarão como as assinaturas funcionam. AWS AppSync

Diretivas de assinatura do esquema do GraphQL

As assinaturas em AWS AppSync são invocadas como resposta a uma mutação. Isso significa que você pode criar qualquer fonte de dados em tempo AWS AppSync real especificando uma diretiva do esquema GraphQL em uma mutação.

As bibliotecas de AWS Amplify cliente gerenciam automaticamente o gerenciamento de conexões por assinatura. As bibliotecas usam pure WebSockets como protocolo de rede entre o cliente e o serviço.

Note

Para controlar a autorização no momento da conexão com uma assinatura, você pode usar AWS Identity and Access Management (IAM) AWS Lambda, grupos de identidade do

Amazon Cognito ou grupos de usuários do Amazon Cognito para autorização em nível de campo. Para controles de acesso refinados em assinaturas, você pode anexar resolvedores aos seus campos de assinatura e executar a lógica usando a identidade do chamador e das fontes de dados. AWS AppSync Para obter mais informações, consulte [Configurando autorização e autenticação para proteger seu GraphQL APIs](#).

As assinaturas são acionadas a partir de mutações e o conjunto de seleção da mutação é enviado aos assinantes.

O exemplo a seguir mostra como trabalhar com assinaturas do GraphQL. Ele não especifica uma fonte de dados porque a fonte de dados pode ser Lambda, Amazon DynamoDB ou Amazon Service. OpenSearch

Para começar a usar assinaturas, é necessário adicionar um ponto de entrada de assinatura ao esquema, da seguinte forma:

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

Digamos que você tenha um site de postagens de blog e deseje assinar novos blogs e alterações aos blogs existentes. Para fazer isso, adicione a seguinte definição Subscription ao esquema:

```
type Subscription {  
  addedPost: Post  
  updatedPost: Post  
  deletedPost: Post  
}
```

Digamos também que possui as seguintes mutações:

```
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!
```

```
}
```

Você pode tornar esses campos em tempo real ao adicionar uma diretiva `@aws_subscribe(mutations: ["mutation_field_1", "mutation_field_2"])` para cada uma das assinaturas sobre as quais deseja receber notificações, da seguinte forma:

```
type Subscription {  
  addedPost: Post  
    @aws_subscribe(mutations: ["addPost"])  
  updatedPost: Post  
    @aws_subscribe(mutations: ["updatePost"])  
  deletedPost: Post  
    @aws_subscribe(mutations: ["deletePost"])  
}
```

Como a `@aws_subscribe(mutations: ["", ..., ""])` recebe uma matriz de entradas de mutação, você pode especificar diversas mutações, que iniciam uma assinatura. Se estiver assinando a partir de um cliente, a consulta do GraphQL poderá ser semelhante ao seguinte:

```
subscription NewPostSub {  
  addedPost {  
    __typename  
    version  
    title  
    content  
    author  
    url  
  }  
}
```

A consulta de assinatura é necessária para conexões de cliente e ferramentas.

Com o WebSockets cliente puro, a filtragem do conjunto de seleção é feita por cliente, pois cada cliente pode definir seu próprio conjunto de seleção. Nesse caso, o conjunto de seleção da assinatura deve ser um subconjunto do conjunto de seleção da mutação. Por exemplo, uma assinatura `addedPost{author title}` vinculada à mutação `addPost(...){id author title url version}` recebe somente o autor e o título da postagem. Ela não recebe os outros campos. No entanto, se a mutação não tivesse o autor no conjunto de seleção, o assinante obteria um valor `null` para o campo autor (ou um erro, caso o campo de autor fosse definido como necessário/não nulo no esquema).

O conjunto de seleção de assinaturas é essencial ao usar o pure WebSockets. Se um campo não estiver definido explicitamente na assinatura, AWS AppSync ele não retornará.

No exemplo anterior, as assinaturas não tinham argumentos. Digamos que o seu esquema tenha a seguinte aparência:

```
type Subscription {  
  updatedPost(id:ID! author:String): Post  
  @aws_subscribe(mutations: ["updatePost"])  
}
```

Nesse caso, o cliente define uma assinatura da seguinte forma:

```
subscription UpdatedPostSub {  
  updatedPost(id:"XYZ", author:"ABC") {  
    title  
    content  
  }  
}
```

O tipo de retorno de um campo do `subscription` no esquema deve corresponder ao tipo de retorno do campo de mutação correspondente. No exemplo anterior, isso foi mostrado como `addPost` e `addedPost` retornados como um tipo de `Post`.

Para configurar assinaturas no cliente, consulte [Como criar um aplicativo cliente usando o cliente Amplify](#).

Usar argumentos de assinatura

Uma parte importante do uso de assinaturas do GraphQL é entender quando e como usar argumentos. Você pode fazer alterações sutis para modificar como e quando notificar os clientes sobre as mutações que ocorreram. Para fazer isso, consulte o esquema de amostra do capítulo de início rápido, que cria "Todos". Para este exemplo de esquema, são definidas as seguintes mutações:

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

No exemplo padrão, os clientes podem assinar atualizações de qualquer Todo usando o `onUpdateTodo` `subscription` sem argumentos:

```
subscription OnUpdateTodo {  
  onUpdateTodo {  
    description  
    id  
    name  
    when  
  }  
}
```

Você pode filtrar seu `subscription` usando seus argumentos. Por exemplo, para acionar somente um `subscription` quando um todo com um ID específico for atualizado, especifique o valor de ID:

```
subscription OnUpdateTodo {  
  onUpdateTodo(id: "a-todo-id") {  
    description  
    id  
    name  
    when  
  }  
}
```

Você também pode enviar vários argumentos. Por exemplo, o seguinte `subscription` demonstra como receber notificações sobre qualquer atualização do Todo em um local e horário específicos:

```
subscription todosAtHome {  
  onUpdateTodo(when: "tomorrow", where: "at home") {  
    description  
    id  
    name  
    when  
    where  
  }  
}
```

Observe que todos os argumentos são opcionais. Se você não especificar nenhum argumento em seu `subscription`, será inscrito em todas as atualizações do Todo que ocorrerem em seu

aplicativo. No entanto, você pode atualizar sua definição do campo do `subscription` para exigir o argumento do ID. Isso forçaria a resposta de um todo específico em vez de todos os todos:

```
onUpdateTodo(  
  id: ID!,  
  name: String,  
  when: String,  
  where: String,  
  description: String  
): Todo
```

Valor de argumento nulo tem significado

Ao fazer uma consulta de assinatura AWS AppSync, um valor de `null` argumento filtrará os resultados de forma diferente de omitir totalmente o argumento.

Vamos voltar ao exemplo da API todos, onde poderíamos criar todos. Veja o esquema de amostra no capítulo de início rápido.

Vamos modificar nosso esquema para incluir um novo campo de `owner`, no tipo `Todo`, descrevendo quem é o proprietário. O campo de `owner` não é obrigatório e só pode ser ativado em `UpdateTodoInput`. Veja a seguinte versão simplificada do esquema:

```
type Todo {  
  id: ID!  
  name: String!  
  when: String!  
  where: String!  
  description: String!  
  owner: String  
}  
  
input CreateTodoInput {  
  name: String!  
  when: String!  
  where: String!  
  description: String!  
}  
  
input UpdateTodoInput {  
  id: ID!  
  name: String
```

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

type Subscription {
  onUpdateTodo(
    id: ID!,
    name: String!,
    when: String!,
    where: String!,
    description: String!
  ): Todo @aws_subscribe(mutations: ["updateTodo"])
}
```

A assinatura a seguir retorna todas as atualizações de Todo:

```
subscription MySubscription {
  onUpdateTodo {
    description
    id
    name
    when
    where
  }
}
```

Se você modificar a assinatura anterior para adicionar o argumento do campo `owner: null`, estará fazendo uma pergunta diferente. Essa assinatura agora registra o cliente para ser notificado sobre todas as atualizações de Todo que não foram fornecidas a um proprietário.

```
subscription MySubscription {
  onUpdateTodo(owner: null) {
    description
    id
    name
    when
    where
  }
}
```

 Note

A partir de 1º de janeiro de 2022, o MQTT over não WebSockets está mais disponível como protocolo para assinaturas do GraphQL no. AWS AppSync APIs O Pure WebSockets é o único protocolo compatível com AWS AppSync.

Clientes baseados no AWS AppSync SDK ou nas bibliotecas do Amplify, lançados após novembro de 2019, usam automaticamente o WebSockets pure por padrão. Atualizar os clientes para a versão mais recente permite que eles usem AWS AppSync o WebSockets motor puro.

O Pure WebSockets vem com um tamanho de carga útil maior (240 KB), uma variedade maior de opções de clientes e CloudWatch métricas aprimoradas. Para obter mais informações sobre o uso de WebSocket clientes puros, consulte [Construindo um WebSocket cliente em tempo real em AWS AppSync](#).

Criação de genéricos pub/sub APIs com tecnologia sem servidor em WebSockets AWS AppSync

 Important

A partir de 13 de março de 2025, você pode criar uma PubSub API em tempo real baseada no WebSockets uso de AWS AppSync Eventos. Para obter mais informações, consulte [Publicar eventos WebSocket](#) no Guia do desenvolvedor de AWS AppSync eventos.

Alguns aplicativos exigem apenas WebSocket APIs que os clientes ouçam um canal ou tópico específico. Dados JSON genéricos sem formato específico ou requisitos fortemente informados podem ser enviados a clientes que recebem um desses canais em um padrão puro e simples de publicação-assinatura (pub/sub).

Use AWS AppSync para implementar algo simples pub/sub WebSocket APIs com pouco ou nenhum conhecimento em GraphQL em minutos, gerando automaticamente o código GraphQL no back-end da API e no lado do cliente.

Crie e configure o pub-sub APIs

Para começar, faça o seguinte:

1. Faça login no AWS Management Console e abra o [AppSync console](#).
 - No Painel, escolha Criar API.
2. Na próxima tela, escolha Criar uma API em tempo real e, em seguida, escolha Avançar.
3. Insira um nome amigável para sua pub/sub API.
4. Você pode ativar os atributos da [API privada](#), mas recomendamos mantê-los desativados por enquanto. Escolha Próximo.
5. Você pode optar por gerar automaticamente uma pub/sub API funcional usando WebSockets o. Também recomendamos desativar esse atributo por enquanto. Escolha Próximo.
6. Escolha Criar API e aguarde alguns minutos. Uma nova API AWS AppSync pub/sub pré-configurada será criada em sua conta. AWS

A API usa AWS AppSync os resolvedores locais integrados (para obter mais informações sobre o uso de resolvedores locais, consulte o [Tutorial: Resolvers locais](#) no Guia do AWS AppSync desenvolvedor) para gerenciar vários pub/sub canais e WebSocket conexões temporários, o que entrega e filtra automaticamente os dados aos clientes inscritos com base apenas no nome do canal. As chamadas de API são autorizadas com uma chave da API.

Depois que a API for implantada, você verá algumas etapas extras para gerar o código do cliente e integrá-lo ao aplicativo cliente. Para obter um exemplo de como integrar rapidamente um cliente, este guia usará um aplicativo web React simples.

1. Comece criando um aplicativo React padronizado usando o [NPM](#) na máquina local:

```
$ npx create-react-app mypubsub-app
$ cd mypubsub-app
```

Note

Este exemplo usa as [bibliotecas do Amplify](#) para conectar clientes à API de back-end. No entanto, não é necessário criar um projeto Amplify CLI localmente. Embora o React seja o cliente preferido neste exemplo, as bibliotecas do Amplify também oferecem suporte a clientes iOS, Android e Flutter, fornecendo os mesmos recursos nesses diferentes runtimes. [Os clientes Amplify suportados fornecem abstrações simples para interagir com os back-ends da API AWS AppSync GraphQL com poucas linhas de código, incluindo WebSocket recursos integrados totalmente compatíveis com o protocolo em tempo real:AWS AppSync WebSocket](#)

```
$ npm install @aws-amplify/api
```

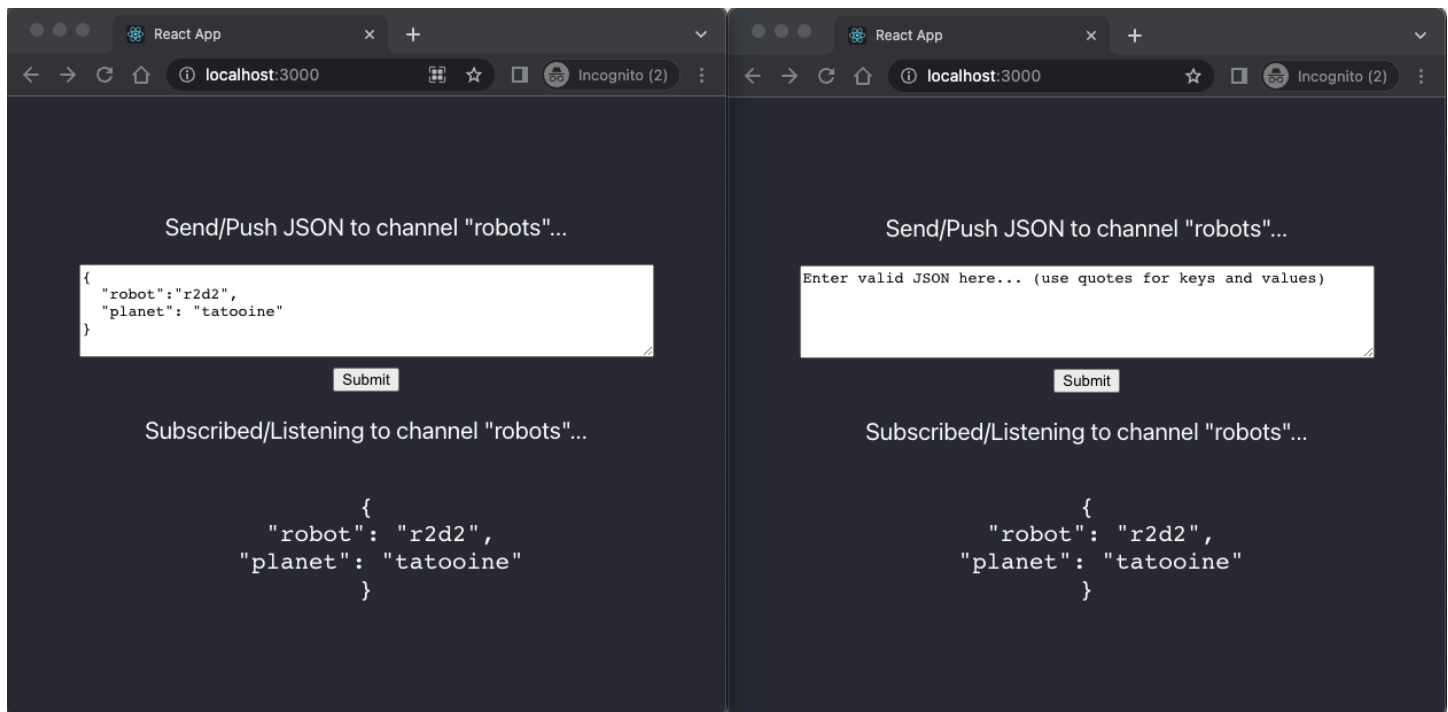
2. No AWS AppSync console, selecione e, em seguida JavaScript, Baixe para baixar um único arquivo com os detalhes de configuração da API e o código de operações do GraphQL gerado.
3. Copie o arquivo baixado para a pasta do `/src` no projeto do React.
4. Em seguida, substitua o conteúdo do arquivo padrão existente do `src/App.js` pelo código de cliente de amostra disponível no console.
5. Use o seguinte comando para criar e iniciar o aplicativo localmente.

```
$ npm start
```

6. Para testar o envio e o recebimento de dados em tempo real, abra duas janelas do navegador e acesse `localhost:3000`. O aplicativo de amostra está configurado para enviar dados JSON genéricos para um canal codificado chamado. `robots`
7. Em uma das janelas do navegador, insira o seguinte blob JSON na caixa de texto e clique em Enviar:

```
{
  "robot": "r2d2",
  "planet": "tatooine"
}
```

Ambas as instâncias do navegador são inscritas no `robots` canal e recebem os dados publicados em tempo real, exibidos na parte inferior do aplicativo web:



Todo o código necessário da API GraphQL, incluindo o esquema, os resolvedores e as operações, é gerado automaticamente para permitir um caso de uso genérico. pub/sub No back-end, os dados são publicados no endpoint em tempo real com uma mutação AWS AppSync do GraphQL, como a seguinte:

```
mutation PublishData {
  publish(data: "{\"msg\": \"hello world!\"}", name: "channel") {
    data
    name
  }
}
```

Os assinantes acessam os dados publicados enviados para o canal temporário específico com uma assinatura relacionada do GraphQL:

```
subscription SubscribeToData {
  subscribe(name:"channel") {
    name
    data
  }
}
```

Implementando o pub-sub APIs em aplicativos existentes

Caso você só precise implementar um recurso em tempo real em um aplicativo existente, essa configuração genérica de pub/sub API pode ser facilmente integrada a qualquer aplicativo ou tecnologia de API. Embora haja vantagens em usar um único endpoint de API para acessar, manipular e combinar com segurança dados de uma ou mais fontes de dados em uma única chamada de rede com o GraphQL, não há necessidade de converter ou reconstruir um aplicativo existente baseado em REST do zero para aproveitar os recursos em tempo real do GraphQL. AWS AppSync Por exemplo, você pode ter uma carga de trabalho CRUD existente em um endpoint de API separado com clientes enviando e recebendo mensagens ou eventos do aplicativo existente para a pub/sub API genérica somente para fins de pub/sub em tempo real.

Definindo filtros de assinaturas aprimorados em AWS AppSync

Important

A partir de 13 de março de 2025, você pode criar uma PubSub API em tempo real baseada no WebSockets uso de AWS AppSync Eventos. Para obter mais informações, consulte [Publicar eventos WebSocket](#) no Guia do desenvolvedor de AWS AppSync eventos.

Em AWS AppSync, você pode definir e habilitar a lógica de negócios para filtragem de dados no back-end diretamente nos resolvedores de assinatura da API GraphQL usando filtros que suportam operadores lógicos adicionais. Você pode configurar esses filtros, diferentemente dos argumentos de assinatura definidos na consulta de assinatura no cliente. Para obter mais informações sobre como usar argumentos de assinatura, consulte [Usar argumentos de assinatura](#). Para obter uma lista de operadores, consulte [AWS AppSync referência do utilitário do modelo de mapeamento do resolvedor](#).

Para os fins deste documento, dividimos a filtragem de dados em tempo real nas seguintes categorias:

- Filtragem básica: filtragem com base nos argumentos definidos pelo cliente na consulta de assinatura.
- Filtragem aprimorada - Filtragem baseada na lógica definida centralmente no back-end do AWS AppSync serviço.

As seções a seguir explicam como configurar filtros de assinatura avançados e mostrar seu uso prático.

Definir assinaturas no esquema do GraphQL

Para usar filtros de assinatura avançados, defina a assinatura no esquema do GraphQL e, em seguida, defina o filtro avançado usando uma extensão de filtragem. Para ilustrar como a filtragem aprimorada de assinaturas funciona AWS AppSync, use o seguinte esquema do GraphQL, que define uma API do sistema de gerenciamento de tickets, como exemplo:

```
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!
```

Suponha que você crie uma fonte de dados do NOME para a API e, em seguida, anexe um resolvedor à mutação do `createTicket` usando essa fonte de dados. Os manipuladores podem ser semelhantes a estes:

```
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

Filtros aprimorados são habilitados no manipulador do resolvedor do GraphQL em uma determinada assinatura. Para obter mais informações, consulte a [Referência do resolvedor](#).

Para implementar o comportamento do filtro aprimorado, você deve usar a função `extensions.setSubscriptionFilter()` para definir uma expressão de filtro avaliada em

relação aos dados publicados de uma mutação do GraphQL na qual os clientes inscritos possam estar interessados. Para ter mais informações sobre as extensões de filtragem, consulte [Extensões](#).

A seção a seguir explica como usar extensões de filtragem para implementar filtros avançados.

Criar filtros avançados de assinatura usando extensões de filtragem

Os filtros avançados são escritos em JSON no manipulador de respostas dos resolvedores da assinatura. Os filtros podem ser agrupados em uma lista chamada `filterGroup`. Os filtros são definidos usando pelo menos uma regra, cada uma com campos, operadores e valores. Vamos definir um novo resolvedor para `onSpecialTicketCreated` que configura um filtro avançado. Você pode configurar várias regras em um filtro que são avaliadas usando a lógica AND, enquanto vários filtros em um grupo de filtros são avaliados usando a lógica OR:

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

export function request(ctx) {
  // simplify 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;
}
```

Com base nos filtros definidos no exemplo anterior, tíquetes importantes serão automaticamente enviados para clientes da API inscritos se um tíquete for criado com:

- nível de `priority` `high` ou `medium`

E

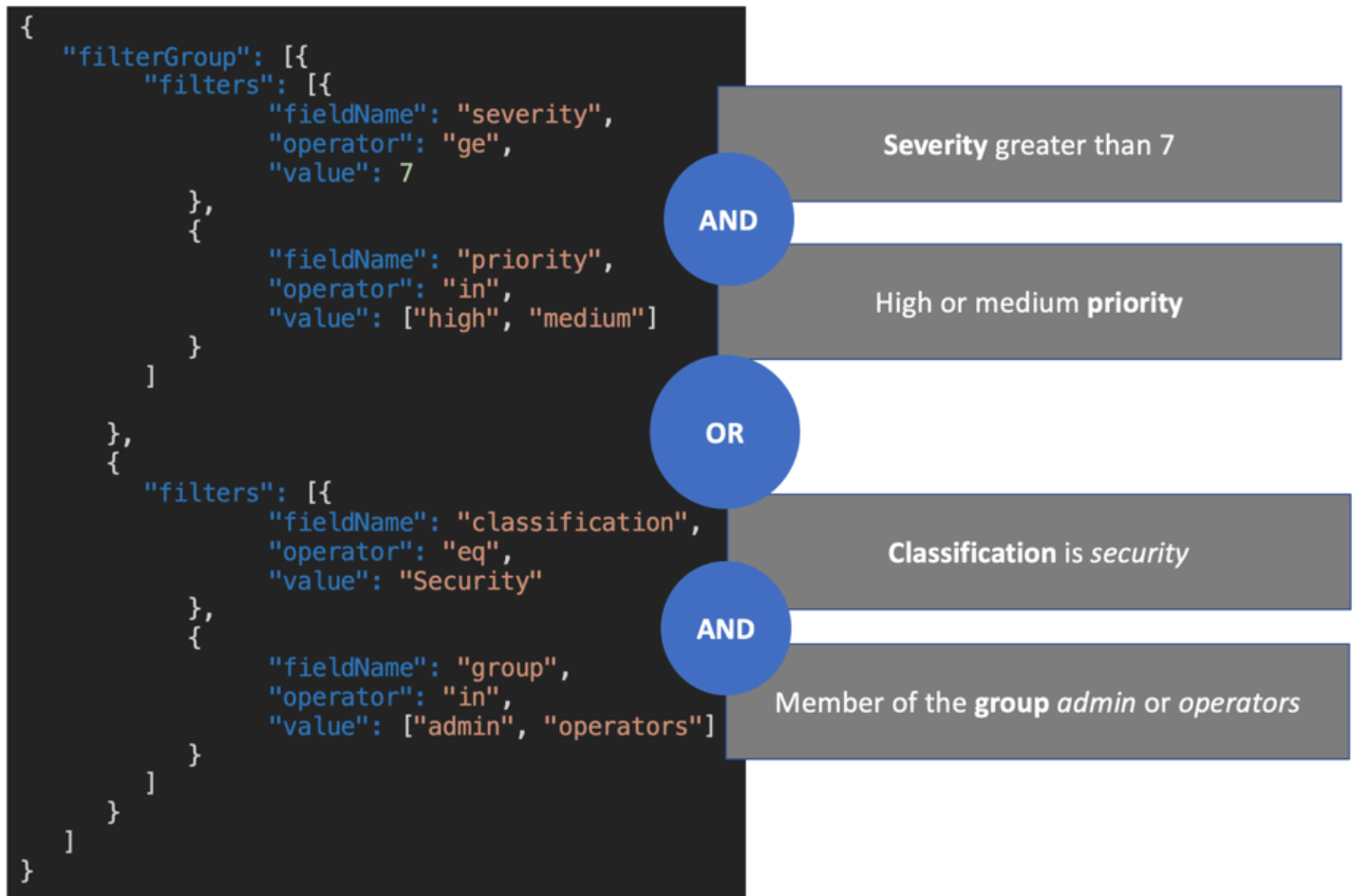
- nível de `severity` maior ou igual a 7 (`ge`)

OU

- `classification` bilhete definido como `Security`

E

- atribuição de `group` definida como `admin` ou `operators`



Os filtros definidos no resolvedor de assinaturas (filtragem avançada) têm precedência sobre a filtragem baseada somente nos argumentos da assinatura (filtragem básica). Para obter mais informações sobre como usar argumentos de assinatura, consulte [Usar argumentos de assinatura](#)).

Se um argumento for definido e exigido no esquema GraphQL da assinatura, a filtragem com base no argumento fornecido ocorrerá somente se o argumento for definido como uma regra no método `extensions.setSubscriptionFilter()` do resolvedor. No entanto, se não houver métodos de filtragem de `extensions` no resolvedor de assinaturas, os argumentos definidos no cliente

serão usados somente para a filtragem básica. Não é possível usar a filtragem básica e a filtragem avançada ao mesmo tempo.

Você pode usar a [variável de context](#) na lógica de extensão de filtro da assinatura para acessar informações contextuais sobre a solicitação. Por exemplo, ao usar grupos de usuários do Amazon Cognito, OIDC ou autorizadores personalizados de Lambda para autorização, você pode recuperar informações sobre os usuários no `context.identity` quando a assinatura é estabelecida. Você pode usar essas informações para estabelecer filtros com base na identidade dos seus usuários.

Agora, suponha que você queira implementar o comportamento de filtro avançado para `onGroupTicketCreated`. A assinatura do `onGroupTicketCreated` exige um nome de grupo obrigatório como argumento. Quando criados, os tickets recebem automaticamente um status `pending`. Você pode configurar um filtro de assinatura para receber somente tickets recém-criados que pertencem ao grupo fornecido:

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

export function request(ctx) {
  // simplify 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;
}
```

Quando os dados são publicados usando uma mutação, como no exemplo a seguir:

```
mutation CreateTicket {
  createTicket(input: {priority: medium, severity: 2, group: "aws"}) {
    id
    priority
    severity
    status
    group
    createdAt
  }
}
```

Os clientes inscritos ouvem os dados a serem enviados automaticamente WebSockets assim que um ticket é criado com a `createTicket` mutação:

```
subscription OnGroup {
  onGroupTicketCreated(group: "aws") {
    category
    status
    severity
    priority
    id
    group
    createdAt
    content
  }
}
```

Os clientes podem ser inscritos sem argumentos porque a lógica de filtragem é implementada no AWS AppSync serviço com filtragem aprimorada, o que simplifica o código do cliente. Os clientes receberão dados somente se os critérios de filtro definidos forem atendidos.

Definição de filtros aprimorados para campos de esquema aninhados

Você pode usar a filtragem de assinatura avançada para filtrar campos de esquema aninhados. Suponha que tenhamos modificado o esquema da seção anterior para incluir tipos de localização e endereço:

```
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
}
```

Com esse esquema, você pode usar um separador . para representar o aninhamento. O exemplo a seguir adiciona uma regra de filtro para um campo de esquema aninhado em `location.address.country`. A assinatura será acionada se o endereço do tíquete estiver definido como USA:

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

export const request = (ctx) => ({ payload: null });
```

```
export function response(ctx) {
  const filter = {
    or: [
      { severity: { ge: 7 }, priority: { in: ['high', 'medium'] } },
      { category: { eq: 'security' }, group: { in: ['admin', 'operators'] } },
      { 'location.address.country': { eq: 'USA' } },
    ],
  };
  extensions.setSubscriptionFilter(util.transform.toSubscriptionFilter(filter));
  return null;
}
```

No exemplo acima, `location` representa o nível de aninhamento um, `address` representa o nível de aninhamento dois e `country` representa o nível de aninhamento três, todos separados pelo separador `..`

Você pode testar essa assinatura usando a mutação `createTicket`:

```
mutation CreateTicketInUSA {
  createTicket(input: {location: "{\"address\":{\"country\":\"USA\"}}"}) {
    category
    content
    createdAt
    group
    id
    location {
      address {
        country
      }
    }
    priority
    severity
    status
  }
}
```

Definindo filtros avançados do cliente

Você pode usar a filtragem básica no GraphQL com [argumentos de assinaturas](#). O cliente que faz a chamada na consulta de assinatura define os valores dos argumentos. Quando filtros aprimorados

são habilitados em um resolvidor de AWS AppSync assinatura com a `extensions` filtragem, os filtros de back-end definidos no resolvidor têm precedência e prioridade.

Configure filtros aprimorados dinâmicos definidos pelo cliente usando um argumento `filter` na assinatura. Ao configurar esses filtros, você deve atualizar o esquema do GraphQL para refletir o novo argumento:

```
...
type Subscription {
  onSpecialTicketCreated(filter: String): Ticket
    @aws_subscribe(mutations: ["createTicket"])
}
...
```

O cliente pode então enviar uma consulta de assinatura, como no exemplo a seguir:

```
subscription onSpecialTicketCreated($filter: String) {
  onSpecialTicketCreated(filter: $filter) {
    id
    group
    description
    priority
    severity
  }
}
```

Você pode configurar a variável de consulta como no exemplo a seguir:

```
{"filter" : "{\"severity\":{\"le\":\"2\"}}"}"
```

O utilitário do resolvidor `util.transform.toSubscriptionFilter()` pode ser implementado no modelo de mapeamento de resposta da assinatura para aplicar o filtro definido no argumento da assinatura para cada cliente:

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

export function request(ctx) {
  // simplify 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;
}
```

Com essa estratégia, os clientes podem definir seus próprios filtros que usam lógica de filtragem avançada e operadores adicionais. Os filtros são atribuídos quando um determinado cliente invoca a consulta de assinatura em uma conexão segura WebSocket. Para obter mais informações sobre o utilitário de transformação para filtragem aprimorada, incluindo o formato da carga útil da variável de `filter` consulta, consulte a visão geral dos [JavaScriptresolvedores](#).

Restrições adicionais de filtragem avançada

Veja abaixo vários casos de uso em que restrições adicionais são colocadas em filtros avançados:

- Os filtros avançados não oferecem suporte à filtragem de listas de objetos de nível superior. Nesse caso de uso, os dados publicados da mutação serão ignorados para assinaturas aprimoradas.
- AWS AppSync suporta até cinco níveis de aninhamento. Os filtros nos campos do esquema após o nível cinco de aninhamento serão ignorados. Veja a resposta do GraphQL abaixo. O campo `continent` em `venue.address.country.metadata.continent` é permitido porque é um ninho de nível cinco. No entanto, `financial` em `venue.address.country.metadata.capital.financial` é um ninho de nível seis, então o filtro não funcionará:

```
{
  "data": {
    "onCreateFilterEvent": {
      "venue": {
        "address": {
          "country": {
            "metadata": {
              "capital": {
                "financial": "New York"
              },
              "continent" : "North America"
            }
          },
          "state": "WA"
        },
        "builtYear": 2023
      }
    }
  }
}
```

```
    },  
    "private": false,  
  }  
}  
}
```

Cancelando a assinatura de WebSocket conexões usando filtros no AWS AppSync

Important

A partir de 13 de março de 2025, você pode criar uma PubSub API em tempo real baseada no WebSockets uso de AWS AppSync Eventos. Para obter mais informações, consulte [Publicar eventos WebSocket](#) no Guia do desenvolvedor de AWS AppSync eventos.

Em AWS AppSync, você pode cancelar a assinatura à força e fechar (invalidar) uma WebSocket conexão de um cliente conectado com base em uma lógica de filtragem específica. Isso será útil em cenários relacionados à autorização, como quando você remove um usuário de um grupo.

A invalidação da assinatura ocorre em resposta a uma carga definida em uma mutação. Recomendamos que você trate as mutações usadas para invalidar conexões de assinatura como operações administrativas em sua API e permissões de escopo de modo adequado, limitando o uso a um usuário administrador, grupo ou serviço de back-end. Por exemplo, usando diretivas de autorização de esquema, como `@aws_auth(cognito_groups: ["Administrators"])` ou `@aws_iam`. Para obter mais informações, consulte [Usar modos de autorização adicionais](#).

Os filtros de invalidação usam a mesma sintaxe e lógica dos [filtros de assinatura aprimorados](#). Defina esses filtros usando os seguintes utilitários:

- `extensions.invalidateSubscriptions()` — Definido no manipulador de resposta do resolvidor do GraphQL para uma mutação.
- `extensions.setSubscriptionInvalidationFilter()` — Definido no manipulador de resposta do resolvidor do GraphQL das assinaturas vinculadas à mutação.

Para obter mais informações sobre extensões de filtragem de invalidação, consulte a visão geral dos [JavaScriptresolvedores](#).

Usar a invalidação da assinatura

Para ver como funciona a invalidação de assinatura AWS AppSync, use o seguinte esquema do GraphQL:

```
type User {
  userId: ID!
  groupId: ID!
}

type Group {
  groupId: ID!
  name: String!
  members: [ID!]!
}

type GroupMessage {
  userId: ID!
  groupId: ID!
  message: String!
}

type Mutation {
  createGroupMessage(userId: ID!, groupId : ID!, message: String!): GroupMessage
  removeUserFromGroup(userId: ID!, groupId : ID!) : User @aws_iam
}

type Subscription {
  onGroupMessageCreated(userId: ID!, groupId : ID!): GroupMessage
    @aws_subscribe(mutations: ["createGroupMessage"])
}

type Query {
  none: String
}
```

Defina um filtro de invalidação no código do resolvidor de mutação `removeUserFromGroup`:

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

export function request(ctx) {
  return { payload: null };
}
```



```
export function response(ctx) {
  const { userId, groupId } = ctx.args;
  extensions.invalidateSubscriptions({
    subscriptionField: 'onGroupMessageCreated',
    payload: { userId, groupId },
  });
  return { userId, groupId };
}
```

Quando a mutação é invocada, os dados definidos no objeto `payload` são usados para cancelar a assinatura definida em `subscriptionField`. Um filtro de invalidação também é definido no modelo de mapeamento de resposta da assinatura do `onGroupMessageCreated`.

Se a `extensions.invalidateSubscriptions()` carga contiver uma ID que corresponda à IDs do cliente inscrito, conforme definido no filtro, a assinatura correspondente será cancelada. Além disso, a WebSocket conexão está fechada. Defina o código do resolvidor de assinatura para a assinatura `onGroupMessageCreated`:

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

export function request(ctx) {
  // simplify 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;
}
```

Observe que o gerenciador de respostas da assinatura pode ter filtros de assinatura e filtros de invalidação definidos ao mesmo tempo.

Por exemplo, suponha que o cliente A inscreva um novo usuário com o ID *user-1* no grupo com o ID *group-1* usando a seguinte solicitação de assinatura:

```
onGroupMessageCreated(userId : "user-1", groupId : "group-1"){...}
```

AWS AppSync executa o resolvidor de assinaturas, que gera filtros de assinatura e invalidação conforme definido no modelo de mapeamento de `onGroupMessageCreated` resposta anterior. Para o cliente A, os filtros de assinatura permitem que os dados sejam enviados somente para `group-1`, e os filtros de invalidação são definidos para `user-1` e `group-1`.

Agora suponha que o cliente B inscreva um usuário com o ID `user-2` no grupo com o ID `group-2` usando a seguinte solicitação de assinatura:

```
onGroupMessageCreated(userId : "user-2", groupId : "group-2"){...}
```

AWS AppSync executa o resolvidor de assinaturas, que gera filtros de assinatura e invalidação. Para o cliente B, os filtros de assinatura permitem que os dados sejam enviados somente para `group-2`, e os filtros de invalidação são definidos para `user-2` e `group-2`.

Em seguida, suponha que uma nova mensagem de grupo com o ID `message-1` seja criada usando uma solicitação de mutação, como no exemplo a seguir:

```
createGroupMessage(id: "message-1", groupId :  
    "group-1", message: "test message"){...}
```

Os clientes inscritos que correspondem aos filtros definidos recebem automaticamente a seguinte carga de dados por meio de: WebSockets

```
{  
  "data": {  
    "onGroupMessageCreated": {  
      "id": "message-1",  
      "groupId": "group-1",  
      "message": "test message",  
    }  
  }  
}
```

O cliente A recebe a mensagem porque os critérios de filtragem correspondem ao filtro de assinatura definido. No entanto, o cliente B não recebe a mensagem, pois o usuário não faz parte de `group-1`. Além disso, a solicitação não corresponde ao filtro de assinatura definido no resolvidor de assinatura.

Por fim, suponha que *user-1* seja removido de *group-1* usando a seguinte solicitação de mutação:

```
removeUserFromGroup(userId: "user-1", groupId : "group-1"){...}
```

A mutação inicia uma invalidação de assinatura, conforme definido no código do manipulador de respostas do `extensions.invalidateSubscriptions()` resolvidor. AWS AppSync em seguida, cancela a assinatura do cliente A e fecha sua conexão. WebSocket O cliente B não é afetado, pois a carga de invalidação definida na mutação não corresponde ao usuário ou grupo.

Quando AWS AppSync invalida uma conexão, o cliente recebe uma mensagem confirmando que sua assinatura foi cancelada:

```
{
  "message": "Subscription complete."
}
```

Usar variáveis de contexto em filtros de invalidação de assinatura

Assim como nos filtros de assinatura aprimorados, você pode usar a [variável context](#) na extensão do filtro de invalidação de assinatura para acessar determinados dados.

Por exemplo, é possível configurar um endereço de e-mail como a carga de invalidação na mutação e, em seguida, compará-lo ao atributo de e-mail ou à solicitação de um usuário inscrito autorizado com grupos de usuários do Amazon Cognito ou OpenID Connect. O filtro de invalidação definido no invalidador de assinatura `extensions.setSubscriptionInvalidationFilter()` verifica se o endereço de e-mail definido pela carga `extensions.invalidateSubscriptions()` da mutação corresponde ao endereço de e-mail recuperado do token JWT do usuário em `context.identity.claims.email`, iniciando a invalidação.

Construindo um WebSocket cliente em tempo real em AWS AppSync

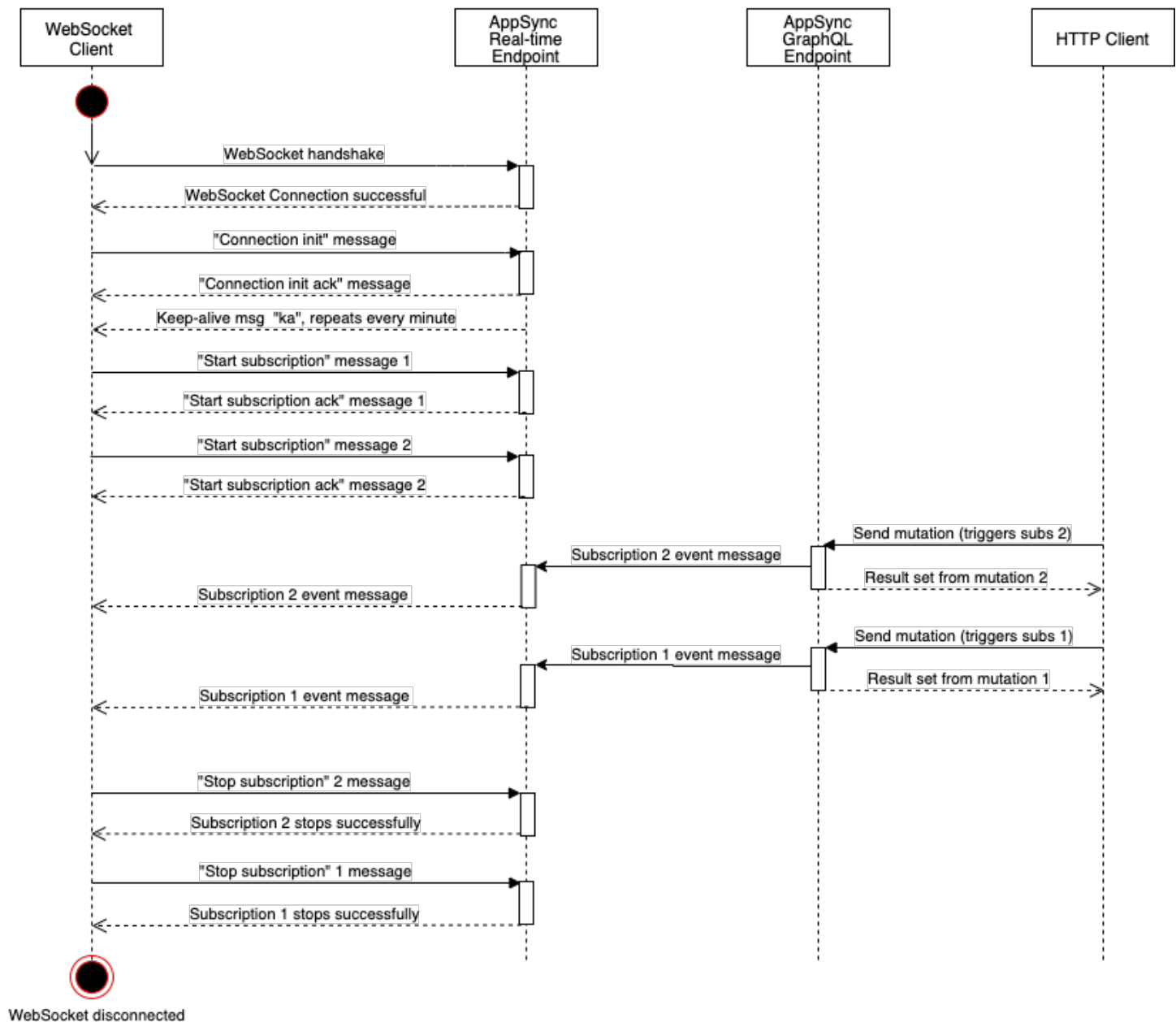
Important

A partir de 13 de março de 2025, você pode criar uma PubSub API em tempo real baseada no WebSockets uso de AWS AppSync Eventos. Para obter mais informações, consulte [Publicar eventos WebSocket](#) no Guia do desenvolvedor de AWS AppSync eventos.

AWS AppSyncO WebSocket cliente em tempo real da GraphQL permite assinaturas do GraphQL por meio de um processo de várias etapas. O cliente primeiro estabelece uma WebSocket conexão com o endpoint AWS AppSync em tempo real, envia uma mensagem de inicialização da conexão e aguarda a confirmação. Após a conexão bem-sucedida, o cliente registra as assinaturas enviando mensagens iniciais com consultas exclusivas e do IDs GraphQL. AWS AppSync confirma assinaturas bem-sucedidas com mensagens de confirmação. O cliente recebe os eventos de assinatura, que são enviados pelas mutações correspondentes. Para manter a conexão, AWS AppSync envia mensagens periódicas de manutenção de atividade. Ao terminar, o cliente cancela o registro das assinaturas enviando mensagens de interrupção. Esse sistema suporta várias assinaturas em uma única WebSocket conexão e acomoda vários modos de autorização, incluindo chaves de API, grupos de usuários do Amazon Cognito, IAM e Lambda.

Implementação de WebSocket cliente em tempo real para assinaturas do GraphQL

O diagrama de sequência e as etapas a seguir mostram o fluxo de trabalho de assinaturas em tempo real entre o WebSocket cliente, o cliente HTTP e AWS AppSync



1. O cliente estabelece uma WebSocket conexão com o endpoint AWS AppSync em tempo real. Se houver um erro de rede, o cliente deverá fazer um recuo exponencial com variação. Para obter mais informações, consulte [Recuo exponencial e instabilidade no blog de arquitetura](#). AWS
2. (Opcional) Depois de estabelecer a WebSocket conexão com sucesso, o cliente envia uma `connection_init` mensagem.
3. Se `connection_init` for enviado, o cliente espera por uma `connection_ack` mensagem de AWS AppSync. Essa mensagem inclui um parâmetro `connectionTimeoutMs`, que é o tempo de espera máximo em milissegundos para uma mensagem "ka", keep-alive.

4. AWS AppSync envia "ka" mensagens periodicamente. O cliente monitora o horário de recebimento de cada mensagem "ka". Se o cliente não receber uma mensagem "ka" em `connectionTimeoutMs` milissegundos, ele deverá fechar a conexão.
5. O cliente registra a assinatura enviando uma mensagem de assinatura `start`. Uma única WebSocket conexão suporta várias assinaturas, mesmo que elas estejam em modos de autorização diferentes.
6. O cliente espera o envio de `start_ack` mensagens AWS AppSync para confirmar as assinaturas bem-sucedidas. Se houver um erro, AWS AppSync retornará uma `"type": "error"` mensagem.
7. O cliente recebe os eventos de assinatura, que são enviados após a chamada de uma mutação correspondente. As consultas e mutações geralmente são enviadas `https://` para o endpoint do GraphQL AWS AppSync. As assinaturas fluem pelo endpoint AWS AppSync em tempo real usando o `secure WebSocket ()`. `wss://`
8. O cliente cancela o registro da assinatura enviando uma mensagem de assinatura `stop`.
9. Depois de cancelar o registro de todas as assinaturas e verificar se não há mensagens sendo transferidas pelo WebSocket, o cliente pode se desconectar da conexão. WebSocket

Detalhes do aperto de mão para estabelecer a conexão WebSocket

Para se conectar e iniciar um handshake bem-sucedido com AWS AppSync, um WebSocket cliente precisa do seguinte:

- O endpoint AWS AppSync em tempo real
- Cabeçalhos — contêm informações relevantes para o AWS AppSync endpoint e a autorização. AWS AppSync suporta os três métodos a seguir para fornecer cabeçalhos:
 - Cabeçalhos via string de consulta
 - As informações do cabeçalho são codificadas como uma string base64, derivada de um objeto JSON stringificado. Esse objeto JSON contém detalhes relevantes para o AWS AppSync endpoint e a autorização. O conteúdo do objeto JSON varia de acordo com o modo de autorização.
 - Cabeçalhos via `Sec-WebSocket-Protocol`
 - Uma string codificada em Base64URL do objeto JSON stringificado que contém informações relevantes para o AWS AppSync endpoint e a autorização é passada como protocolo no cabeçalho. `Sec-WebSocket-Protocol` O conteúdo do objeto JSON varia de acordo com o modo de autorização.

- Cabeçalhos por meio de cabeçalhos HTTP padrão:
 - Os cabeçalhos podem ser passados como cabeçalhos HTTP padrão na solicitação de conexão, da mesma forma que os cabeçalhos são passados para consultas e mutações do GraphQL. AWS AppSync No entanto, a transmissão de cabeçalhos por meio de cabeçalhos HTTP padrão não é compatível com solicitações de conexão de API privadas.
- payload— Cadeia de caracteres codificada em Base64 de. payload A carga útil é necessária somente se os cabeçalhos forem fornecidos usando a string de consulta

Com esses requisitos, um WebSocket cliente pode se conectar à URL, que contém o endpoint em tempo real com a string de consulta, usando `graphql-ws` como WebSocket protocolo.

Descobrir o endpoint em tempo real com base no endpoint do GraphQL

O endpoint do AWS AppSync GraphQL e o endpoint AWS AppSync em tempo real são ligeiramente diferentes em protocolo e domínio. Você pode recuperar o endpoint do GraphQL usando AWS Command Line Interface o AWS CLI comando `()`. `aws appsync get-graphql-api`

AWS AppSync Ponto final do GraphQL:

```
https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql
```

AWS AppSync endpoint em tempo real:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Os aplicativos podem se conectar ao endpoint do AWS AppSync GraphQL (`https://`) usando qualquer cliente HTTP para consultas e mutações. Os aplicativos podem se conectar ao endpoint AWS AppSync em tempo real (`wss://`) usando qualquer WebSocket cliente para assinaturas.

Os nomes de domínio personalizados permitem interagir com os dois endpoints usando um único domínio. Por exemplo, se você configurar `api.example.com` como seu domínio personalizado, poderá interagir com seus endpoints do GraphQL e em tempo real usando estes: URLs

AWS AppSync endpoint GraphQL de domínio personalizado:

```
https://api.example.com/graphql
```

AWS AppSync endpoint em tempo real de domínio personalizado:

```
wss://api.example.com/graphql/realtime
```

Formato do parâmetro de cabeçalho com base no modo de autorização AWS AppSync da API

O formato do header objeto usado na string de consulta de conexão varia de acordo com o modo de autorização da AWS AppSync API. O `host` campo no objeto se refere ao endpoint do AWS AppSync GraphQL, que é usado para validar a conexão mesmo se a `wss://` chamada for feita no endpoint em tempo real. Para iniciar o handshake e estabelecer a conexão autorizada, `payload` deve ser um objeto JSON vazio. A carga útil é necessária somente se os cabeçalhos forem passados por meio da string de consulta.

As seções a seguir demonstram os formatos de cabeçalho para cada modo de autorização.

Chave de API

Cabeçalho da chave da API

Conteúdo do cabeçalho

- `"host": <string>`: o host do endpoint do AWS AppSync GraphQL ou do seu nome de domínio personalizado.
- `"x-api-key": <string>`: a chave de API configurada para a AWS AppSync API.

Exemplo

```
{
  "host": "example1234567890000.appsync-api.us-east-1.amazonaws.com",
  "x-api-key": "da2-12345678901234567890123456"
}
```

Cabeçalhos via string de consulta

Primeiro, um objeto JSON contendo o `host` e o `x-api-key` é convertido em uma string. Em seguida, essa string é codificada usando a codificação base64. A string codificada em base64 resultante é adicionada como um parâmetro de consulta nomeado `header` ao WebSocket URL para estabelecer a conexão com o AWS AppSync endpoint em tempo real. O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJ0b3N0IjoizXhhbXBsZTEyMzQ1Njc4OTAwMDAuYXBwc3luYy1hcGkudXMtZWZzdC0xLmFtYXpvbmF3cy5jb20i
```


É importante observar que, além do objeto de cabeçalho codificado em base64, um objeto JSON vazio {} também é codificado em base64 e incluído como um parâmetro de consulta separado nomeado na URL. `payload WebSocket`

Cabeçalhos via **Sec-WebSocket-Protocol**

Um objeto JSON contendo o `host` e o `x-api-key` é convertido em uma string e depois codificado usando a codificação Base64URL. A string codificada em Base64URL resultante é prefixada com `header-`. Essa string prefixada é então usada como um novo subprotocolo, além do `Sec-WebSocket-Protocol` cabeçalho, ao `graphql-ws` estabelecer a WebSocket conexão com o endpoint AWS AppSync em tempo real.

O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

O `Sec-WebSocket-Protocol` cabeçalho contém o seguinte valor:

```
"sec-websocket-protocol" : ["graphql-ws", "header-ewogICAgImhvc3QiOiJleGFtcGx1MTIzNDU2Nzg5MDAwMC5hcHBzeW5jLWFWaS51cy1lYXN0LTEuYW1hem9uYXdzLmNvbSI
```

Cabeçalhos via cabeçalhos HTTP padrão

Nesse método, as informações da chave do host e da API são transmitidas usando cabeçalhos HTTP padrão ao estabelecer a WebSocket conexão com o endpoint AWS AppSync em tempo real.

O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Os cabeçalhos da solicitação incluiriam o seguinte:

```
"sec-websocket-protocol" : ["graphql-ws"]
"host": "example1234567890000.appsync-api.us-east-1.amazonaws.com",
"x-api-key": "da2-12345678901234567890123456"
```

Grupos de usuários do Amazon Cognito e OpenID Connect (OIDC)

Amazon Cognito e OIDCheader

Conteúdo do cabeçalho:

- "Authorization": <string>: Um token de ID JWT. O cabeçalho pode usar um [esquema Bearer](#).
- "host": <string>: o host do endpoint do AWS AppSync GraphQL ou do seu nome de domínio personalizado.

Exemplo:

```
{
  "Authorization": "eyJEXAMPLEiJjbG5xb3A5eW5MK09QYXIrMTJHWEFLSXBieU5WNHhsQjEXAMPLEnM2WldvPSIsImFsZS4EE2DJH7sH0l2zxYi7f-SmEGoh2AD8emxQRYajByz-rE4Jh0Q0ymN2Ys-ZIkMpVBTPgu-TMWdy0HhDumUj20P82yeZ3w1Zatr_gM4LzjXUXmI_K2yGjuXfXTaa1mvQEBG0mQfVd7SfwXB-jcv4RYVi6j25qgow9Ew52ufurPqaK-3WAKG32KpV8J4-Wejq8t0c-yA7sb8EnB551b7TU93uKRiVVK3E55Nk5ADPoam_WYE45i3s5qVAP_-InW75NUo0CGTsS8YWMfb6ecHYJ-1j-bzA27zaT9VjctXn9byNFZmEXAMPLExw",
  "host": "example1234567890000.appsync-api.us-east-1.amazonaws.com"
}
```

Cabeçalhos via string de consulta

Primeiro, um objeto JSON contendo o host e o Authorization é convertido em uma string. Em seguida, essa string é codificada usando a codificação base64. A string codificada em base64 resultante é adicionada como um parâmetro de consulta nomeado header ao WebSocket URL para estabelecer a conexão com o AWS AppSync endpoint em tempo real. O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJBdXRob3JpemF0aW9uIjoiZXlKcmFXUWlPaUpqYkc1eGIzQTVlVzVNSzA5UVlYSXJNVEpIV0VGTFNYQm1lVTVX
```

É importante observar que, além do objeto de cabeçalho codificado em base64, um objeto JSON vazio {} também é codificado em base64 e incluído como um parâmetro de consulta separado nomeado na URL. payload WebSocket

Cabeçalhos via **Sec-WebSocket-Protocol**

Um objeto JSON contendo o host e o Authorization é convertido em uma string e depois codificado usando a codificação Base64URL. A string codificada em Base64URL resultante é prefixada com. header - Essa string prefixada é então usada como um novo subprotocolo, além do

Sec-WebSocket-Protocol cabeçalho, ao graphql-ws estabelecer a WebSocket conexão com o endpoint AWS AppSync em tempo real.

O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

O Sec-WebSocket-Protocol cabeçalho contém o seguinte valor:

```
"sec-websocket-protocol" : ["graphql-ws", "header-ewogICAgImhvc3QiOiJleGFtcGx1MTIzNDU2Nzg5MDAwMC5hcHBzeW5jLWFwaS51cy11YXN0LTEuYW1hem9uYXdzLmNvbSI
```

Cabeçalhos via cabeçalhos HTTP padrão

Nesse método, as informações do host e da autorização são transmitidas usando cabeçalhos HTTP padrão ao estabelecer a WebSocket conexão com o endpoint AWS AppSync em tempo real. O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Os cabeçalhos da solicitação incluiriam o seguinte:

```
"sec-websocket-protocol" : ["graphql-ws"]
"Authorization": "eyJEXAMPLEiJjbG5xb3A5eW5MK09QYXIrMTJHWEFLSXBieU5WNHhsQjEXAMPLEnM2WldvPSIsImFsZy9lbnQiOiJleGFtcGx1MTIzNDU2Nzg5MDAwMC5hcHBzeW5jLWFwaS51cy11YXN0LTEuYW1hem9uYXdzLmNvbSI",
"host": "example1234567890000.appsync-api.us-east-1.amazonaws.com"
```

IAM

Cabeçalho IAM

Conteúdo do cabeçalho

- "accept": "application/json, text/javascript": Um parâmetro <string> constante.
- "content-encoding": "amz-1.0": Um parâmetro <string> constante.
- "content-type": "application/json; charset=UTF-8": Um parâmetro <string> constante.

- "host": <string>: Esse é o host do endpoint do AWS AppSync GraphQL.
- "x-amz-date": <string>: O carimbo de data/hora deve ser em UTC e no seguinte formato: ISO 8601 YYYYMMDD'T'HHMMSS'Z '. Por exemplo, 20150830T123600Z é um carimbo de data/hora válido. Não inclua milissegundos no carimbo de data/hora. Para obter mais informações, consulte [Como lidar com datas no Signature versão 4](#) na Referência geral da AWS.
- "X-Amz-Security-Token": <string>: o token da AWS sessão, que é necessário ao usar credenciais de segurança temporárias. Para obter mais informações, consulte [Usar credenciais temporárias com recursos da AWS](#) no Guia do usuário do IAM.
- "Authorization": <string>: Informações de assinatura do Signature Version 4 (SigV4) para o AWS AppSync endpoint. Para obter mais informações sobre o processo de assinatura, consulte [Tarefa 4: Adicionar a assinatura à solicitação HTTP](#) na Referência geral da AWS.

A solicitação HTTP de assinatura do SigV4 inclui um URL canônico, que é o endpoint do GraphQL do AWS AppSync com /connect anexado. A AWS região do endpoint do serviço é a mesma região em que você está usando a AWS AppSync API, e o nome do serviço é 'appsync'. A solicitação HTTP para assinar é a seguinte:

```
{
  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",
  }
}
```

Exemplo

```
{
  "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":
  "AgEXAMPLEZ2luX2VjEAoaDmFwLXNvdXR0ZWFEXAMPLECwRQIgAh97C1jq7w0PL8KsxP3YtDuyC/9hAj8PhJ7Fvf38SgoC"
```

```
+
+pEagWCveZUjKE0zyUhBEXAMPLEjj//////////8BEXAMPLEx0Dk2NDgyNzg1NSIMo1mWnpESWUoYw4BkKqEFSrm3DXuL8
+ZbVc4JKjDP4vUCKNR6Le9C9pZp9PsW0NoFy3vLBudAXEXAMPLE0VG8feXfiEEA+1khgFK/
wEtWR+9zF7NaMMmSe07wN2gG2tH0eKMEXAMPLEQX+sMbytQo8iepP9PZ0z1ZsSFb/
dP5Q8hk6YEXAMPLEYcKZsTkDAq2uKFQ8mYUVA9EtQnNRiFLEY83aKvG/tqLWNnG1SNVx7SMcfovkFDqQamm
+88y10wwAEYK7qcoceX6Z7GGcaYuIfGpaX2MCCELeQvZ+8WxEg0nIfz7GYvsYNjLZSaRnV4G
+ILY1F0QNW64S9Nvj
+BwDg3ht2CrNvpwjVY1j9U3nmxE0UG5ne83LL5hhqMpm25kmL7enVgw2kQzmU2id4IKu0C/
WaoDRu02F5zE63vJbxN8AYs7338+4B4HbB6BZ60Ugg96Q15RA41/
gIqxaVPxyTpDfTU5GfSLxocdYeniqqpFMtZG2n9d0u7GsQNcFkNcG3qDZm4tDo8tZbuym0a2VcF2E5hFEgXBa
+XLJCfXi/770qAEjP0x7Qdk3B43p8KG/BaioP5RsV8zBGvH1zAgyPha2rN70/
tT13yrmPd5QYEFwzexjKrV4mWIuRg8NTHYSZJUaeyCwTom80VFUJXG
+GYTUyv5W22aBcnoRGiCiKEYTL0kgXecdKFTHmcIAejQ9We1r0a196Kq87w5KNMCKcCGFnwBNFLmfmbpNqT6rUBxxs3X5nt
aox0FtHX21eF6qIGT8j1z+l2opU+ggwUgkhUUgCH2TfqBj+MLMVvpvgqJsPKt582caFKArIFIv0
+9QupxLnEH2hz04TMTfnU6bQC6z1buVe7h
+t0Lnh1YPFsLQ88anib/7TTC8k9DsBTq0ASE8R2GbSEsm09qbbMwgEaYUh0KtGeyQsSJdhSk6XxXThrWL9EnwBCXDkICMqd
+WgtPtK00weDlCaRs3R2qXcbNgVhleMk4IwnF8D1695AenU1LwHj0JLkCjxgNFfiWAFEPH9aEXAMPLExA="",
  "Authorization": "AWS4-HMAC-SHA256 Credential=XXXXXXXXXXXXXXXXXX/20200401/
us-east-1/appsync/aws4_request, SignedHeaders=accept;content-
encoding;content-type;host;x-amz-date;x-amz-security-token,
  Signature=83EXAMPLEbcc1fe3ee69f75cd5ebbf4cb4f150e4f99cec869f149c5EXAMPLEdc"
}
```

Cabeçalhos via string de consulta

Primeiro, um objeto JSON contendo o host (endpoint do AWS AppSync GraphQL) e os outros cabeçalhos de autorização é convertido em uma string. Em seguida, essa string é codificada usando a codificação base64. A string codificada em base64 resultante é adicionada ao WebSocket URL como um parâmetro de consulta chamado `header`. O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?
header=eyJBdXRob3JpemF0aW9uIjoizXlKcmFXUWlPaUpqYkc1eGIzQTVlVzVNSzA5UVlYSXJNVEpIV0VGTFNYQm11VTVX
```

É importante observar que, além do objeto de cabeçalho codificado em base64, um objeto JSON vazio `{}` também é codificado em base64 e incluído como um parâmetro de consulta separado nomeado na URL. `payload` WebSocket

Cabeçalhos via **Sec-WebSocket-Protocol**

Um objeto JSON contendo o host e os outros cabeçalhos de autorização é convertido em uma string e depois codificado usando a codificação Base64URL. A string codificada em Base64URL

resultante é prefixada com. header - Essa string prefixada é então usada como um novo subprotocolo, além do Sec-WebSocket-Protocol cabeçalho, ao graphql-ws estabelecer a WebSocket conexão com o endpoint AWS AppSync em tempo real.

O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

O Sec-WebSocket-Protocol cabeçalho contém o seguinte valor:

```
"sec-websocket-protocol" : ["graphql-ws", "header-  
ew0KICAiYWNjZXB0IjogImFwcGxpY2F0aW9uL2pzY24sIHRleHQvamF2YXNjcm1wdCIuDQogICJjb250ZW50LWVuY29kaW50"]
```

Cabeçalhos via cabeçalhos HTTP padrão

Nesse método, o host e as outras informações de autorização são transmitidas usando cabeçalhos HTTP padrão ao estabelecer a WebSocket conexão com o endpoint AWS AppSync em tempo real.

O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Os cabeçalhos da solicitação incluiriam o seguinte:

```
"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":  
  "AgEXAMPLEZ2luX2VjEAoaDmFwLXNvdXR0ZWFEXAMPLECwRQIgAh97Cljq7w0PL8KsxP3YtDuyC/9hAj8PhJ7Fvf38SgoC  
  +  
  +pEagWCveZUjKE0zyUhBEXAMPLEjj/////////8BEXAMPLExODk2NDgyNzg1NSIMo1mWnpESWUoYw4BkKqEFS1m3DXuL8  
  +ZbVc4JKjDP4vUCKNR6Le9C9pZp9PsW0NoFy3vLBUDAXEXAMPLE0VG8feXfiEEA+1khgFK/  
  wEtwR+9zF7NaMMmSe07wN2gG2tH0eKMEXAMPLEQX+sMbytQo8iepP9PZ0z1ZsSFb/  
  dP5Q8hk6YEXAMPLEYcKZsTkDAq2uKFQ8mYUVA9EtQnNRiFLEY83aKvG/tqLWNnG1SNVx7SMcfovkFDqQamm  
  +88y10wwAEYK7qcocX6Z7GGcaYuIfGpaX2MCCELeQvZ+8WxEgOnIfz7GYvsYNjLZSaRnV4G  
  +ILY1F0QNW64S9Nvj  
  +BwDg3ht2CrNvpwjVY1j9U3nmxE0UG5ne83LL5hhqMpm25kmL7enVgw2kQzmU2id4IKu0C/  
  WaoDRu02F5zE63vJbxN8AYs7338+4B4Hb6BZ60Ugg96Q15RA41/  
  gIqxaVPxyTpDfTU5GfSLxocdYeniqqpFMtZG2n9d0u7GsQNCfKNCg3qDZm4tDo8tZbuym0a2VcF2E5hFEgXBa  
  +XLJCfXi/770qAEjP0x7Qdk3B43p8KG/BaioP5RsV8zBGvH1zAgyPha2rN70/
```

```
tT13yrmPd5QYefwzexjKrV4mWIuRg8NTHYSZJUaeyCwTom80VFUJXG
+GYUyv5W22aBcnoRGiCiKEYTL0kgXecdKFTHmcIAejQ9We1r0a196Kq87w5KNMckcCGFnwBNFLmfmbpNqT6rUBxss3X5nt
aox0FtHX21eF6qIGT8j1z+l2opU+ggwUgkhUUgCH2TfqBj+MLMVVvpqgJsPKt582caFKArIFiv0
+9QupxLnEH2hz04TMTfnU6bQC6z1buVe7h
+t0Lnh1YPFsLQ88anib/7TTC8k9DsBTq0ASe8R2GbSEsm09qbbMwgEaYUh0KtGeyQsSJdhSk6XxXThrWL9EnwBCXDkICMqd
+WgtPtK00weDlCaRs3R2qXcbNgVhleMk4IWnF8D1695AenU1LwHj0JLkCjxgNFfiWAFEPH9aEXAMPLExA==" ,
"Authorization": "AWS4-HMAC-SHA256 Credential=XXXXXXXXXXXXXXXXXXXX/20200401/
us-east-1/appsycn/aws4_request, SignedHeaders=accept;content-
encoding;content-type;host;x-amz-date;x-amz-security-token,
Signature=83EXAMPLEebcc1fe3ee69f75cd5ebbf4cb4f150e4f99cec869f149c5EXAMPLEdc"
```

Para assinar a solicitação usando um domínio personalizado:

```
{
  url: "https://api.example.com/graphql/connect",
  data: "{}",
  method: "POST",
  headers: {
    "accept": "application/json, text/javascript",
    "content-encoding": "amz-1.0",
    "content-type": "application/json; charset=UTF-8",
  }
}
```

Exemplo

```
{
  "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":
    "AgEXAMPLEZ2luX2VjEAoaDmFwLXNvdXR0ZWFEXAMPLECwRQIgAh97Cljq7w0PL8KsxP3YtDuyC/9hAj8PhJ7Fvf38SgoC
    +
    +pEagWCveZUjKE0zyUhBEXAMPLEjj/////////8BEXAMPLEx0Dk2NDgyNzg1NSIMo1mWnpESWUoYw4BkKqEFS1m3DXuL8
    +ZbVc4JKjDP4vUCKNR6Le9C9pZp9PsW0NoFy3vLBUDAXEXAMPLE0VG8feXfiEEA+1khgFK/
    wEtWr+9zF7NaMMmSe07wN2gG2tH0eKMEXAMPLEEQX+sMbytQo8iepP9PZ0z1ZsSFb/
    dP5Q8hk6YEXAMPLEYcKZsTkDAq2uKFQ8mYUVA9EtQnNRiFLEY83aKvG/tqLWNnG1SNVx7SMcfovkFDqQamm
    +88y10wwAEYK7qcocex6Z7GGcaYuIfGpaX2MCCELeQvZ+8WxEg0nIfz7GYvsYNjLZSaRnV4G
    +ILY1F0QNW64S9Nvj
    +BwDg3ht2CrNvpwjVY1j9U3nmxE0UG5ne83LL5hhqMpm25kmL7enVgw2kQzmU2id4IKu0C/
    WaoDRu02F5zE63vJbxN8AYs7338+4B4HBb6BZ60Ugg96Q15RA41/
```

```
gIqxaVPxyTpDfTU5GfSLxocdYeniqqpFMtZG2n9d0u7GsQNCfKNCg3qDZm4tDo8tZbuym0a2VcF2E5hFEgXBa
+XLJCfXi/770qAEjP0x7Qdk3B43p8KG/BaioP5RsV8zBGvH1zAgyPha2rN70/
tT13yrmPd5QYefwzexjKrV4mWIuRg8NTHYSZJUaeyCwTom80VFUJXG
+GYTUyv5W22aBcnoRgiCiKEYTL0kgXecdKFTHmcIAejQ9We1r0a196Kq87w5KNMckcCGFnwBNFLmfmbpNqT6rUBxxs3X5nt
aox0FtHX21eF6qIGT8j1z+12opU+ggwUgkhUUgCH2TfqBj+MLMVVvpqgJsPKt582caFKArIFiv0
+9QupxLnEH2hz04TMTfnU6bQC6z1buVe7h
+t0Lnh1YPFsLQ88anib/7TTC8k9DsBTq0ASe8R2GbSEsm09qbbMwgEaYUh0KtGeyQsSJdhSk6XxXThrWL9EnwBCXDkICMqQ
+WgtPtK00weDlCaRs3R2qXcbNgVhleMk4IwnF8D1695AenU1LwHj0JLkCjxgNFfiWAFEPH9aEXAMPLExA="",
  "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=83EXAMPLEebcc1fe3ee69f75cd5ebbf4cb4f150e4f99cec869f149c5EXAMPLEdc"
}
```

URL de solicitação com sequência de caracteres de consulta

```
wss://api.example.com/graphql?
header=eyJEXAMPLEHQiOiJhcHBsaWNhdGlvbi9qc29uLCB0ZXh0L2phdmFEXAMPLEQiLCJjb250ZW50LWVuY29kaW5nIjoE
```

Note

Uma WebSocket conexão pode ter várias assinaturas (mesmo com modos de autenticação diferentes). Uma forma de implementar isso é criar uma WebSocket conexão para a primeira assinatura e fechá-la quando a última assinatura não for registrada. Você pode otimizar isso aguardando alguns segundos antes de fechar a WebSocket conexão, caso o aplicativo seja inscrito imediatamente após o cancelamento do registro da última assinatura. Para um exemplo de aplicativo móvel, ao mudar de uma tela para outra, no evento de desmontagem, ele interrompe uma assinatura e, no evento de montagem, ele inicia uma assinatura diferente.

Autorização do Lambda

Cabeçalho de autorização do Lambda

Conteúdo do cabeçalho

- "Authorization": <string>: O valor que é passado como authorizationToken.
- "host": <string>: o host do endpoint do AWS AppSync GraphQL ou do seu nome de domínio personalizado.

Exemplo

```
{
  "Authorization": "M0UzQzM1MkQtMkI0Ni00OTZCLUI1NkQtMUM0MTQ0QjVBRTczCkI1REEzRTIxLTk5NzItNDJENiBQ",
  "host": "example1234567890000.appsync-api.us-east-1.amazonaws.com"
}
```

Cabeçalhos via string de consulta

Primeiro, um objeto JSON contendo o `host` e o `Authorization` é convertido em uma string. Em seguida, essa string é codificada usando a codificação `base64`. A string codificada em `base64` resultante é adicionada como um parâmetro de consulta nomeado `header` ao WebSocket URL para estabelecer a conexão com o AWS AppSync endpoint em tempo real. O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql?  
header=eyJBdXRob3JpemF0aw9uIjoizXlKcmFXUWlPaUpqYkc1eGIzQTVlVzVNSzA5UVlYSXJNVEpIV0VGTfNYQmllVTXV
```

É importante observar que, além do objeto de cabeçalho codificado em base64, um objeto JSON vazio {} também é codificado em base64 e incluído como um parâmetro de consulta separado nomeado na URL. `payload WebSocket`

Cabeçalhos via **Sec-WebSocket-Protocol**

Um objeto JSON contendo o host e o Authorization é convertido em uma string e depois codificado usando a codificação Base64URL. A string codificada em Base64URL resultante é prefixada com. header- Essa string prefixada é então usada como um novo subprotocolo, além do Sec-WebSocket-Protocol cabeçalho, ao graphql-ws estabelecer a WebSocket conexão com o endpoint AWS AppSync em tempo real.

O URL da solicitação resultante tem o seguinte formato:

wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql

O Sec-WebSocket-Protocol cabeçalho contém o seguinte valor:

```
"sec-websocket-protocol" : ["graphql-ws", "header-ewogICAqImhvc3QiOiJleGFtcGxlMTIzNDU2Nzg5MDAwMC5hcHBzeW5jLWFwaS51cy1lYXN0LTEuYW1hem9uYXdzLmNvbSI
```

Cabeçalhos via cabeçalhos HTTP padrão

Nesse método, as informações do host e da autorização são transmitidas usando cabeçalhos HTTP padrão ao estabelecer a WebSocket conexão com o endpoint AWS AppSync em tempo real. O URL da solicitação resultante tem o seguinte formato:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Os cabeçalhos da solicitação incluiriam o seguinte:

```
"sec-websocket-protocol" : ["graphql-ws"]
"Authorization": "eyJEXAMPLEiJjbG5xb3A5eW5MK09QYXIrMTJHWEFLSXBieU5WNHhsQjEXAMPLEnM2WldvPSIsImFsZy9zEE2DJH7sH0l2zxYi7f-SmEGoh2AD8emxQRYajByz-rE4Jh0Q0ymN2Ys-ZIKMpVBTPgu-TMWdyOHhDUMUj20P82yeZ3w1ZAtr_gM4LzjXUXmI_K2yGjuXfXTaa1mvQEBG0mQfVd7SfwXB-jcv4RYVi6j25qgow9Ew52ufurPqaK-3WAKG32KpV8J4-Wejq8t0c-yA7sb8EnB551b7TU93uKRiVVK3E55Nk5ADPoam_WYE45i3s5qVAP_-InW75NUo0CGTsS8YWMfb6ecHYJ-1j-bzA27zaT9VjctXn9byNFZmEXAMPLExw",
"host": "example1234567890000.appsync-api.us-east-1.amazonaws.com"
```

WebSocketOperação em tempo real

Depois de iniciar um WebSocket handshake bem-sucedido com AWS AppSync, o cliente deve enviar uma mensagem subsequente à qual se conectar AWS AppSync para diferentes operações. Essas mensagens exigem os seguintes dados:

- **type**: O tipo da operação.
- **id**: Um identificador exclusivo para a assinatura. Recomendamos usar um UUID para esse fim.
- **payload**: A carga associada dependendo do tipo de operação.

O campo **type** é o único campo obrigatório; os campos **id** e **payload** são opcionais.

Sequência de eventos

Para iniciar, estabelecer, registrar e processar com êxito a solicitação de assinatura, o cliente deve percorrer a seguinte sequência:

1. Inicializar conexão (`connection_init`)
2. Confirmação de conexão (`connection_ack`)
3. Registro de assinatura (`start`)

4. Confirmação de assinatura (`start_ack`)
5. Processamento de assinatura (`data`)
6. Cancelamento de registro de assinatura (`stop`)

Mensagem de inicialização de conexão

(Opcional) Depois de um aperto de mão bem-sucedido, o cliente pode enviar a `connection_init` mensagem para começar a se comunicar com o endpoint AWS AppSync em tempo real. A mensagem é uma string obtida pela string do objeto JSON da seguinte forma:

```
{ "type": "connection_init" }
```

Mensagem de confirmação de conexão

Depois de enviar a mensagem `connection_init`, o cliente deve aguardar a mensagem `connection_ack`. Todas as mensagens enviadas antes de receber `connection_ack` são ignoradas. A mensagem deve ser a seguinte:

```
{
  "type": "connection_ack",
  "payload": {
    // Time in milliseconds waiting for ka message before the client should terminate
    the WebSocket connection
    "connectionTimeoutMs": 300000
  }
}
```

Mensagem keep-alive

Além da mensagem de confirmação de conexão, o cliente recebe mensagens keep-alive periodicamente. Se o cliente não receber uma mensagem de manutenção de atividade dentro do período de tempo limite da conexão, ele deverá fechar a conexão. AWS AppSync continua enviando essas mensagens e atendendo às assinaturas registradas até encerrar a conexão automaticamente (após 24 horas). As mensagens keep-alive são heartbeats e não precisam ser confirmadas pelo cliente.

```
{ "type": "ka" }
```

Mensagem de registro de assinatura

Depois que o cliente recebe uma `connection_ack` mensagem, ele pode enviar mensagens de registro de assinatura para AWS AppSync. Esse tipo de mensagem é um objeto JSON em string que contém os seguintes campos:

- `"id": <string>`: O nome da assinatura. Esse ID deve ser exclusivo para cada assinatura, caso contrário, o servidor retornará um erro indicando que o ID da assinatura está duplicado.
- `"type": "start"`: Um parâmetro `<string>` constante.
- `"payload": <Object>`: Um objeto que contém as informações relevantes para a assinatura.
 - `"data": <string>`: Um objeto JSON em string que contém variáveis e consulta do GraphQL.
 - `"query": <string>`: Uma operação do GraphQL.
 - `"variables": <Object>`: Um objeto que contém as variáveis para a consulta.
 - `"extensions": <Object>`: Um objeto que contém um objeto de autorização.
- `"authorization": <Object>`: Um objeto que contém os campos necessários para autorização.

Objeto de autorização para registro de assinatura

As mesmas regras na seção [Formato do parâmetro de cabeçalho com base no modo de autorização AWS AppSync da API](#) se aplicam ao objeto de autorização. A única exceção é o IAM, no qual as informações de assinatura do SigV4 são um pouco diferentes. Para obter mais detalhes, consulte o exemplo do IAM.

Exemplo usando grupos de usuários do Amazon Cognito:

```
{
  "id": "ee849ef0-cf23-4cb8-9fcb-152ae4fd1e69",
  "payload": {
    "data": "{\"query\":\"subscription onCreateMessage {\n\n onCreateMessage {\n\n\n__typename\n message\n }\n\n }\", \"variables\": {}}",
    "extensions": {
      "authorization": {
        "Authorization":
          "eyJEXAMPLEiJjbG5xb3A5eW5MK09QYXIrMTJEXAMPLEBieU5WNHhsQjhPVW9YMNm2WldvPSIsImFsZyI6I1EXAMPLEn0.e
          qTCtrYeboUJ4luRSTPXaNewNeEXAMPLE14C6sfg05t00f0MpiUwj9k19gtNCCMqoSsjtQoUweFnH4JYa5EXAMPLEVx0yQEQ
          RWvW7yQU3sNQRLEXAMPLEcd0yufBiCYs3dfQxTTdvR1B6Wz6CD781fNeKqfzzUn2beMoup2h6EXAMPLE4ow8cUPUPvG0DzR
          "host": "example1234567890000.appsync-api.us-east-1.amazonaws.com"
```

```

    }
  }
},
"type": "start"
}

```

Exemplo usando o 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\":
          \"AgEXAMPLEZ2luX2VjEAoaDmFwLXNvdXR0ZWFEEXAMPLEcwRQIgAh97Cljq7w0PL8KsxP3YtDuyC/9hAj8PhJ7Fvf38SgoC
          +
          +pEagWCveZUjKE0zyUhBEXAMPLEjj//////////8BEXAMPLEx0Dk2NDgyNzg1NSIMo1mWnpESWUoYw4BkKqEFSrm3DXuL8
          +ZbVc4JKjDP4vUCKNR6Le9C9pZp9Psw0NoFy3vLBUDAXEXAMPLE0VG8feXfiEEA+1khgFK/
          wEtwR+9zF7NaMMmSe07wN2gG2tH0eKMEXAMPLEEQX+sMbytQo8iepP9PZ0z1ZsSFb/
          dP5Q8hk6YEXAMPLEYcKZsTkDAq2uKFQ8mYUVA9EtQnNRiFLEY83aKvG/tqLWNnG1SNVx7SMcfovkFDqQamm
          +88y10wwAEYK7qcocex6Z7GGcaYuIfGpaX2MCCELeQvZ+8WxEg0nIfz7GYvsYNjLZSaRnV4G
          +ILY1F0QNW64S9Nvj
          +BwDg3ht2CrNvpwVY1j9U3nmxE0UG5ne83LL5hhqMpm25kmL7enVgw2kQzmU2id4IKu0C/
          WaoDRu02F5zE63vJbxN8AYs7338+4B4HBB6BZ60Ugg96Q15RA41/
          gIqxaVPxyTpDfTU5GfSLxocdYeniqqpFMtZG2n9d0u7GsQNcFkNcG3qDZm4tDo8tZbuym0a2VcF2E5hFEgXBa
          +XLJCfXi/770qAEjP0x7Qdk3B43p8KG/BaioP5RsV8zBGvH1zAgyPha2rN70/
          tT13yrmPd5QYEFwzexjKrV4mWIuRg8NTHYSZJUaeyCwTom80VFUJXG
          +GYTUyv5W22aBcnoRGiCiKEYTL0kgXecdKFTHmcIAejQ9Welr0a196Kq87w5KNMckcCGFnwBNFLmfnbpNqT6rUBxss3X5nt
          aox0FtHX21eF6qIGT8j1z+l2opU+ggwUgkhUUgCH2TfqBj+MLMVvpvgQJsPKt582caFKArIFIv0
          +9QupxLnEH2hz04TMTfnU6bQC6z1buVe7h
          +t0Lnh1YPFsLQ88anib/7TTC8k9DsBTq0ASE8R2GbSEsm09qbbMwgEaYUh0KtGeyQsSJdhSk6XxXThrWL9EnwBCXDkICMqd
          +WgtPtK00weDlCaRs3R2qXcbNgVhleMk4IwnF8D1695AenU1LwHj0JLkCjxgNFfiWAFEPH9aEXAMPLExA==\",
            \"Authorization\": \"AWS4-HMAC-SHA256 Credential=XXXXXXXXXXXXXXXXXXXX/20200401/
us-east-1/appsycn/aws4_request, SignedHeaders=accept;content-
encoding;content-type;host;x-amz-date;x-amz-security-token,
Signature=b90131a61a7c4318e1c35ead5dbfdeb46339a7585bbdbceceaff51f4022eb1fd\",
            \"content-encoding\": \"amz-1.0\",
            \"host\": \"example1234567890000.appsycn-api.us-east-1.amazonaws.com\",
            \"x-amz-date\": \"20200401T001010Z\"

```

```
    }
  }
},
"type": "start"
}
```

Exemplo do uso de um nome de domínio personalizado:

```
{
  "id": "key-cf23-4cb8-9fcb-152ae4fd1e69",
  "payload": {
    "data": "{\"query\":\"subscription onCreateMessage {\\n  onCreateMessage {\\n    __typename\\n  }\\n }\\n }\", \"variables\":{}}",
    "extensions": {
      "authorization": {
        "x-api-key": "da2-12345678901234567890123456",
        "host": "api.example.com"
      }
    }
  },
  "type": "start"
}
```

A assinatura do SigV4 não exige que `/connect` seja anexado ao URL, e a operação do GraphQL em string do JSON substitui `data`. Veja a seguir um exemplo de uma solicitação de assinatura do SigV4:

```
{
  url: "https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql",
  data: "{\"query\":\"subscription onCreateMessage {\n onCreateMessage {\n __typename\n message\n }\n}\",\"variables\":{}}",
  method: "POST",
  headers: {
    "accept": "application/json, text/javascript",
    "content-encoding": "amz-1.0",
    "content-type": "application/json; charset=UTF-8",
  }
}
```

Mensagem de confirmação de assinatura

Depois de enviar a mensagem de início da assinatura, o cliente deve AWS AppSync aguardar o envio da `start_ack` mensagem. A mensagem `start_ack` indica que a assinatura foi bem-sucedida.

Exemplo de confirmação de assinatura:

```
{
  "type": "start_ack",
  "id": "eEXAMPLE-cf23-1234-5678-152EXAMPLE69"
}
```

Mensagem de erro

Se o início da conexão ou o registro da assinatura falhar, ou se uma assinatura for encerrada pelo servidor, o servidor enviará uma mensagem de erro ao cliente. Se o erro ocorrer durante o tempo de inicialização da conexão, a conexão será fechada pelo servidor.

- `"type": "error"`: Um parâmetro `<string>` constante.
- `"id": <string>`: O ID da assinatura registrada correspondente, se relevante.
- `"payload" <Object>`: Um objeto que contém as informações de erro correspondentes.

Exemplo:

```
{
  "type": "error",
  "payload": {
    "errors": [
      {
        "errorType": "LimitExceededError",
        "message": "Rate limit exceeded"
      }
    ]
  }
}
```

Processar mensagens de dados

Quando um cliente envia uma mutação, AWS AppSync identifica todos os assinantes interessados nela e envia uma "type": "data" mensagem para cada um usando a assinatura correspondente da operação id de assinatura. "start" Espera-se que o cliente mantenha o controle do id de assinatura enviado para que, quando uma mensagem de dados for recebida, o cliente possa combiná-lo com a assinatura correspondente.

- "type": "data": Um parâmetro <string> constante.
- "id": <string>: O ID da assinatura registrada correspondente.
- "payload" <Object>: Um objeto que contém as informações da assinatura.

Exemplo:

```
{
  "type": "data",
  "id": "ee849ef0-cf23-4cb8-9fcb-152ae4fd1e69",
  "payload": {
    "data": {
      "onCreateMessage": {
        "__typename": "Message",
        "message": "test"
      }
    }
  }
}
```

Mensagem de cancelamento de registro de assinatura

Quando o aplicativo deseja parar de receber os eventos de assinatura, o cliente deve enviar uma mensagem com o seguinte objeto JSON em string:

- "type": "stop": Um parâmetro <string> constante.
- "id": <string>: O ID da assinatura com registro cancelado.

Exemplo:

```
{
  "type": "stop",
```



```
"id": "ee849ef0-cf23-4cb8-9fcb-152ae4fd1e69"
}
```

AWS AppSync envia de volta uma mensagem de confirmação com o seguinte objeto JSON stringificado:

- "type": "complete": Um parâmetro <string> constante.
- "id": <string>: O ID da assinatura com registro cancelado.

Depois que o cliente recebe a mensagem de confirmação, ele não recebe mais mensagens para essa assinatura específica.

Exemplo:

```
{
  "type": "complete",
  "id": "eEXAMPLE-cf23-1234-5678-152EXAMPLE69"
}
```

Desconectando o WebSocket

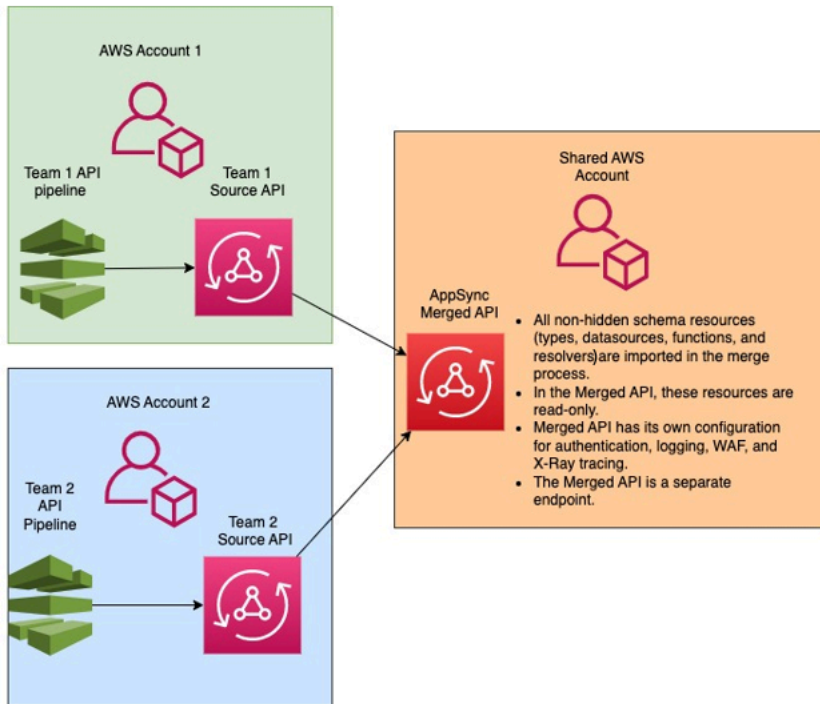
Antes de se desconectar, para evitar perda de dados, o cliente deve ter a lógica necessária para verificar se nenhuma operação está sendo realizada atualmente por meio da WebSocket conexão. Todas as assinaturas devem ser canceladas antes de se desconectar do WebSocket.

Mesclando em APIs AWS AppSync

À medida que o uso do GraphQL se expande dentro de uma organização, podem surgir compensações entre a API e a velocidade de desenvolvimento da API ease-of-use. Por um lado, as organizações adotam AWS AppSync o GraphQL para simplificar o desenvolvimento de aplicativos, oferecendo aos desenvolvedores uma API flexível que eles podem usar para acessar, manipular e combinar com segurança dados de um ou mais domínios de dados com uma única chamada de rede. Por outro lado, as equipes de uma organização responsáveis pelos diferentes domínios de dados combinados em um único endpoint da API GraphQL podem querer a capacidade de criar, gerenciar e implantar atualizações de API independentes umas das outras para aumentar suas velocidades de desenvolvimento.

Para resolver essa tensão, o APIs recurso AWS AppSync Merged permite que equipes de diferentes domínios de dados criem e implantem de forma independente AWS AppSync APIs (por

exemplo, esquemas, resolvedores, fontes de dados e funções do GraphQL), que podem então ser combinadas em uma única API mesclada. Isso dá às organizações a capacidade de manter uma API multidomínio simples de usar, e uma forma de as diferentes equipes que contribuem com essa API poderem fazer atualizações de API de forma rápida e independente.



Usando o Merged APIs, as organizações podem importar os recursos de várias fontes independentes AWS AppSync APIs em um único endpoint de API AWS AppSync mesclada. Para fazer isso, você AWS AppSync pode criar uma lista de fontes de AWS AppSync origem e APIs, em seguida, mesclar todos os metadados associados à fonte, APIs incluindo esquema, tipos, fontes de dados, resolvedores e funções, em uma nova API mesclada. AWS AppSync

Durante as mesclagens, existe a possibilidade de ocorrer um conflito de mesclagem devido a inconsistências no conteúdo dos dados da API de origem, como conflitos de nomenclatura de tipos ao combinar vários esquemas. Para casos de uso simples em que nenhuma definição na fonte está em APIs conflito, não há necessidade de modificar os esquemas da API de origem. A API mesclada simplesmente importa todos os tipos, resolvedores, fontes de dados e funções da fonte original. AWS AppSync APIs Para casos de uso complexos em que surjam conflitos, eles users/teams terão que resolvê-los por vários meios. AWS AppSync fornece aos usuários várias ferramentas e exemplos que podem reduzir os conflitos de mesclagem.

As mesclagens subsequentes configuradas em AWS AppSync propagarão as alterações feitas na fonte APIs para a API mesclada associada.

Fusão APIs e federação

Há muitas soluções e padrões na comunidade GraphQL para combinar esquemas do GraphQL e permitir a colaboração em equipe por meio de um gráfico compartilhado. AWS AppSync O Merged APIs adota uma abordagem de tempo de construção para a composição do esquema, em que APIs as fontes são combinadas em uma API mesclada separada. Uma abordagem alternativa é colocar um roteador em camadas em tempo de execução em várias fontes APIs ou subgráficos. Nessa abordagem, o roteador recebe uma solicitação, faz referência a um esquema combinado que ele mantém como metadados, constrói um plano de solicitação e, em seguida, distribui os elementos da solicitação em seus subgráficos/servidores subjacentes. A tabela a seguir compara a abordagem de tempo de construção da API AWS AppSync mesclada com abordagens de tempo de execução baseadas em roteador para a composição do esquema GraphQL:

Recurso	AppSync API mesclada	Soluções baseadas em roteadores
Subgráficos gerenciados de forma independente	Sim	Sim
Subgráficos endereçáveis de forma independente	Sim	Sim
Composição automatizada do esquema	Sim	Sim
Deteção automatizada de conflitos	Sim	Sim
Resolução de conflitos por meio de diretivas de esquema	Sim	Sim
Servidores de subgráficos compatíveis	AWS AppSync*	Varia

Complexidade da rede	Uma API única e mesclada significa que não há saltos extras na rede.	A arquitetura multicamada requer planejamento e delegação de consultas, análise e serialização/deserialização de subconsultas e resolvedores de referência em subgráficos para realizar junções.
Suporte de observabilidade	Monitoramento, registro e rastreamento integrados. Um único servidor de API mesclado significa depuração simplificada.	Build-your-own observabilidade no roteador e em todos os servidores de subgráficos associados. Depuração complexa em sistemas distribuídos.
Suporte de autorização	Suporte integrado para vários modos de autorização.	Build-your-own regras de autorização.
Segurança entre contas	Suporte integrado para associações de contas entre AWS nuvens.	Build-your-own modelo de segurança.
Suporte para assinaturas	Sim	Não

* O AWS AppSync mesclado só APIs pode ser associado à AWS AppSync fonte APIs. Se precisar de suporte para composição de esquemas entre AWS AppSync e sem AWS AppSync subgráficos, você pode conectar um ou mais GraphQL and/or APIs Merged em uma solução AWS AppSync baseada em roteador. [Por exemplo, consulte o blog de referência para adicionar AWS AppSync APIs como subgráfico usando uma arquitetura baseada em roteador com o Apollo Federation v2: Apollo GraphQL Federation with. AWS AppSync](#)

Tópicos

- [Resolução de conflitos de API mesclada](#)
- [Configurar esquemas](#)
- [Configurar modos de autorização](#)

- [Configurar perfis de execução](#)
- [Configurando contas cruzadas mescladas usando APIs AWS RAM](#)
- [Mesclar](#)
- [Suporte adicional para Merged APIs](#)
- [Limitações de APIs mescladas](#)
- [Criando mesclado APIs](#)

Resolução de conflitos de API mesclada

No caso de um conflito de mesclagem, AWS AppSync fornece aos usuários várias ferramentas e exemplos para ajudar a solucionar o (s) problema (s).

Diretivas de esquema de API mescladas

AWS AppSync introduziu várias diretivas do GraphQL que podem ser usadas para reduzir ou resolver conflitos na fonte: APIs

- **@canonical:** essa diretiva define a precedência de tipos/campos com nomes e dados semelhantes. Se duas ou mais fontes APIs tiverem o mesmo tipo ou campo do GraphQL, uma delas APIs poderá anotar seu tipo ou campo como canônico, o que será priorizado durante a mesclagem. Os conflitos types/fields que não estão anotados com esta diretiva em outra fonte APIs são ignorados quando mesclados.
- **@hidden:** Esta diretiva encapsula certos types/fields para removê-los do processo de fusão. As equipes podem querer remover ou ocultar tipos ou operações específicos na API de origem para que somente clientes internos possam acessar dados digitados específicos. Com essa diretiva anexada, os tipos ou campos não são mesclados na API mesclada.
- **@renamed:** Essa diretiva altera os nomes de types/fields para reduzir os conflitos de nomenclatura. Há situações em que diferentes APIs têm o mesmo tipo ou nome de campo. No entanto, todos eles precisam estar disponíveis no esquema mesclado. Uma maneira simples de incluir todos eles na API mesclada é renomear o campo para algo semelhante, mas diferente.

Para mostrar o esquema de utilitário fornecido pelas diretivas, considere o seguinte exemplo:

Neste exemplo, vamos supor que queremos mesclar duas fontes APIs. Temos dois esquemas que criam e recuperam postagens (por exemplo, seção de comentários ou postagens em mídias

sociais). Supondo que os tipos e campos sejam muito semelhantes, há uma grande chance de conflito durante uma operação de mesclagem. Os trechos abaixo mostram os tipos e campos de cada esquema.

O primeiro arquivo, chamado `Source1.graphql`, é um esquema do GraphQL que permite ao usuário criar Posts usando a mutação `putPost`. Cada `Post` contém um título e um ID. O ID é usado para referenciar as informações do autor ou do `User` (e-mail e endereço) e a `Message`, ou a carga útil (conteúdo). O tipo `User` é anotado com a tag `@canonical`.

```
# This snippet represents a file called Source1.graphql

type Mutation {
  putPost(id: ID!, title: String!): Post
}

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

type Message {
  id: ID!
  content: String
}

type User @canonical {
  id: ID!
  email: String!
  address: String!
}

type Query {
  singlePost(id: ID!): Post
  getMessage(id: ID!): Message
}
```

O segundo arquivo, chamado `Source2.graphql`, é um esquema do GraphQL que funciona muito semelhante ao `Source1.graphql`. No entanto, observe que os campos de cada tipo são diferentes. Ao mesclar esses dois esquemas, haverá conflitos de mesclagem devido a essas diferenças.

Além disso, observe como o `Source2.graphql` também contém várias diretivas para reduzir esses conflitos. O tipo `Post` é anotado com uma tag `@hidden` para se ofuscar durante a operação de

mesclagem. O tipo `Message` é anotado com a tag `@renamed` para modificar o nome do tipo `ChatMessage` no caso de um conflito de nomenclatura com outro tipo `Message`.

```
# This snippet represents a file called Source2.graphql

type Post @hidden {
  id: ID!
  title: String!
  internalSecret: String!
}

type Message @renamed(to: "ChatMessage") {
  id: ID!
  chatId: ID!
  from: User!
  to: User!
}

# Stub user so that we can link the canonical definition from Source1
type User {
  id: ID!
}

type Query {
  getPost(id: ID!): Post
  getMessage(id: ID!): Message @renamed(to: "getChatMessage")
}
```

Quando a mesclagem ocorrer, o resultado produzirá o arquivo `MergedSchema.graphql`:

```
# This snippet represents a file called MergedSchema.graphql

type Mutation {
  putPost(id: ID!, title: String!): Post
}

# Post from Source2 was hidden so only uses the Source1 definition.
type Post {
  id: ID!
  title: String!
}

# Renamed from Message to resolve the conflict
```

```
type ChatMessage {
  id: ID!
  chatId: ID!
  from: User!
  to: User!
}

type Message {
  id: ID!
  content: String
}

# Canonical definition from Source1
type User {
  id: ID!
  email: String!
  address: String!
}

type Query {
  singlePost(id: ID!): Post
  getMessage(id: ID!): Message

  # Renamed from getMessage
  getChatMessage(id: ID!): ChatMessage
}
```

Várias coisas ocorreram na mesclagem:

- O tipo `User` de `Source1.graphql` foi priorizado em relação ao de `Source2.graphql` devido à `User` anotação `@canonical`.
- O `Message` tipo do `Source1.graphql` foi incluído na mesclagem. No entanto, o `Message` do `source2.graphql` teve um conflito de nomenclatura. Devido à anotação `@renamed`, ele também foi incluído na mesclagem, mas com o nome alternativo `ChatMessage`.
- O tipo `Post` de `Source1.graphql` foi incluído, mas o tipo `Post` de `Source2.graphql` não. Normalmente, haveria um conflito nesse tipo, mas como o tipo `Post` de `Source2.graphql` tinha uma anotação `@hidden`, seus dados foram ofuscados e não incluídos na mesclagem. Isso não resultou em conflitos.

- O tipo Query foi atualizado para incluir o conteúdo dos dois arquivos. No entanto, uma consulta GetMessage foi renomeada para GetChatMessage devido à diretiva. Isso resolveu o conflito de nomenclatura entre as duas consultas com o mesmo nome.

Também existe o caso de nenhuma diretiva ser adicionada a um tipo conflitante. Nesse caso, o tipo mesclado incluirá a união de todos os campos de todas as definições de origem desse tipo. Por exemplo, considere o exemplo a seguir:

Esse esquema, chamado Source1.graphql, permite criar e recuperar Posts. A configuração é semelhante à do exemplo anterior, mas com menos informações.

```
# 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
}
```

Esse esquema, chamado Source2.graphql, permite criar e recuperar Reviews (por exemplo, classificação de filmes ou resenhas de restaurantes). As Reviews estão associadas ao Post do mesmo valor de ID. Juntos, eles contêm o título, o ID da postagem e a mensagem da payload da postagem de avaliação completa.

Ao mesclar, haverá um conflito entre os dois tipos de Post. Como não há anotações para resolver esse problema, o comportamento padrão é realizar uma operação de união nos tipos conflitantes.

```
# 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
  }
```

Quando a mesclagem ocorrer, o resultado produzirá o arquivo `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
}
```

Várias coisas ocorreram na mesclagem:

- O tipo `Mutation` não enfrentou conflitos e foi mesclado.

- Os campos do tipo Post foram combinados por meio da operação de união. Observe como a união entre os dois produziu um único id, um title e um único reviews.
- O tipo Review não enfrentou conflitos e foi mesclado.
- O tipo Query não enfrentou conflitos e foi mesclado.

Gerenciar resolvedores em tipos compartilhados

No exemplo acima, considere o caso em que o `Source1.graphql` configurou um resolvedor de unidades em `Query.getPost`, que usa uma fonte de dados do DynamoDB chamada `PostDatasource`. Esse resolvedor retornará o `id` e `title` de um tipo `Post`. Agora, considere que o `Source2.graphql` configurou um resolvedor de pipeline em `Post.reviews`, que executa duas funções. `Function1` tem uma fonte `None` de dados anexada para realizar verificações de autorização personalizadas. `Function2` tem uma fonte de dados do DynamoDB anexada para consultar a tabela. `reviews`

```
query GetPostQuery {
  getPost(id: "1") {
    id,
    title,
    reviews
  }
}
```

Quando a consulta acima é executada por um cliente no endpoint da API mesclada, o AWS AppSync serviço primeiro executa o resolvedor de unidades para `Query.getPost` from `Source1`, que chama `PostDatasource` e retorna os dados do DynamoDB. Em seguida, ele executa o resolvedor de pipeline `Post.reviews`, no qual `Function1` executa a lógica de autorização personalizada e `Function2` retorna as avaliações fornecidas ao `id` encontradas em `$context.source`. O serviço processa a solicitação como uma única execução do GraphQL, e essa solicitação simples exigirá apenas um único token de solicitação.

Gerenciar conflitos de resolvedor em tipos compartilhados

Considere o seguinte caso em que também implementamos um resolvedor em `Query.getPost` para fornecer vários campos ao mesmo tempo além do resolvedor de campo em `Source2`. `Source1.graphql` pode parecer da seguinte forma:

```
# This snippet represents a file called Source1.graphql
```

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

type Query {
  getPost(id: ID!): Post
}
```

Source2.graphql pode parecer da seguinte forma:

```
# This snippet represents a file called Source2.graphql

type Post {
  id: ID!
  content: String!
  contentHash: String!
  author: String!
}

type Query {
  getPost(id: ID!): Post
}
```

A tentativa de mesclar esses dois esquemas gerará um erro de mesclagem porque o AWS AppSync Merged APIs não permite que vários resolvedores de origem sejam anexados ao mesmo campo. Para resolver esse conflito, você pode implementar um padrão de resolvedor de campo que exigiria que o Source2.graphql adicionasse um tipo separado que definirá os campos que ele possui do tipo Post. No exemplo a seguir, adicionamos um tipo chamado PostInfo, que contém os campos de conteúdo e autor que serão resolvidos pelo Source2.graphql. O Source1.graphql implementará o resolvedor anexado a Query.getPost, enquanto o Source2.graphql agora anexará um resolvedor a Post.postInfo para garantir que todos os dados possam ser recuperados com sucesso:

```
type Post {
  id: ID!
  postInfo: PostInfo
}

type PostInfo {
```

```
    content: String!
    contentHash: String!
    author: String!
  }

  type Query {
    getPost(id: ID!): Post
  }
```

Embora a resolução desse conflito exija que os esquemas da API de origem sejam reescritos e, potencialmente, que os clientes alterem suas consultas, a vantagem dessa abordagem é que a propriedade dos resolvedores mesclados permanece clara entre todas as equipes de origem.

Configurar esquemas

Duas partes são responsáveis por configurar os esquemas para criar uma API mesclada:

- Proprietários da API mesclada - Os proprietários da API mesclada devem definir a lógica de autorização e as configurações avançadas da API mesclada, como registro em log, rastreamento, armazenamento em cache e suporte ao WAF.
- Proprietários da API de origem associada - Os proprietários da API associada devem configurar os esquemas, os resolvedores e as fontes de dados que compõem a API mesclada.

Como o esquema da API mesclada é criado a partir dos esquemas da fonte associada APIs, ele é somente para leitura. Isso significa que as alterações no esquema devem ser iniciadas em sua fonte APIs. No AWS AppSync console, você pode alternar entre o esquema mesclado e os esquemas individuais da fonte APIs incluídos na API mesclada usando a lista suspensa acima da janela Esquema.

Configurar modos de autorização

Vários modos de autorização estão disponíveis para proteger sua API mesclada. Para saber mais sobre os modos de autorização em AWS AppSync, consulte [Autorização e autenticação](#).

Os seguintes modos de autorização estão disponíveis para uso com o Merged APIs:

- Chave de API: a estratégia de autorização mais simples. Todas as solicitações devem incluir uma chave de API no cabeçalho da solicitação `x-api-key`. As chaves de API expiradas são mantidas por 60 dias após a data de expiração.

- AWS Identity and Access Management (IAM): a estratégia de autorização AWS do IAM autoriza todas as solicitações assinadas com sigv4.
- Grupos de usuários do Amazon Cognito: autorize seus usuários por meio dos grupos de usuários do Amazon Cognito para obter um controle mais refinado.
- AWS Autorizadores Lambda: uma função sem servidor que permite autenticar e autorizar o acesso à sua API usando lógica personalizada. AWS AppSync
- OpenID Connect: esse tipo de autorização impõe tokens do OpenID Connect (OIDC) fornecidos por um serviço compatível com OIDC. O aplicativo pode aproveitar os usuários e os privilégios definidos pelo provedor de OIDC para controlar o acesso.

Os modos de autorização de uma API mesclada são configurados pelo proprietário da API mesclada. No momento de uma operação de mesclagem, a API mesclada deve incluir o modo de autorização principal configurado em uma API de origem como seu próprio modo de autorização principal ou como um modo de autorização secundário. Caso contrário, ela será incompatível e a operação de mesclagem falhará devido a um conflito. Ao usar diretivas de autenticação múltipla na origem APIs, o processo de mesclagem é capaz de mesclar automaticamente essas diretivas no endpoint unificado. Caso o modo de autorização principal da API de origem não corresponda ao modo de autorização principal da API mesclada, ele adicionará automaticamente essas diretivas de autenticação para garantir que o modo de autorização dos tipos na API de origem seja consistente.

Configurar perfis de execução

Ao criar uma API mesclada, você precisa definir um perfil de serviço. Uma função AWS de serviço é uma função de AWS Identity and Access Management (IAM) usada AWS pelos serviços para realizar tarefas em seu nome.

Nesse contexto, é necessário que sua API mesclada execute resolvedores que acessem dados das fontes de dados configuradas em sua fonte. APIs A função de serviço necessária para isso é `amergedApiExecutionRole`, e ela deve ter acesso explícito para executar solicitações na fonte APIs incluída em sua API mesclada por meio da permissão do `appsync:SourceGraphQL` IAM. Durante a execução de uma solicitação do GraphQL, o AWS AppSync serviço assumirá essa função de serviço e autorizará a função a realizar a ação. `appsync:SourceGraphQL`

AWS AppSync suporta permitir ou negar essa permissão em campos específicos de nível superior dentro da solicitação, como o modo de autorização do IAM funciona para o IAM. APIs Para non-top-level campos, é AWS AppSync necessário definir a permissão no próprio ARN da API de origem. Para restringir o acesso a non-top-level campos específicos na API mesclada, recomendamos

implementar uma lógica personalizada em seu Lambda ou ocultar os campos da API de origem da API mesclada usando a diretiva `@hidden`. Se você quiser permitir que o perfil execute todas as operações de dados em uma API de origem, adicione a política abaixo. Observe que a primeira entrada de recurso permite acesso a todos os campos de nível superior e a segunda entrada abrange resolvedores secundários que autorizam o próprio atributo da API de origem:

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" ]
  }]
}
```

Se quiser limitar o acesso somente a um campo específico de nível superior, você pode usar uma política como esta:

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" ]
  }]
}
```

Você também pode usar o assistente de criação de API do AWS AppSync console para gerar uma função de serviço para permitir que sua API mesclada acesse recursos configurados na fonte APIs que estão na mesma conta da sua API mesclada. Caso sua fonte não APIs esteja na mesma conta da API mesclada, você deve primeiro compartilhar seus AWS recursos usando o Resource Access Manager (AWS RAM).

Configurando contas cruzadas mescladas usando APIs AWS RAM

Ao criar uma API mesclada, você pode, opcionalmente, associar a fonte APIs de outras contas que foram compartilhadas por meio do AWS Resource Access Manager (AWS RAM). AWS RAM ajuda você a compartilhar seus recursos com segurança entre AWS contas, dentro de sua organização ou unidades organizacionais (OUs) e com funções e usuários do IAM.

AWS AppSync se integra AWS RAM para oferecer suporte à configuração e ao acesso à fonte APIs em várias contas a partir de uma única API mesclada. AWS RAM permite criar um compartilhamento de recursos ou um contêiner de recursos e os conjuntos de permissões que serão compartilhados para cada um deles. Você pode adicionar AWS AppSync APIs a um compartilhamento de recursos no AWS RAM. Em um compartilhamento de recursos, AWS AppSync fornece três conjuntos de permissões diferentes que podem ser associados a uma AWS AppSync API na RAM:

1. `AWSRAMPermissionAppSyncSourceApiOperationAccess`: o conjunto de permissões padrão que é adicionado ao compartilhar uma AWS AppSync API AWS RAM se nenhuma outra permissão for especificada. Esse conjunto de permissões é usado para compartilhar uma AWS AppSync API de origem com um proprietário da API mesclada. Esse conjunto de permissões inclui a permissão para `appsync:AssociateMergedGraphQLApi` a API de origem, bem como a permissão `appsync:SourceGraphQL` necessária para acessar os atributos da API de origem em runtime.
2. `AWSRAMPermissionAppSyncMergedApiOperationAccess`: esse conjunto de permissões deve ser configurado ao compartilhar uma API mesclada com o proprietário da API de origem. Esse conjunto de permissões dará à API de origem a capacidade de configurar a API mesclada, incluindo a capacidade de associar qualquer fonte APIs pertencente ao principal de destino à API mesclada e de ler e atualizar as associações de API de origem da API mesclada.
3. `AWSRAMPermissionAppSyncAllowSourceGraphQLAccess`: esse conjunto de permissões permite que a `appsync:SourceGraphQL` permissão seja usada com uma AWS AppSync API. Ele deve ser usado para compartilhar uma API de origem com um proprietário da API mesclada. Ao contrário do conjunto de permissões padrão para acesso à operação da API de origem, esse conjunto de permissões inclui apenas a permissão de runtime `appsync:SourceGraphQL`. Se

um usuário optar por compartilhar o acesso à operação da API mesclada com um proprietário da API de origem, ele também precisará compartilhar essa permissão da API de origem com o proprietário da API mesclada para ter acesso de runtime por meio do endpoint da API mesclada.

AWS AppSync também oferece suporte a permissões gerenciadas pelo cliente. Quando uma das permissões AWS gerenciadas fornecidas não funciona, você pode criar sua própria permissão gerenciada pelo cliente. As permissões gerenciadas pelo cliente são permissões gerenciadas que você cria e mantém especificando com precisão quais ações podem ser executadas sob quais condições com o uso de recursos compartilhados. AWS RAM AWS AppSync permite que você escolha entre as seguintes ações ao criar sua própria permissão:

1. `appsync:AssociateSourceGraphQLApi`
2. `appsync:AssociateMergedGraphQLApi`
3. `appsync:GetSourceApiAssociation`
4. `appsync:UpdateSourceApiAssociation`
5. `appsync:StartSchemaMerge`
6. `appsync:ListTypesByAssociation`
7. `appsync:SourceGraphQL`

Depois de compartilhar adequadamente uma API de origem ou API mesclada AWS RAM e, se necessário, o convite de compartilhamento de recursos ter sido aceito, ele ficará visível no AWS AppSync console quando você criar ou atualizar as associações da API de origem na sua API mesclada. Você também pode listar tudo o AWS AppSync APIs que foi compartilhado usando AWS RAM sua conta, independentemente da permissão definida, chamando a `ListGraphQLApis` operação fornecida por AWS AppSync e usando o filtro do `OTHER_ACCOUNTS` proprietário.

 Note

O compartilhamento via AWS RAM exige que o chamador tenha permissão para realizar a `appsync:PutResourcePolicy` ação em qualquer API que esteja sendo compartilhada. AWS RAM

Mesclar

Gerenciar mesclagens

APIs Os mesclados têm como objetivo apoiar a colaboração da equipe em um AWS AppSync endpoint unificado. As equipes podem desenvolver de forma independente sua própria fonte isolada de APIs GraphQL no back-end, enquanto AWS AppSync o serviço gerencia a integração dos recursos em um único endpoint de API mesclada, a fim de reduzir o atrito na colaboração e diminuir os prazos de desenvolvimento.

Mesclagens automáticas

A fonte APIs associada à sua API AWS AppSync mesclada pode ser configurada para ser mesclada automaticamente (mesclagem automática) na API mesclada após qualquer alteração ser feita na API de origem. Isso garante que as alterações da API de origem sejam sempre propagadas para o endpoint da API mesclada em segundo plano. Qualquer alteração no esquema da API de origem será atualizada na API mesclada, desde que isso não introduza um conflito de mesclagem com uma definição existente na API mesclada. Se a atualização na API de origem for atualizar um resolvedor, fonte de dados ou função, o atributo importado também será atualizado. Quando um novo conflito é introduzido e não pode ser resolvido automaticamente (resolvido automaticamente), a atualização do esquema da API mesclada é rejeitada devido a um conflito não compatível durante a operação de mesclagem. A mensagem de erro está disponível no console para cada associação de API de origem que tenha um status de MERGE_FAILED. Você também pode inspecionar a mensagem de erro chamando a `GetSourceApiAssociation` operação de uma determinada associação de API de origem usando o AWS SDK ou usando a AWS CLI da seguinte forma:

```
aws appsync get-source-api-association --merged-api-identifier <Merged API ARN> --  
association-id <SourceApiAssociation id>
```

Isso produzirá um resultado no seguinte formato:

```
{  
  "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."
  }
}
```

Mesclagens manuais

A configuração padrão para uma API de origem é uma mesclagem manual. Para mesclar as alterações que ocorreram na fonte APIs desde a última atualização da API mesclada, o proprietário da API de origem pode invocar uma mesclagem manual a partir do AWS AppSync console ou por meio da `StartSchemaMerge` operação disponível no SDK AWS e na CLI. AWS

Suporte adicional para Merged APIs

Configurar assinaturas

Ao contrário das abordagens baseadas em roteador para a composição do esquema do GraphQL, o AWS AppSync Merged fornece suporte APIs integrado para assinaturas do GraphQL. Todas as operações de assinatura definidas em sua fonte associada APIs serão mescladas e funcionarão automaticamente em sua API mesclada sem modificação. [Para saber mais sobre como AWS AppSync oferece suporte a assinaturas por meio de WebSockets conexão sem servidor, consulte Dados em tempo real.](#)

Configurar a observabilidade

AWS AppSync O Merged APIs fornece registro, monitoramento e métricas integrados via [Amazon CloudWatch](#). AWS AppSync também fornece suporte integrado para rastreamento via [AWS X-Ray](#).

Configurar domínios personalizados

AWS AppSync [O Merged APIs fornece suporte integrado para o uso de domínios personalizados com o GraphQL e os endpoints em tempo real da API Merged.](#)

Configurar o cache

AWS AppSync O Merged APIs fornece suporte integrado para armazenar, opcionalmente, em cache respostas em nível de solicitação e/ou em nível de resolvedor, bem como compactação de respostas. Para saber mais, consulte [Armazenamento em cache e compactação](#).

Configurando o privado APIs

AWS AppSync [O Merged APIs fornece suporte integrado para Private, APIs que limita o acesso aos endpoints GraphQL e Real-time da API mesclada ao tráfego proveniente de endpoints VPC que você pode configurar.](#)

Configurar regras de firewall

AWS AppSync O Merged APIs fornece suporte integrado para AWS WAF, o que permite que você proteja você APIs definindo [regras de firewall de aplicativos web](#).

Configurar logs de auditoria

AWS AppSync O Merged APIs fornece suporte integrado para AWS CloudTrail, o que permite [configurar e gerenciar registros de auditoria](#).

Limitações de APIs mescladas

Ao desenvolver o Merged APIs, observe as seguintes regras:

1. Uma API mesclada não pode ser uma API de origem para outra API mesclada.
2. Uma API de origem não pode ser associada a mais de uma API mesclada.
3. O limite de tamanho padrão para um documento do esquema da API mesclada é de 10 MB.
4. O número padrão de fontes APIs que pode ser associado a uma API mesclada é 10. No entanto, você pode solicitar um aumento de limite se precisar de mais de 10 fontes APIs em sua API mesclada.

Criando mesclado APIs

Para criar uma API mesclada no console

1. Faça login no AWS Management Console e abra o [AWS AppSync console](#).
 - No Painel, selecione Criar API.
2. Selecione API mesclada e, em seguida, Avançar.
3. Na página Especificar detalhes da API, insira as seguintes informações:
 - a. Em Detalhes da API, insira as seguintes informações:

- i. Especifique o nome da API da API mesclada. Esse campo é uma forma de rotular sua API do GraphQL para diferenciá-la convenientemente de outros GraphQL APIs
 - ii. Especifique os detalhes de contato. Esse campo é opcional e anexa um nome ou grupo à API do GraphQL. Ele não é vinculado ou gerado por outros atributos e funciona da mesma forma que o campo de nome da API.
- b. Em Função de serviço, você deve anexar uma função de execução do IAM à sua API mesclada para que AWS AppSync possa importar e usar seus recursos com segurança em tempo de execução. Você pode escolher criar e usar uma nova função de serviço, o que permitirá especificar as políticas e os recursos que AWS AppSync serão usados. Você também pode importar um perfil do IAM existente escolhendo Usar um perfil de serviço existente e selecionando um perfil na lista suspensa.
- c. Em Configuração da API privada, é possível ativar os atributos da API privada. Observe que essa opção não pode ser alterada após a criação da API mesclada. Para obter mais informações sobre privacidade APIs, consulte Como [usar o AWS AppSync privado APIs](#).

Quando terminar, selecione Avançar.

4. Em seguida, você deve adicionar o GraphQL APIs que será usado como base para sua API mesclada. Na APIs página Selecionar fonte, insira as seguintes informações:
- a. Na tabela APIs da sua AWS conta, escolha Adicionar fonte APIs. Na lista do GraphQL APIs, cada entrada conterá os seguintes dados:
 - i. Nome: o campo nome da API da API do GraphQL.
 - ii. ID da API: o valor de ID exclusivo da API do GraphQL.
 - iii. Modo de autenticação primária: o modo de autorização padrão para a API do GraphQL. Para obter mais informações sobre os modos de autorização no AWS AppSync, consulte [Autorização e autenticação](#).
 - iv. Modo de autenticação adicional: os modos de autorização secundários que foram configurados na API do GraphQL.
 - v. Escolha o APIs que você usará na API mesclada marcando a caixa de seleção ao lado do campo Nome da API. Depois, escolha Adicionar fonte APIs. O GraphQL selecionado APIs aparecerá na tabela APIs de suas AWS contas.
 - b. Na tabela APIs De outras AWS contas, escolha Adicionar fonte APIs. O GraphQL APIs nesta lista vem de outras contas que estão compartilhando seus recursos com a sua por meio de AWS Resource Access Manager (AWS RAM). O processo para selecionar o

GraphQL APIs nesta tabela é o mesmo da seção anterior. Para obter mais informações sobre o compartilhamento de recursos por meio de AWS RAM, consulte [O que é AWS Resource Access Manager?](#) .

Quando terminar, selecione Avançar.

- c. Adicione seu modo de autenticação principal. Consulte [Autorização e autenticação](#) para obter mais informações. Selecione Avançar.
- d. Revise suas entradas e selecione Criar API.

Construindo o GraphQL APIs com introspecção do RDS

O utilitário de introspecção do AWS AppSync pode descobrir modelos de tabelas de banco de dados e propor tipos GraphQL. O assistente Create API do AWS AppSync console pode gerar instantaneamente uma API a partir de um banco de dados Aurora MySQL ou PostgreSQL. Ele cria automaticamente tipos e JavaScript resolvedores para ler e gravar dados.

AWS AppSync fornece integração direta com bancos de dados Amazon Aurora por meio da API de dados do Amazon RDS. Em vez de exigir uma conexão persistente com o banco de dados, a API de dados do Amazon RDS oferece um endpoint HTTP seguro que AWS AppSync se conecta para execução SQL declarações. É possível usá-lo para criar uma API de banco de dados relacional para as workloads MySQL e PostgreSQL no Aurora.

A criação de uma API para seu banco de dados relacional AWS AppSync tem várias vantagens:

- O banco de dados não é diretamente exposto aos clientes, separando o ponto de acesso do próprio banco de dados.
- Você pode criar aplicativos APIs personalizados de acordo com as necessidades de diferentes aplicativos, eliminando a necessidade de lógica de negócios personalizada nos front-ends. Isso se alinha com o padrão Backend-For-Frontend (BFF).
- A autorização e o controle de acesso podem ser implementados na AWS AppSync camada usando vários modos de autorização para controlar o acesso. Nenhum recurso computacional adicional é necessário para se conectar ao banco de dados, como hospedar um servidor web ou fazer conexões por proxy.
- Recursos em tempo real podem ser adicionados por meio de assinaturas, com mutações de dados feitas por meio de envio AppSync automático para clientes conectados.
- Os clientes podem se conectar à API via HTTPS usando portas comuns, como 443.

AWS AppSync facilita a criação APIs a partir de bancos de dados relacionais existentes. O utilitário de introspecção pode descobrir modelos por meio de tabelas de banco de dados e propor tipos de GraphQL. O assistente de API de criação do console do AWS AppSync pode gerar instantaneamente uma API por meio de um banco de dados Aurora MySQL ou PostgreSQL. Ele cria automaticamente tipos e JavaScript resolvedores para ler e gravar dados.

AWS AppSync fornece JavaScript utilitários integrados para simplificar a gravação de instruções SQL em resolvedores. Você pode usar os modelos AWS AppSync de `sql` tags para instruções estáticas com valores dinâmicos ou os utilitários do `rds` módulo para criar instruções programaticamente. Consulte as fontes de dados da [referência de funções de resolvedores do RDS](#) e [módulos integrados](#) para obter mais informações.

Usar o recurso de introspecção (console)

Para ver um tutorial detalhado e um guia de introdução, consulte [Tutorial: Aurora PostgreSQL Serverless with Data API](#).

O AWS AppSync console permite que você crie uma API AWS AppSync GraphQL a partir do seu banco de dados Aurora existente configurado com a API de dados em apenas alguns minutos. Isso gera rapidamente um esquema operacional com base na configuração do banco de dados. É possível usar a API no estado em que se encontra ou desenvolvê-la para adicionar recursos.

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
 - No Painel, escolha Criar API.
2. Em Opções de API, escolha GraphQL APIs, Comece com um cluster Amazon Aurora e, em seguida, Avançar.
 - a. Insira um Nome da API. Isso será usado como um identificador para a API no console.
 - b. Para obter detalhes de contato, você pode inserir um ponto de contato para identificar um gerente para a API. Esse é um campo opcional.
 - c. Em Configuração da API privada, é possível habilitar os atributos da API privada. Uma API privada só pode ser acessada de um endpoint da VPC (VPCE) configurado. Para obter mais informações, consulte [Privado APIs](#).

Não recomendamos habilitar esse atributo para este exemplo. Após analisar suas entradas, selecione Próximo.

3. Na página Banco de dados, escolha Selecionar banco de dados.

- a. É necessário escolher o banco de dados no cluster. A primeira etapa é selecionar a região na qual o cluster existe.
 - b. Selecione o Cluster do Aurora na lista suspensa. Observe que é necessário ter criado e [habilitado](#) uma API de dados correspondente antes de usar o recurso.
 - c. Depois, é necessário adicionar as credenciais do banco de dados ao serviço. Isso é feito principalmente usando AWS Secrets Manager. Selecione a região onde existe o segredo. Para obter mais informações sobre como recuperar informações de segredos, consulte [Find secrets](#) ou [Retrieve secrets](#).
 - d. Adicione o segredo da lista suspensa. Observe que o usuário deve ter [permissões de leitura](#) para o banco de dados.
4. Escolha Importar.

AWS AppSync começará a introspectar seu banco de dados, descobrindo tabelas, colunas, chaves primárias e índices. Ele confere se as tabelas descobertas podem ser compatíveis com uma API do GraphQL. Observe que, para oferecer suporte à criação de novas linhas, as tabelas precisam de uma chave primária, que pode usar várias colunas. AWS AppSync mapeia as colunas da tabela para campos de texto da seguinte forma:

Tipo de dados	Tipo de campo
VARCHAR	String
CHAR	String
BINARY	String
VARBINARY	String
TINYBLOB	String
TINYTEXT	String
TEXT	String
BLOB	String
MEDIUMTEXT	String

MEDIUMBLOB	String
LONGTEXT	String
LOBLOB	String
BOOL	Booleano
BOOLEAN	Booleano
BIT	Int
TINYINT	Int
SMALLINT	Int
MEDIUMINT	Int
INT	Int
INTEGER	Int
BIGINT	Int
YEAR	Int
FLOAT	Float
DOUBLE	Float
DECIMAL	Float
DEC	Float
NUMERIC	Float
DATE	AWSDDate
TIMESTAMP	String
DATETIME	String
TIME	AWSTime

JSON

AWSJson

ENUM

ENUM

- Quando a descoberta da tabela for concluída, a seção Banco de dados será preenchida com as informações. Na nova seção Tabelas do banco de dados, os dados da tabela podem já estar preenchidos e convertidos em um tipo para o esquema. Se você não vê alguns dos dados necessários, pode verificá-los escolhendo Adicionar tabelas, clicando nas caixas de seleção desses tipos no modal exibido e escolhendo Adicionar.

Para remover um tipo da seção Tabelas do banco de dados, clique na caixa de seleção ao lado do tipo a ser removido e selecione Remover. Os tipos removidos serão colocados no modal Adicionar tabelas se quiser adicioná-los novamente mais tarde.

Observe que AWS AppSync usa os nomes das tabelas como nomes de tipo, mas você pode renomeá-los, por exemplo, alterando o nome de uma tabela plural *movies* para o nome do tipo. *Movie* Para renomear um tipo na seção Tabelas do banco de dados, clique na caixa de seleção do tipo a ser renomeado e, depois, clique no ícone de lápis na coluna Nome do tipo.

Para visualizar o conteúdo do esquema com base nas seleções, selecione Visualizar esquema. Observe que esse esquema não pode estar vazio. Será necessário ter, pelo menos, uma tabela convertida em um tipo. Além disso, esse esquema não pode exceder 1 MB.

- Em Perfil de serviço, decida se deseja criar um perfil de serviço especificamente para essa importação ou usar um perfil existente.
- Escolha Próximo.
 - Depois, decida se deseja criar uma API somente leitura (somente consultas) ou uma API para leitura e gravação de dados (com consultas e mutações). O último também aceita assinaturas em tempo real acionadas por mutações.
 - Escolha Próximo.
 - Revise as escolhas e, depois, selecione Criar API. O AWS AppSync criará a API e anexará resolvedores a consultas e mutações. A API gerada é totalmente funcional e pode ser estendida conforme necessário.

Usar o recurso de introspecção (API)

É possível usar a API de introspecção `StartDataSourceIntrospection` para descobrir modelos no banco de dados de forma programática. Para obter mais detalhes sobre o comando, consulte Usar a API [StartDataSourceIntrospection](#).

Para usar `StartDataSourceIntrospection`, forneça seu cluster Aurora Amazon Resource Name (ARN), nome do banco de dados e ARN secreto. AWS Secrets Manager O comando inicia o processo de introspecção. É possível recuperar os resultados com o comando `GetDataSourceIntrospection`. É possível especificar se o comando deve exibir a string SDL (linguagem de definição de armazenamento) para os modelos descobertos. Ela é útil para gerar uma definição de esquema SDL diretamente pelos modelos descobertos.

Por exemplo, se você tiver a seguinte declaração de linguagem de definição de dados (DDL) para uma tabela `Todos` simples:

```
create table if not exists public.todos
(
  id serial constraint todos_pk primary key,
  description text,
  due timestamp,
  "createdAt" timestamp default now()
);
```

Você deve começar a introspecção com o seguinte:

```
aws appsync start-data-source-introspection \
  --rds-data-api-config resourceArn=<cluster-arn>,secretArn=<secret-arn>,databaseName=database
```

Depois, use o comando `GetDataSourceIntrospection` para recuperar o resultado.

```
aws appsync get-data-source-introspection \
  --introspection-id a1234567-8910-abcd-efgh-identifier \
  --include-models-sdl
```

Ele exibirá o resultado a seguir.

```
{
  "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  description: String\n  due: AWSDateTime\n  id: Int!\n  createdAt: AWSDateTime\n}"
}
"nextToken": null
}
}

```

Como criar um aplicativo cliente usando o cliente Amplify

Você pode se conectar à sua API AWS AppSync GraphQL usando qualquer cliente GraphQL, mas recomendamos fortemente o cliente Amplify v6. O Amplify não apenas gera automaticamente um cliente fortemente tipado para SDKs sua API do GraphQL, mas também oferece suporte para dados em tempo real e recursos aprimorados de consulta do GraphQL em aplicativos clientes. Para aplicativos web, o Amplify pode produzir um JavaScript cliente. Para aqueles que visam ambientes multiplataforma ou móveis, o Amplify é compatível com Android, iOS e React Native. Para se aprofundar na geração de código do cliente para essas plataformas, consulte a [documentação](#) do Amplify. Aqui está um guia para começar sua jornada com um aplicativo JavaScript React:

Note

Você precisa instalar e configurar o [npm](#) e a [Amazon CLI](#) antes de começar. Se estiver usando o cliente Amplify v6, [siga este guia](#).

Para começar:

1. Em sua máquina local, navegue até o diretório do projeto. Instale a biblioteca do Amplify usando o comando abaixo:

```
npm install aws-amplify
```

2. Baixe o arquivo de configuração e coloque-o na pasta do projeto. Seu arquivo de configuração normalmente conterá uma variável `config` com algumas configurações (endpoint, região, modo de autorização etc.) definidas. Por exemplo, poderia ser assim:

```
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. No seu código, importe a biblioteca do Amplify e sua configuração para configurar o Amplify:

```
import { Amplify } from 'aws-amplify';
import config from './aws-exports.js';

Amplify.configure(config);
```

Como alternativa, use o snippet na configuração da sua API para configurar o Amplify diretamente:

```
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. Ao usar o conjunto de ferramentas do Amplify, você tem a opção de gerar operações automaticamente com base em seu esquema, o que poupa o esforço de criação manual de scripts. No diretório raiz da sua aplicação, use o seguinte comando da CLI:

```
npx @aws-amplify/cli codegen add --apiId <id goes here> --region <region goes here>
```

Isso baixará o esquema da sua API e, por padrão, gerará o código auxiliar do cliente na pasta `src/graphql`. Depois de cada implantação da API, você pode executar novamente o comando a seguir para gerar instruções e tipos atualizados do GraphQL:

```
npx @aws-amplify/cli codegen
```

5. Agora você pode gerar modelos para Android, Swift, Flutter e JavaScript DataStore Use o comando a seguir para baixar seu esquema:

```
aws appsync get-introspection-schema --api-id <id goes here> --region <region goes here> --format SDL schema.graphql
```

Depois execute o comando a seguir do diretório raiz da aplicação:

```
npx @aws-amplify/cli codegen models \
--model-schema schema.graphql \
--target [android|ios|flutter|javascript|typescript] \
--output-dir ./
```


JavaScript tutoriais de resolvidor para AWS AppSync

Fontes de dados e resolvidores são usados AWS AppSync para traduzir solicitações do GraphQL e buscar informações de seus recursos. AWS AppSync oferece suporte ao provisionamento automático e às conexões com determinados tipos de fonte de dados. AWS AppSync também oferece suporte AWS Lambda ao Amazon DynamoDB, bancos de dados relacionais (Amazon Aurora Serverless), OpenSearch Amazon Service e endpoints HTTP como fontes de dados. Você pode usar uma API GraphQL com seus AWS recursos existentes ou criar fontes de dados e resolvidores do zero. As seções a seguir têm como objetivo elucidar alguns dos casos de uso mais comuns do GraphQL na forma de tutoriais.

Tópicos

- [Criação de um aplicativo de postagem simples usando resolvidores do DynamoDB JavaScript](#)
- [Usando AWS Lambda resolvidores em AWS AppSync](#)
- [Usando resolvidores locais em AWS AppSync](#)
- [Combinação de resolvidores do GraphQL no AWS AppSync](#)
- [Usando resolvidores OpenSearch do Amazon Service em AWS AppSync](#)
- [Executando transações do DynamoDB no AWS AppSync](#)
- [Usando operações em lote do DynamoDB em AWS AppSync](#)
- [Usando resolvidores HTTP em AWS AppSync](#)
- [Uso do Aurora PostgreSQL com a API de dados no AWS AppSync](#)

Criação de um aplicativo de postagem simples usando resolvidores do DynamoDB JavaScript

Neste tutorial, você importará suas tabelas do Amazon DynamoDB e as conectará AWS AppSync para criar uma API GraphQL totalmente funcional usando resolvidores de pipeline que você pode JavaScript utilizar em seu próprio aplicativo.

Você usará o AWS AppSync console para provisionar seus recursos do Amazon DynamoDB, criar seus resolvidores e conectá-los às suas fontes de dados. Você também poderá ler e gravar em seu banco de dados do Amazon DynamoDB por meio de instruções do GraphQL e assinar dados em tempo real.

Existem etapas específicas que devem ser concluídas para que as instruções do GraphQL sejam traduzidas para operações do Amazon DynamoDB e para que as respostas sejam traduzidas novamente para o GraphQL. Esse tutorial descreve o processo de configuração por meio de vários cenários reais e padrões de acesso aos dados.

Criação da API GraphQL

Para criar uma API GraphQL no AWS AppSync

1. Abra o AppSync console e escolha Criar API.
2. Selecione Design do zero e escolha Próximo.
3. Nomeie sua API PostTutorialAPI e escolha Próximo. Vá para a página de revisão, mantendo o restante das opções definidas com seus valores padrão e escolha Create.

O AWS AppSync console cria uma nova API GraphQL para você. Por padrão, ele está usando o modo de autenticação de chave de API. Você pode usar o console para configurar o restante da API GraphQL e executar consultas nela durante o restante desse tutorial.

Definição de uma API Post básica

Agora que você tem sua API GraphQL, você pode configurar um esquema básico que permite a criação, recuperação e exclusão básica de dados publicados.

Para adicionar dados ao seu esquema

1. Na sua API, escolha a guia Esquema.
2. Criaremos um esquema que define um tipo Post e uma operação addPost para adicionar e obter objetos Post. No painel Esquema, substitua o conteúdo pelo seguinte código:

```
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. Escolha Salvar esquema.

Configuração de sua tabela do Amazon DynamoDB

O AWS AppSync console pode ajudar a provisionar os AWS recursos necessários para armazenar seus próprios recursos em uma tabela do Amazon DynamoDB. Nesta etapa, você criará uma tabela do Amazon DynamoDB, para armazenar suas postagens. Você também configurará um [índice secundário](#) que usaremos posteriormente.

Para criar sua tabela do Amazon DynamoDB

1. Na página Esquema, escolha Criar recursos.
2. Escolha Usar tipo existente e depois o tipo Post.
3. Na seção Índices adicionais, escolha Adicionar índice.
4. Nomeie o índice `author-index`.
5. Defina `Primary key` como `author` e a chave `Sort` como `None`.
6. Desative `Automatically generate GraphQL`. Neste exemplo, nós mesmos criaremos o resolvedor.
7. Escolha Criar.

Agora você tem uma nova fonte de dados chamada `PostTable`, que pode ser vista em Fontes de dados na guia lateral. Você usará essa fonte de dados para vincular suas consultas e mutações à sua tabela do Amazon DynamoDB.

Configurando um resolvedor `AddPost` (Amazon DynamoDB) `PutItem`

Agora que conhece AWS AppSync a tabela do Amazon DynamoDB, você pode vinculá-la a consultas e mutações individuais definindo resolvedores. O primeiro resolvedor que você cria é o resolvedor de `addPost` pipeline usando JavaScript, que permite criar uma publicação na sua tabela do Amazon DynamoDB. Um resolvedor de pipeline possui os seguintes componentes:

- A local no esquema do GraphQL para anexar o resolvedor. Nesse caso, você está configurando um resolvedor no campo `createPost` no tipo `Mutation`. Esse resolvedor será invocado quando o chamador chamar a mutação `{ addPost(...){...} }`.
- A fonte de dados a ser usada para esse resolvedor. Nesse caso, você deseja usar a fonte de dados do DynamoDB definida anteriormente para poder adicionar entradas à tabela do DynamoDB de `post-table-for-tutorial`.
- O manipulador de solicitações. O manipulador de solicitações é uma função que manipula a solicitação recebida do chamador e a traduz em instruções para AWS AppSync execução no DynamoDB.
- O manipulador de respostas. A tarefa do manipulador de respostas é manipular a resposta do DynamoDB e traduzi-la de volta em algo que o GraphQL espera. Isso é útil se a forma dos dados no DynamoDB for diferente para o tipo `Post` no GraphQL, mas, nesse caso, elas têm a mesma forma, portanto basta transmitir os dados.

Para configurar seu resolvedor

1. Na sua API, escolha a guia Esquema.
2. No painel Resolvedores, encontre o campo `addPost` no tipo `Mutation` e depois escolha Anexar.
3. Escolha sua fonte de dados e selecione Criar.
4. No seu editor de código, substitua o código por este trecho:

```
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. Escolha Salvar.

Note

Nesse código, você usa os utilitários do módulo do DynamoDB que permitem criar facilmente solicitações do DynamoDB.

AWS AppSync vem com um utilitário para geração automática de ID chamado `util.autoId()`, que é usado para gerar um ID para sua nova postagem. Se você não especificar uma ID, o utilitário a gerará automaticamente para você.

```
const key = { id: ctx.args.id ?? util.autoId() }
```

Para obter mais informações sobre os utilitários disponíveis para JavaScript, consulte [recursos de JavaScript tempo de execução para resolvedores e funções](#).

Chamar a API para adicionar uma publicação

Agora que o resolvedor foi configurado, é AWS AppSync possível traduzir uma `addPost` mutação recebida em uma operação do Amazon DynamoDB. `PutItem` Agora você pode executar uma mutação para colocar algo na tabela.

Para executar a operação

1. Na sua API, escolha a guia Consultas.
2. No painel Consultas, adicione a seguinte mutação:

```
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. Escolha Executar (o botão de reprodução laranja) e selecione addPost. Os resultados da publicação recém-criada devem aparecer no painel Resultados à direita do painel Consultas. A aparência deve ser semelhante à seguinte:

```

{
  "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
    }
  }
}

```

A explicação a seguir mostra o que ocorreu:

1. AWS AppSync recebeu uma solicitação de addPost mutação.
2. AWS AppSync executa o manipulador de solicitações do resolvidor. A função `ddb.put` cria uma solicitação `PutItem` semelhante a esta:

```

{

```

```
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 usa esse valor para gerar e executar uma solicitação do Amazon PutItem DynamoDB.
4. AWS AppSync pegou os resultados da PutItem solicitação e os converteu novamente em tipos GraphQL.

```
{
  "id" : "123",
  "author": "AUTHORNAME",
  "title": "Our first post!",
  "content": "This is our first post.",
  "url": "https://aws.amazon.com/appsync/",
  "ups" : 1,
  "downs" : 0,
  "version" : 1
}
```

5. O manipulador de resposta retorna o resultado imediatamente (`return ctx.result`).
6. O resultado final é visível na resposta do GraphQL.

Configuração do resolvedor GetPost (Amazon DynamoDB) GetItem

Agora que você pode adicionar dados à tabela do Amazon DynamoDB, é necessário configurar a consulta `getPost` para que ela possa recuperar esses dados da tabela. Para fazer isso, vamos configurar outro resolvedor.

Para adicionar seu resolvedor

1. Na sua API, escolha a guia Esquema.

2. No painel Resolvedores à direita, encontre o campo `getPost` no tipo Query e escolha Anexar.
3. Escolha sua fonte de dados e selecione Criar.
4. No editor de código, substitua o código por este trecho:

```
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. Salve o resolvedor.

Note

Nesse resolvedor, usamos uma expressão de função de seta para o manipulador de resposta.

Chamar a API para obter uma publicação

Agora que o resolvedor foi configurado, AWS AppSync sabe como traduzir uma `getPost` consulta recebida em uma operação do Amazon DynamoDB `GetItem`. Agora é possível executar uma consulta para recuperar a publicação criada anteriormente.

Para executar sua consulta

1. Na sua API, escolha a guia Consultas.
2. No painel Consultas, adicione o seguinte código e use a ID que você copiou após criar sua publicação:

```
query getPost {
  getPost(id: "123") {
    id
    author
    title
    content
    url
  }
}
```



```
    ups
    downs
    version
  }
}
```

3. Escolha Executar (o botão de reprodução laranja) e selecione `getPost`. Os resultados da publicação recém-criada devem aparecer no painel Resultados à direita do painel Consultas.
4. A publicação recuperada do Amazon DynamoDB deve aparecer no painel Results à direita do painel Queries. A aparência deve ser semelhante à seguinte:

```
{
  "data": {
    "getPost": {
      "id": "123",
      "author": "AUTHORNAME",
      "title": "Our first post!",
      "content": "This is our first post.",
      "url": "https://aws.amazon.com/appsync/",
      "ups": 1,
      "downs": 0,
      "version": 1
    }
  }
}
```

Como alternativa, utilize o exemplo a seguir:

```
query getPost {
  getPost(id: "123") {
    id
    author
    title
  }
}
```

Se sua consulta `getPost` precisar apenas de `id`, `author` e `title`, você poderá alterar sua função de solicitação para usar expressões de projeção para especificar apenas os atributos que deseja da tabela do DynamoDB para evitar transferência desnecessária de dados do DynamoDB para AWS AppSync. Por exemplo, a função de solicitação pode se parecer com o trecho abaixo:

```
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
```

Você também pode usar um [selectionSetList](#) com `getPost` para representar `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
```

Crie uma mutação UpdatePost (Amazon DynamoDB) UpdateItem

Até agora, você pode criar e recuperar objetos Post no Amazon DynamoDB. Depois, você configurará uma nova mutação para atualizar um objeto. Em comparação com a mutação `addPost` que exige que todos os campos sejam especificados, essa mutação permite especificar apenas os campos que deseja alterar. Também introduziu um novo argumento `expectedVersion` que permite especificar a versão que deseja modificar. Você configurará uma condição que garante que você esteja modificando a versão mais recente do objeto. Você fará isso usando o `Operation.sc` do Amazon DynamoDB `UpdateItem`

Para criar seu resolvidor

1. Na sua API, escolha a guia Esquema.
2. No painel Schema, modifique o tipo `Mutation` para adicionar uma nova mutação `updatePost` da seguinte forma:

```
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. Escolha Salvar esquema.

4. No painel Resolvedores à direita, encontre o campo `updatePost` recém-criado no tipo `Mutation` e escolha Anexar. Crie seu resolvedor usando o trecho abaixo:

```

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. Salve todas as alterações feitas.

Esse resolvedor usa `ddb.update` para criar uma solicitação `UpdateItem` do Amazon DynamoDB. Em vez de escrever o item inteiro, você está apenas pedindo ao Amazon DynamoDB para atualizar determinados atributos. Isso é feito usando expressões de atualização do Amazon DynamoDB.

A função `ddb.update` recebe uma chave e um objeto de atualização como argumentos. Depois, você verifica os valores dos argumentos recebidos. Quando um valor é definido como `null`, use a operação `remove` do DynamoDB para sinalizar que o valor deve ser removido do item do DynamoDB.

Existe também uma nova seção `condition`. Uma expressão de condição permite que você diga ao AWS AppSync Amazon DynamoDB se a solicitação deve ou não ser bem-sucedida com base no estado do objeto que já está no Amazon DynamoDB antes da execução da operação. Nesse caso, você deseja que a solicitação `UpdateItem` seja bem-sucedida apenas se o campo `version` do item atualmente no Amazon DynamoDB corresponder exatamente ao argumento `expectedVersion`. Quando o item é atualizado, queremos incrementar o valor de `version`. Isso é fácil de fazer com a função de operação `increment`.

Para obter mais informações sobre expressões de condição, consulte a documentação de [expressões de condição](#).

Para obter mais informações sobre a `UpdateItem` solicitação, consulte a [UpdateItem](#) documentação e a documentação do módulo [DynamoDB](#).

Para obter mais informações sobre como escrever expressões de atualização, consulte a documentação do [UpdateExpressionsDynamoDB](#).

Chamar a API para atualizar uma publicação

Vamos tentar atualizar o objeto `Post` com o novo resolvedor.

Para atualizar seu objeto

1. Na sua API, escolha a guia Consultas.
2. No painel Consultas, adicione a seguinte mutação. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente:

```
mutation updatePost {
  updatePost(
    id:123
    title: "An empty story"
    content: null
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

3. Escolha Executar (o botão de reprodução laranja) e selecione updatePost.
4. A publicação atualizada no Amazon DynamoDB deve aparecer no painel Resultados à direita do painel Consultas. A aparência deve ser semelhante à seguinte:

```
{
  "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
    }
  }
}
```

Nessa solicitação, você pediu ao AWS AppSync Amazon DynamoDB que atualizasse somente `title` os campos e `content`. Todos os outros campos foram ignorados (exceto o incremento do campo `version`). Definiu-se o atributo `title` para um novo valor e o atributo `content` foi removido

da publicação. Os campos `author`, `url`, `ups` e `downs` foram mantidos. Tente executar a solicitação de mutação novamente, deixando a solicitação exatamente como está. Você verá uma resposta semelhante à seguinte:

```
{
  "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: 1RR3QN5F35CS8IV5VR40Q09NNBVV4KQNS05AEMVJF66Q9ASUAAJG)"
    }
  ]
}
```

A solicitação falha porque a expressão da condição é avaliada como `false`:

1. Na primeira vez que você executou a solicitação, o valor do campo `version` da publicação no Amazon DynamoDB era 1, que correspondia ao argumento `expectedVersion`. A solicitação foi bem-sucedida, o que significa que o campo `version` foi incrementado no Amazon DynamoDB para 2.
2. Na segunda vez que a solicitação foi executada, o valor do campo `version` da publicação no Amazon DynamoDB era 2, que não correspondeu ao argumento `expectedVersion`.

Esse padrão é geralmente chamado de bloqueio positivo.

Crie mutações de votos (Amazon DynamoDB) UpdateItem

O tipo `Post` contém campos `ups` e `downs` para permitir o registro de votos positivos e negativos. Contudo, no momento, a API não nos permite fazer nada com eles. Vamos adicionar uma mutação para permitir votos positivos e negativos nas publicações.

Para adicionar sua mutação

1. Na sua API, escolha a guia Esquema.
2. No painel Esquema, modifique o tipo `Mutation` e adicione o enumerador `DIRECTION` para adicionar novas mutações de voto:

```
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. Escolha Salvar esquema.
4. No painel Resolvedores à direita, encontre o campo `voterecém-criado` no tipo `Mutation` e escolha Anexar. Crie um resolvedor criando e substituindo o código pelo seguinte trecho:

```
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. Salve todas as alterações feitas.

Chamar a API para realizar voto positivo e negativo em uma publicação

Agora que os novos resolvedores foram configurados, AWS AppSync sabe como traduzir uma entrada `upvotePost` ou uma `downvote` mutação em uma operação do Amazon DynamoDB. `UpdateItem` Agora é possível executar mutações para realizar votos positivos ou negativos na publicação criada anteriormente.

Para executar sua mutação

1. Na sua API, escolha a guia Consultas.
2. No painel Consultas, adicione a seguinte mutação. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente:

```
mutation votePost {
  vote(id:123, direction: UP) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```


3. Escolha Executar (o botão de reprodução laranja) e selecione `votePost`.
4. A publicação atualizada no Amazon DynamoDB deve aparecer no painel Resultados à direita do painel Consultas. A aparência deve ser semelhante à seguinte:

```
{
  "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. Escolha Executar mais algumas vezes. Você deverá ver os campos `ups` e `version` aumentando em 1 cada vez que executar a consulta.
6. Altere a consulta para chamá-la com um arquivo `DIRECTION`.

```
mutation votePost {
  vote(id:123, direction: DOWN) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

7. Escolha Executar (o botão de reprodução laranja) e selecione `votePost`.

Desta vez, você deverá ver os campos `downs` e `version` aumentando em 1 cada vez que executar a consulta.

Configurando um resolvidor DeletePost (Amazon DynamoDB) DeleteItem

Depois, talvez você deseje criar uma mutação para excluir uma publicação. Você fará isso usando a operação DeleteItem do Amazon DynamoDB.

Para adicionar sua mutação

1. Em seu esquema, escolha a guia Esquema.
2. No painel Esquema, modifique o tipo Mutation para adicionar uma nova mutação deletePost:

```
type Mutation {  
  deletePost(id: ID!, expectedVersion: Int): Post  
  vote(id: ID!, direction: DIRECTION!): Post  
  updatePost(  
    id: ID!,  
    author: String,  
    title: String,  
    content: String,  
    url: String,  
    expectedVersion: Int!  
  ): Post  
  addPost(  
    id: ID  
    author: String!,  
    title: String!,  
    content: String!,  
    url: String!  
  ): Post!  
}
```

3. Desta vez, você tornou o campo expectedVersion opcional. Depois, escolha Salvar esquema.
4. No painel Resolvedores à direita, encontre o campo deleterecém-criado no tipo Mutation e escolha Anexar. Crie um resolvidor usando o seguinte código:

```
import { util } from '@aws-appsync/utils'  
  
import { util } from '@aws-appsync/utils';  
import * as ddb from '@aws-appsync/utils/dynamodb';  
  
export function request(ctx) {  
  let condition = null;  
  if (ctx.args.expectedVersion) {
```

```
    condition = {
      or: [
        { id: { attributeExists: false } },
        { version: { eq: ctx.args.expectedVersion } },
      ],
    };
  }
  return ddb.remove({ key: { id: ctx.args.id }, condition });
}

export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type);
  }
  return result;
}
```

Note

O argumento `expectedVersion` é opcional. Se o chamador definir um argumento `expectedVersion` na solicitação, o manipulador de solicitação adiciona uma condição que permite que a solicitação `DeleteItem` seja bem-sucedida somente se o item já estiver excluído ou se o atributo `version` da publicação no Amazon DynamoDB corresponder exatamente ao `expectedVersion`. Se omitido, nenhuma expressão de condição será especificada na solicitação `DeleteItem`. Ele será bem-sucedida independentemente do valor de `version`, ou se o item existe ou não no Amazon DynamoDB.

Mesmo que você esteja excluindo um item, é possível retornar o item que foi excluído, caso ainda não tenha sido excluído.

Para obter mais informações sobre a `DeleteItem` solicitação, consulte a [DeleteItem](#) documentação.

Chamar a API para excluir uma publicação

Agora que o resolvidor foi configurado, AWS AppSync sabe como traduzir uma `delete` mutação recebida em uma operação do Amazon DynamoDB. `DeleteItem` Agora você pode executar uma mutação para excluir algo na tabela.

Para executar sua mutação

1. Na sua API, escolha a guia Consultas.
2. No painel Consultas, adicione a seguinte mutação. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente:

```
mutation deletePost {  
  deletePost(id:123) {  
    id  
    author  
    title  
    content  
    url  
    ups  
    downs  
    version  
  }  
}
```

3. Escolha Executar (o botão de reprodução laranja) e selecione `deletePost`.
4. A publicação foi excluída do Amazon DynamoDB. Observe que AWS AppSync retorna o valor do item que foi excluído do Amazon DynamoDB, que deve aparecer no painel Resultados à direita do painel Consultas. A aparência deve ser semelhante à seguinte:

```
{  
  "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. O valor só será retornado se essa chamada para `deletePost` for aquela que realmente o exclui do Amazon DynamoDB. Escolha Executar novamente.

6. A chamada ainda será bem-sucedida, mas nenhum valor é retornado:

```
{
  "data": {
    "deletePost": null
  }
}
```

7. Agora vamos tentar excluir uma publicação, mas dessa vez especificando um `expectedValue`. Primeiro é necessário criar uma nova publicação, pois você acabou de excluir aquela com a qual estava trabalhando.

8. No painel Consultas, adicione a seguinte mutação:

```
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. Escolha Executar (o botão de reprodução laranja) e selecione `addPost`.

10. Os resultados da publicação recém-criada devem aparecer no painel Resultados à direita do painel Consultas. Registre a `id` do objeto recém-criado porque você precisará dele em alguns instantes. A aparência deve ser semelhante à seguinte:

```
{
  "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 Agora, vamos tentar excluir essa publicação com um valor ilegal para `expectedVersion`. No painel Consultas, adicione a seguinte mutação. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente:

```

mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 9999
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}

```

12 Escolha Executar (o botão de reprodução laranja) e selecione `deletePost`. O seguinte resultado é retornado:

```

{
  "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: 70830037M1FTFRK038A4CI9H43VV4KQNS05AEMVJF66Q9ASUAAJG)"
  }
]
}

```

13A solicitação falhou porque a expressão da condição foi avaliada como false. O valor para `version` da publicação no Amazon DynamoDB não corresponde ao `expectedValue` especificado nos argumentos. O valor atual do objeto é retornada no campo `data` na seção `errors` da resposta do GraphQL. Repita a solicitação, mas corrija o `expectedVersion`:

```

mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}

```

14Escolha Executar (o botão de reprodução laranja) e selecione `deletePost`.

Dessa vez, a solicitação é bem-sucedida e o valor que foi excluído do Amazon DynamoDB é retornado:

```

{

```

```

"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 Escolha Executar novamente. A chamada ainda é bem-sucedida, mas dessa vez nenhum valor é retornado porque a publicação já estava excluída no Amazon DynamoDB.

```
{ "data": { "deletePost": null } }
```

Configurar o resolvidor allPost (Scan do Amazon DynamoDB)

Até agora, a API só será útil se você souber o id de cada publicação a ser examinada. Vamos adicionar um novo resolvidor que retornará todas as publicações na tabela.

Para adicionar sua mutação

1. Na sua API, escolha a guia Esquema.
2. No painel Esquema, modifique o tipo Query para adicionar uma nova consulta allPost da seguinte forma:

```

type Query {
  allPost(limit: Int, nextToken: String): PaginatedPosts!
  getPost(id: ID): Post
}

```

3. Adicione um novo tipo PaginationPosts:

```

type PaginatedPosts {
  posts: [Post!]!
  nextToken: String
}

```



```
}
```

4. Escolha Salvar esquema.

5. No painel Resolvedores à direita, encontre o campo `allPost` recém-criado no tipo `Query` e escolha Anexar. Crie um resolvedor com o seguinte código:

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const { limit = 20, nextToken } = ctx.arguments;
  return ddb.scan({ limit, nextToken });
}

export function response(ctx) {
  const { items: posts = [], nextToken } = ctx.result;
  return { posts, nextToken };
}
```

O manipulador de solicitações desse resolvedor espera dois argumentos opcionais:

- `limit` - Especifica o número máximo de itens a serem retornados em uma única chamada.
- `nextToken` - Usado para recuperar o próximo conjunto de resultados (mostraremos de onde vem o valor `nextToken` mais tarde).

6. Salve todas as alterações feitas no seu resolvedor.

Para obter mais informações sobre solicitação `Scan`, consulte a documentação de referência do [Scan](#).

Chamar a API para verificar todas as publicações

Agora que o resolvedor foi configurado, AWS AppSync sabe como traduzir uma `allPost` consulta recebida em uma operação do Amazon DynamoDB `Scan`. Agora você pode verificar a tabela para recuperar todas as publicações. No entanto, antes de testar, é necessário preencher a tabela com alguns dados, pois tudo que foi usado até agora já foi excluído.

Para adicionar e consultar dados

1. Na sua API, escolha a guia Consultas.
2. No painel Consultas, adicione a seguinte mutação:

```
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. Escolha Executar (o botão de reprodução laranja).
4. Agora vamos examinar a tabela, retornando cinco resultados por vez. No painel Consultas, cole a seguinte consulta:

```
query allPost {
  allPost(limit: 5) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

5. Escolha Executar (o botão de reprodução laranja) e selecione allPost.

As primeiras cinco publicações devem aparecer no painel Results à direita do painel Queries. A aparência deve ser semelhante à seguinte:

```
{
```

```
"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. Você recebeu cinco resultados e um `nextToken` que pode ser usado para obter o próximo conjunto de resultados. Atualize a consulta `allPost` para incluir o `nextToken` do conjunto de resultados anterior:

```
query allPost {
  allPost(
    limit: 5
    nextToken: "<token>"
  ) {
    posts {
      id
      author
    }
    nextToken
  }
}
```

```
}  
}
```

7. Escolha Executar (o botão de reprodução laranja) e selecione allPost.

As quatro publicações restantes devem aparecer no painel Results à direita do painel Queries. Não há nextToken nesse conjunto de resultados pois todos os nove resultados foram encontrados, sem nenhum restante. A aparência deve ser semelhante à seguinte:

```
{  
  "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  
    }  
  }  
}
```

Configurando um resolvedor de allPostsBy autor (Amazon DynamoDB Query)

Além de verificar todas as publicações do Amazon DynamoDB, também é possível consultar o Amazon DynamoDB para recuperar as publicações criadas por um determinado autor. A tabela do Amazon DynamoDB que você criou anteriormente já tem um GlobalSecondaryIndex chamado

`author-index` que pode ser usada com uma operação Query do Amazon DynamoDB para recuperar todas as publicações criadas por um autor específico.

Para adicionar sua consulta

1. Na sua API, escolha a guia Esquema.
2. No painel Esquema, modifique o tipo Query para adicionar uma nova consulta `allPostsByAuthor` da seguinte forma:

```
type Query {  
  allPostsByAuthor(author: String!, limit: Int, nextToken: String): PaginatedPosts!  
  allPost(limit: Int, nextToken: String): PaginatedPosts!  
  getPost(id: ID): Post  
}
```

Observe que isso usa o mesmo tipo `PaginatedPosts` usado na consulta `allPost`.

3. Escolha Salvar esquema.
4. No painel Resolvedores à direita, encontre o campo `allPostsByAuthor` recém-criado no tipo Query e escolha Anexar. Crie um resolvedor usando o trecho abaixo:

```
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 };  
}
```

Assim como o resolvedor `allPost`, esse resolvedor possui dois argumentos opcionais:

- `limit` - Especifica o número máximo de itens a serem retornados em uma única chamada.

- `nextToken` - Recupera o próximo conjunto de resultados (o valor `nextToken` pode ser obtido em uma chamada anterior).

5. Salve todas as alterações feitas no seu resolvedor.

Para obter mais informações sobre a solicitação Query, consulte a documentação de referência da [consulta](#).

Chamar a API para consultar todas as publicações por autor

Agora que o resolvedor foi configurado, AWS AppSync sabe como traduzir uma `allPostsByAuthor` mutação recebida em uma operação do DynamoDB em relação ao índice `author-index`. Agora você pode consultar a tabela para recuperar todas as publicações de um determinado autor.

Antes disso, no entanto, vamos preencher a tabela com mais algumas publicações pois até agora todas têm o mesmo autor.

Para adicionar dados e consultar

1. Na sua API, escolha a guia Consultas.
2. No painel Consultas, adicione a seguinte mutação:

```
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. Escolha Executar (o botão de reprodução laranja) e selecione `addPost`.
4. Agora, vamos consultar a tabela, retornando todas as publicações de autoria da Nadia. No painel Consultas, cole a seguinte consulta:

```
query allPostsByAuthor {
  allPostsByAuthor(author: "Nadia") {
    posts {
      id
```

```
    title
  }
  nextToken
}
```

5. Escolha Executar (o botão de reprodução laranja) e selecione `allPostsByAuthor`. Todas as publicações de autoria de Nadia devem aparecer no painel Resultados à direita do painel Consultas. A aparência deve ser semelhante à seguinte:

```
{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world"
        },
        {
          "id": "11",
          "title": "Did you know...?"
        }
      ],
      "nextToken": null
    }
  }
}
```

6. A paginação funciona para Query da mesma forma que funciona para Scan. Por exemplo, vamos procurar todas as publicações por `AUTHORNAME`, obtendo cinco por vez.
7. No painel Consultas, cole a seguinte consulta:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    limit: 5
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

```
}  
}
```

8. Escolha Executar (o botão de reprodução laranja) e selecione `allPostsByAuthor`. Todas as publicações de autoria de `AUTHORNAME` devem aparecer no painel Resultados à direita do painel Consultas. A aparência deve ser semelhante à seguinte:

```
{  
  "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. Atualize o argumento `nextToken` com o valor retornado pela consulta anterior da seguinte forma:

```
query allPostsByAuthor {  
  allPostsByAuthor(  
    author: "AUTHORNAME"  
    limit: 5  
    nextToken: "<token>"  
  )  
}
```



```

    nextToken: "<token>"
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}

```

10 Escolha Executar (o botão de reprodução laranja) e selecione `allPostsByAuthor`. As publicações restantes de autoria de `AUTHORNAME` devem aparecer no painel Resultados à direita do painel Consultas. A aparência deve ser semelhante à seguinte:

```

{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "8",
          "title": "A series of posts, Volume 8"
        },
        {
          "id": "5",
          "title": "A series of posts, Volume 5"
        },
        {
          "id": "3",
          "title": "A series of posts, Volume 3"
        },
        {
          "id": "9",
          "title": "A series of posts, Volume 9"
        }
      ],
      "nextToken": null
    }
  }
}

```

Uso de conjuntos

Até agora, o tipo `Post` foi um objeto de chave/valor plano. Também é possível modelar objetos complexos com seu resolvidor, como conjuntos, listas e mapas. Vamos atualizar o tipo `Post` para incluir tags. Uma publicação pode ter zero ou mais tags, que são armazenadas no DynamoDB como um Conjunto de strings. Vamos configurar também algumas mutações para adicionar e remover tags, e uma nova consulta para verificar as publicações com uma tag específica.

Para configurar seus dados

1. Na sua API, escolha a guia Esquema.
2. No painel Esquema, modifique o tipo `Post` para adicionar um novo campo `tags` da seguinte forma:

```
type Post {  
  id: ID!  
  author: String  
  title: String  
  content: String  
  url: String  
  ups: Int!  
  downs: Int!  
  version: Int!  
  tags: [String!]  
}
```

3. No painel Esquema, modifique o tipo `Query` para adicionar uma nova consulta `allPostsByTag` da seguinte forma:

```
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. No painel Esquema, modifique o tipo `Mutation` para adicionar novas mutações `addTag` e `removeTag` da seguinte forma:

```
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. Escolha Salvar esquema.
6. No painel Resolvedores à direita, encontre o campo `allPostsByTag` recém-criado no tipo Query e escolha Anexar. Crie seu resolvedor usando o trecho abaixo:

```

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. Salve todas as alterações feitas no seu resolvedor.
8. Agora, faça o mesmo para o campo Mutation `addTag` usando o trecho abaixo:

 Note

Embora os utilitários do DynamoDB atualmente não sejam compatíveis com operações de conjuntos, você ainda pode interagir com os conjuntos criando a solicitação sozinho.

```
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. Salve todas as alterações feitas no seu resolvedor.

10 Repita isso mais uma vez para o campo `Mutation removeTag` usando o trecho abaixo:

```
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. Salve todas as alterações feitas no seu resolvedor.

Chamar a API para trabalhar com tags

Agora que você configurou os resolvedores, AWS AppSync sabe como traduzir as entradas `addTag` e as `allPostsByTag` solicitações em DynamoDB e operações. `removeTag` `UpdateItem` `Scan`. Para testar, vamos selecionar uma das publicações criadas anteriormente. Por exemplo, vamos usar uma publicação de autoria da Nadia.

Para usar tags

1. Na sua API, escolha a guia Consultas.
2. No painel Consultas, cole a seguinte consulta:

```
query allPostsByAuthor {  
  allPostsByAuthor(  
    author: "Nadia"  
  ) {  
    posts {  
      id  
      title  
    }  
    nextToken  
  }  
}
```

3. Escolha Executar (o botão de reprodução laranja) e selecione `allPostsByAuthor`.
4. Todas as publicações de Nadia devem aparecer no painel Resultados à direita do painel Consultas. A aparência deve ser semelhante à seguinte:

```
{  
  "data": {  
    "allPostsByAuthor": {  
      "posts": [  
        {
```

```
      "id": "10",
      "title": "The cutest dog in the world"
    },
    {
      "id": "11",
      "title": "Did you known...?"
    }
  ],
  "nextToken": null
}
```

5. Vamos usar aquela com o título The cutest dog in the world. Registre sua `id` porque você a usará mais tarde. Agora, vamos tentar adicionar uma tag dog.
6. No painel Consultas, adicione a seguinte mutação. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente.

```
mutation addTag {
  addTag(id:10 tag: "dog") {
    id
    title
    tags
  }
}
```

7. Escolha Executar (o botão de reprodução laranja) e selecione `addTag`. A publicação é atualizada com a nova tag:

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

8. É possível adicionar mais tags. Atualize a mutação para alterar o argumento `tag` para puppy:

```
mutation addTag {
  addTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

9. Escolha Executar (o botão de reprodução laranja) e selecione addTag. A publicação é atualizada com a nova tag:

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog",
        "puppy"
      ]
    }
  }
}
```

10. Também é possível excluir tags. No painel Consultas, adicione a seguinte mutação. Também será necessário atualizar o argumento id para o valor anotado anteriormente:

```
mutation removeTag {
  removeTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

11. Escolha Executar (o botão de reprodução laranja) e selecione removeTag. A publicação é atualizada e a tag puppy é excluída.

```
{
  "data": {
    "addTag": {
```

```
    "id": "10",
    "title": "The cutest dog in the world",
    "tags": [
      "dog"
    ]
  }
}
```

12. Também é possível pesquisar todas as publicações com uma tag. No painel Consultas, cole a seguinte consulta:

```
query allPostsByTag {
  allPostsByTag(tag: "dog") {
    posts {
      id
      title
      tags
    }
    nextToken
  }
}
```

13. Escolha Executar (o botão de reprodução laranja) e selecione `allPostsByTag`. Todas as publicações com a tag `dog` são retornadas da seguinte forma:

```
{
  "data": {
    "allPostsByTag": {
      "posts": [
        {
          "id": "10",
          "title": "The cutest dog in the world",
          "tags": [
            "dog",
            "puppy"
          ]
        }
      ],
      "nextToken": null
    }
  }
}
```



```
}
```

Conclusão

Neste tutorial, você criou uma API que permite manipular objetos Post no DynamoDB usando AWS AppSync e GraphQL.

Para limpar, você pode excluir a API AWS AppSync GraphQL do console.

Para excluir a função associada à tabela do DynamoDB, selecione sua fonte de dados na tabela Fontes de dados e clique em editar. Observe o valor da função em Criar ou usar uma função existente. Acesse o console do IAM, para excluir a função.

Para excluir sua tabela do DynamoDB, clique no nome da tabela na lista de fontes de dados. Isso leva você ao console do DynamoDB, onde você pode excluir a tabela.

Usando AWS Lambda resolvedores em AWS AppSync

Você pode usar AWS Lambda with AWS AppSync para resolver qualquer campo do GraphQL. Por exemplo, uma consulta do GraphQL pode enviar uma chamada para uma instância do Amazon Relational Database Service (Amazon RDS), e uma mutação do GraphQL pode ser gravada em um stream do Amazon Kinesis. Nesta seção, mostraremos como escrever uma função do Lambda que executa lógica de negócios com base na invocação de uma operação de campo do GraphQL.

Criar uma função do Lambda

O exemplo a seguir mostra uma função do Lambda escrita em Node.js (runtime: Node.js 18.x) que executa diferentes operações em publicações de blog como parte de um aplicativo de publicações de blog. Observe que o código deve ser salvo em um nome de arquivo com extensão .mis.

```
export const handler = async (event) => {
  console.log('Received event {}'.format(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)
  }
}

```

Essa função do Lambda recupera uma publicação por ID, adiciona uma publicação, recupera uma lista de publicações e busca publicações relacionadas para determinada publicação.

Note

A função do Lambda usa a instrução `switch` em `event.field` para determinar qual campo está sendo resolvido no momento.

Crie essa função Lambda usando o AWS Management Console.

Configurar a fonte de dados para o Lambda

Depois de criar a função do Lambda, navegue até a API GraphQL no console AWS AppSync e escolha a guia Fontes de dados.

Escolha Criar fonte de dados, insira um Nome de fonte de dados fácil de usar (por exemplo, **Lambda**) e, em seguida, para Tipo de fonte de dados, escolha Função AWS Lambda . Em Região, escolha a mesma região da sua função. Para ARN da função, escolha o nome do recurso da Amazon (ARN) da sua função do Lambda.

Depois de escolher sua função Lambda, você pode criar uma nova função AWS Identity and Access Management (IAM) (para a qual AWS AppSync atribui as permissões apropriadas) ou escolher uma função existente que tenha a seguinte política embutida:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": "arn:aws:lambda:REGION:ACCOUNTNUMBER:function/
LAMBDA_FUNCTION"
    }
  ]
}
```

Você também deve configurar uma relação de confiança com AWS AppSync a função do IAM da seguinte forma:

JSON

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

Criar um esquema do GraphQL

Agora que a fonte de dados está conectada à função do Lambda, crie um esquema do GraphQL.

No editor de esquemas no AWS AppSync console, certifique-se de que seu esquema corresponda ao seguinte esquema:

```
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]
  }
```

Configurar resolvedores

Agora que você registrou uma fonte de dados do Lambda e um esquema do GraphQL válido, pode conectar seus campos do GraphQL à sua fonte de dados do Lambda usando resolvedores.

Você criará um resolvedor que usa o tempo de execução AWS AppSync JavaScript (APPSYNC_JS) e interagirá com suas funções do Lambda. Para saber mais sobre como escrever AWS AppSync resolvedores e funções com JavaScript, consulte [recursos de JavaScript tempo de execução para resolvedores e funções](#).

Para obter mais informações sobre os modelos de mapeamento do Lambda, consulte a [referência da função de JavaScript resolução para o Lambda](#).

Nesta etapa, você anexa um resolvedor à função do Lambda para os seguintes campos: `getPost(id:ID!): Post`, `allPosts: [Post]`, `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!` e `Post.relatedPosts: [Post]`. No editor de esquemas no AWS AppSync console, no painel Resolvedores, escolha Anexar ao lado do campo. `getPost(id:ID!): Post` Escolha sua fonte de dados do Lambda. Depois, forneça o seguinte código:

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

export function request(ctx) {
  const {source, args} = ctx
  return {
    operation: 'Invoke',
    payload: { field: ctx.info.fieldName, arguments: args, source },
  };
}

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

Esse código do resolvidor passa o nome do campo, a lista de argumentos e o contexto sobre o objeto de origem para a função do Lambda quando ela o invoca. Escolha Salvar.

Você anexou seu primeiro resolvidor com sucesso. Repita essa operação para os campos restantes.

Teste da sua API GraphQL

Agora que a função do Lambda está conectada aos resolvidores do GraphQL, você pode executar algumas mutações e consultas usando o console ou um aplicativo cliente.

No lado esquerdo do AWS AppSync console, escolha Consultas e cole o código a seguir:

Mutação addPost

```
mutation AddPost {  
  addPost(  
    id: 6  
    author: "Author6"  
    title: "Sixth book"  
    url: "https://www.amazon.com/"  
    content: "This is the book is a tutorial for using GraphQL with AWS AppSync."  
  ) {  
    id  
    author  
    title  
    content  
    url  
    ups  
    downs  
  }  
}
```

Consulta getPost

```
query GetPost {  
  getPost(id: "2") {  
    id  
    author  
    title  
    content  
    url  
  }  
}
```

```
        ups
        downs
    }
}
```

Consulta allPosts

```
query AllPosts {
  allPosts {
    id
    author
    title
    content
    url
    ups
    downs
    relatedPosts {
      id
      title
    }
  }
}
```

Retornar erros

Qualquer resolução de campo determinada pode resultar em um erro. Com AWS AppSync, você pode gerar erros das seguintes fontes:

- Manipulador de respostas do resolvedor
- Função do Lambda

No manipulador de respostas do resolvedor

Para gerar erros intencionais, você pode usar o método utilitário `util.error`. Ele utiliza como argumento uma `errorMessage`, um `errorType` e um valor `data` opcional. O `data` é útil para retornar dados adicionais de volta ao cliente quando ocorre um erro. O objeto `data` é adicionado aos `errors` na resposta final do GraphQL.

O exemplo a seguir mostra como usá-lo no manipulador de respostas do resolvedor `Post.relatedPosts: [Post]`.

```
// the Post.relatedPosts response handler
export function response(ctx) {
  util.error("Failed to fetch relatedPosts", "LambdaFailure", ctx.result)
  return ctx.result;
}
```

Isso produz uma resposta do GraphQL semelhante à seguinte:

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



```

    ]
  }
]
}

```

Onde `allPosts[0].relatedPosts` é nulo pois o erro e a `errorMessage`, o `errorType` e o `data` estão presentes no objeto `data.errors[0]`.

A partir da função do Lambda

AWS AppSync também compreende os erros que a função Lambda gera. O modelo de programação do Lambda permite gerar erros processados. Se a função Lambda gerar um erro, AWS AppSync não conseguirá resolver o campo atual. Somente a mensagem de erro retornada a partir do Lambda é definida na resposta. No momento, não é possível enviar quaisquer dados adicionais de volta para o cliente ao gerar um erro a partir da função do Lambda.

Note

Se sua função Lambda gerar um erro não tratado, AWS AppSync use a mensagem de erro que o Lambda definiu.

A seguinte função do Lambda gera um erro:

```

export const handler = async (event) => {
  console.log('Received event {}'.format(JSON.stringify(event, 3)))
  throw new Error('I always fail.')
}

```

O erro é recebido em seu manipulador de respostas. Você pode enviá-lo de volta na resposta do GraphQL anexando o erro à resposta com `util.appendError`. Para fazer isso, altere seu manipulador de resposta de AWS AppSync função para isso:

```

// the lambdaInvoke response handler
export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type, result);
  }
  return result;
}

```

```
}
```

Isso retorna uma resposta do GraphQL semelhante à seguinte:

```
{
  "data": {
    "allPosts": null
  },
  "errors": [
    {
      "path": [
        "allPosts"
      ],
      "data": null,
      "errorType": "Lambda:Unhandled",
      "errorInfo": null,
      "locations": [
        {
          "line": 2,
          "column": 3,
          "sourceName": null
        }
      ],
      "message": "I fail. always"
    }
  ]
}
```

Caso de uso avançado: agrupamento em lotes

Neste exemplo, a função do Lambda tem um campo `relatedPosts` que retorna uma lista de publicações relacionadas para uma publicação. Nas consultas de exemplo, a invocação do campo `allPosts` a partir da função do Lambda retorna cinco publicações. Como também especificamos que queremos resolver `relatedPosts` para cada publicação retornada, a operação do campo `relatedPosts` será invocada cinco vezes.

```
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
    }
  }
}

```

Embora isso possa não parecer substancial neste exemplo específico, essa busca excessiva combinada pode prejudicar rapidamente o aplicativo.

Se você buscasse `relatedPosts` novamente nas `Posts` relacionadas retornadas na mesma consulta, o número de invocações aumentaria drasticamente.

```

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
        id
        title
        author
      }
    }
  }
}

```

Nessa consulta relativamente simples, AWS AppSync invocaria a função Lambda $1 + 5 + 25 = 31$ vezes.

Esse é um desafio bastante comum e normalmente é chamado de problema N+1 (nesse caso, $N = 5$) e pode incorrer em maior latência e mais custo para o aplicativo.

Uma forma de resolver esse problema é agrupar solicitações do resolvidor de campo semelhantes em lotes. Nesse exemplo, em vez da função do Lambda resolver uma lista de publicações relacionadas para uma única publicação, ela resolveria uma lista de publicações relacionadas para um determinado lote de publicações.

Para demonstrar isso, vamos atualizar o resolvidor para `relatedPosts` lidar com lotes.

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

export function request(ctx) {
  const {source, args} = ctx
  return {
    operation: ctx.info.fieldName === 'relatedPosts' ? 'BatchInvoke' : 'Invoke',
    payload: { field: ctx.info.fieldName, arguments: args, source },
  };
}

export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type, result);
  }
  return result;
}
```

O código agora altera a operação de `Invoke` para `BatchInvoke` quando `fieldName` está sendo resolvido é `relatedPosts`. Agora, habilite o lote na função na seção `Configurar lotes`. Defina o tamanho máximo do lote definido como 5. Escolha `Salvar`.

Com essa alteração, ao resolver `relatedPosts`, a função do Lambda recebe como entrada o seguinte:

```
[
  {
    "field": "relatedPosts",
    "source": {
      "id": 1
    }
  },
  {
    "field": "relatedPosts",
    "source": {
```

```

      "id": 2
    }
  },
  ...
]

```

Quando BatchInvoke for especificado na solicitação, a função do Lambda recebe uma lista de solicitações e retorna uma lista de resultados.

Especificamente, a lista de resultados deve corresponder ao tamanho e à ordem das entradas da carga útil da solicitação para que AWS AppSync possa corresponder aos resultados adequadamente.

Neste exemplo de processamento em lotes, a função do Lambda retorna um lote de resultados da seguinte forma:

```

[
  [{"id":"2","title":"Second book"}, {"id":"3","title":"Third book"}], //
  relatedPosts for id=1
  [{"id":"3","title":"Third book"}] //
  relatedPosts for id=2
]

```

Você pode atualizar seu código do Lambda para lidar com lotes para relatedPosts:

```

export const handler = async (event) => {
  console.log('Received event {}', JSON.stringify(event, 3))
  //throw new Error('I fail. always')

  const posts = {
    1: { id: '1', title: 'First book', author: 'Author1', url: 'https://amazon.com/',
      content: 'SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT
AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1', ups: '100', downs: '10', },
    2: { id: '2', title: 'Second book', author: 'Author2', url: 'https://amazon.com',
      content: 'SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT', ups: '100', downs:
'10', },
    3: { id: '3', title: 'Third book', author: 'Author3', url: null, content: null,
      ups: null, downs: null },
    4: { id: '4', title: 'Fourth book', author: 'Author4', url: 'https://
www.amazon.com/', content: 'SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
AUTHOR 4 SAMPLE TEXT AUTHOR 4', ups: '1000', downs: '0', },

```

```

    5: { id: '5', title: 'Fifth book', author: 'Author5', url: 'https://
www.amazon.com/', content: 'SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT
AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT', ups: '50', downs: '0', },
  }

const relatedPosts = {
  1: [posts['4']],
  2: [posts['3'], posts['5']],
  3: [posts['2'], posts['1']],
  4: [posts['2'], posts['1']],
  5: [],
}

if (!event.field && event.length){
  console.log(`Got a BatchInvoke Request. The payload has ${event.length} items to
resolve.`);
  return event.map(e => relatedPosts[e.source.id])
}

console.log('Got an Invoke Request.')
let result
switch (event.field) {
  case 'getPost':
    return posts[event.arguments.id]
  case 'allPosts':
    return Object.values(posts)
  case 'addPost':
    // return the arguments back
    return event.arguments
  case 'addPostErrorWithData':
    result = posts[event.arguments.id]
    // attached additional error information to the post
    result.errorMessage = 'Error with the mutation, data has changed'
    result.errorType = 'MUTATION_ERROR'
    return result
  case 'relatedPosts':
    return relatedPosts[event.source.id]
  default:
    throw new Error('Unknown field, unable to resolve ' + event.field)
}
}

```

Retornar erros individuais

Os exemplos anteriores mostram que é possível retornar um único erro da função do Lambda ou gerar um erro do seu manipulador de respostas. Para invocações em lote, gerar um erro a partir da função do Lambda sinaliza um lote inteiro como falho. Isso pode ser aceitável em cenários específicos onde ocorreu um erro irrecuperável, como uma conexão com falha a um armazenamento de dados. No entanto, nos casos em que alguns itens no lote são bem-sucedidos e outros falham, é possível retornar ambos os erros e os dados válidos. Como AWS AppSync exige a resposta em lote para listar elementos que correspondam ao tamanho original do lote, você deve definir uma estrutura de dados que possa diferenciar dados válidos de um erro.

Por exemplo, se espera-se que a função do Lambda retorne um lote de publicações relacionadas, você pode, em vez disso, retornar uma lista de objetos Response em que cada objeto possui campos opcionais de dados, errorMessage e errorType. Se o campo errorMessage estiver presente, isso significa que ocorreu um erro.

O código a seguir mostra como você pode atualizar a função do Lambda:

```
export const handler = async (event) => {
  console.log('Received event {}', JSON.stringify(event, 3))
  // throw new Error('I fail. always')
  const posts = [
    1: { id: '1', title: 'First book', author: 'Author1', url: 'https://amazon.com/',
      content: 'SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT
      AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1', ups: '100', downs: '10', },
    2: { id: '2', title: 'Second book', author: 'Author2', url: 'https://amazon.com',
      content: 'SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT', ups: '100', downs:
      '10', },
    3: { id: '3', title: 'Third book', author: 'Author3', url: null, content: null,
      ups: null, downs: null },
    4: { id: '4', title: 'Fourth book', author: 'Author4', url: 'https://
www.amazon.com/', content: 'SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT
AUTHOR 4 SAMPLE TEXT AUTHOR 4', ups: '1000', downs: '0', },
    5: { id: '5', title: 'Fifth book', author: 'Author5', url: 'https://
www.amazon.com/', content: 'SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT
AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT', ups: '50', downs: '0', },
  ]

  const relatedPosts = {
    1: [posts['4']],
    2: [posts['3'], posts['5']],
  }
```

```

    3: [posts['2'], posts['1']],
    4: [posts['2'], posts['1']],
    5: [],
  }

  if (!event.field && event.length){
    console.log(`Got a BatchInvoke Request. The payload has ${event.length} items to
    resolve.`);
    return event.map(e => {
    // return an error for post 2
    if (e.source.id === '2') {
    return { 'data': null, 'errorMessage': 'Error Happened', 'errorType': 'ERROR' }
    }
    return {data: relatedPosts[e.source.id]}
    })
  }

  console.log('Got an Invoke Request.')
  let result
  switch (event.field) {
case 'getPost':
  return posts[event.arguments.id]
case 'allPosts':
  return Object.values(posts)
case 'addPost':
  // return the arguments back
return event.arguments
case 'addPostErrorWithData':
  result = posts[event.arguments.id]
  // attached additional error information to the post
  result.errorMessage = 'Error with the mutation, data has changed'
  result.errorType = 'MUTATION_ERROR'
return result
case 'relatedPosts':
  return relatedPosts[event.source.id]
default:
  throw new Error('Unknown field, unable to resolve ' + event.field)
  }
}

```

Atualize o código do resolvedor relatedPosts:

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



```
export function request(ctx) {
  const {source, args} = ctx
  return {
    operation: ctx.info.fieldName === 'relatedPosts' ? 'BatchInvoke' : 'Invoke',
    payload: { field: ctx.info.fieldName, arguments: args, source },
  };
}

export function response(ctx) {
  const { error, result } = ctx;
  if (error) {
    util.appendError(error.message, error.type, result);
  } else if (result.errorMessage) {
    util.appendError(result.errorMessage, result.errorType, result.data)
  } else if (ctx.info.fieldName === 'relatedPosts') {
    return result.data
  } else {
    return result
  }
}
```

O manipulador de respostas agora verifica erros retornados pela função do Lambda nas operações Invoke, verifica erros retornados para itens individuais das operações BatchInvoke e, por fim, verifica `fieldName`. Para `relatedPosts`, a função retorna `result.data`. Para todos os outros campos, a função retorna apenas `result`. Por exemplo, veja a consulta abaixo:

```
query AllPosts {
  allPosts {
    id
    title
    content
    url
    ups
    downs
    relatedPosts {
      id
    }
    author
  }
}
```

Essa consulta retorna uma resposta do GraphQL semelhante à seguinte:

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

Configuração do tamanho máximo do lote

Para configurar o tamanho máximo de lotes em um resolvidor, use o seguinte comando no 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

Ao fornecer um modelo de mapeamento de solicitação, você deve usar a operação `BatchInvoke` para usar lotes.

Usando resolvedores locais em AWS AppSync

AWS AppSync permite que você use fontes de dados compatíveis (AWS Lambda Amazon DynamoDB ou OpenSearch Amazon Service) para realizar várias operações. No entanto, em determinados cenários, uma chamada para uma fonte de dados compatível pode não ser necessária.

É aí que o resolvedor local se torna útil. Em vez de chamar uma fonte de dados remota, o resolvedor local apenas encaminhará o resultado do manipulador de solicitação para o manipulador de resposta. A resolução do campo não deixará o AWS AppSync.

Os resolvedores locais são úteis em uma infinidade de situações. O caso de uso mais popular é a publicação de notificações sem acionar uma chamada da fonte de dados. Para demonstrar esse caso de uso, vamos criar um aplicativo pub/sub no qual os usuários possam publicar e assinar mensagens. Esse exemplo utiliza Assinaturas, portanto se não estiver familiarizado com Assinaturas, siga o tutorial [Dados em tempo real](#).

Criação do aplicativo pub/sub

Primeiro, crie uma API GraphQL em branco escolhendo a opção Design do zero e configurando os detalhes opcionais ao criar sua API GraphQL.

Em nosso aplicativo pub/sub, os clientes podem assinar e publicar mensagens. Cada mensagem publicada inclui um nome e dados. Adicione-o ao esquema:

```
type Channel {
  name: String!
  data: AWSJSON!
}

type Mutation {
  publish(name: String!, data: AWSJSON!): Channel
}

type Query {
  getChannel: Channel
}

type Subscription {
  subscribe(name: String!): Channel
  @aws_subscribe(mutations: ["publish"])
```

```
}
```

Depois, vamos anexar um resolvedor ao campo `Mutation.publish`. No painel Resolvedores próximo ao painel Esquema, encontre o tipo `Mutation`, o campo `publish(...): Channel` e clique em Anexar.

Crie uma fonte de dados `None` e nomeie-a `PageDataSource`. Anexe-a ao seu resolvedor.

Adicione a implementação do seu resolvedor usando o seguinte trecho:

```
export function request(ctx) {  
  return { payload: ctx.args };  
}  
  
export function response(ctx) {  
  return ctx.result;  
}
```

Certifique-se de criar o resolvedor e salvar as alterações feitas.

Enviar e assinar mensagens

Para que os clientes recebam mensagens, primeiro eles devem assinar uma caixa de entrada.

No painel Consultas, execute a assinatura `SubscribeToData`:

```
subscription SubscribeToData {  
  subscribe(name:"channel") {  
    name  
    data  
  }  
}
```

O assinante receberá mensagens sempre que a mutação `publish` for invocada, mas somente quando a mensagem for enviada para a assinatura `channel`. Vamos tentar isso no painel Consultas. Enquanto sua assinatura ainda estiver em execução no console, abra outro console e execute a seguinte solicitação no painel Consultas:

Note

Estamos usando strings JSON válidas neste exemplo.

```
mutation PublishData {  
  publish(data: "{\"msg\": \"hello world!\"}", name: "channel") {  
    data  
    name  
  }  
}
```

O resultado será semelhante a este:

```
{  
  "data": {  
    "publish": {  
      "data": "{\"msg\": \"hello world!\"}",  
      "name": "channel"  
    }  
  }  
}
```

Acabamos de demonstrar o uso de resolvedores locais, publicando uma mensagem e recebendo-a sem sair do AWS AppSync serviço.

Combinação de resolvedores do GraphQL no AWS AppSync

Os resolvedores e campos em um esquema do GraphQL têm relacionamentos 1:1 com um grande grau de flexibilidade. Como uma fonte de dados é configurada em um resolvedor independentemente de um esquema, você tem a capacidade de resolver ou manipular seus tipos do GraphQL por meio de diferentes fontes de dados, permitindo misturar e combinar um esquema para melhor atender às suas necessidades.

Os cenários a seguir demonstram como misturar e combinar fontes de dados no seu esquema. Antes de começar, você deve estar familiarizado com a configuração de fontes de dados e resolvedores para o Amazon AWS Lambda DynamoDB e o Amazon Service. OpenSearch

Esquema de exemplo

O esquema a seguir tem um tipo de Post com três Query e operações Mutation cada:

```
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
  }
```

Neste exemplo, você teria um total de seis resolvedores, cada um precisando de uma fonte de dados. Uma forma de resolver esse problema seria conectá-los a uma única tabela do Amazon DynamoDB, Posts chamada, na qual o campo executa uma varredura e AllPost o campo executa uma consulta ([JavaScriptconsulte searchPosts a referência da função de resolução](#) do DynamoDB). No entanto, você não está limitado ao Amazon DynamoDB; existem fontes de dados diferentes, como Lambda OpenSearch ou Service, para atender às suas necessidades comerciais.

Alteração de dados por meio de resolvedores

Talvez seja necessário retornar resultados de um banco de dados de terceiros que não seja diretamente suportado por fontes AWS AppSync de dados. Talvez você também precise realizar modificações complexas nos dados antes que eles sejam retornados aos clientes da API. Isso pode ser causado pela formatação inadequada dos tipos de dados, como diferenças de carimbo de data/hora nos clientes ou pelo tratamento de problemas de compatibilidade com versões anteriores. Nesse caso, conectar AWS Lambda funções como fonte de dados à sua AWS AppSync API é a solução apropriada. Para fins ilustrativos, no exemplo a seguir, uma AWS Lambda função manipula dados obtidos de um armazenamento de dados de terceiros:

```
export const handler = (event, context, callback) => {  
  // fetch data  
  const result = fetcher()  
  
  // apply complex business logic  
  const data = transform(result)  
  
  // return to AppSync  
  return data  
};
```

Essa é uma função do Lambda perfeitamente válida e pode ser anexada ao campo `AllPost` no esquema do GraphQL para que qualquer consulta que retorne todos os resultados obtenha números aleatórios para os votos positivos/negativos.

DynamoDB e serviços OpenSearch

Para alguns aplicativos, você pode realizar mutações ou consultas de pesquisa simples no DynamoDB e fazer com que um processo em segundo plano transfira documentos para o Service. OpenSearch Você pode simplesmente anexar o `searchPosts` resolvedor à fonte de dados do OpenSearch Service e retornar os resultados da pesquisa (de dados originados no DynamoDB) usando uma consulta GraphQL. Isso pode ser extremamente poderoso ao adicionar operações de pesquisa avançadas aos aplicativos, como palavra-chave, correspondências de palavras confusas ou até mesmo pesquisas geoespaciais. A transferência de dados do DynamoDB pode ser feita por meio de um processo ETL ou, como alternativa, você pode transmitir do DynamoDB usando o Lambda.

Para começar a usar essas fontes de dados específicas, consulte nossos tutoriais do [DynamoDB](#) e [Lambda](#).

Por exemplo, usando o esquema do nosso tutorial anterior, a seguinte mutação adiciona um item ao DynamoDB:

```
mutation addPost {
  addPost(
    id: 123
    author: "Nadia"
    title: "Our first post!"
    content: "This is our first post."
    url: "https://aws.amazon.com/appsync/"
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

Isso grava dados no DynamoDB, que então transmite dados via Lambda para o OpenSearch Amazon Service, que você usa para pesquisar publicações em campos diferentes. Por exemplo, como os dados estão no Amazon OpenSearch Service, você pode pesquisar os campos do autor ou do conteúdo com texto de formato livre, mesmo com espaços, da seguinte forma:

```
query searchName{
  searchAuthor(name:"  Nadia  "){
    id
    title
    content
  }
}

----- or -----

query searchContent{
  searchContent(text:"test"){
```

```
        id
        title
        content
    }
}
```

Como os dados são gravados diretamente no DynamoDB, você ainda pode realizar operações de pesquisa de item ou lista eficientes na tabela com as consultas `allPost{...}` e `getPost{...}`. Essa pilha usa o seguinte código de exemplo para transmissões do DynamoDB:

Note

Este código Python é um exemplo e não deve ser usado em código de produção.

```
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.'
```

Em seguida, você pode usar os streams do DynamoDB para anexá-los a uma tabela do DynamoDB com uma chave primária de, e qualquer alteração na origem do DynamoDB será id transmitida para o seu domínio de serviço. OpenSearch Para obter mais informações sobre como configurar isso, consulte a [documentação de Transmissões do DynamoDB](#).

Usando resolvedores OpenSearch do Amazon Service em AWS AppSync

AWS AppSync suporta o uso do Amazon OpenSearch Service a partir de domínios que você provisionou em sua própria AWS conta, desde que eles não existam dentro de uma VPC. Assim que os domínios forem provisionados, conecte-se a eles usando uma fonte de dados, no momento em que pode configurar um resolvedor no esquema para realizar operações do GraphQL, como consultas, mutações e assinaturas. Esse tutorial apresentará alguns exemplos comuns.

Para obter mais informações, consulte nossa [referência JavaScript de função de resolução para OpenSearch](#).

Criar um novo domínio OpenSearch de serviço

Para começar com este tutorial, você precisa de um domínio OpenSearch de serviço existente. Caso não tenha um, use o exemplo a seguir. Observe que pode levar até 15 minutos para que um domínio OpenSearch de serviço seja criado antes que você possa prosseguir com a integração com uma fonte de AWS AppSync dados.

```
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
```

Você pode lançar a seguinte AWS CloudFormation pilha na região US-West-2 (Oregon) em sua conta: AWS

Launch Stack

Configurar uma fonte de dados para o OpenSearch serviço

Depois que o domínio OpenSearch Service for criado, navegue até a API do AWS AppSync GraphQL e escolha a guia Fontes de dados. Escolha Criar fonte de dados e insira um nome amigável para a fonte de dados, como “**oss**”. Em seguida, escolha o OpenSearch domínio da Amazon para o tipo de fonte de dados, escolha a região apropriada e você deverá ver seu domínio OpenSearch de serviço listado. Depois de selecioná-la, você pode criar uma nova função e atribuir AWS AppSync as permissões apropriadas à função ou escolher uma função existente, que tenha a seguinte política embutida:

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:REGION:ACCOUNTNUMBER:domain/democluster/*"
      ]
    }
  ]
}
```

Você também precisará estabelecer uma relação de confiança com AWS AppSync essa função:

JSON

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

Além disso, o domínio do OpenSearch serviço tem sua própria política de acesso que você pode modificar por meio do console do Amazon OpenSearch Service. Você deve adicionar uma política semelhante à abaixo com as ações e os recursos apropriados para o domínio do OpenSearch Serviço. Observe que o principal será a função da fonte de AWS AppSync dados, que pode ser encontrada no console do IAM se você permitir que o console a crie.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::ACCOUNTNUMBER:role/service-role/
APPSYNC_DATASOURCE_ROLE"
      },
      "Action": [
        "es:ESHttpDelete",
        "es:ESHttpHead",
        "es:ESHttpGet",
        "es:ESHttpPost",
        "es:ESHttpPut"
      ],
      "Resource": "arn:aws:es:REGION:ACCOUNTNUMBER:domain/DOMAIN_NAME/*"
    }
  ]
}
```

```
}  
]  
}
```

Conexão de um resolvedor

Agora que a fonte de dados está conectada ao seu domínio de OpenSearch serviço, você pode conectá-la ao seu esquema do GraphQL com um resolvedor, conforme mostrado no exemplo a seguir:

```
type Query {  
  getPost(id: ID!): Post  
  allPosts: [Post]  
}  
  
type Mutation {  
  addPost(id: ID!, author: String, title: String, url: String, ups: Int, downs: Int,  
  content: String): AWSJSON  
}  
  
type Post {  
  id: ID!  
  author: String  
  title: String  
  url: String  
  ups: Int  
  downs: Int  
  content: String  
}
```

Observe que há um tipo `Post` definido pelo usuário com um campo de `id`. Nos exemplos a seguir, presumimos que há um processo (que pode ser automatizado) para colocar esse tipo em seu domínio de OpenSearch serviço, que seria mapeado para uma raiz de caminho de `/post/_doc` onde `post` está o índice. A partir desse caminho raiz, você pode executar pesquisas de documento individuais, pesquisas com curingas com `/id/post*` ou pesquisas de vários documentos com um caminho de `/post/_search`. Por exemplo, se você tiver outro tipo chamado `User`, poderá indexar documentos em um novo índice chamado `user` e depois realizar pesquisas com um caminho de `/user/_search`.

No editor de esquemas no AWS AppSync console, modifique o Posts esquema anterior para incluir uma consulta: `searchPosts`

```
type Query {  
  getPost(id: ID!): Post  
  allPosts: [Post]  
  searchPosts: [Post]  
}
```

Salve o esquema. No painel Resolvedores, encontre `searchPosts` e escolha Anexar. Escolha sua fonte OpenSearch de dados de serviço e salve o resolvedor. Atualize o código do seu resolvedor usando o trecho abaixo:

```
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)  
}
```

Isso pressupõe que o esquema anterior tenha documentos que foram indexados no OpenSearch Serviço sob o campo. `post`. Se você estruturar os dados de forma diferente, será necessário atualizar de forma adequada.

Modificação das pesquisas

O manipulador de solicitação do resolvedor anterior executa uma consulta simples para todos os registros. Digamos que você queira pesquisar um autor específico. Além disso, digamos que queira que esse autor seja um argumento definido na consulta do GraphQL. No editor de esquemas do AWS AppSync console, adicione uma `allPostsByAuthor` consulta:

```
type Query {  
  getPost(id: ID!): Post  
  allPosts: [Post]  
  allPostsByAuthor(author: String!): [Post]  
  searchPosts: [Post]  
}
```

No painel Resolvedores, encontre `allPostsByAuthor` e escolha Anexar. Escolha a fonte OpenSearch de dados do serviço e use o código a seguir:

```
import { util } from '@aws-appsync/utils'  
  
/**  
 * Searches for documents by `author`  
 * @param {import('@aws-appsync/utils').Context} ctx the context  
 * @returns {*} the request  
 */  
export function request(ctx) {  
  return {  
    operation: 'GET',  
    path: '/post/_search',  
    params: {  
      body: {  
        from: 0,  
        size: 50,  
        query: { match: { author: ctx.args.author } },  
      },  
    },  
  },  
}
```



```
/**
 * Returns the fetched items
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the result
 */
export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type)
  }
  return ctx.result.hits.hits.map((hit) => hit._source)
}
```

Observe que o body é preenchido com uma consulta de termo para o campo `author`, que é enviada a partir do cliente como um argumento. Também é possível usar informações pré-preenchidas, como texto padrão.

Adicionando dados ao OpenSearch serviço

Talvez você queira adicionar dados ao seu domínio de OpenSearch serviço como resultado de uma mutação do GraphQL. Esse é um poderoso mecanismo para pesquisa e outras finalidades. Como você pode usar assinaturas do GraphQL para [tornar seus dados em tempo real](#), ele pode servir como um mecanismo para notificar os clientes sobre atualizações nos dados em seu domínio de serviço. OpenSearch

Volte para a página Esquema no AWS AppSync console e selecione Anexar para a `addPost()` mutação. Selecione a fonte OpenSearch de dados do serviço novamente e use o código a seguir:

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

/**
 * Searches for documents by `author`
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the request
 */
export function request(ctx) {
  return {
    operation: 'PUT',
    path: `/post/_doc/${ctx.args.id}`,
    params: { body: ctx.args },
  }
}
```

```
/**
 * Returns the inserted post
 * @param {import('@aws-appsync/utils').Context} ctx the context
 * @returns {*} the result
 */
export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type)
  }
  return ctx.result
}
```

Como antes, este é um exemplo de como os dados podem ser estruturados. Se tiver diferentes nomes de campos ou índices, é necessário atualizar o path e o body. Este exemplo também mostra como usar `context.arguments`, que também pode ser escrito como `ctx.args`, em seu manipulador de solicitações.

Recuperação de um único documento

Por fim, se você quiser usar a **`getPost(id:ID)`** consulta em seu esquema para retornar um documento individual, localize essa consulta no editor de esquemas do AWS AppSync console e escolha Anexar. Selecione a fonte OpenSearch de dados do serviço novamente e use o código a seguir:

```
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
}
```

Executar consultas e mutações

Agora você deve ser capaz de realizar operações do GraphQL em seu domínio OpenSearch Service. Navegue até a guia Consultas do AWS AppSync console e adicione um novo registro:

```
mutation AddPost {
  addPost (
    id:"12345"
    author: "Fred"
    title: "My first book"
    content: "This will be fun to write!"
    url: "publisher website",
    ups: 100,
    downs:20
  )
}
```

Você verá o resultado da mutação à direita. Da mesma forma, agora você pode executar uma `searchPosts` consulta em seu domínio OpenSearch de serviço:

```
query search {
  searchPosts {
    id
    title
    author
    content
  }
}
```

Práticas recomendadas

- **OpenSearch** O serviço deve ser para consultar dados, não como seu banco de dados principal. [Talvez você queira usar o OpenSearch Service em conjunto com o Amazon DynamoDB, conforme descrito em Combinação de resolvedores GraphQL.](#)
- Só conceda acesso ao seu domínio permitindo que a função AWS AppSync de serviço acesse o cluster.
- Você pode começar pequeno no desenvolvimento, com o cluster de menor custo e, em seguida, migrar para um cluster maior com alta disponibilidade (HA) à medida que entrar na produção.

Executando transações do DynamoDB no AWS AppSync

AWS AppSync suporta o uso de operações de transação do Amazon DynamoDB em uma ou mais tabelas em uma única região. As operações compatíveis são `TransactGetItems` e `TransactWriteItems`. Ao usar esses recursos no AWS AppSync, você pode realizar tarefas como:

- Enviar uma lista de chaves em uma única consulta e retornar os resultados de uma tabela
- Ler os registros de uma ou mais tabelas em uma única consulta
- Gravando registros em transações em uma ou mais tabelas de uma all-or-nothing forma
- Executar transações quando algumas condições são atendidas

Permissões

Como outros resolvedores, você precisa criar uma fonte de dados AWS AppSync e criar uma função ou usar uma existente. Como operações de transação exigem diferentes permissões em tabelas do DynamoDB, é necessário conceder as permissões de função configuradas para ações de leitura e gravação:

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:region:accountId:table/TABLENAME",
        "arn:aws:dynamodb:region:accountId:table/TABLENAME/*"
    ]
}

```

Note

As funções são vinculadas às fontes de dados em AWS AppSync, e os resolvedores nos campos são invocados em uma fonte de dados. As fontes de dados configuradas para buscar no DynamoDB têm apenas uma tabela especificada, para manter as configurações simples. Portanto, ao executar uma operação de transação para várias tabelas em um único resolvedor, que é uma tarefa mais avançada, é necessário conceder acesso à função na fonte de dados para qualquer tabela com a qual o resolvedor interage. Isso é feito no campo **Recurso** na política do IAM acima. A configuração das chamadas de transação nas tabelas é feita no código de resolvedor, descrita abaixo.

Fonte de dados

Para simplificar, usaremos a mesma fonte de dados para todos os resolvedores usados neste tutorial.

Teremos duas tabelas chamadas `savingAccounts` e `checkingAccounts`, ambas com `accountNumber` como chave de partição, e uma tabela `transactionHistory` com `transactionId` como chave de partição. É possível usar os comandos da CLI abaixo para criar suas tabelas. Substitua `region` pela região.

Com a 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
```

No AWS AppSync console, em Fontes de dados, crie uma nova fonte de dados do DynamoDB e nomeie-a. TransactTutorial Selecione savingAccounts como tabela (a tabela especificada não importa ao usar transações). Selecione para criar uma fonte de dados. Você pode revisar a configuração da fonte de dados para ver o nome da função gerada. No console do IAM, você pode adicionar uma política em linha que permite que a fonte de dados interaja com todas as tabelas.

Substitua region e accountID por sua região e seu ID da conta:

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:region:accountId:table/savingAccounts",
      "arn:aws:dynamodb:region:accountId:table/savingAccounts/*",
      "arn:aws:dynamodb:region:accountId:table/checkingAccounts",
      "arn:aws:dynamodb:region:accountId:table/checkingAccounts/*",
      "arn:aws:dynamodb:region:accountId:table/transactionHistory",
      "arn:aws:dynamodb:region:accountId:table/transactionHistory/*"
    ]
  }
}
```

Transações

Neste exemplo, o contexto é uma transação bancária clássica, onde usaremos `TransactWriteItems` para:

- Transferir dinheiro de contas poupanças para contas correntes
- Gerar novos registros para cada transação

E, então, vamos usar `TransactGetItems` para recuperar detalhes de contas poupanças e contas correntes.

Warning

`TransactWriteItems` não é compatível quando usado com detecção e resolução de conflitos. Essas configurações devem ser desativadas para evitar possíveis erros.

Nós definimos nosso esquema GraphQL da seguinte forma:

```
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 - Preencher contas

Para transferir dinheiro entre contas, precisamos preencher a tabela com os detalhes. Usaremos a operação do GraphQL `Mutation.populateAccounts` para fazer isso.

Na seção Esquema, clique em Anexar ao lado da operação `Mutation.populateAccounts`. Escolha a fonte de dados do `TransactTutorial` e escolha Criar.

Agora use o seguinte código:

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

export function request(ctx) {
  const { savingAccounts, checkingAccounts } = ctx.args

  const savings = savingAccounts.map(({ accountNumber, ...rest }) => {
    return {
      table: 'savingAccounts',
      operation: 'PutItem',
      key: util.dynamodb.toMapValues({ accountNumber }),
      attributeValues: util.dynamodb.toMapValues(rest),
    }
  })

  const checkings = checkingAccounts.map(({ accountNumber, ...rest }) => {
    return {
      table: 'checkingAccounts',
      operation: 'PutItem',
      key: util.dynamodb.toMapValues({ accountNumber }),
      attributeValues: util.dynamodb.toMapValues(rest),
    }
  })

  return {
    version: '2018-05-29',
    operation: 'TransactWriteItems',
    transactItems: [...savings, ...checkings],
  }
}

export function response(ctx) {
  if (ctx.error) {
```

```
    util.error(ctx.error.message, ctx.error.type, null, ctx.result.cancellationReasons)
  }
  const { savingAccounts: sInput, checkingAccounts: cInput } = ctx.args
  const keys = ctx.result.keys
  const savingAccounts = sInput.map((_, i) => keys[i])
  const sLength = sInput.length
  const checkingAccounts = cInput.map((_, i) => keys[sLength + i])
  return { savingAccounts, checkingAccounts }
}
```

Salve o resolvidor e navegue até a seção Consultas do AWS AppSync console para preencher as contas.

Execute a seguinte mutação:

```
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
    }
  }
}
```

Nós preenchemos três contas poupanças e três contas correntes em uma mutação.

Use o console do DynamoDB para confirmar se os dados aparecem nas tabelas savingAccounts e checkingAccounts.

TransactWriteItems - Transferir dinheiro

Anexe um resolvidor à mutação `transferMoney` com o código a seguir. Para cada transferência, precisamos de um modificador de sucesso para contas correntes e de poupança, e precisamos rastrear a transferência nas transações.

```
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 }
}

```

Agora, navegue até a seção Consultas do AWS AppSync console e execute a mutação `TransferMoney` da seguinte forma:

```

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
    }
  }
}

```

```
}
```

Nós enviamos três transações bancárias em uma mutação. Use o console do DynamoDB para validar se os dados aparecem nas tabelas `savingAccounts`, `checkingAccounts` e `transactionHistory`.

TransactGetItems - Recuperar contas

Para recuperar os detalhes das contas poupança e corrente em uma única solicitação transacional, anexaremos um resolvidor à operação `Query.getAccount` do GraphQL em nosso esquema. Selecione Anexar e escolha a mesma fonte de dados do `TransactTutorial` criada no início do tutorial. Use o seguinte código:

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

export function request(ctx) {
  const { savingAccountNumbers, checkingAccountNumbers } = ctx.args

  const savings = savingAccountNumbers.map((accountNumber) => {
    return { table: 'savingAccounts', key: util.dynamodb.toMapValues({ accountNumber }) }
  })
  const checkings = checkingAccountNumbers.map((accountNumber) => {
    return { table: 'checkingAccounts', key:
util.dynamodb.toMapValues({ accountNumber }) }
  })
  return {
    version: '2018-05-29',
    operation: 'TransactGetItems',
    transactItems: [...savings, ...checkings],
  }
}

export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type, null, ctx.result.cancellationReasons)
  }

  const { savingAccountNumbers: sInput, checkingAccountNumbers: cInput } = ctx.args
  const items = ctx.result.items
  const savingAccounts = sInput.map((_, i) => items[i])
  const sLength = sInput.length
  const checkingAccounts = cInput.map((_, i) => items[sLength + i])
  return { savingAccounts, checkingAccounts }
```

```
}
```

Salve o resolvidor e navegue até as seções Consultas do console do AWS AppSync . Para recuperar as contas poupança e corrente, execute a seguinte consulta:

```
query getAccounts {  
  getAccounts(  
    savingAccountNumbers: ["1", "2", "3"],  
    checkingAccountNumbers: ["1", "2"]  
  ) {  
    savingAccounts {  
      accountNumber  
      username  
      balance  
    }  
    checkingAccounts {  
      accountNumber  
      username  
      balance  
    }  
  }  
}
```

Demonstramos com sucesso o uso das transações do DynamoDB usando AWS AppSync

Usando operações em lote do DynamoDB em AWS AppSync

AWS AppSync suporta o uso de operações em lote do Amazon DynamoDB em uma ou mais tabelas em uma única região. As operações compatíveis são BatchGetItem, BatchPutItem e BatchDeleteItem. Ao usar esses recursos no AWS AppSync, você pode realizar tarefas como:

- Enviar uma lista de chaves em uma única consulta e retornar os resultados de uma tabela
- Ler os registros de uma ou mais tabelas em uma única consulta
- Gravar registros em lote em uma ou mais tabelas
- Gravar ou excluir registros condicionalmente em várias tabelas que podem ter uma relação

As operações em lote AWS AppSync têm duas diferenças principais em relação às operações sem lote:

- A função da fonte de dados deve ter permissões para todas as tabelas que o resolvidor acessará.
- A especificação de tabela para um resolvidor faz parte do objeto de solicitação.

Lotes de tabela única

Warning

BatchPutItem e BatchDeleteItem não são compatíveis quando usados com detecção e resolução de conflitos. Essas configurações devem ser desativadas para evitar possíveis erros.

Para começar, vamos criar uma API GraphQL. No AWS AppSync console, escolha Create API, GraphQL APIs e Design from scratch. Nomeie sua API BatchTutorial API, escolha Próximo e, na etapa Especificar recursos do GraphQL, selecione Criar recursos do GraphQL mais tarde e clique em Próximo. Revise seus detalhes e crie a API. Acesse a página Esquema e cole o esquema a seguir, observando que, para a consulta, passaremos uma lista de 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]
}
```

Salve seu esquema e escolha Criar recursos na parte superior da página. Escolha Usar tipo existente e selecione o tipo Post. Nomeie sua tabela Posts. Certifique-se de que a Chave primária

esteja definida como `id`, desmarque Gerar GraphQL automaticamente (você fornecerá seu próprio código) e selecione Criar. Para começar, AWS AppSync cria uma tabela do DynamoDB e uma fonte de dados conectada à tabela com as funções apropriadas. No entanto, ainda há algumas permissões que você precisa adicionar à função. Acesse a página Fontes de dados e escolha a nova fonte de dados. Em Selecionar uma função existente, você notará que uma função foi criada automaticamente para a tabela. Anote a função (deve ter uma aparência parecida com `comappsync-ds-ddb-aaabbbccddd-Posts`) e, em seguida, acesse o console do IAM (<https://console.aws.amazon.com/iam/>). No console do IAM, escolha Funções e selecione sua função na tabela. Na sua função, em Políticas de permissões, clique em "+" ao lado da política (deve ter um nome semelhante ao nome da função). Escolha Editar na parte superior do expansível quando a política aparece. Você precisa adicionar permissões em lote à sua política, especificamente `dynamodb:BatchGetItem` e `dynamodb:BatchWriteItem`. Essa lista será semelhante a:

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:..."
      ]
    }
  ]
}
```

Escolha Avançar e Salvar alterações. Sua política deve permitir o processamento em lote agora.

De volta ao AWS AppSync console, acesse a página Esquema e selecione Anexar ao lado do `Mutation.batchAdd` campo. Crie seu resolvidor usando a tabela `Posts` como fonte de dados. No editor de código, substitua os manipuladores pelo trecho abaixo. Esse trecho recebe automaticamente cada item no tipo `input PostInput` do GraphQL e cria um mapa, que é necessário para a operação `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;
}
```

Navegue até a página Consultas do AWS AppSync console e execute a seguinte `batchAdd` mutação:

```
mutation add {
  batchAdd(posts:[{
    id: 1 title: "Running in the Park"},{
    id: 2 title: "Playing fetch"
  }]){
    id
    title
  }
}
```

Você deverá ver os resultados impressos na tela; isso pode ser validado revisando o console do DynamoDB para verificar os valores gravados na tabela `Posts`.

Em seguida, repita o processo de anexar um resolvedor, mas para o campo `Query.batchGet` usando a tabela `Posts` como fonte de dados. Substitua os manipuladores pelo código abaixo. Isso recebe automaticamente cada item no tipo `ids: []` do GraphQL e cria um mapa, que é necessário para a operação `BatchGetItem`:

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

export function request(ctx) {
  return {
    operation: "BatchGetItem",
    tables: {
      Posts: {
        keys: ctx.args.ids.map((id) => util.dynamodb.toMapValues({ id })),
        consistentRead: true,
      },
    },
  };
}

export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type);
  }
  return ctx.result.data.Posts;
}
```

Agora, volte para a página Consultas do AWS AppSync console e execute a seguinte `batchGet` consulta:

```
query get {
  batchGet(ids:[1,2,3]){
    id
    title
  }
}
```

Isso deve retornar os resultados para os dois valores `id` adicionados anteriormente. Observe que um valor `null` foi retornado para o `id` com um valor de 3. Isso ocorre porque não havia registro na tabela `Posts` com esse valor ainda. Observe também que AWS AppSync retorna os resultados na mesma ordem das chaves passadas para a consulta, que é um recurso adicional que AWS AppSync

funciona em seu nome. Portanto, se você alternar para `batchGet(ids: [1, 3, 2])`, verá a ordem alterada. Você também saberá qual id retornou um valor `null`.

Por fim, anexe mais um resolvidor ao campo `Mutation.batchDelete` usando a tabela `Posts` como fonte de dados. Substitua os manipuladores pelo código abaixo. Isso recebe automaticamente cada item no tipo `ids: []` do GraphQL e cria um mapa, que é necessário para a operação `BatchGetItem`:

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

export function request(ctx) {
  return {
    operation: "BatchDeleteItem",
    tables: {
      Posts: ctx.args.ids.map((id) => util.dynamodb.toMapValues({ id })),
    },
  };
}

export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type);
  }
  return ctx.result.data.Posts;
}
```

Agora, volte para a página de consultas do AWS AppSync console e execute a seguinte `batchDelete` mutação:

```
mutation delete {
  batchDelete(ids:[1,2]){ id }
}
```

Os registros com id, 1 e 2 devem ser excluídos agora. Se você executar novamente a consulta `batchGet()` de anteriormente, eles devem retornar `null`.

Lote de várias tabelas

Warning

`BatchPutItem` e `BatchDeleteItem` não são compatíveis quando usados com detecção e resolução de conflitos. Essas configurações devem ser desativadas para evitar possíveis erros.

AWS AppSync também permite que você execute operações em lote nas tabelas. Vamos criar um aplicativo mais complexo. Imagine que estamos criando um aplicativo de saúde para animais de estimação em que sensores informam a localização e a temperatura corporal do animal. Os sensores são alimentados por bateria e tentam se conectar à rede a cada cinco minutos. Quando um sensor estabelece uma conexão, ele envia suas leituras para nossa AWS AppSync API. Em seguida, os gatilhos analisam os dados para que um painel seja apresentado ao dono do animal. Vamos nos concentrar na representação das interações entre o sensor e o armazenamento de dados de back-end.

No AWS AppSync console, escolha `Create API, GraphQL APIs` e `Design from scratch`. Nomeie sua API `MultiBatchTutorial` API, escolha `Próximo` e, na etapa `Especificar recursos do GraphQL`, selecione `Criar recursos do GraphQL` mais tarde e clique em `Próximo`. Revise seus detalhes e crie a API. Vá até a página `Esquema`, cole e salve o seguinte esquema:

```
type Mutation {
  # Register a batch of readings
  recordReadings(tempReadings: [TemperatureReadingInput], locReadings:
[LocationReadingInput]): RecordResult
  # Delete a batch of readings
  deleteReadings(tempReadings: [TemperatureReadingInput], locReadings:
[LocationReadingInput]): RecordResult
}

type Query {
  # Retrieve all possible readings recorded by a sensor at a specific time
  getReadings(sensorId: ID!, timestamp: String!): [SensorReading]
}

type RecordResult {
  temperatureReadings: [TemperatureReading]
  locationReadings: [LocationReading]
```

```
}

interface SensorReading {
  sensorId: ID!
  timestamp: String!
}

# Sensor reading representing the sensor temperature (in Fahrenheit)
type TemperatureReading implements SensorReading {
  sensorId: ID!
  timestamp: String!
  value: Float
}

# Sensor reading representing the sensor location (lat,long)
type LocationReading implements SensorReading {
  sensorId: ID!
  timestamp: String!
  lat: Float
  long: Float
}

input TemperatureReadingInput {
  sensorId: ID!
  timestamp: String
  value: Float
}

input LocationReadingInput {
  sensorId: ID!
  timestamp: String
  lat: Float
  long: Float
}
```

Precisamos criar duas tabelas do DynamoDB:

- `locationReadings` armazenará as leituras de localização do sensor.
- `temperatureReadings` armazenará as leituras de temperatura do sensor.

Ambas as tabelas compartilharão a mesma estrutura de chave primária: `sensorId (String)` como chave de partição e `timestamp (String)` como chave de classificação.

Escolha Criar recursos na parte superior da página. Escolha Usar tipo existente e selecione o tipo `locationReadings`. Nomeie sua tabela `locationReadings`. Certifique-se de que a Chave primária esteja definida como `sensorId` e a chave de classificação como `timestamp`. Desmarque Gerar GraphQL automaticamente (você fornecerá seu próprio código) e selecione Criar. Repita esse processo para `temperatureReadings` usar `temperatureReadings` como tipo e nome da tabela. Use as mesmas teclas acima.

Suas novas tabelas conterão funções geradas automaticamente. Ainda existem algumas permissões que você precisa adicionar a essas funções. Vá para a página Fontes de dados e escolha `locationReadings`. Em Selecionar uma função existente, é possível ver a função. Anote a função (deve ter uma aparência parecida com `apps-sync-ds-ddb-aaabbbcccddd-locationReadings`) e, em seguida, acesse o console do IAM (<https://console.aws.amazon.com/iam/>). No console do IAM, escolha Funções e selecione sua função na tabela. Na sua função, em Políticas de permissões, clique em "+" ao lado da política (deve ter um nome semelhante ao nome da função). Escolha Editar na parte superior do expansível quando a política aparece. Você precisa adicionar permissões a essa política. Essa lista será semelhante a:

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:region:account:table/locationReadings",
        "arn:aws:dynamodb:region:account:table/locationReadings/*",
        "arn:aws:dynamodb:region:account:table/temperatureReadings",
        "arn:aws:dynamodb:region:account:table/temperatureReadings/*"
      ]
    }
  ]
}
```

```
    }  
  ]  
}
```

Escolha Avançar e Salvar alterações. Repita esse processo para a fonte de dados `temperatureReadings` usando o mesmo trecho de política acima.

BatchPutItem - Gravação de leituras do sensor

Nossos sensores precisam ser capazes de enviar suas leituras assim que se conectarem à internet. O campo do GraphQL Mutation `recordReadings` é a API que será usada para fazer isso. Precisaremos adicionar um resolvedor a esse campo.

Na página Esquema do AWS AppSync console, selecione Anexar ao lado do `Mutation.recordReadings` campo. Na próxima tela, crie seu resolvedor usando a tabela `locationReadings` como fonte de dados.

Depois de criar seu resolvedor, substitua os manipuladores pelo código a seguir no editor. Essa operação `BatchPutItem` nos permite especificar múltiplas tabelas:

```
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
}
```

```
}
```

Com operações em lote, podem haver erros e resultados retornados na invocação. Nesse caso, podemos fazer uma manipulação de erros adicional.

Note

O uso de `utils.appendError()` é semelhante ao `util.error()`, com a principal diferença de que a avaliação do manipulador de solicitação ou resposta não é interrompida. Em vez disso, sinaliza que houve um erro no campo, mas permite que o manipulador seja avaliado e, conseqüentemente, retorne os dados ao chamador. Recomendamos que você use `utils.appendError()` quando o aplicativo precisar retornar resultados parciais.

Salve o resolvidor e navegue até a página Consultas no AWS AppSync console. Agora podemos enviar algumas leituras do sensor.

Execute a seguinte mutação:

```
mutation sendReadings {
  recordReadings(
    tempReadings: [
      {sensorId: 1, value: 85.5, timestamp: "2018-02-01T17:21:05.000+08:00"},
      {sensorId: 1, value: 85.7, timestamp: "2018-02-01T17:21:06.000+08:00"},
      {sensorId: 1, value: 85.8, timestamp: "2018-02-01T17:21:07.000+08:00"},
      {sensorId: 1, value: 84.2, timestamp: "2018-02-01T17:21:08.000+08:00"},
      {sensorId: 1, value: 81.5, timestamp: "2018-02-01T17:21:09.000+08:00"}
    ]
    locReadings: [
      {sensorId: 1, lat: 47.615063, long: -122.333551, timestamp:
"2018-02-01T17:21:05.000+08:00"},
      {sensorId: 1, lat: 47.615163, long: -122.333552, timestamp:
"2018-02-01T17:21:06.000+08:00"},
      {sensorId: 1, lat: 47.615263, long: -122.333553, timestamp:
"2018-02-01T17:21:07.000+08:00"},
      {sensorId: 1, lat: 47.615363, long: -122.333554, timestamp:
"2018-02-01T17:21:08.000+08:00"},
      {sensorId: 1, lat: 47.615463, long: -122.333555, timestamp:
"2018-02-01T17:21:09.000+08:00"}
    ]) {
    locationReadings {
      sensorId
```



```

    timestamp
    lat
    long
  }
  temperatureReadings {
    sensorId
    timestamp
    value
  }
}
}

```

Enviamos dez leituras do sensor em uma mutação, com as leituras divididas em duas tabelas. Use o console do DynamoDB para validar se os dados aparecem nas tabelas `locationReadings` e `temperatureReadings`.

BatchDeleteItem - Excluindo as leituras do sensor

Da mesma forma, também será necessário excluir lotes de leituras do sensor. Vamos usar o campo do GraphQL `Mutation.deleteReadings` para essa finalidade. Na página Esquema do AWS AppSync console, selecione Anexar ao lado do `Mutation.deleteReadings` campo. Na próxima tela, crie seu resolvedor usando a tabela `locationReadings` como fonte de dados.

Depois de criar seu resolvedor, substitua os manipuladores no editor de código pelo trecho abaixo. Neste resolvedor, usamos um mapeador de função auxiliar que extrai `sensorId` e `timestamp` das entradas fornecidas.

```

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
}
```

Salve o resolvedor e navegue até a página Consultas no AWS AppSync console. Agora, vamos excluir algumas leituras do sensor.

Execute a seguinte mutação:

```
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

Ao contrário da operação `DeleteItem`, o item totalmente excluído não é retornado na resposta. Somente a chave passada é retornada. Para saber mais, consulte a [referência da função `BatchDeleteItem` no JavaScript resolvedor do DynamoDB](#).

Valide por meio do console do DynamoDB que essas duas leituras foram excluídas das tabelas `locationReadings` e `temperatureReadings`.

BatchGetItem - Recuperar leituras

Outra operação comum para o nosso aplicativo seria recuperar as leituras de um sensor em um determinado momento. Vamos anexar um resolvedor ao campo do GraphQL Query `getReadings` em nosso esquema. Na página Esquema do AWS AppSync console, selecione Anexar ao lado do Query `getReadings` campo. Na próxima tela, crie seu resolvedor usando a tabela `locationReadings` como fonte de dados.

Vamos usar o seguinte código:

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

export function request(ctx) {
  const keys = [util.dynamodb.toMapValues(ctx.args)]
  const consistentRead = true
  return {
    operation: 'BatchGetItem',
    tables: {
      locationReadings: { keys, consistentRead },
      temperatureReadings: { keys, consistentRead },
    },
  }
}

export function response(ctx) {
  if (ctx.error) {
    util.appendError(ctx.error.message, ctx.error.type)
  }
  const { locationReadings: locs, temperatureReadings: temps } = ctx.result.data

  return [
    ...locs.map((l) => ({ ...l, __typename: 'LocationReading' })),
    ...temps.map((t) => ({ ...t, __typename: 'TemperatureReading' })),
  ]
}
```

Salve o resolvedor e navegue até a página Consultas no AWS AppSync console. Agora, vamos recuperar as leituras dos nossos sensores.

Execute a seguinte consulta:

```
query getReadingsForSensorAndTime {
```

```
# Let's retrieve the very first two readings
getReadings(sensorId: 1, timestamp: "2018-02-01T17:21:06.000+08:00") {
  sensorId
  timestamp
  ...on TemperatureReading {
    value
  }
  ...on LocationReading {
    lat
    long
  }
}
```

Demonstramos com sucesso o uso das operações em lote do DynamoDB usando AWS AppSync

Tratamento de erros

Em AWS AppSync, as operações da fonte de dados às vezes podem retornar resultados parciais. Resultados parciais será o termo usado para indicar quando a saída de uma operação for composta por alguns dados e um erro. Como o tratamento de erros é inerentemente específico do aplicativo, AWS AppSync você tem a oportunidade de lidar com erros no manipulador de respostas. O erro de invocação do resolvidor, se presente, está disponível no contexto como `ctx.error`. Os erros de invocação sempre incluem uma mensagem e um tipo, acessível como propriedades `ctx.error.message` e `ctx.error.type`. No manipulador de respostas, é possível manipular resultados parciais de três maneiras:

1. Absorver o erro de invocação apenas retornando dados.
2. Gere um erro (usando `util.error(...)`) interrompendo a avaliação do manipulador, que não retornará nenhum dado.
3. Anexe um erro (usando `util.appendError(...)`) e também retorne dados.

Vamos demonstrar cada um dos três pontos acima com operações em lote do DynamoDB!

Operações em lote do DynamoDB

Com as operações em lote do DynamoDB, é possível que um lote seja concluído parcialmente. Ou seja, é possível que alguns dos itens solicitados ou chaves não seja processados. Se não AWS AppSync for possível concluir um lote, os itens não processados e um erro de invocação serão definidos no contexto.

Vamos implementar a manipulação de erros usando a configuração de campo `Query.getReadings` da operação `BatchGetItem` da seção anterior desse tutorial. Dessa vez, vamos fingir que ao executar o campo `Query.getReadings`, a tabela do DynamoDB `temperatureReadings` esgotou sua throughput provisionada. O DynamoDB gerou `ProvisionedThroughputExceededException` um durante a segunda tentativa AWS AppSync de processar os elementos restantes no lote.

O JSON a seguir representa o contexto serializado após a invocação em lote do DynamoDB, mas antes do manipulador de resposta ser chamado:

```
{
  "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": []
}
```

Algumas observações sobre o contexto:

- O erro de invocação foi definido no contexto em `ctx.error` by e AWS AppSync o tipo de erro foi definido como. `DynamoDB:ProvisionedThroughputExceededException`
- Os resultados são mapeados por tabela em `ctx.result.data`, mesmo que haja um erro presente.
- As chaves não processadas estão disponíveis em `ctx.result.data.unprocessedKeys`. Aqui, não AWS AppSync foi possível recuperar o item com a chave (`sensorId:1, timestamp:2018-02-01T17:21:05.000 + 08:00`) devido à taxa de transferência insuficiente da tabela.

Note

Em `BatchPutItem`, é `ctx.result.data.unprocessedItems`. Em `BatchDeleteItem`, é `ctx.result.data.unprocessedKeys`.

Vamos lidar com esse erro de três maneiras diferentes.

1. Absorção do erro de invocação

Retornar dados sem manipular o erro de invocação efetivamente absorve o erro, tornando o resultado para o determinado campo do GraphQL sempre bem-sucedido.

O código que gravamos é familiar e concentra-se apenas nos dados do resultado.

Manipulador de resposta

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

Resposta do GraphQL

```
{
```

```
"data": {
  "getReadings": [
    {
      "sensorId": "1",
      "timestamp": "2018-02-01T17:21:05.000+08:00",
      "lat": 47.615063,
      "long": -122.333551
    },
    {
      "sensorId": "1",
      "timestamp": "2018-02-01T17:21:05.000+08:00",
      "value": 85.5
    }
  ]
}
```

Nenhum erro será adicionado à resposta do erro uma vez que apenas dados foram modificados.

2. Gerar um erro para abortar a execução do manipulador de resposta

Quando falhas parciais devem ser tratadas como falhas completas do ponto de vista do cliente, você pode abortar a execução do manipulador de respostas para evitar o retorno de dados. O método utilitário `util.error(...)` atinge exatamente esse comportamento.

Código do manipulador de respostas

```
export function response(ctx) {
  if (ctx.error) {
    util.error(ctx.error.message, ctx.error.type, null,
      ctx.result.data.unprocessedKeys);
  }
  return ctx.result.data;
}
```

Resposta do GraphQL

```
{
  "data": {
    "getReadings": null
  },
  "errors": [
```

```

{
  "path": [
    "getReadings"
  ],
  "data": null,
  "errorType": "DynamoDB:ProvisionedThroughputExceededException",
  "errorInfo": {
    "temperatureReadings": [
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00"
      }
    ],
    "locationReadings": []
  },
  "locations": [
    {
      "line": 58,
      "column": 3
    }
  ],
  "message": "You exceeded your maximum allowed provisioned throughput for a table
or for one or more global secondary indexes. (...)"
}
]
}

```

Embora alguns resultados possam ter sido retornados da operação em lote do DynamoDB, escolhemos gerar um erro tal que o campo do GraphQL `getReadings` seja nulo e o erro seja adicionado ao bloco erros da resposta do GraphQL.

3. Anexar um erro para retornar dados e erros

Em alguns casos, para oferecer uma experiência melhor ao usuário, os aplicativos podem retornar resultados parciais e notificar seus clientes sobre os itens não processados. Os clientes podem decidir implementar uma nova tentativa ou traduzir o erro de volta para o usuário final. O `util.appendError(...)` é o método utilitário que permite esse comportamento, permitindo que o designer do aplicativo anexe erros no contexto sem interferir na avaliação do manipulador de respostas. Depois de avaliar o manipulador de respostas, AWS AppSync processará quaisquer erros de contexto anexando-os ao bloco de erros da resposta do GraphQL.

Código do manipulador de respostas


```
export function response(ctx) {
  if (ctx.error) {
    util.appendError(ctx.error.message, ctx.error.type, null,
ctx.result.data.unprocessedKeys);
  }
  return ctx.result.data;
}
```

Encaminhamos o erro de invocação e o elemento `unprocessedKeys` dentro do bloco de erros da resposta do GraphQL. O campo `getReadings` também retorna dados parciais da tabela `locationReadings` como você pode ver na resposta abaixo.

Resposta do GraphQL

```
{
  "data": {
    "getReadings": [
      null,
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00",
        "value": 85.5
      }
    ]
  },
  "errors": [
    {
      "path": [
        "getReadings"
      ],
      "data": null,
      "errorType": "DynamoDB:ProvisionedThroughputExceededException",
      "errorInfo": {
        "temperatureReadings": [
          {
            "sensorId": "1",
            "timestamp": "2018-02-01T17:21:05.000+08:00"
          }
        ],
        "locationReadings": []
      }
    },
    {
      "locations": [
        {
```

```
      "line": 58,  
      "column": 3  
    }  
  ],  
  "message": "You exceeded your maximum allowed provisioned throughput for a table  
or for one or more global secondary indexes. (...)"  
}  
]  
}
```

Usando resolvedores HTTP em AWS AppSync

AWS AppSync permite que você use fontes de dados compatíveis (ou seja, Amazon DynamoDB, AWS Lambda, Amazon Service ou OpenSearch Amazon Aurora) para realizar várias operações, além de quaisquer endpoints HTTP arbitrários para resolver campos do GraphQL. Depois que os endpoints HTTP estiverem disponíveis, conecte-se a eles usando uma fonte de dados. Em seguida, configure um resolvedor no esquema para executar operações do GraphQL, como consultas, mutações e assinaturas. Esse tutorial apresentará alguns exemplos comuns.

Neste tutorial, você usa uma API REST (criada usando o Amazon API Gateway e o Lambda) com um endpoint GraphQL AWS AppSync .

Criar uma API REST

Você pode usar o AWS CloudFormation modelo a seguir para configurar um endpoint REST que funcione para este tutorial:

A button with a yellow gradient background, a blue border, and a blue play icon on the right. The text "Launch Stack" is in bold black font.

A AWS CloudFormation pilha executa as seguintes etapas:

1. Configura uma função do Lambda, que contém a lógica de negócios para o seu microsserviço.
2. Configura uma API REST do API Gateway com a seguinte combinação endpoint/method/content de tipos:

Caminho do recurso da API	Método HTTP	Tipo de conteúdo compatível
/v1/users	POST	application/json

Caminho do recurso da API	Método HTTP	Tipo de conteúdo compatível
/v1/users	GET	application/json
/v1/users/1	GET	application/json
/v1/users/1	PUT	application/json
/v1/users/1	DELETE	application/json

Criação da API GraphQL

Para criar a API GraphQL em: AWS AppSync

1. Abra o AWS AppSync console e escolha Criar API.
2. Escolha GraphQL APIs e, em seguida, escolha Design do zero. Escolha Próximo.
3. Para o nome da API, digite UserData. Escolha Próximo.
4. Selecione Create GraphQL resources later. Escolha Próximo.
5. Revise suas entradas e selecione Criar API.

O AWS AppSync console cria uma nova API GraphQL para você usando o modo de autenticação por chave de API. Você pode usar o console para configurar ainda mais sua API GraphQL e executar solicitações.

Criar um esquema do GraphQL

Agora que você tem uma API GraphQL, vamos criar um esquema do GraphQL. No editor de esquemas no AWS AppSync console, use o trecho abaixo:

```
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  
}
```

Configurar a fonte de dados HTTP

Para configurar a fonte de dados HTTP, faça o seguinte:

1. Na página Fontes de dados na sua API AWS AppSync GraphQL, escolha Criar fonte de dados.
2. Insira um nome para a fonte de dados como HTTP_Example.
3. Em Tipo de fonte de dados, escolha Endpoint de HTTP.
4. Configure o endpoint para o endpoint da API Gateway que foi criado no início do tutorial.
Você pode encontrar seu endpoint gerado pela pilha navegando até o console do Lambda e encontrando seu aplicativo em Aplicativos. Dentro das configurações do seu aplicativo, você verá um endpoint de API que será seu endpoint no AWS AppSync. Lembre-se de não incluir o nome do estágio como parte do endpoint. Por exemplo, se o seu endpoint fosse `https://aaabbbcccd.execute-api.us-east-1.amazonaws.com/v1`, você digitaria `https://aaabbbcccd.execute-api.us-east-1.amazonaws.com`.

Note

No momento, somente endpoints públicos são suportados pelo AWS AppSync.

Para obter mais informações sobre as autoridades de certificação que são reconhecidas pelo AWS AppSync serviço, consulte [Autoridades de certificação \(CA\) reconhecidas por AWS AppSync para endpoints HTTPS](#).

Configuração de resolvedores do

Nesta etapa, você conectará a fonte de dados HTTP às consultas `getUser` e `addUser`.

Como configurar o resolvedor `getUser`:

1. Na sua API do AWS AppSync GraphQL, escolha a guia Esquema.
2. À direita do editor Esquema, no painel Resolvedores e em tipo Consulta, encontre o campo `getUser` e escolha Anexar.
3. Mantenha o tipo de resolvedor como `Unit` e o runtime como `APPSYNC_JS`.
4. Em Nome da fonte de dados, escolha o endpoint HTTP que você criou anteriormente.
5. Escolha Criar.
6. No editor de código Resolvedor, adicione o seguinte trecho como seu manipulador de solicitação:

```
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. Adicione o seguinte trecho como seu manipulador de respostas:

```
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. Escolha a guia Consulta e, depois, execute a seguinte consulta:

```
query GetUser{  
  getUser(id:1){  
    id  
    username  
  }  
}
```

Isso deve retornar a seguinte resposta:

```
{  
  "data": {  
    "getUser": {  
      "id": "1",  
      "username": "nadia"  
    }  
  }  
}
```

Como configurar o resolvidor addUser:

1. Escolha a guia Esquema.
2. À direita do editor Esquema, no painel Resolvedores e em tipo Consulta, encontre o campo addUser e escolha Anexar.
3. Mantenha o tipo de resolvidor como `Unit` e o runtime como `APPSYNC_JS`.
4. Em Nome da fonte de dados, escolha o endpoint HTTP que você criou anteriormente.
5. Escolha Criar.
6. No editor de código Resolvidor, adicione o seguinte trecho como seu manipulador de solicitação:

```
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. Adicione o seguinte trecho como seu manipulador de respostas:

```

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. Escolha a guia Consulta e, depois, execute a seguinte consulta:

```

mutation addUser{
  addUser(userInput:{
    id:"2",
    username:"shaggy"
  }){
    id
    username
  }
}

```

Se você executar a consulta `getUser` novamente, ela deverá retornar a seguinte resposta:

```

{
  "data": {
    "getUser": {
      "id": "2",

```

```
    "username": "shaggy"
  }
}
```

Invocando serviços AWS

Você pode usar resolvedores HTTP para configurar uma interface AWS de API GraphQL para serviços. As solicitações HTTP AWS devem ser assinadas com o [processo Signature Version 4](#) para que seja AWS possível identificar quem as enviou. AWS AppSync calcula a assinatura em seu nome quando você associa uma função do IAM à fonte de dados HTTP.

Você fornece dois componentes adicionais para invocar AWS serviços com resolvedores HTTP:

- Uma função do IAM com permissões para chamar o AWS serviço APIs
- Configurar a assinatura na fonte de dados

Por exemplo, se você quiser chamar a [ListGraphqlApis operação](#) com resolvedores HTTP, primeiro [crie uma função do IAM](#) que AWS AppSync assuma com a seguinte política anexada:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "appsync:ListGraphqlApis"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Em seguida, crie a fonte de dados HTTP para AWS AppSync. Neste exemplo, você chamará AWS AppSync na região Oeste dos EUA (Oregon). Configure a seguinte configuração HTTP em um arquivo chamado `http.json`, que inclui a região de assinatura e o nome do serviço:


```
{
  "endpoint": "https://appsync.us-west-2.amazonaws.com/",
  "authorizationConfig": {
    "authorizationType": "AWS_IAM",
    "awsIamConfig": {
      "signingRegion": "us-west-2",
      "signingServiceName": "appsync"
    }
  }
}
```

Em seguida, use o AWS CLI para criar a fonte de dados com uma função associada da seguinte forma:

```
aws appsync create-data-source --api-id <API-ID> \
                               --name AWSAppSync \
                               --type HTTP \
                               --http-config file:///http.json \
                               --service-role-arn <ROLE-ARN>
```

Ao anexar um resolvidor ao campo no esquema, use o seguinte modelo de mapeamento de solicitações para chamar AWS AppSync:

```
{
  "version": "2018-05-29",
  "method": "GET",
  "resourcePath": "/v1/apis"
}
```

Quando você executa uma consulta GraphQL para essa fonte de dados, AWS AppSync assina a solicitação usando a função fornecida e inclui a assinatura na solicitação. A consulta retorna uma lista do AWS AppSync GraphQL APIs na sua conta nessa AWS região.

Uso do Aurora PostgreSQL com a API de dados no AWS AppSync

Saiba como conectar sua API GraphQL aos bancos de dados Aurora PostgreSQL usando o. AWS AppSync Essa integração permite que você crie aplicativos escaláveis e orientados por dados executando consultas e mutações SQL por meio de operações do GraphQL. AWS AppSync fornece uma fonte de dados para executar instruções SQL em clusters do Amazon Aurora que são

habilitados com uma API de dados. Você pode usar AWS AppSync resolvedores para executar instruções SQL na API de dados com consultas, mutações e assinaturas do GraphQL.

Antes de começar este tutorial, você deve ter familiaridade básica com AWS serviços e conceitos do GraphQL.

 Note

Este tutorial usa a Região US-EAST-1.

Tópicos

- [Configurar seu banco de dados Aurora PostgreSQL](#)
- [Criar o banco de dados e uma tabela](#)
- [Criar um esquema do GraphQL](#)
- [Resolvedores para RDS](#)
- [Excluir o cluster](#)

Configurar seu banco de dados Aurora PostgreSQL

Antes de adicionar uma fonte de dados do Amazon RDS AWS AppSync, faça o seguinte.

1. Ative uma API de dados em um cluster Aurora Serverless v2.
2. Configure um segredo usando AWS Secrets Manager
3. Crie o cluster usando o AWS CLI comando a seguir.

```
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
```

Isso retornará um ARN para o cluster. Depois de criar um cluster, você deve adicionar uma instância Serverless v2 com o comando a seguir. AWS CLI

```
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

Esses endpoints demoram para serem ativados. Você pode verificar seu status no console do RDS na guia Conectividade e segurança do cluster.

Verifique o status do cluster com o AWS CLI comando a seguir.

```
aws rds describe-db-clusters \  
  --db-cluster-identifier appsync-tutorial \  
  --query "DBClusters[0].Status"
```

Crie um segredo por meio do AWS Secrets Manager console ou AWS CLI com um arquivo de entrada, como o seguinte, usando o USERNAME e COMPLEX_PASSWORD da etapa anterior:

```
{  
  "username": "USERNAME",  
  "password": "COMPLEX_PASSWORD"  
}
```

Passe isso como um parâmetro para AWS CLI:

```
aws secretsmanager create-secret \  
  --name appsync-tutorial-rds-secret \  
  --secret-string file://creds.json
```

Isso retornará um ARN para o segredo. Anote o ARN do seu cluster Aurora Serverless v2 e do Secret para mais tarde ao criar uma fonte de dados no console. AWS AppSync

Criar o banco de dados e uma tabela

Primeiro, crie um banco de dados chamado TESTDB. No PostgreSQL, um banco de dados é um contêiner que contém tabelas e outros objetos SQL. Valide se seu cluster Aurora Serverless v2 está

configurado corretamente antes de adicioná-lo à sua API. AWS AppSync Primeiro, crie um banco de dados TESTDB com o `--sql` parâmetro a seguir.

```
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"
```

Se isso for executado sem erros, inclua duas tabelas com o comando `create table`:

```
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);'
```

Se for bem-sucedido, adicione o cluster como fonte de dados na sua API.

Criar um esquema do GraphQL

Agora que sua API de dados Aurora Serverless v2 está sendo executada com tabelas configuradas, criaremos um esquema GraphQL. Você pode criar rapidamente sua API importando configurações de tabela de um banco de dados existente usando o assistente de criação de API.

Para começar:

1. No AWS AppSync console, escolha **Create API** e, em seguida, **Start with an Amazon Aurora cluster**.

2. Especifique os detalhes da API, como Nome da API, e selecione o banco de dados para gerar a API.
3. Selecione o banco de dados. Se necessário, atualize a região e selecione o cluster do Aurora e o banco de dados TESTDB.
4. Selecione o segredo e escolha Importar.
5. Depois que as tabelas forem descobertas, atualize os nomes dos tipos. Altere Todos para Todo e Tasks para Task.
6. Visualize o esquema gerado selecionando Visualizar esquema. O esquema terá a seguinte aparência:

```
type Todo {  
  id: Int!  
  description: String!  
  due: AWSDate!  
  createdAt: String  
}  
  
type Task {  
  id: Int!  
  todoId: Int!  
  description: String  
}
```

7. Para a função, você pode AWS AppSync criar uma nova função ou criar uma com uma política semelhante à abaixo:

```
{  
  "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:*"
    ]
  }
}
```

Observe que há duas declarações nesta política à qual você está concedendo acesso ao perfil. O primeiro recurso é o cluster do Aurora e o segundo é o ARN do AWS Secrets Manager .

Selecione Próximo, revise os detalhes da configuração e escolha Criar API. Agora você tem uma API totalmente operacional. É possível revisar os detalhes completos da API na página Esquema.

Resolvedores para RDS

O fluxo de criação da API criou automaticamente os resolvedores para interagir com nossos tipos. Se você consultar a página Esquema, encontrará alguns dos seguintes resolvedores.

- Criar um todo por meio do campo `Mutation.createTodo`.
- Atualizar um todo por meio do campo `Mutation.updateTodo`.
- Excluir um todo por meio do campo `Mutation.deleteTodo`.
- Obter um único todo por meio do campo `Query.getTodo`.
- Listar todos os todos por meio do campo `Query.listTodos`.

Você encontrará campos e resolvedores semelhantes anexados para o tipo Task. Vamos examinar com mais cuidado alguns dos resolvedores.

Mutation.createTodo

No editor de esquemas no AWS AppSync console, no lado direito, escolha `testdb` próximo a `createTodo(...)`: Todo O código do resolvedor usa a função `insert` do módulo do `rds` para criar dinamicamente uma declaração de inserção que adiciona dados à tabela `todos`. Como

estamos trabalhando com o Postgres, podemos aproveitar a declaração `returning` para recuperar os dados inseridos.

Atualize o resolvidor a seguir para especificar corretamente o DATE tipo do due campo.

```
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]
}
```

Salve o resolvidor. A dica de tipo marca a propriedade `due` no objeto de entrada como um tipo DATE. Isso permite que o mecanismo Postgres interprete adequadamente o valor. Depois, atualize o esquema para remover o `id` da entrada `CreateTodo`. Como nosso banco de dados Postgres pode retornar o ID gerado, você pode confiar nele para criar e retornar o resultado como uma única solicitação, da seguinte forma.

```
input CreateTodoInput {
```

```
    due: AWSDate!
    createdAt: String
    description: String!
  }
```

Faça a alteração e atualize o esquema. Acesse o editor de consultas para adicionar um item ao banco de dados da seguinte maneira.

```
mutation CreateTodo {
  createTodo(input: {description: "Hello World!", due: "2023-12-31"}) {
    id
    due
    description
    createdAt
  }
}
```

Você obtém o seguinte resultado.

```
{
  "data": {
    "createTodo": {
      "id": 1,
      "due": "2023-12-31",
      "description": "Hello World!",
      "createdAt": "2023-11-14 20:47:11.875428"
    }
  }
}
```

Query.listTodos

No editor de esquemas no console, à direita, selecione testdb ao lado de listTodos(id: ID!): Todo. O manipulador de solicitações usa a função de utilitário select para criar uma solicitação dinamicamente em runtime.

```
export function request(ctx) {
  const { filter = {}, limit = 100, nextToken } = ctx.args;
  const offset = nextToken ? +util.base64Decode(nextToken) : 0;
  const statement = select({
    table: 'todos',
```



```

        columns: '*',
        limit,
        offset,
        where: filter,
    });
    return createPgStatement(statement)
}

```

Queremos filtrar todos com base na data due. Vamos atualizar o resolvedor no qual converter valores due em DATE. Atualize a lista de importações e o manipulador de solicitações da seguinte maneira.

```

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 };
}
```

No editor de consultas, faça o seguinte.

```
query LIST {
  listTodos(limit: 10, filter: {due: {between: ["2021-01-01", "2025-01-02"]}}) {
    items {
      id
      due
      description
    }
  }
}
```

Mutation.updateTodo

Também é possível update um Todo. No editor de consultas, vamos atualizar o primeiro item Todo de id 1.

```
mutation UPDATE {
  updateTodo(input: {id: 1, description: "edits"}) {
    description
    due
    id
  }
}
```

Observe que é preciso especificar o id do item que você está atualizando. Também é possível determinar uma condição para atualizar somente um item que atenda a condições específicas. Por exemplo, podemos querer editar o item somente se a descrição começar da edits seguinte forma.

```
mutation UPDATE {
  updateTodo(input: {id: 1, description: "edits: make a change"}, condition:
{description: {beginsWith: "edits"}}) {
    description
    due
    id
  }
}
```

Assim como processamos as operações `create` e `list`, podemos atualizar o resolvidor para converter o campo `due` em `DATE`. Salve essas alterações da `updateTodo` seguinte forma.

```
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];
}
```

Agora tente uma atualização com uma condição:

```
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

É possível delete um Todo com a mutação deleteTodo. Isso funciona como a updateTodo mutação, e você deve especificar o item que deseja excluir id da seguinte maneira.

```
mutation DELETE {
  deleteTodo(input: {id: 1}) {
    description
    due
    id
  }
}
```

Redigir consultas personalizadas

Usamos os utilitários do módulo do rds para criar declarações SQL. Também podemos redigir a própria declaração estática personalizada para interagir com o banco de dados. Primeiro, atualize o esquema para remover o id da entrada CreateTask.

```
input CreateTaskInput {
  todoId: Int!
  description: String
}
```

Depois, crie algumas tarefas. Uma tarefa tem uma relação de chave estrangeira com a Todo seguinte.

```
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 }
}
```

Crie um novo campo em seu Query tipo chamado da `getTodoAndTasks` seguinte forma.

```
getTodoAndTasks(id: Int!): Todo
```

Adicione um `tasks` campo ao `Todo` tipo da seguinte maneira.

```
type Todo {
  due: AWSDate!
  id: Int!
  createdAt: String
  description: String!
  tasks:TaskConnection
}
```

Salve o esquema. No editor de esquemas no console, à direita, selecione Anexar resolvidor para `getTodosAndTasks(id: Int!): Todo`. Selecione a fonte de dados do Amazon RDS. Atualize seu resolvidor com o código a seguir.

```
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;
}
```

Nesse código, usamos o modelo de tag `sql` para redigir uma declaração SQL para a qual podemos transmitir com segurança um valor dinâmico em runtime. `createPgStatement` pode receber até duas solicitações SQL por vez. Usamos isso para enviar uma consulta ao todo e outra para a `tasks`. É possível fazer isso com uma declaração `JOIN` ou qualquer outro método. A ideia é poder redigir a própria declaração SQL para implementar a lógica de negócios. Para usar a consulta no editor de consultas, faça o seguinte.

```
query TodoAndTasks {
  getTodosAndTasks(id: 2) {
    id
    due
    description
    tasks {
      items {
        id
        description
      }
    }
  }
}
```

Excluir o cluster

Important

A exclusão de um cluster é permanente. Revise o projeto com cuidado antes de realizar essa ação.

Para excluir o cluster:

```
$ aws rds delete-db-cluster \  
  --db-cluster-identifier appsync-tutorial \  
  --skip-final-snapshot
```

Tutoriais do resolvidor VTL para AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Fontes de dados e resolvidores são usados AWS AppSync para traduzir solicitações do GraphQL e buscar informações de seus recursos. AWS AppSync oferece suporte ao provisionamento automático e às conexões com determinados tipos de fonte de dados. AWS AppSync também oferece suporte AWS Lambda ao Amazon DynamoDB, bancos de dados relacionais (Amazon Aurora Serverless), OpenSearch Amazon Service e endpoints HTTP como fontes de dados. Você pode usar uma API GraphQL com seus AWS recursos existentes ou criar fontes de dados e resolvidores do zero. As seções a seguir têm como objetivo elucidar alguns dos casos de uso mais comuns do GraphQL na forma de tutoriais.

AWS AppSync usa modelos de mapeamento escritos em Apache Velocity Template Language (VTL) para resolvidores. Para obter mais informações sobre como usar modelos de mapeamento, consulte a [Referência de modelo de mapeamento do resolvidor](#). Mais informações sobre como trabalhar com VTL estão disponíveis no [Guia de programação do modelo de mapeamento do resolvidor](#).

AWS AppSync suporta o provisionamento automático de tabelas do DynamoDB a partir de um esquema do GraphQL, conforme descrito em Provisionar a partir do esquema (opcional) e Iniciar um esquema de amostra. Você também pode importar a partir de uma tabela do DynamoDB existente que criará os esquemas e conectará os resolvidores. Isso está descrito em Importar a partir do Amazon DynamoDB (opcional).

Tópicos

- [Criação de uma aplicação de postagem simples usando resolvidores do DynamoDB](#)
- [Usando AWS Lambda resolvidores em AWS AppSync](#)
- [Usando resolvidores OpenSearch do Amazon Service em AWS AppSync](#)
- [Usando resolvidores locais em AWS AppSync](#)
- [Combinação de resolvidores do GraphQL no AWS AppSync](#)
- [Usando operações em lote do DynamoDB em AWS AppSync](#)

- [Executando transações do DynamoDB no AWS AppSync](#)
- [Usando resolvedores HTTP em AWS AppSync](#)
- [Usando o Aurora Serverless v2 com AWS AppSync](#)
- [Usando resolvedores de pipeline em AWS AppSync](#)
- [Usando as operações do Delta Sync em fontes de dados versionadas em AWS AppSync](#)

Criação de uma aplicação de postagem simples usando resolvedores do DynamoDB

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Este tutorial mostra como você pode trazer suas próprias tabelas do Amazon DynamoDB e conectá-las AWS AppSync a uma API do GraphQL.

Você pode permitir o AWS AppSync provisionamento de recursos do DynamoDB em seu nome. Ou, se preferir, você pode conectar as tabelas existentes a um esquema do GraphQL criando uma fonte de dados e um resolvedor. Em ambos os casos, você poderá ler e gravar no banco de dados do DynamoDB por meio de instruções do GraphQL e assinar dados em tempo real.

Existem etapas de configuração específicas que precisam ser concluídas para que as instruções do GraphQL sejam traduzidas para operações do DynamoDB e para que as respostas sejam traduzidas novamente para o GraphQL. Esse tutorial descreve o processo de configuração por meio de vários cenários reais e padrões de acesso aos dados.

Configuração de tabelas do DynamoDB

Para começar este tutorial, primeiro você precisa seguir as etapas abaixo para provisionar AWS recursos.

1. Provisione AWS recursos usando o seguinte AWS CloudFormation modelo na CLI:

```
aws cloudformation create-stack \
```

```
--stack-name AWSAppSyncTutorialForAmazonDynamoDB \  
--template-url https://s3.us-west-2.amazonaws.com/awssappsync/resources/  
dynamodb/AmazonDynamoDBCFTemplate.yaml \  
--capabilities CAPABILITY_NAMED_IAM
```

Como alternativa, você pode lançar a seguinte AWS CloudFormation pilha na região Oeste dos EUA 2 (Oregon) em sua conta. AWS



Isso cria o seguinte:

- Uma tabela do DynamoDB chamada AppSyncTutorial-Post que armazenará os dados de Post.
 - Uma função do IAM e uma política gerenciada do IAM associada AWS AppSync para permitir a interação com a Post tabela.
2. Para ver mais detalhes sobre a pilha e os recursos criados, execute o seguinte comando da CLI:

```
aws cloudformation describe-stacks --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

3. Para excluir os recursos mais tarde, execute o seguinte:

```
aws cloudformation delete-stack --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

Criação da API GraphQL

Para criar a API GraphQL em: AWS AppSync

1. Faça login no AWS Management Console e abra o [AppSync console](#).
 - No APIs painel, escolha Criar API.
2. Na janela Personalizar sua API ou importar a partir do Amazon DynamoDB, escolha Criar a partir do zero.
 - Escolha Iniciar à direita da mesma janela.
3. No campo Nome da API, defina o nome da API para AWSAppSyncTutorial1.
4. Escolha Criar.

O AWS AppSync console cria uma nova API GraphQL para você usando o modo de autenticação por chave de API. Você pode usar o console para configurar o restante da API GraphQL e executar consultas nela durante o restante desse tutorial.

Definição de uma API Post básica

Agora que você criou uma API AWS AppSync GraphQL, você pode configurar um esquema básico que permite a criação, recuperação e exclusão básicas de dados de postagem.

1. Faça login no AWS Management Console e abra o [AppSync console](#).
 - No APIs painel, escolha a API que você acabou de criar.
2. Na barra lateral, escolha Esquema.
 - No painel Esquema, substitua o conteúdo pelo seguinte código:

```
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. Escolha Salvar.

Esse esquema define um tipo Post e operações para adicionar e obter objetos Post.

Configuração da fonte de dados para as tabelas do DynamoDB

Em seguida, vincule as consultas e mutações definidas no esquema à tabela AppSyncTutorial-Post do DynamoDB.

Primeiro, AWS AppSync precisa estar atento às suas tabelas. Faça isso ao configurar uma fonte de dados no AWS AppSync:

1. Faça login no AWS Management Console e abra o [AppSync console](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, escolha Fontes de dados.
2. Escolha Criar fonte de dados.
 - a. Para Nome da fonte de dados, insira em PostDynamoDBTable.
 - b. Para o tipo de fonte de dados, escolha Tabela do Amazon DynamoDB.
 - c. Para Região, escolha US-WEST-2.
 - d. Em Nome da tabela, escolha a tabela AppSyncTutorial-Post DynamoDB.
 - e. Crie um perfil do IAM (recomendado) ou escolha uma função existente que tenha permissão do IAM `lambda:invokeFunction`. Os perfis existentes precisam de uma política de confiança, conforme explicado na seção [Anexar uma fonte de dados](#).

Veja a seguir um exemplo de política do IAM que tem as permissões necessárias para executar as operações no recurso:

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. Escolha Criar.

Configurando o resolvidor AddPost (DynamoDB) PutItem

Depois de conhecer AWS AppSync a tabela do DynamoDB, você pode vinculá-la a consultas e mutações individuais definindo Resolvers. O primeiro resolvidor criado é o resolvidor addPost, que permite criar uma publicação na tabela AppSyncTutorial-Post do DynamoDB.

Um resolvidor tem os seguintes componentes:

- A local no esquema do GraphQL para anexar o resolvidor. Nesse caso, você está configurando um resolvidor no campo addPost no tipo Mutation. Esse resolvidor será invocado quando o chamador chamar `mutation { addPost(...){...} }`.
- A fonte de dados a ser usada para esse resolvidor. Nesse caso, você deseja usar a fonte de dados PostDynamoDBTable definida anteriormente, para que possa adicionar entradas na tabela do DynamoDB AppSyncTutorial-Post.
- O modelo de mapeamento da solicitação. O objetivo do modelo de mapeamento de solicitações é pegar a solicitação recebida do chamador e traduzi-la em instruções AWS AppSync para execução no DynamoDB.
- O modelo de mapeamento da resposta. O trabalho do modelo de mapeamento da resposta é receber a resposta do DynamoDB e traduzi-la de volta para algo esperado pelo GraphQL. Isso é útil se a forma dos dados no DynamoDB for diferente para o tipo Post no GraphQL, mas, nesse caso, elas têm a mesma forma, portanto basta transmitir os dados.

Como configurar o resolvidor :

1. Faça login no AWS Management Console e abra o [AppSync console](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, escolha Fontes de dados.
2. Escolha Criar fonte de dados.
 - a. Para Nome da fonte de dados, insira em PostDynamoDBTable.
 - b. Para o tipo de fonte de dados, escolha Tabela do Amazon DynamoDB.
 - c. Para Região, escolha US-WEST-2.
 - d. Em Nome da tabela, escolha a tabela AppSyncTutorial-Post DynamoDB.
 - e. Crie um perfil do IAM (recomendado) ou escolha uma função existente que tenha permissão do IAM `lambda:invokeFunction`. Os perfis existentes precisam de uma política de confiança, conforme explicado na seção [Anexar uma fonte de dados](#).

Veja a seguir um exemplo de política do IAM que tem as permissões necessárias para executar as operações no recurso:

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. Escolha Criar.
4. Escolha a guia Esquema.

5. No painel Tipos de dados à direita, encontre o campo `addPost` no tipo `Mutação` e, em seguida, escolha `Anexar`.
6. No menu `Ação`, escolha `Atualizar runtime` e selecione `Resolvedor de unidade` (somente `VTL`).
7. Em `Data source name` (Nome da fonte de dados), escolha `PostDynamoDBTable`.
8. Em `Configurar o modelo de mapeamento da solicitação`, cole o seguinte:

```
{
  "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 }
  }
}
```

Observação: um tipo é especificado em todas as chaves e valores de atributo. Por exemplo, defina o campo `author` para `{ "S" : "${context.arguments.author}" }`. A `S` parte indica para o DynamoDB AWS AppSync e para o DynamoDB que o valor será um valor de string. O valor real é preenchido a partir do argumento `author`. Da mesma forma, o campo `version` é um campo de número pois ele usa `N` para o tipo. Finalmente, você também está inicializando os campos `ups`, `downs` e `version`.

Neste tutorial, você especificou que o `ID!` tipo GraphQL, que indexa o novo item inserido no DynamoDB, vem como parte dos argumentos do cliente. AWS AppSync vem com um utilitário para geração automática de `ID$utils.autoId()`, chamado que você também poderia ter usado na forma de `"id" : { "S" : "${utils.autoId()}" }`. Depois, basta deixar o `id`: `ID!` fora da definição do esquema de `addPost()` e ele seria inserido automaticamente. Você não usará essa técnica para esse tutorial, mas deve considerá-la como uma das melhores práticas ao gravar em tabelas do DynamoDB.

Para obter mais informações sobre os modelos de mapeamento, consulte a documentação de referência [Visão geral do modelo de mapeamento do resolvedor](#). Para obter mais informações sobre mapeamento de GetItem solicitações, consulte a documentação de [GetItem](#) referência. Para obter mais informações sobre os tipos, consulte a documentação de referência [Mapeamento da solicitação](#).

9. Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
$utils.toJson($context.result)
```

Observação: como o formato dos dados na tabela AppSyncTutorial-Post corresponde exatamente ao formato do tipo Post no GraphQL, o modelo de mapeamento da resposta apenas transmite os resultados diretamente. Observe também que todos os exemplos desse tutorial usam o mesmo modelo de mapeamento da resposta, portanto você só cria um arquivo.

10. Escolha Salvar.

Chamar a API para adicionar uma publicação

Agora que o resolvedor está configurado, é AWS AppSync possível traduzir uma addPost mutação recebida em uma operação do DynamoDB. PutItem Agora você pode executar uma mutação para colocar algo na tabela.

- Escolha a guia Consultas.
- No painel Consultas, cole a seguinte mutação:

```
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
  }
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- Os resultados da publicação recém-criada devem aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```

{
  "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
    }
  }
}
```

Veja o que aconteceu:

- AWS AppSync recebeu uma solicitação de addPost mutação.
- AWS AppSync pegou a solicitação e o modelo de mapeamento da solicitação e gerou um documento de mapeamento da solicitação. Isso teria a seguinte aparência:

```

{
  "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 usou o documento de mapeamento da solicitação para gerar e executar uma solicitação do PutItem DynamoDB.
- AWS AppSync pegou os resultados da PutItem solicitação e os converteu novamente em tipos GraphQL.

```
{
  "id" : "123",
  "author": "AUTHORNAME",
  "title": "Our first post!",
  "content": "This is our first post.",
  "url": "https://aws.amazon.com/appsync/",
  "ups" : 1,
  "downs" : 0,
  "version" : 1
}
```

- Os transmitiu por meio do documento de mapeamento da resposta, sem alterações.
- Retornou o objeto recém-criado na resposta do GraphQL.

Configurando o GetPost Resolver (DynamoDB) GetItem

Agora que você pode adicionar dados à tabela AppSyncTutorial-Post do DynamoDB, é necessário configurar a consulta getPost para que ela possa recuperar esses dados da tabela AppSyncTutorial-Post. Para fazer isso, vamos configurar outro resolvedor.

- Escolha a guia Esquema.
- No painel Tipos de dados à direita, encontre o campo getPost no tipo Consulta e, em seguida, escolha Anexar.
- No menu Ação, escolha Atualizar runtime e selecione Resolvedor de unidade (somente VTL).
- Em Data source name (Nome da fonte de dados), escolha PostDynamoDBTable.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
{
```

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

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
$utils.toJson($context.result)
```

- Escolha Salvar.

Chamar a API para obter uma publicação

Agora, o resolvidor foi configurado e AWS AppSync sabe como traduzir uma `getPost` consulta recebida em uma operação do DynamoDB. `GetItem` Agora é possível executar uma consulta para recuperar a publicação criada anteriormente.

- Escolha a guia Consultas.
- No painel Consultas, cole o seguinte:

```
query getPost {
  getPost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- A publicação recuperada do DynamoDB deve aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```
{
  "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
}
```

Veja o que aconteceu:

- AWS AppSync recebeu uma solicitação `getPost` de consulta.
- AWS AppSync pegou a solicitação e o modelo de mapeamento da solicitação e gerou um documento de mapeamento da solicitação. Isso teria a seguinte aparência:

```
{
  "version" : "2017-02-28",
  "operation" : "GetItem",
  "key" : {
    "id" : { "S" : "123" }
  }
}
```

- AWS AppSync usou o documento de mapeamento da solicitação para gerar e executar uma solicitação do `GetItem` DynamoDB.
- AWS AppSync pegou os resultados da `GetItem` solicitação e os converteu de volta para os tipos GraphQL.

```
{
  "id" : "123",
  "author": "AUTHORNAME",
  "title": "Our first post!",
  "content": "This is our first post.",
  "url": "https://aws.amazon.com/appsync/",
  "ups" : 1,
  "downs" : 0,
  "version" : 1
}
```

```
}
```

- Os transmitiu por meio do documento de mapeamento da resposta, sem alterações.
- Retornou o objeto recuperado na resposta.

Como alternativa, utilize o exemplo a seguir:

```
query getPost {  
  getPost(id:123) {  
    id  
    author  
    title  
  }  
}
```

Se sua consulta `getPost` precisar apenas de `id`, `author` e `title`, você poderá alterar seu modelo de mapeamento da solicitação para usar expressões de projeção para especificar apenas os atributos que deseja da tabela do DynamoDB para evitar transferência desnecessária de dados do DynamoDB para AWS AppSync. Por exemplo, o modelo de mapeamento da solicitação pode se parecer com o trecho abaixo:

```
{  
  "version" : "2017-02-28",  
  "operation" : "GetItem",  
  "key" : {  
    "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)  
  },  
  "projection" : {  
    "expression" : "#author, id, title",  
    "expressionNames" : { "#author" : "author"  
  }  
}
```

Criar uma mutação `UpdatePost` (DynamoDB) `UpdateItem`

Até agora, você pode criar e recuperar objetos `Post` no DynamoDB. Depois, você configurará uma nova mutação para que possamos atualizar o objeto. Você fará isso usando a operação do `UpdateItem` DynamoDB.

- Escolha a guia `Esquema`.

- No painel Esquema, modifique o tipo Mutation para adicionar uma nova mutação updatePost da seguinte forma:

```
type Mutation {
  updatePost(
    id: ID!,
    author: String!,
    title: String!,
    content: String!,
    url: String!
  ): Post
  addPost(
    author: String!
    title: String!
    content: String!
    url: String!
  ): Post!
}
```

- Escolha Salvar.
- No painel Tipos de dados à direita, encontre o campo updatePost recém-criado no tipo Mutação e, em seguida, escolha Anexar.
- No menu Ação, escolha Atualizar runtime e selecione Resolvedor de unidade (somente VTL).
- Em Data source name (Nome da fonte de dados), escolha PostDynamoDBTable.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
{
  "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 }
      }
    }
  }
}

```

Observação: esse resolvedor está usando o `UpdateItem` DynamoDB, que é significativamente diferente da operação. `PutItem` Em vez de escrever o item inteiro, você está apenas pedindo ao DynamoDB para atualizar determinados atributos. Isso é feito usando expressões de atualização do DynamoDB. A expressão em si é especificada no campo `expression` na seção `update`. Ela diz para definir o `author`, o `title`, o `content` e os atributos de `url` e, em seguida, incrementar o campo `version`. Os valores a serem usados não aparecem na expressão em si; a expressão tem espaços reservados com nomes que começam com "dois pontos", que são definidos no campo `expressionValues`. Finalmente, o DynamoDB tem palavras reservadas que não podem aparecer no `expression`. Por exemplo, `url` é uma palavra reservada, então, para atualizar o campo `url` é possível usar espaços reservados de nome e defini-los no campo `expressionNames`.

Para obter mais informações sobre mapeamento de `UpdateItem` solicitações, consulte a documentação de [UpdateItem](#) referência. Para obter mais informações sobre como escrever expressões de atualização, consulte a documentação do [UpdateExpressions DynamoDB](#).

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
$utils.toJson($context.result)
```

Chamar a API para atualizar uma publicação

Agora, o resolvedor foi configurado e AWS AppSync sabe como traduzir uma `update` mutação recebida em uma operação do DynamoDB. `Update` Agora você pode executar uma mutação para atualizar o item escrito anteriormente.

- Escolha a guia Consultas.
- No painel Consultas, cole a seguinte mutação. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente.

```

mutation updatePost {
  updatePost(

```

```
    id:"123"
    author: "A new author"
    title: "An updated author!"
    content: "Now with updated content!"
    url: "https://aws.amazon.com/appsync/"
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- A publicação atualizada no DynamoDB deve aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```
{
  "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
    }
  }
}
```

Neste exemplo, os campos `ups` e `downs` não foram modificados porque o modelo de mapeamento de solicitações não AWS AppSync solicitou que o DynamoDB fizesse nada com esses campos. Além disso, o campo `version` foi incrementado em 1 porque você pediu ao AWS AppSync DynamoDB que adicionasse 1 ao campo `version`.

Modificando o UpdatePost Resolver (DynamoDB) UpdateItem

Esse é um bom início para a mutação `updatePost`, mas tem dois problemas principais:

- Se quiser atualizar apenas um único campo, é necessário atualizar todos os campos.
- Se duas pessoas estiverem modificando o objeto, possivelmente haverá perda de informações.

Para resolver esses problemas, você modificará a mutação `updatePost` para modificar apenas os argumentos que foram especificados na solicitação e, depois, adicionará uma condição à operação `UpdateItem`.

1. Escolha a guia Esquema.
2. No painel Esquema, modifique o campo `updatePost` no tipo `Mutation` e remova os pontos de exclamação dos argumentos `author`, `title`, `content` e `url`, mantendo o campo `id` como está. Isso os tornará argumento opcional. Além disso, adicione um novo argumento `expectedVersion` obrigatório.

```
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. Escolha Salvar.
4. No painel Tipos de dados, encontre o campo `updatePost` no tipo `Mutação`.
5. Escolha `PostDynamoDBTable` abrir o resolvidor existente.
6. Em `Configurar o modelo de mapeamento da solicitação`, modifique o modelo de mapeamento da solicitação da seguinte forma:

```

{
  "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. Escolha Salvar.

Este modelo é um dos exemplos mais complexos. Ele demonstra a potência e a flexibilidade dos modelos de mapeamento. Ele percorre todos os argumentos, ignorando `id` e `expectedVersion`. Se o argumento for definido como algo, ele solicitará que o DynamoDB AWS AppSync atualize esse atributo no objeto no DynamoDB. Se o atributo for definido como nulo, ele AWS AppSync solicitará que o DynamoDB remova esse atributo do objeto de postagem. Se um argumento não foi especificado, nada será feito com o atributo. Ele também incrementa o campo `version`.

Além disso, há uma nova seção `condition`. Uma expressão de condição permite que você diga ao AWS AppSync DynamoDB se a solicitação deve ou não ser bem-sucedida com base no estado do objeto que já está no DynamoDB antes da execução da operação. Nesse caso, você deseja que a solicitação `UpdateItem` seja bem-sucedida apenas se o campo `version` do item atualmente no DynamoDB corresponder ao argumento `expectedVersion`.

Para obter mais informações sobre as expressões de condições, consulte a documentação de referência [Expressões de condições](#).

Chamar a API para atualizar uma publicação

Vamos tentar atualizar o objeto `Post` com o novo resolvedor:

- Escolha a guia Consultas.
- No painel Consultas, cole a mutação a seguir. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente.

```
mutation updatePost {
  updatePost(
    id:123
    title: "An empty story"
    content: null
    expectedVersion: 2
```

```
    ) {  
      id  
      author  
      title  
      content  
      url  
      ups  
      downs  
      version  
    }  
  }  
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- A publicação atualizada no DynamoDB deve aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```
{  
  "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  
    }  
  }  
}
```

Nessa solicitação, você pediu ao AWS AppSync DynamoDB que atualizasse somente `title` o campo e `content`. Ele ignorou todos os outros campos (além de incrementar o campo `version`). Definiu-se o atributo `title` para um novo valor e o atributo `content` foi removido da publicação. Os campos `author`, `url`, `ups` e `downs` foram mantidos.

Tente executar a solicitação de mutação novamente, deixando a solicitação exatamente como está. Você verá uma resposta semelhante à seguinte:

```
{  
  "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)"
    }
  ]
}

```

A solicitação falha pois a expressão de condição é avaliada como falsa:

- Na primeira vez que você executou a solicitação, o valor do campo `version` da publicação no DynamoDB era 2, que correspondia ao argumento `expectedVersion`. A solicitação foi bem-sucedida, o que significa que o campo `version` foi incrementado no DynamoDB para 3.
- Na segunda vez que a solicitação foi executada, o valor do campo `version` da publicação no DynamoDB era 3, que não correspondeu ao argumento `expectedVersion`.

Esse padrão é geralmente chamado de bloqueio positivo.

Um recurso de um resolvidor do AWS AppSync DynamoDB é que ele retorna o valor atual do objeto post no DynamoDB. Encontre isso no campo data na seção errors da resposta do GraphQL. O aplicativo pode usar essas informações para decidir como deve continuar. Nesse caso, é possível ver que o campo version do objeto no DynamoDB está definido como 3; portanto, podemos apenas atualizar o argumento expectedVersion para 3 e a solicitação teria êxito novamente.

Para obter mais informações sobre o tratamento de falhas da verificação de condição, consulte a documentação de referência do modelo de mapeamento [Expressões de condições](#).

Crie mutações UpVotePost e DownVotePost (DynamoDB) UpdateItem

O tipo Post tem campos ups e downs para habilitar o registro de votos positivos e negativos, mas até agora a API não permite fazer nada com eles. Vamos adicionar algumas mutações para permitir votos positivos e negativos nas publicações.

- Escolha a guia Esquema.
- No painel Esquema, modifique o tipo Mutation para adicionar novas mutações upvotePost e downvotePost da seguinte forma:

```
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!  
}
```

- Escolha Salvar.
- No painel Tipos de dados à direita, encontre o campo upvotePost recém-criado no tipo Mutação e, em seguida, escolha Anexar.

- No menu Ação, escolha Atualizar runtime e selecione Resolvedor de unidade (somente VTL).
- Em Data source name (Nome da fonte de dados), escolha PostDynamoDBTable.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

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

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
$utils.toJson($context.result)
```

- Escolha Salvar.
- No painel Tipos de dados à direita, encontre o campo downvotePost recém-criado no tipo Mutação e, em seguida, escolha Anexar.
- Em Data source name (Nome da fonte de dados), escolha PostDynamoDBTable.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

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



```
}
```

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
$utils.toJson($context.result)
```

- Escolha Salvar.

Chamar a API para realizar voto positivo ou negativo em uma Publicação

Agora que os novos resolvedores foram configurados, AWS AppSync sabe como traduzir uma entrada `upvotePost` ou uma `downvote` mutação para a operação do DynamoDB. `UpdateItem` Agora é possível executar mutações para realizar votos positivos ou negativos na publicação criada anteriormente.

- Escolha a guia Consultas.
- No painel Consultas, cole a mutação a seguir. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente.

```
mutation votePost {  
  upvotePost(id:123) {  
    id  
    author  
    title  
    content  
    url  
    ups  
    downs  
    version  
  }  
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- A publicação é atualizada no DynamoDB e deve aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```
{  
  "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
  }
}
```

- Escolha Executar consulta mais algumas vezes. Você deve ver os campos `ups` e `version` incrementar em 1 a cada vez que você executar a consulta.
- Altere a consulta para chamar a mutação `downvotePost` da seguinte forma:

```
mutation votePost {
  downvotePost(id:123) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Escolha Executar consulta (o botão de reprodução laranja). Dessa vez, você deve ver os campos `downs` e `version` incrementar em 1 a cada vez que você executar a consulta.

```
{
  "data": {
    "downvotePost": {
      "id": "123",
      "author": "A new author",
      "title": "An empty story",
      "content": null,
      "url": "https://aws.amazon.com/appsync/",
      "ups": 6,
      "downs": 4,
      "version": 12
    }
  }
}
```

```
}
```

Configurando o DeletePost Resolver (DynamoDB) DeleteItem

A próxima mutação que você quer configurar é para excluir uma publicação. Você fará isso usando a operação `DeleteItem` do DynamoDB.

- Escolha a guia Esquema.
- No painel Esquema, modifique o tipo `Mutation` para adicionar uma nova mutação `deletePost` da seguinte forma:

```
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!  
}
```

Dessa vez o campo `expectedVersion` é opcional, o que é explicado mais tarde ao adicionar o modelo de mapeamento da solicitação.

- Escolha Salvar.
- No painel Tipos de dados à direita, encontre o campo `delete` recém-criado no tipo `Mutação` e, em seguida, escolha Anexar.
- No menu Ação, escolha Atualizar runtime e selecione Resolvedor de unidade (somente VTL).
- Em Data source name (Nome da fonte de dados), escolha `PostDynamoDBTable`.

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
{
  "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
}
```

Observação: o argumento `expectedVersion` é opcional. Se o chamador definir um argumento `expectedVersion` na solicitação, o modelo adiciona uma condição que permite que a solicitação `DeleteItem` seja bem-sucedida somente se o item já estiver excluído ou se o atributo `version` da publicação no DynamoDB corresponder exatamente ao `expectedVersion`. Se omitido, nenhuma expressão de condição será especificada na solicitação `DeleteItem`. Ele será bem-sucedida independentemente do valor de `version`, ou se o item existe ou não no DynamoDB.

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
$utils.toJson($context.result)
```

Observação: mesmo que você esteja excluindo um item, é possível retornar o item que foi excluído, caso ainda não tenha sido excluído.

- Escolha Salvar.

Para obter mais informações sobre mapeamento de `DeleteItem` solicitações, consulte a documentação de [DeleteItem](#) referência.

Chamar a API para excluir uma publicação

Agora, o resolvidor foi configurado e AWS AppSync sabe como traduzir uma `delete` mutação recebida em uma operação do DynamoDB. `DeleteItem` Agora você pode executar uma mutação para excluir algo na tabela.

- Escolha a guia Consultas.
- No painel Consultas, cole a mutação a seguir. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente.

```
mutation deletePost {  
  deletePost(id:123) {  
    id  
    author  
    title  
    content  
    url  
    ups  
    downs  
    version  
  }  
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- A publicação é excluída do DynamoDB. Observe que AWS AppSync retorna o valor do item que foi excluído do DynamoDB, que deve aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```
{  
  "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  
    }  
  }  
}
```

```
}
```

O valor é retornado somente se essa chamada para `deletePost` foi aquela que realmente a excluiu do DynamoDB.

- Escolha Executar consulta novamente.
- A chamada ainda será bem-sucedida, mas nenhum valor é retornado.

```
{
  "data": {
    "deletePost": null
  }
}
```

Agora vamos tentar excluir uma publicação, mas dessa vez especificando um `expectedValue`. No entanto, primeiro é necessário criar uma nova publicação, pois você acabou de excluir aquela com a qual estava trabalhando.

- No painel Consultas, cole a seguinte mutação:

```
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
  }
}
```

- Escolha Executar consulta (o botão de reprodução laranja).

- Os resultados da publicação recém-criada devem aparecer no painel de resultados à direita do painel de consulta. Anote o `id` do objeto recém-criado, pois será necessário em alguns instantes. A aparência deve ser semelhante à seguinte:

```
{
  "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
    }
  }
}
```

Agora vamos tentar excluir a publicação, mas coloque o valor errado para `expectedVersion`:

- No painel Consultas, cole a mutação a seguir. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente.

```
mutation deletePost {
  deletePost(
    id:123
    expectedVersion: 9999
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Escolha Executar consulta (o botão de reprodução laranja).

```
{
  "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:
ABCDEFGHIJKLMN0PQRSTUVWXYZABCDEFGHIJKLMN0PQRSTUVWXYZ)"
    }
  ]
}
```

A solicitação falhou porque a expressão de condição é avaliada como falsa: o valor para `version` da publicação no DynamoDB não corresponde ao `expectedValue` especificado nos argumentos. O valor atual do objeto é retornada no campo `data` na seção `errors` da resposta do GraphQL.

- Repita a solicitação, mas corrija o `expectedVersion`:

```
mutation deletePost {
  deletePost(
    id:123
```



```
    expectedVersion: 1
  ) {
    id
    author
    title
    content
    url
    ups
    downs
    version
  }
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- Dessa vez, a solicitação é bem-sucedida e o valor que foi excluído do DynamoDB é retornado:

```
{
  "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
    }
  }
}
```

- Escolha Executar consulta novamente.
- A chamada ainda é bem-sucedida, mas dessa vez nenhum valor é retornado porque a publicação já estava excluída no DynamoDB.

```
{
  "data": {
    "deletePost": null
  }
}
```

Configurar o resolvidor allPost (Scan do DynamoDB)

Até agora, a API só será útil se você souber o id de cada publicação a ser examinada. Vamos adicionar um novo resolvidor que retornará todas as publicações na tabela.

- Escolha a guia Esquema.
- No painel Esquema, modifique o tipo Query para adicionar uma nova consulta allPost da seguinte forma:

```
type Query {  
  allPost(count: Int, nextToken: String): PaginatedPosts!  
  getPost(id: ID): Post  
}
```

- Adicione um novo tipo PaginationPosts:

```
type PaginatedPosts {  
  posts: [Post!]!  
  nextToken: String  
}
```

- Escolha Salvar.
- No painel Tipos de dados à direita, encontre o campo allPost recém-criado no tipo Consulta e, em seguida, escolha Anexar.
- No menu Ação, escolha Atualizar runtime e selecione Resolvidor de unidade (somente VTL).
- Em Data source name (Nome da fonte de dados), escolha PostDynamoDBTable.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
{  
  "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  
}
```

Esse resolvedor tem dois argumentos opcionais: `count`, que especifica o número máximo de itens que serão retornados em uma única chamada, e `nextToken`, que pode ser usado para recuperar o próximo conjunto de resultados, (você mostrará de onde vem o valor para `nextToken` posteriormente).

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
{
  "posts": $utils.toJson($context.result.items)
  #if( ${context.result.nextToken} )
    , "nextToken": $util.toJson($context.result.nextToken)
  #end
}
```

Observação: esse modelo de mapeamento da resposta é diferente de todos os outros até agora. O resultado da consulta `allPost` é um `PaginatedPosts`, que contém uma lista de publicações e um token de paginação. A forma desse objeto é diferente da retornada do AWS AppSync DynamoDB Resolver: a lista de publicações é chamada `items` nos resultados do AWS AppSync DynamoDB Resolver, mas é chamada de entrada. `posts PaginatedPosts`

- Escolha Salvar.

Para obter mais informações sobre o mapeamento da solicitação `Scan`, consulte a documentação de referência [Scan](#).

Chamar a API para verificar todas as publicações

Agora, o resolvedor foi configurado e AWS AppSync sabe como traduzir uma `allPost` consulta recebida em uma operação do DynamoDB. `Scan` Agora você pode verificar a tabela para recuperar todas as publicações.

No entanto, antes de testar, é necessário preencher a tabela com alguns dados, pois tudo que foi usado até agora já foi excluído.

- Escolha a guia Consultas.
- No painel Consultas, cole a seguinte mutação:

```
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 }
  }

```

- Escolha Executar consulta (o botão de reprodução laranja).

Agora vamos examinar a tabela, retornando cinco resultados por vez.

- No painel Consultas, cole a seguinte consulta:

```

query allPost {
  allPost(count: 5) {
    posts {
      id
      title
    }
    nextToken
  }
}

```

- Escolha Executar consulta (o botão de reprodução laranja).
- As primeiras cinco publicações devem aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```

{
  "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":
"eyJ2ZXJzaW9uIjoxLCJ0b2t1biI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eTh0SmRvcG9ad2RHYjI
}
}
}

```

Você tem cinco resultados e um `nextToken` que pode ser usado para obter o próximo conjunto de resultados.

- Atualize a consulta `allPost` para incluir o `nextToken` do conjunto de resultados anterior:

```

query allPost {
  allPost(
    count: 5
    nextToken:
"eyJ2ZXJzaW9uIjoxLCJ0b2t1biI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eTh0SmRvcG9ad2RHYjI
  ) {
    posts {
      id
      author
    }
  }
}

```

```
    nextToken
  }
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- As quatro publicações restantes devem aparecer no painel de resultados à direita do painel de consulta. Não há `nextToken` nesse conjunto de resultados pois todos os nove resultados foram encontrados, sem nenhum restante. A aparência deve ser semelhante à seguinte:

```
{
  "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
    }
  }
}
```

Configurando o allPostsBy Author Resolver (DynamoDB Query)

Além de verificar todas as publicações do DynamoDB, também é possível consultar o DynamoDB para recuperar as publicações criadas por um determinado autor. A tabela do DynamoDB criada anteriormente já possui um `GlobalSecondaryIndex` chamado `author-index` que podemos

usar com uma operação Query do DynamoDB para recuperar todos as publicações criadas por um determinado autor.

- Escolha a guia Esquema.
- No painel Esquema, modifique o tipo Query para adicionar uma nova consulta `allPostsByAuthor` da seguinte forma:

```
type Query {
  allPostsByAuthor(author: String!, count: Int, nextToken: String): PaginatedPosts!
  allPost(count: Int, nextToken: String): PaginatedPosts!
  getPost(id: ID): Post
}
```

Observação: isso usa o mesmo tipo `PaginatedPosts` usado com a consulta `allPost`.

- Escolha Salvar.
- No painel Tipos de dados à direita, localize o campo `allPostsByAutor` recém-criado no Tipo de consulta e escolha Anexar.
- No menu Ação, escolha Atualizar runtime e selecione Resolvedor de unidade (somente VTL).
- Em Data source name (Nome da fonte de dados), escolha `PostDynamoDBTable`.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
{
  "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
}
```

Como o resolvedor `allPost`, esse resolvedor tem dois argumentos opcionais: `count`, que especifica o número máximo de itens que serão retornados em uma única chamada e `nextToken`, que pode ser usado para recuperar o próximo conjunto de resultados (o valor para `nextToken` pode ser obtido de uma chamada anterior).

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
{
  "posts": $utils.toJson($context.result.items)
  #if( ${context.result.nextToken} )
    , "nextToken": $util.toJson($context.result.nextToken)
  #end
}
```

Observação: esse é o mesmo modelo de mapeamento da resposta usado no resolvedor `allPost`.

- Escolha Salvar.

Para obter mais informações sobre o mapeamento da solicitação Query, consulte a documentação de referência [Consulta](#).

Chamar a API para consultar todas as publicações de um autor

Agora, o resolvedor foi configurado e AWS AppSync sabe como traduzir uma `allPostsByAuthor` mutação recebida em uma operação do DynamoDB em relação ao índiceQuery. `author-index`. Agora você pode consultar a tabela para recuperar todas as publicações de um determinado autor.

Antes de fazer isso, no entanto, vamos preencher a tabela com mais algumas publicações pois até agora todas têm o mesmo autor.

- Escolha a guia Consultas.
- No painel Consultas, cole a seguinte mutação:

```
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 }
```



```
}
```

- Escolha Executar consulta (o botão de reprodução laranja).

Agora, vamos consultar a tabela, retornando todas as publicações de autoria da Nadia.

- No painel Consultas, cole a seguinte consulta:

```
query allPostsByAuthor {  
  allPostsByAuthor(author: "Nadia") {  
    posts {  
      id  
      title  
    }  
    nextToken  
  }  
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- Todas as publicações de autoria da Nadia devem aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```
{  
  "data": {  
    "allPostsByAuthor": {  
      "posts": [  
        {  
          "id": "10",  
          "title": "The cutest dog in the world"  
        },  
        {  
          "id": "11",  
          "title": "Did you know...?"  
        }  
      ],  
      "nextToken": null  
    }  
  }  
}
```

A paginação funciona para Query da mesma forma que funciona para Scan. Por exemplo, vamos procurar todas as publicações por AUTHORNAME, obtendo cinco por vez.

- No painel Consultas, cole a seguinte consulta:

```
query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    count: 5
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- Todas as publicações de autoria da AUTHORNAME devem aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```
{
  "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":
"eyJ2ZXJzaW9uIjoxLCJ0b2t1biI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eTh0SmRvcG9ad2RHYjI
  }
}

```

- Atualize o argumento `nextToken` com o valor retornado pela consulta anterior da seguinte forma:

```

query allPostsByAuthor {
  allPostsByAuthor(
    author: "AUTHORNAME"
    count: 5
    nextToken:
"eyJ2ZXJzaW9uIjoxLCJ0b2t1biI6IkFRSUNBSGo4eHR0RG0xWXhUa1F0cEhXMEp1R3B0M1B3eTh0SmRvcG9ad2RHYjI
  ) {
    posts {
      id
      title
    }
    nextToken
  }
}

```

- Escolha Executar consulta (o botão de reprodução laranja).
- As publicações restantes de autoria de `AUTHORNAME` devem aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```

{
  "data": {
    "allPostsByAuthor": {
      "posts": [
        {
          "id": "8",
          "title": "A series of posts, Volume 8"
        },
        {
          "id": "5",
          "title": "A series of posts, Volume 5"
        }
      ]
    }
  }
}

```

```
    },
    {
      "id": "3",
      "title": "A series of posts, Volume 3"
    },
    {
      "id": "9",
      "title": "A series of posts, Volume 9"
    }
  ],
  "nextToken": null
}
}
```

Uso de conjuntos

Até esse ponto, o Post tipo era um key/value objeto plano. Você também pode modelar objetos complexos com o resolvidor de AWS AppSyncDynamo banco de dados, como conjuntos, listas e mapas.

Vamos atualizar o tipo Post para incluir tags. Uma publicação pode ter 0 ou mais tags, que são armazenadas no DynamoDB como um Conjunto de strings. Vamos configurar também algumas mutações para adicionar e remover tags, e uma nova consulta para verificar as publicações com uma tag específica.

- Escolha a guia Esquema.
- No painel Esquema, modifique o tipo Post para adicionar um novo campo tags da seguinte forma:

```
type Post {
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
  tags: [String!]
```

```
}
```

- No painel Esquema, modifique o tipo Query para adicionar uma nova consulta `allPostsByTag` da seguinte forma:

```
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  
}
```

- No painel Esquema, modifique o tipo Mutation para adicionar novas mutações `addTag` e `removeTag` da seguinte forma:

```
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!  
}
```

- Escolha Salvar.
- No painel Tipos de dados à direita, localize o campo `allPostsByTag` recém-criado no Tipo de consulta e escolha Anexar.
- Em Data source name (Nome da fonte de dados), escolha `PostDynamoDBTable`.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
{
  "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
}
```

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
{
  "posts": $utils.toJson($context.result.items)
  #if( ${context.result.nextToken} )
    , "nextToken": $util.toJson($context.result.nextToken)
  #end
}
```

- Escolha Salvar.
- No painel Tipos de dados à direita, encontre o campo addTag recém-criado no tipo Mutação e, em seguida, escolha Anexar.
- Em Data source name (Nome da fonte de dados), escolha PostDynamoDBTable.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
{
  "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 }
    }
  }
}

```

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
$utils.toJson($context.result)
```

- Escolha Salvar.
- No painel Tipos de dados à direita, encontre o campo removeTag recém-criado no tipo Mutação e, em seguida, escolha Anexar.
- Em Data source name (Nome da fonte de dados), escolha PostDynamoDBTable.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```

{
  "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 }
    }
  }
}

```

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
$utils.toJson($context.result)
```

- Escolha Salvar.

Chamar a API para trabalhar com tags

Agora que você configurou os resolvedores, AWS AppSync sabe como traduzir as entradas addTag e as allPostsByTag solicitações em DynamoDB e operações. removeTag UpdateItem Scan

Para testar, vamos selecionar uma das publicações criadas anteriormente. Por exemplo, vamos usar uma publicação de autoria da Nadia.

- Escolha a guia Consultas.
- No painel Consultas, cole a seguinte consulta:

```
query allPostsByAuthor {  
  allPostsByAuthor(  
    author: "Nadia"  
  ) {  
    posts {  
      id  
      title  
    }  
    nextToken  
  }  
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- Todas as publicações da Nadia devem aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```
{  
  "data": {  
    "allPostsByAuthor": {  
      "posts": [  
        {  
          "id": "10",  
          "title": "The cutest dog in the world"  
        },  
        {  
          "id": "11",  
          "title": "Did you known...?"  
        }  
      ],  
      "nextToken": null  
    }  
  }  
}
```

- Vamos usar aquela com o título "The cutest dog in the world". Anote seu id pois ele será usado posteriormente.

Agora vamos tentar adicionar uma tag dog.

- No painel Consultas, cole a mutação a seguir. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente.

```
mutation addTag {  
  addTag(id:10 tag: "dog") {  
    id  
    title  
    tags  
  }  
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- A publicação é atualizada com a nova tag.

```
{  
  "data": {  
    "addTag": {  
      "id": "10",  
      "title": "The cutest dog in the world",  
      "tags": [  
        "dog"  
      ]  
    }  
  }  
}
```

Adicione uma ou mais tags da seguinte forma:

- Atualize a mutação para alterar o argumento `tag` para `puppy`.

```
mutation addTag {  
  addTag(id:10 tag: "puppy") {  
    id  
    title  
    tags  
  }  
}
```

- Escolha Executar consulta (o botão de reprodução laranja).

- A publicação é atualizada com a nova tag.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog",
        "puppy"
      ]
    }
  }
}
```

Também é possível excluir tags:

- No painel Consultas, cole a mutação a seguir. Também será necessário atualizar o argumento `id` para o valor anotado anteriormente.

```
mutation removeTag {
  removeTag(id:10 tag: "puppy") {
    id
    title
    tags
  }
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- A publicação é atualizada e a tag puppy é excluída.

```
{
  "data": {
    "addTag": {
      "id": "10",
      "title": "The cutest dog in the world",
      "tags": [
        "dog"
      ]
    }
  }
}
```

```
}
```

Também é possível pesquisar todas as publicações com uma tag:

- No painel Consultas, cole a seguinte consulta:

```
query allPostsByTag {  
  allPostsByTag(tag: "dog") {  
    posts {  
      id  
      title  
      tags  
    }  
    nextToken  
  }  
}
```

- Escolha Executar consulta (o botão de reprodução laranja).
- Todas as publicações com a tag dog são retornadas da seguinte forma:

```
{  
  "data": {  
    "allPostsByTag": {  
      "posts": [  
        {  
          "id": "10",  
          "title": "The cutest dog in the world",  
          "tags": [  
            "dog",  
            "puppy"  
          ]  
        }  
      ],  
      "nextToken": null  
    }  
  }  
}
```

Uso de listas e mapas

Além de usar conjuntos do DynamoDB, também é possível usar listas e mapas do DynamoDB para modelar dados complexos em um único objeto.

Vamos adicionar a capacidade de adicionar comentários em publicações. Isso será modelado como uma lista de objetos de mapa no objeto Post no DynamoDB.

Observação: em um aplicativo real, os comentários seriam modelados em suas próprias tabelas. Para este tutorial, basta adicioná-los à tabela Post.

- Escolha a guia Esquema.
- No painel Esquema, adicione um novo tipo Comment da seguinte forma:

```
type Comment {  
  author: String!  
  comment: String!  
}
```

- No painel Esquema, modifique o tipo Post para adicionar um novo campo comments da seguinte forma:

```
type Post {  
  id: ID!  
  author: String  
  title: String  
  content: String  
  url: String  
  ups: Int!  
  downs: Int!  
  version: Int!  
  tags: [String!]  
  comments: [Comment!]  
}
```

- No painel Esquema, modifique o tipo Mutation para adicionar uma nova mutação addComment da seguinte forma:

```
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!
}

```

- Escolha Salvar.
- No painel Tipos de dados à direita, encontre o campo addComment recém-criado no tipo Mutação e, em seguida, escolha Anexar.
- Em Data source name (Nome da fonte de dados), escolha PostDynamoDBTable.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```

{
  "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)  
  }  
}  
}
```

Essa expressão de atualização anexará uma lista que contém o novo comentário à lista `comments` existente. Se a lista ainda não existir, ela será criada.

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte:

```
$utils.toJson($context.result)
```

- Escolha Salvar.

Chamar a API para adicionar um comentário

Agora que você configurou os resolvedores, AWS AppSync sabe como traduzir as `addComment` solicitações recebidas em operações do DynamoDB. `UpdateItem`

Vamos testar adicionando um comentário à mesma publicação na qual as tags foram adicionadas.

- Escolha a guia Consultas.
- No painel Consultas, cole a seguinte consulta:

```
mutation addComment {  
  addComment(  
    id:10  
    author: "Steve"  
    comment: "Such a cute dog."  
  ) {  
    id  
    comments {  
      author  
      comment  
    }  
  }  
}
```

- Escolha Executar consulta (o botão de reprodução laranja).

- Todas as publicações da Nadia devem aparecer no painel de resultados à direita do painel de consulta. A aparência deve ser semelhante à seguinte:

```
{
  "data": {
    "addComment": {
      "id": "10",
      "comments": [
        {
          "author": "Steve",
          "comment": "Such a cute dog."
        }
      ]
    }
  }
}
```

Se você executar a solicitação várias vezes, vários comentários serão anexados à lista.

Conclusão

Neste tutorial, você criou uma API que nos permite manipular objetos Post no DynamoDB usando o GraphQL. AWS AppSync Para obter mais informações, consulte a [Referência do modelo de mapeamento do resolvedor](#).

Para limpar, você pode excluir a API AppSync GraphQL do console.

Para excluir a tabela do DynamoDB e a função do IAM que você criou para este tutorial, você pode executar o seguinte para excluir a pilha ou visitar `AWSAppSyncTutorialForAmazonDynamoDB` o console e excluir AWS CloudFormation a pilha:

```
aws cloudformation delete-stack \
  --stack-name AWSAppSyncTutorialForAmazonDynamoDB
```

Usando AWS Lambda resolvedores em AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Você pode usar AWS Lambda with AWS AppSync para resolver qualquer campo do GraphQL. Por exemplo, uma consulta do GraphQL pode enviar uma chamada para uma instância do Amazon Relational Database Service (Amazon RDS), e uma mutação do GraphQL pode ser gravada em um stream do Amazon Kinesis. Nesta seção, mostraremos como escrever uma função do Lambda que executa lógica de negócios com base na invocação de uma operação de campo do GraphQL.

Criar uma função do Lambda

O exemplo a seguir mostra uma função do Lambda escrita em Node.js que realiza operações diferentes em publicações de blog como parte de um aplicativo de publicação de blog.

```
exports.handler = (event, context, callback) => {
  console.log("Received event {}", JSON.stringify(event, 3));
  var posts = {
    "1": {"id": "1", "title": "First book", "author": "Author1", "url": "https://amazon.com/", "content": "SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1", "ups": "100", "downs": "10"},
    "2": {"id": "2", "title": "Second book", "author": "Author2", "url": "https://amazon.com", "content": "SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT", "ups": "100", "downs": "10"},
    "3": {"id": "3", "title": "Third book", "author": "Author3", "url": null, "content": null, "ups": null, "downs": null },
    "4": {"id": "4", "title": "Fourth book", "author": "Author4", "url": "https://www.amazon.com/", "content": "SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4", "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;
  }
};
```

Essa função do Lambda recupera uma publicação por ID, adiciona uma publicação, recupera uma lista de publicações e busca publicações relacionadas para determinada publicação.

Observação: a função do Lambda usa a instrução `switch` em `event.field` para determinar qual campo está sendo resolvido no momento.

Crie essa função Lambda usando o console AWS de gerenciamento ou uma AWS CloudFormation pilha. Para criar a função a partir de uma CloudFormation pilha, você pode usar o seguinte comando AWS Command Line Interface (AWS CLI):

```
aws cloudformation create-stack --stack-name AppSyncLambdaExample \  
--template-url https://s3.us-west-2.amazonaws.com/awsappsync/resources/lambda/  
LambdaCFTemplate.yaml \  
--capabilities CAPABILITY_NAMED_IAM
```

Você também pode lançar a AWS CloudFormation pilha na AWS região Oeste dos EUA (Oregon) em sua AWS conta aqui:



Configurar a fonte de dados para o Lambda

Depois de criar a função do Lambda, navegue até a API GraphQL no console AWS AppSync e escolha a guia Fontes de dados.

Escolha Criar fonte de dados, insira um Nome de fonte de dados fácil de usar (por exemplo, **Lambda**) e, em seguida, para Tipo de fonte de dados, escolha Função AWS Lambda . Em Região, escolha a mesma região que a de sua função. (Se você criou a função a partir da CloudFormation pilha fornecida, a função provavelmente está em US-WEST-2.) Para ARN da função do Lambda, escolha o nome do recurso da Amazon (ARN) da sua função do Lambda.

Depois de escolher sua função Lambda, você pode criar uma nova função AWS Identity and Access Management (IAM) (para a qual AWS AppSync atribui as permissões apropriadas) ou escolher uma função existente que tenha a seguinte política embutida:

JSON

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

```
        "Effect": "Allow",
        "Action": [
            "lambda:InvokeFunction"
        ],
        "Resource": "arn:aws:lambda:REGION:ACCOUNTNUMBER:function/
LAMBDA_FUNCTION"
    }
]
```

Você também deve configurar uma relação de confiança com AWS AppSync a função do IAM da seguinte forma:

JSON

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

Criar um esquema do GraphQL

Agora que a fonte de dados está conectada à função do Lambda, crie um esquema do GraphQL.

No editor de esquemas no AWS AppSync console, certifique-se de que seu esquema corresponda ao seguinte esquema:

```
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]  
}
```

Configurar resolvedores

Agora que você registrou uma fonte de dados do Lambda e um esquema do GraphQL válido, pode conectar seus campos do GraphQL à sua fonte de dados do Lambda usando resolvedores.

Para criar um resolvedor, você precisará de modelos de mapeamento. Para saber mais sobre modelos de mapeamento, consulte [Resolver Mapping Template Overview](#).

Para obter mais informações sobre modelos de mapeamento do Lambda, consulte [Resolver mapping template reference for Lambda](#).

Nesta etapa, você anexa um resolvedor à função do Lambda para os seguintes campos:

`getPost(id:ID!): Post`, `allPosts: [Post]`, `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!` e `Post.relatedPosts: [Post]`.

No editor de esquemas no AWS AppSync console, no lado direito, escolha Anexar resolvedor para `getPost(id:ID!): Post`.

Depois, no menu Ação, escolha Atualizar runtime e selecione Resolvedor de unidade (somente VTL).

Depois, escolha sua fonte de dados do Lambda. Na seção modelo de mapeamento da solicitação, escolha Invocar e encaminhar argumentos.

Modifique o objeto `payload` para adicionar o nome do campo. O modelo deve ser semelhante ao seguinte:

```
{
  "version": "2017-02-28",
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": $utils.toJson($context.arguments)
  }
}
```

Na seção modelo de mapeamento da resposta, escolha Retornar o resultado Lambda.

Nesse caso, use o modelo base como está. Ele deve ter a seguinte aparência:

```
$utils.toJson($context.result)
```

Escolha Salvar. Você anexou seu primeiro resolvidor com sucesso. Repita essa operação para os campos restantes da seguinte forma:

Para o modelo de mapeamento da solicitação `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!`:

```
{
  "version": "2017-02-28",
  "operation": "Invoke",
  "payload": {
    "field": "addPost",
    "arguments": $utils.toJson($context.arguments)
  }
}
```

Para o modelo de mapeamento da resposta `addPost(id: ID!, author: String!, title: String, content: String, url: String): Post!`:

```
$utils.toJson($context.result)
```

Para o modelo de mapeamento da solicitação `allPosts`: `[Post]`:

```
{
  "version": "2017-02-28",
  "operation": "Invoke",
  "payload": {
    "field": "allPosts"
  }
}
```

Para o modelo de mapeamento da resposta `allPosts`: `[Post]`:

```
$utils.toJson($context.result)
```

Para o modelo de mapeamento da solicitação `Post.relatedPosts`: `[Post]`:

```
{
  "version": "2017-02-28",
  "operation": "Invoke",
  "payload": {
    "field": "relatedPosts",
    "source": $utils.toJson($context.source)
  }
}
```

Para o modelo de mapeamento da resposta `Post.relatedPosts`: `[Post]`:

```
$utils.toJson($context.result)
```

Teste da sua API GraphQL

Agora que a função do Lambda está conectada aos resolvedores do GraphQL, você pode executar algumas mutações e consultas usando o console ou um aplicativo cliente.

No lado esquerdo do AWS AppSync console, escolha Consultas e cole o código a seguir:

Mutação `addPost`

```
mutation addPost {
  addPost(
    id: 6
  )
}
```

```
    author: "Author6"
    title: "Sixth book"
    url: "https://www.amazon.com/"
    content: "This is the book is a tutorial for using GraphQL with AWS AppSync."
  ) {
    id
    author
    title
    content
    url
    ups
    downs
  }
}
```

Consulta getPost

```
query getPost {
  getPost(id: "2") {
    id
    author
    title
    content
    url
    ups
    downs
  }
}
```

Consulta allPosts

```
query allPosts {
  allPosts {
    id
    author
    title
    content
    url
    ups
    downs
    relatedPosts {
      id
```

```
        title
      }
    }
  }
}
```

Retornar erros

Qualquer resolução de campo determinada pode resultar em um erro. Com AWS AppSync, você pode gerar erros das seguintes fontes:

- Modelo de mapeamento da solicitação ou resposta
- Função do Lambda

A partir do modelo de mapeamento

Para gerar erros intencionais, você pode usar o método auxiliar `$utils.error` do modelo VTL (Velocity Template Language). Ele utiliza como argumento uma `errorMessage`, um `errorType` e um valor `data` opcional. O `data` é útil para retornar dados adicionais de volta ao cliente quando ocorre um erro. O objeto `data` é adicionado ao `errors` na resposta final do GraphQL.

O exemplo a seguir mostra como usá-lo no modelo de mapeamento da resposta

`Post.relatedPosts: [Post]:`

```
$utils.error("Failed to fetch relatedPosts", "LambdaFailure", $context.result)
```

Isso produz uma resposta do GraphQL semelhante à seguinte:

```
{
  "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"
      }
    ]
  }
]
}

```

Onde `allPosts[0].relatedPosts` é nulo pois o erro e a `errorMessage`, o `errorType` e o `data` estão presentes no objeto `data.errors[0]`.

A partir da função do Lambda

AWS AppSync também compreende os erros que a função Lambda gera. O modelo de programação do Lambda permite gerar erros processados. Se a função Lambda gerar um erro, AWS AppSync não conseguirá resolver o campo atual. Somente a mensagem de erro retornada a partir do Lambda é definida na resposta. No momento, não é possível enviar quaisquer dados adicionais de volta para o cliente ao gerar um erro a partir da função do Lambda.

Observação: se sua função Lambda gerar um erro não tratado, AWS AppSync use a mensagem de erro que o Lambda definiu.

A seguinte função do Lambda gera um erro:

```
exports.handler = (event, context, callback) => {
  console.log("Received event {}", JSON.stringify(event, 3));
  callback("I fail. Always.");
};
```

Isso retorna uma resposta do GraphQL semelhante à seguinte:

```
{
  "data": {
    "allPosts": [
      {
        "id": "2",
        "title": "Second book",
        "relatedPosts": null
      },
      ...
    ]
  },
  "errors": [
    {
      "path": [
        "allPosts",
        0,
        "relatedPosts"
      ],
      "errorType": "Lambda:Handled",
      "locations": [
        {
          "line": 5,
          "column": 5
        }
      ],
      "message": "I fail. Always."
    }
  ]
}
```

Caso de uso avançado: agrupamento em lotes

Neste exemplo, a função do Lambda tem um campo `relatedPosts` que retorna uma lista de publicações relacionadas para uma publicação. Nas consultas de exemplo, a invocação do campo `allPosts` a partir da função do Lambda retorna cinco publicações. Como também especificamos

que queremos resolver `relatedPosts` para cada publicação retornada, a operação do campo `relatedPosts` será invocada cinco vezes.

```
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
    }
  }
}
```

Embora isso possa não parecer substancial neste exemplo específico, essa busca excessiva combinada pode prejudicar rapidamente o aplicativo.

Se você buscasse `relatedPosts` novamente nas `Posts` relacionadas retornadas na mesma consulta, o número de invocações aumentaria drasticamente.

```
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
        id
        title
        author
      }
    }
  }
}
```

```

    }
  }
}
```

Nessa consulta relativamente simples, AWS AppSync invocaria a função Lambda $1 + 5 + 25 = 31$ vezes.

Esse é um desafio bastante comum e normalmente é chamado de problema N+1 (nesse caso, $N = 5$) e pode incorrer em maior latência e mais custo para o aplicativo.

Uma forma de resolver esse problema é agrupar solicitações do resolvidor de campo semelhantes em lotes. Nesse exemplo, em vez da função do Lambda resolver uma lista de publicações relacionadas para uma única publicação, ela resolveria uma lista de publicações relacionadas para um determinado lote de publicações.

Para demonstrar isso, vamos alternar o resolvidor `Post.relatedPosts: [Post]` para um resolvidor habilitado para lotes.

No lado direito do AWS AppSync console, escolha o `Post.relatedPosts: [Post]` resolvidor existente. Altere o modelo de mapeamento da solicitação para o seguinte:

```

{
  "version": "2017-02-28",
  "operation": "BatchInvoke",
  "payload": {
    "field": "relatedPosts",
    "source": $utils.toJson($context.source)
  }
}
```

Somente o campo `operation` foi alterado de `Invoke` para `BatchInvoke`. O campo de carga útil agora se torna uma matriz do que é especificado no modelo. Neste exemplo, a função do Lambda recebe o seguinte como entrada:

```

[
  {
    "field": "relatedPosts",
    "source": {
      "id": 1
    }
  },
  {
```

```

    "field": "relatedPosts",
    "source": {
      "id": 2
    }
  },
  ...
]

```

Quando `BatchInvoke` for especificado no modelo de mapeamento da solicitação, a função do Lambda recebe uma lista de solicitações e retorna uma lista de resultados.

Especificamente, a lista de resultados deve corresponder ao tamanho e à ordem das entradas da carga útil da solicitação para que AWS AppSync possa corresponder aos resultados adequadamente.

Neste exemplo de processamento em lotes, a função do Lambda retorna um lote de resultados da seguinte forma:

```

[
  [{"id": "2", "title": "Second book"}, {"id": "3", "title": "Third book"}], //
  relatedPosts for id=1
  [{"id": "3", "title": "Third book"}]
  // relatedPosts for id=2
]

```

A seguinte função do Lambda em Node.js demonstra essa funcionalidade de agrupamento em lotes para o campo `Post.relatedPosts` da seguinte forma:

```

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;
}
};

```

Retornar erros individuais

Os exemplos anteriores mostram que é possível retornar um único erro a partir da função do Lambda ou gerar um erro a partir dos modelos de mapeamento. Para invocações em lote, gerar um erro a partir da função do Lambda sinaliza um lote inteiro como falho. Isso pode ser aceitável em cenários específicos onde ocorreu um erro irreversível, como uma conexão com falha a um armazenamento de dados. No entanto, nos casos em que alguns itens no lote são bem-sucedidos e outros falham, é

possível retornar ambos os erros e os dados válidos. Como AWS AppSync exige a resposta em lote para listar elementos que correspondam ao tamanho original do lote, você deve definir uma estrutura de dados que possa diferenciar dados válidos de um erro.

Por exemplo, se espera-se que a função do Lambda retorne um lote de publicações relacionadas, você pode, em vez disso, retornar uma lista de objetos Response em que cada objeto possui campos opcionais de dados, errorMessage e errorType. Se o campo errorMessage estiver presente, isso significa que ocorreu um erro.

O código a seguir mostra como você pode atualizar a função do Lambda:

```
exports.handler = (event, context, callback) => {
  console.log("Received event {}", JSON.stringify(event, 3));
  var posts = {
    "1": {"id": "1", "title": "First book", "author": "Author1", "url": "https://amazon.com/", "content": "SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1", "ups": "100", "downs": "10"},
    "2": {"id": "2", "title": "Second book", "author": "Author2", "url": "https://amazon.com", "content": "SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT", "ups": "100", "downs": "10"},
    "3": {"id": "3", "title": "Third book", "author": "Author3", "url": null, "content": null, "ups": null, "downs": null },
    "4": {"id": "4", "title": "Fourth book", "author": "Author4", "url": "https://www.amazon.com/", "content": "SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4", "ups": "1000", "downs": "0"},
    "5": {"id": "5", "title": "Fifth book", "author": "Author5", "url": "https://www.amazon.com/", "content": "SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT", "ups": "50", "downs": "0"} };

  var relatedPosts = {
    "1": [posts['4']],
    "2": [posts['3'], posts['5']],
    "3": [posts['2'], posts['1']],
    "4": [posts['2'], posts['1']],
    "5": []
  };

  console.log("Got a BatchInvoke Request. The payload has %d items to resolve.", event.length);
  // event is now an array
  var field = event[0].field;
```

```

switch(field) {
  case "relatedPosts":
    var results = [];
    results.push({ 'data': relatedPosts['1'] });
    results.push({ 'data': relatedPosts['2'] });
    results.push({ 'data': null, 'errorMessage': 'Error Happened', 'errorType':
'ERROR' });
    results.push(null);
    results.push({ 'data': relatedPosts['3'], 'errorMessage': 'Error Happened
with last result', 'errorType': 'ERROR' });
    callback(null, results);
    break;
  default:
    callback("Unknown field, unable to resolve" + field, null);
    break;
}
};

```

Para este exemplo, o modelo de mapeamento da resposta a seguir analisa cada item da função do Lambda e sinaliza os erros que ocorrerem:

```

#if( $context.result && $context.result.errorMessage )
  $utils.error($context.result.errorMessage, $context.result.errorType,
$context.result.data)
#else
  $utils.toJson($context.result.data)
#end

```

Esse exemplo retorna uma resposta do GraphQL semelhante à seguinte:

```

{
  "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"
}
]
```

Configuração do tamanho máximo do lote

Por padrão, ao usar `batchInvoke`, AWS AppSync envia solicitações para sua função Lambda em lotes de até cinco itens. Você pode configurar o tamanho máximo do lote dos seus resolvedores do Lambda.

Para configurar o tamanho máximo de lotes em um resolvedor, use o seguinte comando no 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

Ao fornecer um modelo de mapeamento de solicitação, você deve usar a operação `BatchInvoke` para usar lotes.

Você também pode usar o seguinte comando para ativar e configurar o lote em resolvedores diretos do Lambda:

```
$ aws appsync create-resolver --api-id <api-id> --type-name Query --field-name
relatedPosts \
--data-source-name "<lambda-datasource>" \
--max-batch-size X
```

Configuração de tamanho máximo de lote com modelos VTL

Para resolvedores do Lambda que possuem modelos de solicitação de VTL, o tamanho máximo do lote não terá efeito, a menos que o tenham especificado diretamente como uma operação `BatchInvoke` em VTL. Da mesma forma, se você estiver executando uma mutação de nível superior, o processamento em lote não será conduzido para mutações porque a especificação do GraphQL exige que mutações paralelas sejam executadas sequencialmente.

Por exemplo, considere as seguintes mutações:

```
type Mutation {
  putItem(input: Item): Item
  putItems(inputs: [Item]): [Item]
}
```

Usando a primeira mutação, podemos criar 10 `Items` conforme mostrado no trecho abaixo:

```
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
}
}
```

Neste exemplo, os Items não serão agrupados em um grupo de 10, mesmo que o tamanho máximo do lote seja definido como 10 no resolvidor do Lambda. Em vez disso, eles serão executados sequencialmente de acordo com a especificação do GraphQL.

Para realizar uma mutação em lote real, você pode seguir o exemplo abaixo usando a segunda mutação:

```
mutation MyMutation {
```

```
putItems([$someItem1, $someItem2, $someItem3,$someItem4, $someItem5, $someItem6,
$someItem7, $someItem8, $someItem9, $someItem10]) {
  id,
  name
}
```

Para obter mais informações sobre como usar lotes com resolvedores diretos do Lambda, consulte [Resolvedores diretos do Lambda](#).

Usando resolvedores OpenSearch do Amazon Service em AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync suporta o uso do Amazon OpenSearch Service a partir de domínios que você provisionou em sua própria AWS conta, desde que eles não existam dentro de uma VPC. Assim que os domínios forem provisionados, conecte-se a eles usando uma fonte de dados, no momento em que pode configurar um resolvedor no esquema para realizar operações do GraphQL, como consultas, mutações e assinaturas. Esse tutorial apresentará alguns exemplos comuns.

Para obter mais informações, consulte a [Referência do modelo de mapeamento do Resolver para OpenSearch](#).

Configuração com um clique

Para configurar automaticamente um endpoint do GraphQL com o AWS AppSync Amazon OpenSearch Service configurado, você pode usar este modelo: AWS CloudFormation

Launch Stack



Após a conclusão da AWS CloudFormation implantação, você pode pular diretamente para a execução de consultas e mutações do [GraphQL](#).

Criar um novo domínio OpenSearch de serviço

Para começar com este tutorial, você precisa de um domínio OpenSearch de serviço existente. Caso não tenha um, use o exemplo a seguir. Observe que pode levar até 15 minutos para que um domínio OpenSearch de serviço seja criado antes que você possa prosseguir com a integração com uma fonte de AWS AppSync dados.

```
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
```

Você pode lançar a seguinte AWS CloudFormation pilha na região Oeste dos EUA 2 (Oregon) em sua AWS conta:

A button with a yellow gradient background, rounded corners, and a blue play icon on the right. The text "Launch Stack" is in bold black font.

Configurar fonte de dados para OpenSearch serviço

Depois que o domínio OpenSearch Service for criado, navegue até a API do AWS AppSync GraphQL e escolha a guia Fontes de dados. Selecione Novo e insira um nome acessível para a fonte de dados, como "oss". Em seguida, escolha OpenSearch o domínio da Amazon para o tipo de fonte de dados, escolha a região apropriada e você deverá ver seu domínio OpenSearch de serviço listado. Depois de selecioná-la, você pode criar uma nova função e AWS AppSync atribuir as permissões apropriadas à função ou escolher uma função existente, que tenha a seguinte política embutida:

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:REGION:ACCOUNTNUMBER:domain/democluster/*"
    ]
}
]
}

```

Você também precisará estabelecer uma relação de confiança com AWS AppSync essa função:

JSON

```

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

```

Além disso, o domínio do OpenSearch serviço tem sua própria política de acesso, que você pode modificar por meio do console do Amazon OpenSearch Service. Você precisará adicionar uma política semelhante à seguinte, com as ações e os recursos apropriados para o domínio do OpenSearch Serviço. Observe que o principal será a função da fonte de AppSync dados, que, se você permitir que o console a crie, pode ser encontrada no console do IAM.

JSON

```

{
  "Version": "2012-10-17",

```

```

    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::ACCOUNTNUMBER:role/service-role/
APPSYNC_DATASOURCE_ROLE"
        },
        "Action": [
          "es:ESHttpDelete",
          "es:ESHttpHead",
          "es:ESHttpGet",
          "es:ESHttpPost",
          "es:ESHttpPut"
        ],
        "Resource": "arn:aws:es:REGION:ACCOUNTNUMBER:domain/DOMAIN_NAME/*"
      }
    ]
  }
}

```

Conexão de um resolvidor

Agora que a fonte de dados está conectada ao seu domínio de OpenSearch serviço, você pode conectá-la ao seu esquema do GraphQL com um resolvidor, conforme mostrado no exemplo a seguir:

```

schema {
  query: Query
  mutation: Mutation
}

type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
}

type Mutation {
  addPost(id: ID!, author: String, title: String, url: String, ups: Int, downs: Int,
content: String): AWSJSON
}

type Post {
  id: ID!
}

```



```
author: String
title: String
url: String
ups: Int
downs: Int
content: String
}
...
```

Observe que há um tipo `Post` definido pelo usuário com um campo de `id`. Nos exemplos a seguir, presumimos que há um processo (que pode ser automatizado) para colocar esse tipo em seu domínio de `OpenSearch` serviço, que seria mapeado para uma raiz de caminho de `/post/_doc`, onde `post` está o índice. A partir desse caminho raiz, você pode executar pesquisas de documento individuais, pesquisas com curingas com `/id/post*` ou pesquisas de vários documentos com um caminho de `/post/_search`. Por exemplo, se você tiver outro tipo chamado `User`, poderá indexar documentos em um novo índice chamado `user` e depois realizar pesquisas com um caminho de `/user/_search`.

No editor de esquemas no AWS AppSync console, modifique o `Posts` esquema anterior para incluir uma consulta: `searchPosts`

```
type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  searchPosts: [Post]
}
```

Salve o esquema. À direita, em `searchPosts`, selecione `Anexar resolvedor`. No menu `Ação`, escolha `Atualizar runtime` e selecione `Resolvedor de unidade (somente VTL)`. Em seguida, escolha sua fonte `OpenSearch` de dados do Serviço. Na seção modelo de mapeamento da solicitação, selecione o menu suspenso em `Postagens de consulta` para obter um modelo base. Modifique o path para `/post/_search`. Ele deve ter a seguinte aparência:

```
{
  "version": "2017-02-28",
  "operation": "GET",
  "path": "/post/_search",
  "params": {
    "headers": {},
    "queryString": {},
```

```

      "body": {
        "from": 0,
        "size": 50
      }
    }
  }
}

```

Isso pressupõe que o esquema anterior tenha documentos que foram indexados no OpenSearch Serviço sob o campo. `post`. Se você estruturar os dados de forma diferente, será necessário atualizar de forma adequada.

Na seção do modelo de mapeamento de resposta, você precisa especificar o `_source` filtro apropriado se quiser recuperar os resultados dos dados de uma consulta de OpenSearch serviço e traduzir para o GraphQL. Use o modelo a seguir:

```

[
  #foreach($entry in $context.result.hits.hits)
  #if( $velocityCount > 1 ) , #end
  $utils.toJson($entry.get("_source"))
  #end
]

```

Modificação das pesquisas

O modelo de mapeamento da solicitação anterior executa uma consulta simples para todos os registros. Digamos que você queira pesquisar um autor específico. Além disso, digamos que queira que esse autor seja um argumento definido na consulta do GraphQL. No editor de esquema do console do AWS AppSync, adicione uma consulta `allPostsByAuthor`:

```

type Query {
  getPost(id: ID!): Post
  allPosts: [Post]
  allPostsByAuthor(author: String!): [Post]
  searchPosts: [Post]
}

```

Agora escolha Anexar resolvidor e selecione a fonte OpenSearch de dados do Serviço, mas use o exemplo a seguir no modelo de mapeamento de resposta:

```

{

```

```

    "version": "2017-02-28",
    "operation": "GET",
    "path": "/post/_search",
    "params": {
      "headers": {},
      "queryString": {},
      "body": {
        "from": 0,
        "size": 50,
        "query": {
          "match": {
            "author": $util.toJson($context.arguments.author)
          }
        }
      }
    }
  }
}

```

Observe que o body é preenchido com uma consulta de termo para o campo `author`, que é enviada a partir do cliente como um argumento. Também é possível ter informações pré-preenchidas, como texto padrão, ou até mesmo usar outros [utilitários](#).

Se estiver usando esse resolvidor, preencha o modelo de mapeamento da resposta com as mesmas informações que o exemplo anterior.

Adicionando dados ao OpenSearch serviço

Talvez você queira adicionar dados ao seu domínio de OpenSearch serviço como resultado de uma mutação do GraphQL. Esse é um poderoso mecanismo para pesquisa e outras finalidades. Como você pode usar assinaturas do GraphQL para [tornar seus dados em tempo real](#), ele serve como um mecanismo para notificar os clientes sobre atualizações nos dados em seu domínio de serviço. OpenSearch

Retorne à página Esquema no AWS AppSync console e selecione Anexar resolvidor para a `addPost()` mutação. Selecione a fonte OpenSearch de dados do Serviço novamente e use o seguinte modelo de mapeamento de resposta para o `Posts` esquema:

```

{
  "version": "2017-02-28",
  "operation": "PUT",
  "path": $util.toJson("/post/_doc/$context.arguments.id"),

```

```
"params":{
  "headers":{},
  "queryString":{},
  "body":{
    "id": $util.toJson($context.arguments.id),
    "author": $util.toJson($context.arguments.author),
    "ups": $util.toJson($context.arguments.ups),
    "downs": $util.toJson($context.arguments.downs),
    "url": $util.toJson($context.arguments.url),
    "content": $util.toJson($context.arguments.content),
    "title": $util.toJson($context.arguments.title)
  }
}
```

Como antes, este é um exemplo de como os dados podem ser estruturados. Se tiver diferentes nomes de campos ou índices, é necessário atualizar o path e o body conforme apropriado. Esse exemplo também mostra como usar `$context.arguments` para preencher o modelo a partir dos argumentos da mutação do GraphQL.

Antes de prosseguir, use o seguinte modelo de mapeamento de resposta, que retornará o resultado da operação de mutação ou informações de erro como saída:

```
#if($context.error)
  $util.toJson($ctx.error)
#else
  $util.toJson($context.result)
#end
```

Recuperação de um único documento

Finalmente, se quiser usar a consulta `getPost(id:ID)` no esquema para retornar um documento individual, encontre essa consulta no editor de esquema do console do AWS AppSync e selecione Attach resolver (Anexar resolvedor). Selecione a fonte OpenSearch de dados do Serviço novamente e use o seguinte modelo de mapeamento:

```
{
  "version":"2017-02-28",
  "operation":"GET",
  "path": $util.toJson("post/_doc/$context.arguments.id"),
  "params":{
```

```
    "headers": {},
    "queryString": {},
    "body": {}
  }
}
```

Como o path acima usa o argumento `id` com um corpo vazio, isso retorna o único documento. No entanto, é necessário usar o seguinte modelo de mapeamento da resposta, pois agora você está retornando um único item, e não uma lista:

```
$utils.toJson($context.result.get("_source"))
```

Executar consultas e mutações

Agora você deve ser capaz de realizar operações do GraphQL em seu domínio OpenSearch Service. Navegue até a guia Consultas do AWS AppSync console e adicione um novo registro:

```
mutation addPost {
  addPost (
    id: "12345"
    author: "Fred"
    title: "My first book"
    content: "This will be fun to write!"
    url: "publisher website",
    ups: 100,
    downs: 20
  )
}
```

Você verá o resultado da mutação à direita. Da mesma forma, agora você pode executar uma `searchPosts` consulta em seu domínio OpenSearch de serviço:

```
query searchPosts {
  searchPosts {
    id
    title
    author
    content
  }
}
```

Práticas recomendadas

- OpenSearch O serviço deve ser para consultar dados, não como seu banco de dados principal. [Talvez você queira usar o OpenSearch Service em conjunto com o Amazon DynamoDB, conforme descrito em Combinação de resolvedores GraphQL.](#)
- Só conceda acesso ao seu domínio permitindo que a função AWS AppSync de serviço acesse o cluster.
- Você pode começar pequeno no desenvolvimento, com o cluster de menor custo e, em seguida, migrar para um cluster maior com alta disponibilidade (HA) à medida que entrar na produção.

Usando resolvedores locais em AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync permite que você use fontes de dados compatíveis (AWS Lambda Amazon DynamoDB ou OpenSearch Amazon Service) para realizar várias operações. No entanto, em determinados cenários, uma chamada para uma fonte de dados compatível pode não ser necessária.

É aí que o resolvedor local se torna útil. Em vez de chamar uma fonte de dados remota, o resolvedor local apenas encaminhará o resultado do modelo de mapeamento da solicitação para o modelo de mapeamento da resposta. A resolução do campo não deixará o AWS AppSync.

Os resolvedores locais são úteis em vários casos de uso. O caso de uso mais popular é a publicação de notificações sem acionar uma chamada da fonte de dados. Para demonstrar esse caso de uso, vamos criar um aplicativo de paginação, onde os usuários podem pular uns aos outros. Esse exemplo utiliza Assinaturas, portanto se não estiver familiarizado com Assinaturas, siga o tutorial [Dados em tempo real](#).

Criar o aplicativo de paginação

No aplicativo de paginação, os clientes podem assinar uma caixa de entrada e enviar páginas para outros clientes. Cada página inclui uma mensagem. Aqui está o esquema:

```
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  
}
```

Vamos anexar um resolvidor no campo `Mutation.page`. No painel Esquema, clique em Anexar resolvidor ao lado da definição do campo no painel à direita. Crie uma nova fonte de dados do tipo Nenhuma e nomeie-a `PageDataSource`.

Para o modelo de mapeamento da solicitação, insira:

```
{  
  "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()"  
  }  
}
```

E para o modelo de mapeamento da resposta, selecione o padrão Encaminhar o resultado. Salve o resolutor. O aplicativo agora está pronto, vamos paginar!

Enviar e assinar páginas

Para que os clientes recebam páginas, primeiro eles devem assinar uma caixa de entrada.

No painel Consultas vamos executar a assinatura inbox:

```
subscription Inbox {  
  inbox(to: "Nadia") {  
    body  
    to  
    from  
    sentAt  
  }  
}
```

Nadia receberá páginas sempre que a mutação `Mutation.page` for invocada. Vamos invocar a mutação executando a mutação:

```
mutation Page {  
  page(to: "Nadia", body: "Hello, World!") {  
    body  
    to  
    from  
    sentAt  
  }  
}
```

Nós acabamos de demonstrar o uso de resolvedores locais, enviando uma Página e recebendo-a sem sair do AWS AppSync.

Combinação de resolvedores do GraphQL no AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Os resolvedores e campos em um esquema do GraphQL têm relacionamentos 1:1 com um grande grau de flexibilidade. Como uma fonte de dados é configurada em um resolvedor independentemente de um esquema, você tem a capacidade de resolver ou manipular os tipos do GraphQL por meio de diferentes fontes de dados, misturando e combinando em um esquema que melhor atender às suas necessidades.

Os cenários de exemplo a seguir demonstram como misturar e combinar fontes de dados no seu esquema. Antes de começar, recomendamos que você esteja familiarizado com a configuração de fontes de dados e resolvedores para AWS Lambda o Amazon DynamoDB OpenSearch e o Amazon Service, conforme descrito nos tutoriais anteriores.

Esquema de exemplo

O esquema abaixo tem um tipo de Post com três operações Query e três operações Mutation definidas:

```
type Post {
  id: ID!
  author: String!
  title: String
  content: String
  url: String
  ups: Int
  downs: Int
  version: Int!
}

type Query {
  allPost: [Post]
  getPost(id: ID!): Post
  searchPosts: [Post]
}

type Mutation {
  addPost(
    id: ID!,
    author: String!,
    title: String,
    content: String,
    url: String
  ): Post
  updatePost(
```

```

    id: ID!,
    author: String!,
    title: String,
    content: String,
    url: String,
    ups: Int!,
    downs: Int!,
    expectedVersion: Int!
  ): Post
  deletePost(id: ID!): Post
}
```

Neste exemplo, existem seis resolvedores para anexar. Uma maneira possível seria que todos viessem de uma tabela do Amazon DynamoDB, chamada Posts, em que `AllPosts` executa uma verificação e `searchPosts` executa uma consulta, conforme descrito na [Referência do modelo de mapeamento de resolvedor do DynamoDB](#). No entanto, existem alternativas para atender às suas necessidades de negócios, como fazer com que essas consultas do GraphQL sejam resolvidas a partir do Lambda ou do Service. OpenSearch

Alterar os dados por meio de resolvedores

Talvez você precise retornar os resultados de um banco de dados como o DynamoDB (ou o Amazon Aurora) aos clientes com alguns dos atributos alterados. Isso pode ser devido à formatação dos tipos de dados, como diferenças de carimbos de data/hora nos clientes, ou para lidar com problemas de compatibilidade com versões anteriores. Para fins ilustrativos no exemplo a seguir, uma função AWS Lambda manipula os votos positivos e negativos para postagens de blog, atribuindo a elas números aleatórios cada vez que o resolvedor do GraphQL é invocado:

```

'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);
  });
};
```

Essa é uma função do Lambda perfeitamente válida e pode ser anexada ao campo `AllPosts` no esquema do GraphQL para que qualquer consulta que retorne todos os resultados obtenha números aleatórios para os votos positivos/negativos.

DynamoDB e serviços OpenSearch

Para alguns aplicativos, você pode realizar mutações ou consultas de pesquisa simples no DynamoDB e fazer com que um processo em segundo plano transfira documentos para o Service. OpenSearch Em seguida, você pode simplesmente anexar o `searchPosts Resolver` à fonte OpenSearch de dados do Service e retornar os resultados da pesquisa (de dados originados no DynamoDB) usando uma consulta GraphQL. Isso pode ser extremamente poderoso ao adicionar operações de pesquisa avançadas aos aplicativos, como palavra-chave, correspondências de palavras confusas ou até mesmo pesquisas geoespaciais. A transferência de dados do DynamoDB pode ser feita por meio de um processo ETL ou, como alternativa, você pode transmitir do DynamoDB usando o Lambda. Você pode lançar um exemplo completo disso usando a seguinte AWS CloudFormation pilha na região Oeste dos EUA 2 (Oregon) em sua AWS conta:

Launch Stack 

O esquema desse exemplo permitirá que você adicione publicações usando um resolvedor do DynamoDB da seguinte forma:

```
mutation add {
  putPost(author:"Nadia"
    title:"My first post"
    content:"This is some test content"
    url:"https://aws.amazon.com/appsync/"
  ){
    id
    title
  }
}
```

```
}
```

Isso grava dados no DynamoDB, que então transmite dados via Lambda para o OpenSearch Amazon Service, onde você pode pesquisar todas as postagens em campos diferentes. Por exemplo, como os dados estão no Amazon OpenSearch Service, você pode pesquisar os campos do autor ou do conteúdo com texto de formato livre, mesmo com espaços, da seguinte forma:

```
query searchName{
  searchAuthor(name:"  Nadia  "){
    id
    title
    content
  }
}

query searchContent{
  searchContent(text:"test"){
    id
    title
    content
  }
}
```

Como os dados são gravados diretamente no DynamoDB, você ainda pode realizar operações de pesquisa de item ou lista eficientes na tabela com as consultas `allPosts{...}` e `singlePost{...}`. Essa pilha usa o seguinte código de exemplo para transmissões do DynamoDB:

Observação: esse código é apenas um exemplo.

```
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);

```

```
};
```

Em seguida, você pode usar os streams do DynamoDB para anexá-los a uma tabela do DynamoDB com uma chave primária de, e qualquer alteração na origem do DynamoDB será id transmitida para o seu domínio de serviço. OpenSearch Para obter mais informações sobre como configurar isso, consulte a [documentação de Transmissões do DynamoDB](#).

Usando operações em lote do DynamoDB em AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync suporta o uso de operações em lote do Amazon DynamoDB em uma ou mais tabelas em uma única região. As operações compatíveis são BatchGetItem, BatchPutItem e BatchDeleteItem. Ao usar esses recursos no AWS AppSync, execute tarefas como:

- Enviar uma lista de chaves em uma única consulta e retornar os resultados de uma tabela
- Ler os registros de uma ou mais tabelas em uma única consulta
- Gravar registros em lote em uma ou mais tabelas
- Gravar ou excluir registros condicionalmente em várias tabelas que podem ter uma relação

Usar operações em lote com o DynamoDB AWS AppSync in é uma técnica avançada que exige um pouco mais de reflexão e conhecimento sobre suas operações de back-end e estruturas de tabelas. Além disso, as operações em lote AWS AppSync têm duas diferenças principais em relação às operações sem lote:

- A função da fonte de dados deve ter permissões para todas as tabelas acessadas pelo resolvedor.
- A especificação de tabela para um resolvedor faz parte do modelo de mapeamento.

Permissões

Como outros resolvedores, você precisa criar uma fonte de dados AWS AppSync e criar uma função ou usar uma existente. Como operações em lote exigem diferentes permissões em tabelas do

DynamoDB, é necessário conceder as permissões de função configuradas para ações de leitura e gravação:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "dynamodb:BatchGetItem",
        "dynamodb:BatchWriteItem"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:dynamodb:region:account:table/TABLENAME",
        "arn:aws:dynamodb:region:account:table/TABLENAME/*"
      ]
    }
  ]
}
```

Observação: as funções estão vinculadas às fontes de dados em AWS AppSync, e os resolvedores nos campos são invocados em relação a uma fonte de dados. As fontes de dados configuradas para buscar no DynamoDB têm apenas uma tabela especificada, para manter a configuração simples. Portanto, ao executar uma operação em lote para várias tabelas em um único resolvedor, que é uma tarefa mais avançada, é necessário conceder acesso à função na fonte de dados para qualquer tabela com a qual o resolvedor interage. Isso é feito no campo Recurso na política do IAM acima. A configuração das tabelas para realizar chamadas de lote é feita no modelo de resolvedor, descrita abaixo.

Fonte de dados

Para simplificar, usaremos a mesma fonte de dados para todos os resolvedores usados neste tutorial. Na guia Fontes de dados, crie uma nova fonte de dados do DynamoDB e nomeie-a. BatchTutorial O nome da tabela pode ser qualquer coisa pois os nomes de tabelas são especificados como parte do modelo de mapeamento da solicitação para operações em lote. Chamaremos a tabela de empty.

Para esse tutorial, qualquer função com a seguinte política em linha funcionará:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "dynamodb:BatchGetItem",
        "dynamodb:BatchWriteItem"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:dynamodb:region:account:table/Posts",
        "arn:aws:dynamodb:region:account:table/Posts/*",
        "arn:aws:dynamodb:region:account:table/locationReadings",
        "arn:aws:dynamodb:region:account:table/locationReadings/*",
        "arn:aws:dynamodb:region:account:table/temperatureReadings",
        "arn:aws:dynamodb:region:account:table/temperatureReadings/*"
      ]
    }
  ]
}
```

Lote de tabela única

Warning

BatchPutItem e BatchDeleteItem não são compatíveis quando usados com detecção e resolução de conflitos. Essas configurações devem ser desativadas para evitar possíveis erros.

Para este exemplo, digamos que tenha uma única tabela chamada Publicações à qual você deseja adicionar e remover itens com operações em lote. Use o esquema a seguir, observando que, para a consulta, passaremos uma lista de 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
  }

```

Anexe um resolvedor ao campo `batchAdd()` com o seguinte Modelo de mapeamento da solicitação. Isso recebe automaticamente cada item no tipo `input PostInput` do GraphQL e cria um mapa, que é necessário para a operação `BatchPutItem`:

```

#set($postsdata = [])
#foreach($item in ${ctx.args.posts})
    $util.qr($postsdata.add($util.dynamodb.toMapValues($item)))
#end

{
  "version" : "2018-05-29",
  "operation" : "BatchPutItem",
  "tables" : {
    "Posts": $utils.toJson($postsdata)
  }
}

```

Nesse caso, o Modelo de mapeamento da resposta é uma simples passagem, mas observe que o nome da tabela está anexado como `..data.Posts` ao objeto de contexto:

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

Agora, navegue até a página Queries (Consultas) do console do AWS AppSync e execute a seguinte mutação batchAdd:

```
mutation add {  
  batchAdd(posts:[{  
    id: 1 title: "Running in the Park",{  
    id: 2 title: "Playing fetch"  
  }]){  
    id  
    title  
  }  
}
```

Veja os resultados impressos na tela e valide-os de forma independente por meio do console do DynamoDB que ambos os valores gravaram na tabela Publicações.

Em seguida, anexe um resolvedor ao campo batchGet() com o seguinte Modelo de mapeamento da solicitação. Isso recebe automaticamente cada item no tipo ids: [] do GraphQL e cria um mapa, que é necessário para a operação BatchGetItem:

```
#set($ids = [])  
#foreach($id in ${ctx.args.ids})  
  #set($map = {})  
  $util.qr($map.put("id", $util.dynamodb.toString($id)))  
  $util.qr($ids.add($map))  
#end  
  
{  
  "version" : "2018-05-29",  
  "operation" : "BatchGetItem",  
  "tables" : {  
    "Posts": {  
      "keys": $util.toJson($ids),  
      "consistentRead": true,  
      "projection" : {  
        "expression" : "#id, title",  
        "expressionNames" : { "#id" : "id"}  
      }  
    }  
  }  
}
```

```
}
```

O Modelo de mapeamento da resposta é novamente uma simples passagem, de novo com o nome da tabela anexado como `..data.Posts` ao objeto de contexto:

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

Agora volte à página Queries (Consultas) do console do AWS AppSync e execute a seguinte consulta `batchGet`:

```
query get {  
  batchGet(ids:[1,2,3]){  
    id  
    title  
  }  
}
```

Isso deve retornar os resultados para os dois valores `id` adicionados anteriormente. Observe que um valor `null` foi retornado para o `id` com um valor de 3. Isso ocorre porque não havia registro na tabela Publicações com esse valor ainda. Observe também que AWS AppSync retorna os resultados na mesma ordem das chaves passadas para a consulta, que é um recurso adicional que AWS AppSync funciona em seu nome. Portanto, se você alternar para `batchGet(ids:[1,3,2])`, verá a ordem alterada. Você também saberá qual `id` retornou um valor `null`.

Finalmente, anexe um resolvidor ao campo `batchDelete()` com o seguinte Modelo de mapeamento da solicitação. Isso recebe automaticamente cada item no tipo `ids:[]` do GraphQL e cria um mapa, que é necessário para a operação `BatchGetItem`:

```
#set($ids = [])  
#foreach($id in ${ctx.args.ids})  
  #set($map = {})  
  $util.qr($map.put("id", $util.dynamodb.toString($id)))  
  $util.qr($ids.add($map))  
#end  
  
{  
  "version" : "2018-05-29",  
  "operation" : "BatchDeleteItem",  
  "tables" : {  
    "Posts": $util.toJson($ids)  
  }  
}
```

```
}
```

O Modelo de mapeamento da resposta é novamente uma simples passagem, de novo com o nome da tabela anexado como `..data.Posts` ao objeto de contexto:

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

Agora volte à página Queries (Consultas) do console do AWS AppSync e execute a mutação `batchDelete` a seguir:

```
mutation delete {  
  batchDelete(ids:[1,2]){ id }  
}
```

Os registros com `id`, 1 e 2 devem ser excluídos agora. Se você executar novamente a consulta `batchGet()` de anteriormente, eles devem retornar `null`.

Lote de várias tabelas

Warning

`BatchPutItem` e `BatchDeleteItem` não são compatíveis quando usados com detecção e resolução de conflitos. Essas configurações devem ser desativadas para evitar possíveis erros.

AWS AppSync também permite que você execute operações em lote nas tabelas. Vamos criar um aplicativo mais complexo. Imagine que estamos criando um aplicativo de Saúde do animal de estimação, onde sensores relatam a localização do animal e a temperatura do corpo. Os sensores são alimentados por bateria e tentam se conectar à rede a cada cinco minutos. Quando um sensor estabelece uma conexão, ele envia suas leituras para nossa AWS AppSync API. Em seguida, os gatilhos analisam os dados para que um painel seja apresentado ao dono do animal. Vamos nos concentrar na representação das interações entre o sensor e o armazenamento de dados de back-end.

Como pré-requisito, primeiro vamos criar duas tabelas do DynamoDB; `locationReadings` armazenará as leituras de localização do sensor e `temperatureReadings` armazenará as leituras de temperatura do sensor. Ambas as tabelas compartilham a mesma estrutura de chave primária: `sensorId` (`String`) sendo a chave de partição e `timestamp` (`String`) a chave de classificação.

Vamos usar o seguinte esquema do GraphQL:

```
type Mutation {
  # Register a batch of readings
  recordReadings(tempReadings: [TemperatureReadingInput], locReadings:
[LocationReadingInput]): RecordResult
  # Delete a batch of readings
  deleteReadings(tempReadings: [TemperatureReadingInput], locReadings:
[LocationReadingInput]): RecordResult
}

type Query {
  # Retrieve all possible readings recorded by a sensor at a specific time
  getReadings(sensorId: ID!, timestamp: String!): [SensorReading]
}

type RecordResult {
  temperatureReadings: [TemperatureReading]
  locationReadings: [LocationReading]
}

interface SensorReading {
  sensorId: ID!
  timestamp: String!
}

# Sensor reading representing the sensor temperature (in Fahrenheit)
type TemperatureReading implements SensorReading {
  sensorId: ID!
  timestamp: String!
  value: Float
}

# Sensor reading representing the sensor location (lat,long)
type LocationReading implements SensorReading {
  sensorId: ID!
  timestamp: String!
  lat: Float
  long: Float
}

input TemperatureReadingInput {
  sensorId: ID!
  timestamp: String
```

```

    value: Float
  }

  input LocationReadingInput {
    sensorId: ID!
    timestamp: String
    lat: Float
    long: Float
  }

```

BatchPutItem - Gravação de leituras do sensor

Nossos sensores precisam ser capazes de enviar suas leituras assim que se conectarem à internet. O campo do GraphQL `Mutation.recordReadings` é a API que será usada para fazer isso. Vamos anexar um resolvidor para dar vida à nossa API.

Selecione Anexar ao lado do campo `Mutation.recordReadings`. Na próxima tela, escolha a mesma fonte de dados `BatchTutorial1` criada no início do tutorial.

Vamos adicionar o seguinte modelo de mapeamento da solicitação:

Modelo de mapeamento da solicitação

```

## Convert tempReadings arguments to DynamoDB objects
#set($tempReadings = [])
#foreach($reading in ${ctx.args.tempReadings})
    $util.qr($tempReadings.add($util.dynamodb.toMapValues($reading)))
#end

## Convert locReadings arguments to DynamoDB objects
#set($locReadings = [])
#foreach($reading in ${ctx.args.locReadings})
    $util.qr($locReadings.add($util.dynamodb.toMapValues($reading)))
#end

{
  "version" : "2018-05-29",
  "operation" : "BatchPutItem",
  "tables" : {
    "locationReadings": $utils.toJson($locReadings),
    "temperatureReadings": $utils.toJson($tempReadings)
  }
}

```

Como você pode ver, a operação `BatchPutItem` nos permite especificar várias tabelas.

Vamos usar o seguinte modelo de mapeamento da resposta.

Modelo de mapeamento da resposta

```
## 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)
```

Com operações em lote, podem haver erros e resultados retornados na invocação. Nesse caso, podemos fazer uma manipulação de erros adicional.

Observação: o uso de `$utils.appendError()` é semelhante ao `$util.error()`, com a principal distinção de que ele não interrompe a avaliação do modelo de mapeamento. Em vez disso, ele sinaliza que ocorreu um erro com o campo, mas permite que o modelo seja avaliado e, conseqüentemente, retorne dados ao chamador. Recomendamos que você use `$utils.appendError()` quando o aplicativo precisar retornar resultados parciais.

Salve o resolvedor e navegue até a página Consultas do AWS AppSync console. Vamos enviar algumas leituras do sensor!

Execute a seguinte mutação:

```
mutation sendReadings {
  recordReadings(
    tempReadings: [
      {sensorId: 1, value: 85.5, timestamp: "2018-02-01T17:21:05.000+08:00"},
      {sensorId: 1, value: 85.7, timestamp: "2018-02-01T17:21:06.000+08:00"},
      {sensorId: 1, value: 85.8, timestamp: "2018-02-01T17:21:07.000+08:00"},
      {sensorId: 1, value: 84.2, timestamp: "2018-02-01T17:21:08.000+08:00"},
      {sensorId: 1, value: 81.5, timestamp: "2018-02-01T17:21:09.000+08:00"}
    ]
    locReadings: [
      {sensorId: 1, lat: 47.615063, long: -122.333551, timestamp:
"2018-02-01T17:21:05.000+08:00"},
      {sensorId: 1, lat: 47.615163, long: -122.333552, timestamp:
"2018-02-01T17:21:06.000+08:00"}
    ]
  )
}
```



```

    {sensorId: 1, lat: 47.615263, long: -122.333553, timestamp:
"2018-02-01T17:21:07.000+08:00"}
    {sensorId: 1, lat: 47.615363, long: -122.333554, timestamp:
"2018-02-01T17:21:08.000+08:00"}
    {sensorId: 1, lat: 47.615463, long: -122.333555, timestamp:
"2018-02-01T17:21:09.000+08:00"}
  }) {
    locationReadings {
      sensorId
      timestamp
      lat
      long
    }
    temperatureReadings {
      sensorId
      timestamp
      value
    }
  }
}
```

Enviamos dez leituras do sensor em uma mutação, com as leituras divididas em duas tabelas. Use o console do DynamoDB para validar que os dados são exibidos nas tabelas `locationReadings` e `temperatureReadings`.

BatchDeleteItem - Excluindo leituras do sensor

Da mesma forma, também será necessário excluir lotes de leituras do sensor. Vamos usar o campo do GraphQL Mutation `deleteReadings` para essa finalidade. Selecione Anexar ao lado do campo Mutation `recordReadings`. Na próxima tela, escolha a mesma fonte de dados `BatchTutorial` criada no início do tutorial.

Vamos usar o seguinte modelo de mapeamento da solicitação.

Modelo de mapeamento da solicitação

```

## 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)
  }
}
```

O modelo de mapeamento da resposta é o mesmo usado para `Mutation.recordReadings`.

Modelo de mapeamento da resposta

```
## 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)
```

Salve o resolvedor e navegue até a página Consultas do AWS AppSync console. Agora, vamos excluir algumas leituras do sensor!

Execute a seguinte mutação:

```
mutation deleteReadings {
  # Let's delete the first two readings we recorded
  deleteReadings(
    tempReadings: [{sensorId: 1, timestamp: "2018-02-01T17:21:05.000+08:00"}]
    locReadings: [{sensorId: 1, timestamp: "2018-02-01T17:21:05.000+08:00"}]) {
```

```
locationReadings {
  sensorId
  timestamp
  lat
  long
}
temperatureReadings {
  sensorId
  timestamp
  value
}
}
```

Valide por meio do console do DynamoDB que essas duas leituras foram excluídas das tabelas `locationReadings` e `temperatureReadings`.

BatchGetItem - Recuperar leituras

Outra operação comum para o nosso aplicativo de Saúde do animal de estimação seria recuperar as leituras de um sensor em um determinado momento. Vamos anexar um resolvidor ao campo do GraphQL Query `getReadings` em nosso esquema. Selecione Anexar, e na próxima tela escolha a mesma fonte de dados `BatchTutorial` criada no início do tutorial.

Vamos adicionar o seguinte modelo de mapeamento da solicitação.

Modelo de mapeamento da solicitação

```
## Build a single DynamoDB primary key,
## as both locationReadings and tempReadings tables
## share the same primary key structure
#set($pkey = {})
$util.qr($pkey.put("sensorId", $ctx.args.sensorId))
$util.qr($pkey.put("timestamp", $ctx.args.timestamp))

{
  "version" : "2018-05-29",
  "operation" : "BatchGetItem",
  "tables" : {
    "locationReadings": {
      "keys": [$util.dynamodb.toMapValuesJson($pkey)],
      "consistentRead": true
    },
  },
}
```

```

    "temperatureReadings": {
      "keys": [$util.dynamodb.toMapValuesJson($pkey)],
      "consistentRead": true
    }
  }
}

```

Observe que agora estamos usando a `BatchGetItem` operação.

Nosso modelo de mapeamento da resposta será um pouco diferente, pois escolhemos retornar uma lista `SensorReading`. Vamos mapear o resultado da invocação no formato desejado.

Modelo de mapeamento da resposta

```

## 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)

```

Salve o resolvidor e navegue até a página Consultas do AWS AppSync console. Agora, vamos recuperar as leituras do sensor!

Execute a seguinte consulta:

```

query getReadingsForSensorAndTime {
  # Let's retrieve the very first two readings
  getReadings(sensorId: 1, timestamp: "2018-02-01T17:21:06.000+08:00") {
    sensorId
    timestamp
    ...on TemperatureReading {
      value
    }
  }
}

```

```
}  
  ...on LocationReading {  
    lat  
    long  
  }  
}  
}
```

Demonstramos com sucesso o uso das operações em lote do DynamoDB usando AWS AppSync

Tratamento de erros

Em AWS AppSync, as operações da fonte de dados às vezes podem retornar resultados parciais. Resultados parciais será o termo usado para indicar quando a saída de uma operação for composta por alguns dados e um erro. Como a manipulação de erros é inerentemente específica ao aplicativo, o AWS AppSync oferece a oportunidade de lidar com erros no modelo de mapeamento da resposta. O erro de invocação do resolvidor, se presente, está disponível no contexto como `$ctx.error`. Os erros de invocação sempre incluem uma mensagem e um tipo, acessível como propriedades `$ctx.error.message` e `$ctx.error.type`. Durante a invocação do modelo de mapeamento da resposta, você pode lidar com resultados parciais de três maneiras:

1. engolir o erro de invocação apenas retornando dados
2. gerar um erro (usando `$util.error(...)`) ao interromper a avaliação do modelo de mapeamento da resposta, que não retornará quaisquer dados.
3. anexar um erro (usando `$util.appendError(...)`) e também retornar dados

Vamos demonstrar cada um dos três pontos acima com operações em lote do DynamoDB!

Operações em lote do DynamoDB

Com as operações em lote do DynamoDB, é possível que um lote seja concluído parcialmente. Ou seja, é possível que alguns dos itens solicitados ou chaves não seja processados. Se não AWS AppSync for possível concluir um lote, os itens não processados e um erro de invocação serão definidos no contexto.

Vamos implementar a manipulação de erros usando a configuração de campo `Query.getReadings` da operação `BatchGetItem` da seção anterior desse tutorial. Dessa vez, vamos fingir que ao executar o campo `Query.getReadings`, a tabela do DynamoDB `temperatureReadings` esgotou sua throughput provisionada. O DynamoDB gerou

ProvisionedThroughputExceededException na segunda tentativa AWS AppSync de processar os elementos restantes no lote.

O seguinte JSON representa o contexto serializado após a invocação de lote do DynamoDB, mas antes da avaliação do modelo de mapeamento da resposta.

```
{
  "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": []
}
```

Algumas observações sobre o contexto:

- o erro de invocação foi definido no contexto em `$ctx.error` by AWS AppSync e o tipo de erro foi definido como `DynamoDB: ProvisionedThroughputExceededException`
- os resultados são mapeados por tabela em `$ctx.result.data`, mesmo que haja um erro presente
- as chaves não processadas estão disponíveis em `$ctx.result.data.unprocessedKeys`. Aqui, não AWS AppSync foi possível recuperar o item com a chave (`sensorId:1, timestamp:2018-02-01T 17:21:05.000 + 08:00`) devido à taxa de transferência insuficiente da tabela.

Observação: em `BatchPutItem`, é `$ctx.result.data.unprocessedItems`. Em `BatchDeleteItem`, é `$ctx.result.data.unprocessedKeys`.

Vamos lidar com esse erro de três maneiras diferentes.

1. Absorção do erro de invocação

Retornar dados sem manipular o erro de invocação efetivamente absorve o erro, tornando o resultado para o determinado campo do GraphQL sempre bem-sucedido.

O modelo de mapeamento da resposta gravado é familiar e se concentra apenas nos dados do resultado.

Modelo de mapeamento da resposta:

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

Resposta do GraphQL:

```
{
  "data": {
    "getReadings": [
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00",
        "lat": 47.615063,
        "long": -122.333551
      },
    ],
  },
}
```

```
{
  {
    "sensorId": "1",
    "timestamp": "2018-02-01T17:21:05.000+08:00",
    "value": 85.5
  }
}
```

Nenhum erro será adicionado à resposta do erro uma vez que apenas dados foram modificados.

2. Geração de um erro para abortar a execução do modelo

Quando falhas parciais devem ser tratadas como falhas completas do ponto de vista do cliente, você pode abortar a execução do modelo para evitar o retorno de dados. O método utilitário `$util.error(...)` atinge exatamente esse comportamento.

Modelo de mapeamento da resposta:

```
## 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)
```

Resposta do GraphQL:

```
{
  "data": {
    "getReadings": null
  },
  "errors": [
    {
      "path": [
        "getReadings"
      ],
      "data": null,
      "errorType": "DynamoDB:ProvisionedThroughputExceededException",
      "errorInfo": {
```



```

    "temperatureReadings": [
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00"
      }
    ],
    "locationReadings": [],
  },
  "locations": [
    {
      "line": 58,
      "column": 3
    }
  ],
  "message": "You exceeded your maximum allowed provisioned throughput for a table
or for one or more global secondary indexes. (...)"
}
]
}

```

Embora alguns resultados possam ter sido retornados da operação em lote do DynamoDB, escolhemos gerar um erro tal que o campo do GraphQL `getReadings` seja nulo e o erro seja adicionado ao bloco erros da resposta do GraphQL.

3. Anexar um erro para retornar dados e erros

Em alguns casos, para oferecer uma experiência melhor ao usuário, os aplicativos podem retornar resultados parciais e notificar seus clientes sobre os itens não processados. Os clientes podem decidir implementar uma nova tentativa ou traduzir o erro de volta para o usuário final. O `$util.appendError(...)` é o método utilitário que permite esse comportamento, permitindo que o designer do aplicativo anexe erros no contexto sem interferir na avaliação do modelo. Depois de avaliar o modelo, AWS AppSync processará quaisquer erros de contexto anexando-os ao bloco de erros da resposta do GraphQL.

Modelo de mapeamento da resposta:

```

#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)
```

Encaminhamos o erro de invocação e o elemento `unprocessedKeys` dentro do bloco de erros da resposta do GraphQL. O campo `getReadings` também retorna dados parciais da tabela `locationReadings` como você pode ver na resposta abaixo.

Resposta do GraphQL:

```
{
  "data": {
    "getReadings": [
      null,
      {
        "sensorId": "1",
        "timestamp": "2018-02-01T17:21:05.000+08:00",
        "value": 85.5
      }
    ]
  },
  "errors": [
    {
      "path": [
        "getReadings"
      ],
      "data": null,
      "errorType": "DynamoDB:ProvisionedThroughputExceededException",
      "errorInfo": {
        "temperatureReadings": [
          {
            "sensorId": "1",
            "timestamp": "2018-02-01T17:21:05.000+08:00"
          }
        ],
        "locationReadings": []
      },
      "locations": [
        {
          "line": 58,
          "column": 3
        }
      ],
      "message": "You exceeded your maximum allowed provisioned throughput for a table or for one or more global secondary indexes. (...)"
    }
  ]
}
```

```
}  
]  
}
```

Executando transações do DynamoDB no AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync suporta o uso de operações de transação do Amazon DynamoDB em uma ou mais tabelas em uma única região. As operações compatíveis são `TransactGetItems` e `TransactWriteItems`. Ao usar esses recursos no AWS AppSync, você pode realizar tarefas como:

- Enviar uma lista de chaves em uma única consulta e retornar os resultados de uma tabela
- Ler os registros de uma ou mais tabelas em uma única consulta
- Grave registros na transação em uma ou mais tabelas de uma all-or-nothing forma
- Executar transações quando algumas condições são atendidas

Permissões

Como outros resolvedores, você precisa criar uma fonte de dados AWS AppSync e criar uma função ou usar uma existente. Como operações de transação exigem diferentes permissões em tabelas do DynamoDB, é necessário conceder as permissões de função configuradas para ações de leitura e gravação:

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:region:accountId:table/TABLENAME",
        "arn:aws:dynamodb:region:accountId:table/TABLENAME/*"
    ]
}

```

Observação: as funções estão vinculadas às fontes de dados em AWS AppSync, e os resolvedores nos campos são invocados em relação a uma fonte de dados. As fontes de dados configuradas para buscar no DynamoDB têm apenas uma tabela especificada, para manter a configuração simples. Portanto, ao executar uma operação de transação para várias tabelas em um único resolvedor, que é uma tarefa mais avançada, é necessário conceder acesso à função na fonte de dados para qualquer tabela com a qual o resolvedor interage. Isso é feito no campo Recurso na política do IAM acima. A configuração das chamadas de transação nas tabelas é feita no modelo de resolvedor, descrita abaixo.

Fonte de dados

Para simplificar, usaremos a mesma fonte de dados para todos os resolvedores usados neste tutorial. Na guia Fontes de dados, crie uma nova fonte de dados do DynamoDB e nomeie-a. TransactTutorial O nome da tabela pode ser qualquer coisa, pois os nomes de tabelas são especificados como parte do modelo de mapeamento da solicitação para operações de transação. Chamaremos a tabela de empty.

Teremos duas tabelas chamadas savingAccounts e checkingAccounts, ambas com accountNumber como chave de partição, e uma tabela transactionHistory com transactionId como chave de partição.

Para esse tutorial, qualquer função com a seguinte política em linha funcionará. Substitua region e accountId por sua região e seu ID da conta:

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:region:accountId:table/savingAccounts",
        "arn:aws:dynamodb:region:accountId:table/savingAccounts/*",
        "arn:aws:dynamodb:region:accountId:table/checkingAccounts",
        "arn:aws:dynamodb:region:accountId:table/checkingAccounts/*",
        "arn:aws:dynamodb:region:accountId:table/transactionHistory",
        "arn:aws:dynamodb:region:accountId:table/transactionHistory/*"
      ]
    }
  ]
}
```

Transações

Neste exemplo, o contexto é uma transação bancária clássica, onde usaremos `TransactWriteItems` para:

- Transferir dinheiro de contas poupanças para contas correntes
- Gerar novos registros para cada transação

E, então, vamos usar `TransactGetItems` para recuperar detalhes de contas poupanças e contas correntes.

 Warning

`TransactWriteItems` não é compatível quando usado com detecção e resolução de conflitos. Essas configurações devem ser desativadas para evitar possíveis erros.

Nós definimos nosso esquema GraphQL da seguinte forma:

```
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 - Preencher contas

Para transferir dinheiro entre contas, precisamos preencher a tabela com os detalhes. Usaremos a operação do GraphQL `Mutation.populateAccounts` para fazer isso.

Na seção Esquema, clique em Anexar ao lado da operação `Mutation.populateAccounts`. Vá para VTL Unit Resolvers e escolha a mesma fonte de dados do `TransactTutorial`.

Agora, use o seguinte modelo de mapeamento da solicitação:

Modelo de mapeamento da solicitação

```

#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)
}

```

E o seguinte modelo de mapeamento de resposta:

Modelo de mapeamento da resposta


```
#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)
```

Salve o resolvedor e navegue até a seção Consultas do AWS AppSync console para preencher as contas.

Execute a seguinte mutação:

```
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
    }
  }
}
```

```

    }
  }
}
```

Nós preenchemos três contas poupanças e três contas correntes em uma mutação.

Use o console do DynamoDB para confirmar se os dados aparecem nas tabelas `savingAccounts` e `checkingAccounts`.

TransactWriteItems - Transferir dinheiro

Anexe um resolvidor à mutação `transferMoney` com o seguinte Modelo de mapeamento da solicitação. Observe se os valores de `amounts`, `savingAccountNumbers` e `checkingAccountNumbers` são os mesmos.

```

#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)
}

```

```
}
```

Teremos três transações bancárias em uma única operação `TransactWriteItems`. Use o seguinte Modelo de mapeamento de resposta:

```
#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)
```

Agora navegue até a seção Consultas do console AWS AppSync e execute a mutação `transferMoney` da seguinte forma:

```
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  
    }  
  }  
}
```

Nós enviamos duas transações bancárias em uma mutação. Use o console do DynamoDB para validar se os dados aparecem nas tabelas `savingAccounts`, `checkingAccounts` e `transactionHistory`.

TransactGetItems - Recuperar contas

Para recuperar os detalhes das contas poupanças e contas-correntes em uma única solicitação transacional, anexaremos um resolvidor à operação do GraphQL Query `getAccounts` no nosso esquema. Selecione Anexar, vá para VTL Unit Resolvers e, na próxima tela, escolha a mesma fonte de dados do `TransactTutorial` criada no início do tutorial. Configure os modelos da seguinte maneira:

Modelo de mapeamento da solicitação

```
#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)
}

```

Modelo de mapeamento da resposta

```

#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)

```

Salve o resolvidor e navegue até as seções Consultas do AWS AppSync console. Para recuperar as contas poupanças e contas correntes, execute a seguinte consulta:

```

query getAccounts {
  getAccounts(
    savingAccountNumbers: ["1", "2", "3"],
    checkingAccountNumbers: ["1", "2"]
  )
}

```

```
) {  
  savingAccounts {  
    accountNumber  
    username  
    balance  
  }  
  checkingAccounts {  
    accountNumber  
    username  
    balance  
  }  
}
```

Demonstramos com sucesso o uso das transações do DynamoDB usando AWS AppSync

Usando resolvedores HTTP em AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync permite que você use fontes de dados compatíveis (ou seja, Amazon DynamoDB, AWS Lambda, Amazon Service ou OpenSearch Amazon Aurora) para realizar várias operações, além de quaisquer endpoints HTTP arbitrários para resolver campos do GraphQL. Depois que os endpoints HTTP estiverem disponíveis, conecte-se a eles usando uma fonte de dados. Em seguida, configure um resolvidor no esquema para executar operações do GraphQL, como consultas, mutações e assinaturas. Esse tutorial apresentará alguns exemplos comuns.

Neste tutorial, você usa uma API REST (criada usando o Amazon API Gateway e o Lambda) com um endpoint GraphQL AWS AppSync.

Configuração com um clique

Se você quiser configurar automaticamente um endpoint do GraphQL AWS AppSync com um endpoint HTTP configurado (usando o Amazon API Gateway e o Lambda), você pode usar o seguinte modelo: AWS CloudFormation

[Launch Stack](#)

Criar uma API REST

Você pode usar o AWS CloudFormation modelo a seguir para configurar um endpoint REST que funcione para este tutorial:

[Launch Stack](#)

A AWS CloudFormation pilha executa as seguintes etapas:

1. Configura uma função do Lambda, que contém a lógica de negócios para o seu microsserviço.
2. Configura uma API REST do API Gateway com a seguinte combinação endpoint/method/content de tipos:

Caminho do recurso da API	Método HTTP	Tipo de conteúdo compatível
/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

Criação da API GraphQL

Para criar a API GraphQL em: AWS AppSync

- Abra o AWS AppSync console e escolha Criar API.
- Para o nome da API, digite UserData.
- Selecione Esquema personalizado.
- Escolha Criar.

O AWS AppSync console cria uma nova API GraphQL para você usando o modo de autenticação por chave de API. É possível usar o console para configurar o restante da API GraphQL e executar consultas nela durante o restante desse tutorial.

Criar um esquema do GraphQL

Agora que você tem uma API GraphQL, vamos criar um esquema do GraphQL. No editor de esquemas no AWS AppSync console, verifique se o esquema corresponde ao esquema a seguir:

```
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
}
```

Configurar a fonte de dados HTTP

Para configurar a fonte de dados HTTP, faça o seguinte:

- Na DataSource guia, escolha Novo e digite um nome amigável para a fonte de dados (por exemplo, HTTP).
- Em Tipo de fonte de dados, escolha HTTP.
- Defina o endpoint como o endpoint da API Gateway criado. Lembre-se de não incluir o nome do estágio como parte do endpoint.

Observação: no momento, somente endpoints públicos são compatíveis com o AWS AppSync.

Nota: Para obter mais informações sobre as autoridades de certificação que são reconhecidas pelo AWS AppSync serviço, consulte [Autoridades de certificação \(CA\) reconhecidas por AWS AppSync para endpoints HTTPS](#).

Configurar resolvedores

Nesta etapa, conecte a fonte de dados http à consulta getUser.

Como configurar o resolvedor :

- Escolha a guia Esquema.
- No painel Tipos de dados à direita no tipo Consulta, localize o campo getUser e escolha Associar.
- Em Nome da fonte de dados, escolha HTTP.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte código:

```
{
  "version": "2018-05-29",
  "method": "GET",
  "params": {
    "headers": {
      "Content-Type": "application/json"
    }
  },
  "resourcePath": $util.toJson("/v1/users/${ctx.args.id}")
}
```

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte código:

```
## return the body
#if($ctx.result.statusCode == 200)
    ##if response is 200
    $ctx.result.body
#else
    ##if response is not 200, append the response to error block.
    $utils.appendError($ctx.result.body, "$ctx.result.statusCode")
#end
```

- Escolha a guia Consulta e, depois, execute a seguinte consulta:

```
query GetUser{
  getUser(id:1){
    id
    username
  }
}
```

Isso deve retornar a seguinte resposta:

```
{
  "data": {
    "getUser": {
      "id": "1",
      "username": "nadia"
    }
  }
}
```

- Escolha a guia Esquema.
- No painel Tipos de dados à direita em Mutação, localize o campo addUser e escolha Anexar.
- Em Nome da fonte de dados, escolha HTTP.
- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte código:

```
{
```

```

"version": "2018-05-29",
"method": "POST",
"resourcePath": "/v1/users",
"params":{
  "headers":{
    "Content-Type": "application/json",
  },
  "body": $util.toJson($ctx.args.userInput)
}
}

```

- Em Configurar o modelo de mapeamento da solicitação, cole o seguinte código:

```

## Raise a GraphQL field error in case of a datasource invocation error
#if($ctx.error)
  $util.error($ctx.error.message, $ctx.error.type)
#end
## if the response status code is not 200, then return an error. Else return the body
**
#if($ctx.result.statusCode == 200)
  ## If response is 200, return the body.
  $ctx.result.body
#else
  ## If response is not 200, append the response to error block.
  $utils.appendError($ctx.result.body, "$ctx.result.statusCode")
#end

```

- Escolha a guia Consulta e, depois, execute a seguinte consulta:

```

mutation addUser{
  addUser(userInput:{
    id:"2",
    username:"shaggy"
  }){
    id
    username
  }
}

```

Isso deve retornar a seguinte resposta:

```
{
  "data": {
    "getUser": {
      "id": "2",
      "username": "shaggy"
    }
  }
}
```

Invocando serviços AWS

Você pode usar resolvedores HTTP para configurar uma interface AWS de API GraphQL para serviços. As solicitações HTTP AWS devem ser assinadas com o [processo Signature Version 4](#) para que seja AWS possível identificar quem as enviou. AWS AppSync calcula a assinatura em seu nome quando você associa uma função do IAM à fonte de dados HTTP.

Você fornece dois componentes adicionais para invocar AWS serviços com resolvedores HTTP:

- Uma função do IAM com permissões para chamar o AWS serviço APIs
- Configurar a assinatura na fonte de dados

Por exemplo, se você quiser chamar a [ListGraphqlApis operação](#) com resolvedores HTTP, primeiro [crie uma função do IAM](#) que AWS AppSync assuma com a seguinte política anexada:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "appsync:ListGraphqlApis"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Em seguida, crie a fonte de dados HTTP para AWS AppSync. Neste exemplo, você chamará AWS AppSync na região Oeste dos EUA (Oregon). Configure a seguinte configuração HTTP em um arquivo chamado `http.json`, que inclui a região de assinatura e o nome do serviço:

```
{
  "endpoint": "https://appsync.us-west-2.amazonaws.com/",
  "authorizationConfig": {
    "authorizationType": "AWS_IAM",
    "awsIamConfig": {
      "signingRegion": "us-west-2",
      "signingServiceName": "appsync"
    }
  }
}
```

Em seguida, use o AWS CLI para criar a fonte de dados com uma função associada da seguinte forma:

```
aws appsync create-data-source --api-id <API-ID> \
                               --name AWSAppSync \
                               --type HTTP \
                               --http-config file:///http.json \
                               --service-role-arn <ROLE-ARN>
```

Ao anexar um resolvedor ao campo no esquema, use o seguinte modelo de mapeamento de solicitações para chamar AWS AppSync:

```
{
  "version": "2018-05-29",
  "method": "GET",
  "resourcePath": "/v1/apis"
}
```

Quando você executa uma consulta GraphQL para essa fonte de dados, AWS AppSync assina a solicitação usando a função fornecida e inclui a assinatura na solicitação. A consulta retorna uma lista do AWS AppSync GraphQL APIs na sua conta nessa AWS região.

Usando o Aurora Serverless v2 com AWS AppSync

Conecte sua API GraphQL aos bancos de dados Aurora Serverless usando o AWS AppSync. Essa integração permite que você execute instruções SQL por meio de consultas, mutações e assinaturas do GraphQL, oferecendo uma maneira flexível de interagir com seus dados relacionais.

Note

Este tutorial usa a Região US-EAST-1.

Benefícios

- Integração perfeita entre GraphQL e bancos de dados relacionais
- Capacidade de realizar operações SQL por meio de interfaces GraphQL
- Escalabilidade sem servidor com o Aurora Serverless v2
- Acesso seguro aos dados por meio do AWS Secrets Manager
- Proteção contra injeção de SQL por meio de higienização de entrada
- Recursos flexíveis de consulta, incluindo operações de filtragem e alcance

Casos de uso comuns

- Criação de aplicativos escaláveis com requisitos de dados relacionais
- Para criar APIs, isso, é necessário tanto a flexibilidade do GraphQL quanto os recursos do banco de dados SQL
- Gerenciando operações de dados por meio de mutações e consultas do GraphQL
- Implementação de padrões seguros de acesso ao banco

Neste tutorial, você aprenderá o seguinte.

- Configurar um cluster Aurora Serverless v2
- Ativar a funcionalidade da API de dados
- Crie e configure estruturas de banco de dados
- Definir esquemas do GraphQL para operações de banco de dados

- Implemente resolvedores para consultas e mutações
- Proteja seu acesso aos dados por meio da higienização adequada da entrada
- Execute várias operações de banco de dados por meio de interfaces GraphQL

Tópicos

- [Configurando seu cluster de banco de dados](#)
- [Ativar API de dados](#)
- [Criar banco de dados e tabela](#)
- [Esquema do GraphQL](#)
- [Conecte sua API às operações do banco de dados](#)
- [Modifique seus dados por meio da API](#)
- [Recupere seus dados](#)
- [Proteja seu acesso aos dados](#)

Configurando seu cluster de banco de dados

Antes de adicionar uma fonte de dados do Amazon RDS AWS AppSync, você deve primeiro habilitar uma API de dados em um cluster Aurora Serverless v2 e configurar um segredo usando AWS Secrets Manager. Você pode criar um cluster Aurora Serverless v2 usando: 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
```

Isso retornará um ARN para o cluster.

Depois de criar o cluster, você deve adicionar uma instância do Aurora Serverless v2 usando o comando a seguir.

```
aws rds create-db-instance \  
  --db-instance-identifier appsync-tutorial-instance
```



```
--db-cluster-identifier appsync-tutorial \  
--db-instance-identifier appsync-tutorial-instance-1 \  
--db-instance-class db.serverless \  
--engine aurora-mysql
```

Note

Esses endpoints demoram para serem ativados. Você pode verificar seu status no console do Amazon RDS na guia Conectividade e segurança do cluster. Você também pode verificar o status do seu cluster com o AWS CLI comando a seguir.

```
aws rds describe-db-clusters \  
  --db-cluster-identifier appsync-tutorial \  
  --query "DBClusters[0].Status"
```

Você pode criar um segredo usando o AWS Secrets Manager console ou AWS CLI com um arquivo de entrada, como o seguinte, usando o USERNAME e COMPLEX_PASSWORD da etapa anterior.

```
{  
  "username": "USERNAME",  
  "password": "COMPLEX_PASSWORD"  
}
```

Passe isso como um parâmetro para AWS CLI:

```
aws secretsmanager create-secret --name HttpRDSSecret --secret-string file://creds.json  
--region us-east-1
```

Isso retornará um ARN para o segredo.

Anote o ARN do seu cluster Aurora Serverless e Secret para uso posterior no AppSync console ao criar uma fonte de dados.

Ativar API de dados

Você pode ativar a API de dados em seu cluster [seguindo as instruções na documentação do RDS](#). A API de dados deve ser ativada antes de ser adicionada como fonte de AppSync dados.

Criar banco de dados e tabela

Depois de ativar sua API de dados, você pode garantir que ela funcione com o comando `aws rds-data execute-statement` na AWS CLI. Isso garantirá que seu cluster Aurora Serverless seja configurado corretamente antes de adicioná-lo à sua API. AppSync Primeiro crie um banco de dados chamado TESTDB com o parâmetro `--sql` da seguinte forma:

```
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"
```

Se isso for executado sem erro, inclua uma tabela com o comando `create table`:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789000:cluster:http-endpoint-test" \
--schema "mysql" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789000:secret:testHttp2-AmNvc1" \
--region us-east-1 \
--sql "create table Pets(id varchar(200), type varchar(200), price float)" --database "TESTDB"
```

Se tudo tiver sido executado sem problemas, você poderá adicionar o cluster como fonte de dados na sua AppSync API.

Esquema do GraphQL

Agora que sua API de dados do Aurora Serverless está funcionando com uma tabela, criaremos um esquema do GraphQL e anexaremos resolvedores para executar mutações e assinaturas. Crie uma nova API no AWS AppSync console, navegue até a página Esquema e insira o seguinte:

```
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
  }
```

Salve seu esquema e navegue até a página Fontes de dados e crie uma nova fonte de dados. Selecione Banco de dados relacional para o tipo de fonte de dados e forneça um nome amigável. Use o nome do banco de dados que você criou na última etapa, bem como o ARN do cluster em que você o criou. Para a função, você pode AppSync criar uma nova função ou criar uma com uma política semelhante à abaixo:

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:*"
      ]
    }
  ]
}
```

Observe que há duas instruções nesta política que você está concedendo acesso à função. O primeiro recurso é seu cluster Aurora Serverless e o segundo é seu ARN. AWS Secrets Manager. Você precisará fornecer AMBOS ARNs na configuração da fonte de AppSync dados antes de clicar em Criar.

Passa isso como um parâmetro para AWS CLI o.

```
aws secretsmanager create-secret \
```

```
--name HttpRDSSecret \  
--secret-string file://creds.json \  
--region us-east-1
```

Isso retornará um ARN para o segredo. Anote o ARN do seu cluster e segredo sem servidor do Aurora para mais tarde ao criar uma fonte de dados no console. AWS AppSync

Crie sua estrutura de banco de dados

Depois de ativar sua API de dados, você pode garantir que ela funcione com o comando `aws rds-data execute-statement` na AWS CLI. Isso garantirá que seu cluster Aurora Serverless v2 seja configurado corretamente antes de adicioná-lo à sua API. AWS AppSync Primeiro, crie um banco de dados chamado TESTDB com o `--sql` parâmetro a seguir.

```
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"
```

Se isso for executado sem erros, adicione uma tabela com o seguinte comando `create table`.

```
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"
```

Projete sua interface de API

Depois que a API de dados Aurora Serverless v2 estiver em funcionamento com uma tabela, crie um esquema GraphQL e anexe resolvedores para realizar mutações e assinaturas. Crie uma nova API no AWS AppSync console, navegue até a página Esquema no console e insira o seguinte.

```
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
  }
```

Salve seu esquema e navegue até a página Fontes de dados e crie uma nova fonte de dados. Escolha Banco de dados relacional para o tipo de fonte de dados e forneça um nome amigável. Use o nome do banco de dados que você criou na última etapa, bem como o ARN do cluster em que você o criou. Para a função, você pode AWS AppSync criar uma nova função ou criar uma com uma política semelhante à seguinte.

```
{
  "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:mysqlcluster",
        "arn:aws:rds:us-east-1:111122223333:cluster:mysqlcluster:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:111122223333:secret:mysecret",
        "arn:aws:secretsmanager:us-east-1:111122223333:secret:mysecret:*"
      ]
    }
  ]
}
```

Observe que há duas instruções nesta política que você está concedendo acesso à função. O primeiro recurso é seu cluster Aurora Serverless v2 e o segundo é seu ARN. AWS Secrets Manager Você precisará fornecer AMBOS ARNs na configuração da fonte de AWS AppSync dados antes de clicar em Criar.

Conecte sua API às operações do banco de dados

Agora que temos um esquema GraphQL válido e uma fonte de dados RDS, você pode anexar resolvedores aos campos do GraphQL ao seu esquema. Nossa API oferecerá os seguintes recursos:

1. crie um animal de estimação usando o campo `mutation.createPet`
2. atualize um animal de estimação usando o campo `mutation.updatePet`
3. exclua um animal de estimação usando o campo `mutation.deletePet`
4. obtenha um único uso por meio do campo `Query.getPet`
5. listar tudo usando o campo `query.listPets`
6. liste animais de estimação em uma faixa de preço usando o `Query.listPetsByPriceRange` campo

Mutation.createPet

No editor de esquemas no AWS AppSync console, no lado direito, escolha `Attach Resolver` for `createPet(input: CreatePetInput!): Pet`. Selecione a fonte de dados do RDS. Na seção modelo de mapeamento de solicitação, adicione o seguinte modelo:

```
#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)
  }
}
```

O sistema executa instruções SQL sequencialmente, com base na ordem na matriz de instruções. Os resultados retornarão na mesma ordem. Como isso é uma mutação, você executará uma instrução `select` após a inserção para recuperar os valores confirmados a fim de preencher o modelo de mapeamento de resposta do GraphQL.

Na seção modelo de mapeamento de resposta, adicione o seguinte modelo:


```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[1][0])
```

Como as instruções têm duas consultas SQL, precisamos especificar o segundo resultado na matriz que retorna do banco de dados com: `$utils.rds.toJsonString($ctx.result)[1][0]`).

Mutation.updatePet

No editor de esquemas no AWS AppSync console, escolha Anexar resolvidor para `updatePet(input: UpdatePetInput!): Pet`. Escolha sua fonte de dados do RDS. Na seção do modelo de mapeamento da solicitação, adicione o modelo a seguir.

```
{
  "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)
  }
}
```

Na seção do modelo de mapeamento de resposta, adicione o modelo a seguir.

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[1][0])
```

Mutation.deletePet

No editor de esquemas no AWS AppSync console, escolha Anexar resolvidor para `deletePet(input: DeletePetInput!): Pet`. Escolha sua fonte de dados do RDS. Na seção do modelo de mapeamento da solicitação, adicione o modelo a seguir.

```
{
  "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"
}
```

Na seção do modelo de mapeamento de resposta, adicione o modelo a seguir.

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0][0])
```

Query.getPet

Agora que as mutações foram criadas para seu esquema, conecte as três consultas para mostrar como obter itens individuais, listas e aplicar a filtragem SQL. No editor de esquemas no AWS AppSync console, escolha Anexar resolvidor para `getPet(id: ID!): Pet`. Escolha sua fonte de dados do RDS. Na seção do modelo de mapeamento da solicitação, adicione o modelo a seguir.

```
{
  "version": "2018-05-29",
  "statements": [
    $util.toJson("select * from Pets WHERE id=:ID")
  ],
  "variableMap": {
    ":ID": "$ctx.args.id"
  }
}
```

Na seção modelo de mapeamento de resposta, adicione o seguinte modelo:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0][0])
```

Query.listPets

No editor de esquemas no AWS AppSync console, no lado direito, escolha Attach Resolver para `listPets(id: ID!): [Pet]`. Escolha sua fonte de dados do RDS. Na seção do modelo de mapeamento da solicitação, adicione o modelo a seguir.

```
{
  "version": "2018-05-29",
  "statements": [
```

```
    "select * from Pets"
  ]
}
```

Na seção do modelo de mapeamento de resposta, adicione o modelo a seguir.

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0])
```

Consulta. listPetsByPriceRange

No editor de esquemas no AWS AppSync console, no lado direito, escolha Attach Resolver forgetPet(id: ID!): Pet. Escolha sua fonte de dados do RDS. Na seção do modelo de mapeamento da solicitação, adicione o modelo a seguir.

```
{
  "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)
  }
}
```

Na seção modelo de mapeamento de resposta, adicione o seguinte modelo:

```
$utils.toJson($utils.rds.toJsonObject($ctx.result)[0])
```

Modifique seus dados por meio da API

Agora que você configurou todos os seus resolvedores com instruções SQL e conectou a API do GraphQL à API de dados do Aurora Serverless, é possível começar a realizar mutações e consultas. No AWS AppSync console, escolha a guia Consultas e digite o seguinte para criar um animal de estimação:

```
mutation add {
  createPet(input : { type:fish, price:10.0 }){
```

```
    id
    type
    price
  }
}
```

A resposta deve conter o id, o tipo e o preço da seguinte forma:

```
{
  "data": {
    "createPet": {
      "id": "c6fedbbe-57ad-4da3-860a-ffe8d039882a",
      "type": "fish",
      "price": "10.0"
    }
  }
}
```

Você pode modificar este item executando a mutação `updatePet`:

```
mutation update {
  updatePet(input : {
    id: ID_PLACEHOLDER,
    type:bird,
    price:50.0
  }){
    id
    type
    price
  }
}
```

Observe que usamos o id que foi retornado da operação `createPet` anteriormente. Esse será um valor único para o seu registro, já que o resolvidor utilizou `$util.autoId()`. Você pode excluir um registro de maneira semelhante:

```
mutation delete {
  deletePet(input : {id:ID_PLACEHOLDER}){
    id
    type
    price
  }
}
```

```
}
```

Crie alguns registros com a primeira mutação com valores diferentes para preço e execute algumas consultas.

Recupere seus dados

Ainda na guia Consultas do console, use a instrução a seguir para listar todos os registros que você criou.

```
query allpets {  
  listPets {  
    id  
    type  
    price  
  }  
}
```

Aproveite o predicado SQL WHERE que estava `where price > :MIN and price < :MAX` em nosso modelo de mapeamento para o Query. `listPetsByPriceRange` com a seguinte consulta do GraphQL:

```
query petsByPriceRange {  
  listPetsByPriceRange(min:1, max:11) {  
    id  
    type  
    price  
  }  
}
```

Você só deve ver registros com um preço acima de US\$ 1 ou abaixo de US\$ 10. Por fim, é possível realizar consultas para recuperar registros individuais da seguinte maneira:

```
query onePet {  
  getPet(id:ID_PLACEHOLDER){  
    id  
    type  
    price  
  }  
}
```

Proteja seu acesso aos dados

A injeção de SQL é uma vulnerabilidade de segurança em aplicativos de banco de dados. Isso ocorre quando os invasores inserem um código SQL malicioso por meio dos campos de entrada do usuário. Isso pode permitir acesso não autorizado aos dados do banco de dados. Recomendamos que você valide e limpe cuidadosamente todas as entradas do usuário antes do processamento, usando `variableMap` para proteção contra ataques de injeção de SQL. Se os mapas de variáveis não forem usados, você será responsável por limpar os argumentos de suas operações do GraphQL. Uma maneira de fazer isso é fornecer etapas de validação específicas de entrada no modelo de mapeamento de solicitação antes da execução de uma instrução SQL em relação à API de dados. Vamos ver como podemos modificar o modelo de mapeamento de solicitação do exemplo `listPetsByPriceRange`. Em vez de contar apenas com a entrada do usuário, você pode fazer o seguinte:

```
#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)
    }
}
```

Outra maneira de proteger contra entradas invasivas ao executar resolvedores em relação à API de dados é usar instruções preparadas junto com o procedimento armazenado e entradas parametrizadas. Por exemplo, no resolvedor para `listPets`, defina o seguinte procedimento que executa `select` como uma instrução preparada:

```
CREATE PROCEDURE listPets (IN type_param VARCHAR(200))
```

```
BEGIN
  PREPARE stmt FROM 'SELECT * FROM Pets where type=?';
  SET @type = type_param;
  EXECUTE stmt USING @type;
  DEALLOCATE PREPARE stmt;
END
```

Crie isso em sua instância Aurora Serverless v2.

```
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"
```

O código de resolvedor resultante para listPets foi simplificado, pois agora simplesmente chamamos o procedimento armazenado. No mínimo, qualquer entrada de string deve ter aspas simples [com caracteres de escape](#).

```
#set ($validType = $util.isString($ctx.args.type) && !
$util.isNullOrBlank($ctx.args.type))
#if (!$validType)
  $util.error("Input for 'type' is not valid.", "ValidationError")
#end

{
  "version": "2018-05-29",
  "statements": [
    "CALL listPets(:type)"
  ]
  "variableMap": {
    ":type": $util.toJson($ctx.args.type.replace("'", "'"))
  }
}
```

Usando sequências de escape

Use aspas simples para marcar o início e o fim dos literais de string em uma instrução SQL, por exemplo. 'some string value'. Para permitir que valores de string com um ou mais caracteres

de aspas simples (') sejam usados em uma string, cada um deve ser substituído por duas aspas simples (' '). Por exemplo, se a string de entrada for `Nadia 's dog`, você deverá inserir caracteres de escape nela para a instrução SQL, como

```
update Pets set type='Nadia''s dog' WHERE id='1'
```

Usando resolvedores de pipeline em AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync fornece uma maneira simples de conectar um campo GraphQL a uma única fonte de dados por meio de resolvedores de unidades. No entanto, a execução de uma única operação pode não ser suficiente. Os resolvedores de pipeline oferecem a capacidade de executar operações em série mediante fontes de dados. Crie funções na sua API e anexe-as a um resolvedor de pipeline. Cada resultado de execução da função é direcionado para a próxima até que nenhuma função fique sem execução. Com os resolvedores de pipeline, agora você pode criar fluxos de trabalho mais complexos diretamente no AWS AppSync. Neste tutorial, você cria um aplicativo de visualização de fotos simples, no qual os usuários podem publicar e visualizar fotos publicadas por seus amigos.

Configuração com um clique

Se você quiser configurar automaticamente o endpoint do GraphQL AWS AppSync com todos os resolvedores configurados e os AWS recursos necessários, você pode usar o seguinte modelo: AWS CloudFormation

**Launch Stack**

Esta pilha cria os seguintes recursos na sua conta:

- Função do IAM AWS AppSync para acessar os recursos em sua conta
- 2 Tabelas do DynamoDB

- 1 grupo de usuários do Amazon Cognito
- 2 grupos de usuários do Amazon Cognito
- 3 usuários do grupo de usuários do Amazon Cognito
- 1 AWS AppSync API

No final do processo de criação da AWS CloudFormation pilha, você recebe um e-mail para cada um dos três usuários do Amazon Cognito que foram criados. Cada e-mail contém uma senha temporária que você usa para fazer login como usuário do Amazon Cognito no AWS AppSync console. Salve as senhas para o restante do tutorial.

Configuração manual

Se você preferir passar manualmente por um step-by-step processo pelo AWS AppSync console, siga o processo de configuração abaixo.

Configurando seus AWS AppSync recursos não relacionados

A API se comunica com duas tabelas do DynamoDB: uma tabela pictures que armazena fotos e uma tabela friends que armazena os relacionamentos entre os usuários. A API é configurada para usar o grupo de usuários do Amazon Cognito como tipo de autenticação. A AWS CloudFormation pilha a seguir configura esses recursos na conta.



No final do processo de criação da AWS CloudFormation pilha, você recebe um e-mail para cada um dos três usuários do Amazon Cognito que foram criados. Cada e-mail contém uma senha temporária que você usa para fazer login como um usuário do Amazon Cognito no console do AWS AppSync. Salve as senhas para o restante do tutorial.

Criação da API GraphQL

Para criar a API GraphQL em: AWS AppSync

1. Abra o AWS AppSync console, escolha Build From Scratch e escolha Start.
2. Defina o nome da API como AppSyncTutorial-PicturesViewer.
3. Escolha Criar.

O AWS AppSync console cria uma nova API GraphQL para você usando o modo de autenticação por chave de API. Você pode usar o console para configurar o restante da API GraphQL e executar consultas nela durante o restante desse tutorial.

Configurar a API do GraphQL

Você precisa configurar a AWS AppSync API com o grupo de usuários do Amazon Cognito que você acabou de criar.

1. Escolha a guia Configurações.
2. Na seção Authorization Type, escolha Grupo de usuários do Amazon Cognito.
3. Em Configuração do grupo de usuários, escolha US-WEST-2 para a região da AWS .
4. Escolha o AppSyncTutorial- grupo de UserPool usuários.
5. Escolha DENY como Ação padrão.
6. Deixe o campo regex AppId do cliente em branco.
7. Escolha Salvar.

Agora, a API está configurada para usar o grupo de usuários do Amazon Cognito como seu tipo de autorização.

Configuração da fonte de dados para as tabelas do DynamoDB

Depois que as tabelas do DynamoDB forem criadas, navegue até a API do AWS AppSync GraphQL no console e escolha a guia Fontes de dados. Agora, você criará uma fonte de dados AWS AppSync para cada uma das tabelas do DynamoDB que você acabou de criar.

1. Escolha a guia Fonte de dados.
2. Selecione Novo para criar uma nova fonte de dados.
3. Para o nome da fonte de dados, insira PicturesDynamoDBTable.
4. Para o tipo de fonte de dados, escolha Tabela do Amazon DynamoDB.
5. Para a região, escolha US-WEST-2.
6. Na lista de tabelas, escolha a tabela AppSyncTutorial-Pictures DynamoDB.
7. Na seção Criar ou usar um perfil existente, escolha Perfil existente.
8. Escolha a função que acabou de ser criada a partir do CloudFormation modelo. Se você não alterou o ResourceNamePrefix, o nome da função deverá ser AppSyncTutorialDBRole-Dynamo.
9. Escolha Criar.

Repita o mesmo processo para a tabela de amigos. O nome da tabela do DynamoDB deve ser `AppSyncTutorialser-Friends` se você não alterou `ResourceNamePrefix` parâmetro no momento da criação da pilha. CloudFormation

Criação do esquema do GraphQL

Agora que as fontes de dados estão conectadas às suas tabelas do DynamoDB, vamos criar um esquema do GraphQL. No editor de esquemas no AWS AppSync console, verifique se o esquema corresponde ao esquema a seguir:

```
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!
}
```

Escolha **Salvar esquema** para salvar o esquema.

Alguns dos campos do esquema foram anotados com a diretiva `@aws_auth`. Como a configuração de ação padrão da API é definida como `DENY`, a API rejeita todos os usuários que não são membros

dos grupos mencionados na diretiva `@aws_auth`. Para obter mais informações sobre como proteger sua API, você pode ler a página [Segurança](#). Nesse caso, somente usuários administradores têm acesso aos campos `mutation.createPicture` e `mutation.createFriendship`, enquanto usuários que são membros dos grupos Administradores ou Visualizadores podem acessar a Consulta `getPicturesByCampo` do proprietário. Todos os outros usuários não têm acesso.

Configurar resolvedores

Agora que você tem um esquema do GraphQL válido e duas fontes de dados, é possível anexar resolvedores aos campos do GraphQL no esquema. A API oferece os seguintes recursos:

- Crie uma foto por meio do campo `Mutation.createPicture`
- Estabeleça a amizade por meio do campo `Mutation.createFriendship`
- Recupere uma foto por meio do campo `Query.getPicture`

`Mutation.createPicture`

No editor de esquemas no AWS AppSync console, no lado direito, escolha `Attach Resolver` `forcreatePicture(input: CreatePictureInput!): Picture!`. Escolha a fonte de dados do `PicturesDynamoDBTableDynamoDB`. Na seção modelo de mapeamento de solicitação, adicione o seguinte modelo:

```
#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)
}
```

Na seção modelo de mapeamento de resposta, adicione o seguinte modelo:

```
#if($ctx.error)
```

```
$util.error($ctx.error.message, $ctx.error.type)
#end
$util.toJson($ctx.result)
```

A funcionalidade de criação de fotos está concluída. Você está salvando uma foto na tabela Pictures, usando um UUID gerado aleatoriamente como id da foto e usando o nome de usuário do Cognito como proprietário da foto.

Mutation.createFriendship

No editor de esquemas no AWS AppSync console, no lado direito, escolha Attach Resolver for createFriendship(id: ID!, target: ID!): Boolean. Escolha a fonte de dados do FriendsDynamoDBTableDynamoDB. Na seção modelo de mapeamento de solicitação, adicione o seguinte modelo:

```
#set($userToFriendFriendship = { "userId" : "$ctx.args.id", "friendId":
  "$ctx.args.target" })
#set($friendToUserFriendship = { "userId" : "$ctx.args.target", "friendId":
  "$ctx.args.id" })
#set($friendsItems = [$util.dynamodb.toMapValues($userToFriendFriendship),
  $util.dynamodb.toMapValues($friendToUserFriendship)])

{
  "version" : "2018-05-29",
  "operation" : "BatchPutItem",
  "tables" : {
    ## Replace 'AppSyncTutorial-' default below with the ResourceNamePrefix you
    provided in the CloudFormation template
    "AppSyncTutorial-Friends": $util.toJson($friendsItems)
  }
}
```

Importante: no modelo de BatchPutItem solicitação, o nome exato da tabela do DynamoDB deve estar presente. O nome padrão da tabela é AppSyncTutorial-Friends. Se você estiver usando o nome errado da tabela, você receberá um erro ao AppSync tentar assumir a função fornecida.

Para simplificar este tutorial, proceda como se a solicitação de amizade tivesse sido aprovada e salve a entrada do relacionamento diretamente na AppSyncTutorialFriendstabela.

Efetivamente, você está armazenando dois itens para cada amizade, pois o relacionamento é bidirecional. Para obter mais detalhes sobre as melhores práticas do Amazon DynamoDB para many-to-many representar relacionamentos, consulte [DynamoDB Best Practices](#).

Na seção modelo de mapeamento de resposta, adicione o seguinte modelo:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end
true
```

Nota: certifique-se de que o seu modelo de solicitação contenha o nome da tabela correta. O nome padrão é AppSyncTutorial-Friends, mas o nome da tabela pode ser diferente se você alterar o CloudFormation ResourceNamePrefixparâmetro.

Consulta. getPicturesByProprietário

Agora que você tem amizades e fotos, precisa fornecer aos usuários a capacidade de ver as fotos de seus amigos. Para atender a esse requisito, você precisa primeiro verificar se o solicitante é amigo do proprietário e, por fim, consultar as fotos.

Como essa funcionalidade requer duas operações de fonte de dados, você criará duas funções. A primeira função, isFriend, verifica se o solicitante e o proprietário são amigos. A segunda função, getPicturesByProprietário, recupera as fotos solicitadas com o ID do proprietário. Vamos dar uma olhada no fluxo de execução abaixo para o resolvidor proposto na Consulta. getPicturesByCampo do proprietário:

1. Antes do modelo de mapeamento: prepare os argumentos de entrada de contexto e campo.
2. Função isFriend: verifica se o solicitante é o proprietário da foto. Caso contrário, ele verifica se os usuários solicitantes e proprietários são amigos fazendo uma operação do GetItem DynamoDB na tabela de amigos.
3. getPicturesByFunção de proprietário: recupera imagens da tabela Imagens usando uma operação de consulta do DynamoDB no Índice Secundário Global do índice do proprietário.
4. Após o modelo de mapeamento: mapeie o resultado da foto para que os atributos do DynamoDB mapeiem corretamente para os campos esperados do tipo GraphQL.

Primeiro, vamos criar as funções.

Função isFriend

1. Escolha a guia Funções.
2. Escolha Criar função para criar uma função.

3. Para o nome da fonte de dados, insira `FriendsDynamoDBTable`.
4. Para o nome da função, digite `isFriend`.
5. Dentro da área de texto do modelo de mapeamento de solicitação, cole o seguinte modelo:

```
#set($ownerId = $ctx.prev.result.owner)
#set($callerId = $ctx.prev.result.callerId)

## if the owner is the caller, no need to make the check
#if($ownerId == $callerId)
    #return($ctx.prev.result)
#end

{
  "version" : "2018-05-29",

  "operation" : "GetItem",

  "key" : {
    "userId" : $util.dynamodb.toDynamoDBJson($callerId),
    "friendId" : $util.dynamodb.toDynamoDBJson($ownerId)
  }
}
```

6. Dentro da área de texto do modelo de mapeamento de resposta, cole o seguinte modelo:

```
#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. Escolha Criar função.

Resultado: você criou a função `isFriend`.

getPicturesByFunção do proprietário

1. Escolha a guia Funções.
2. Escolha Criar função para criar uma função.
3. Para o nome da fonte de dados, insira PicturesDynamoDBTable.
4. Para o nome da função, digite getPicturesByOwner.
5. Dentro da área de texto do modelo de mapeamento de solicitação, cole o seguinte modelo:

```
{
  "version" : "2018-05-29",
  "operation" : "Query",
  "query" : {
    "expression": "#owner = :owner",
    "expressionNames": {
      "#owner" : "owner"
    },
    "expressionValues" : {
      ":owner" : $util.dynamodb.toDynamoDBJson($ctx.prev.result.owner)
    }
  },
  "index": "owner-index"
}
```

6. Dentro da área de texto do modelo de mapeamento de resposta, cole o seguinte modelo:

```
#if($ctx.error)
  $util.error($ctx.error.message, $ctx.error.type)
#end

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

7. Escolha Criar função.

Resultado: você criou a função getPicturesByProprietário. Agora que as funções foram criadas, anexe um resolvidor de pipeline à consulta. getPicturesByCampo do proprietário.

No editor de esquemas no AWS AppSync console, no lado direito, escolha Attach Resolver forQuery.getPicturesByOwner(id: ID!): [Picture]. Na página seguinte, escolha o link Converter para resolvidor de pipeline exibido abaixo da lista suspensa da fonte de dados. Use o seguinte para o modelo de mapeamento anterior:

```
#set($result = { "owner": $ctx.args.id, "callerId": $ctx.identity.username })
$util.toJson($result)
```

Na seção modelo de mapeamento posterior, use o seguinte modelo:

```
#foreach($picture in $ctx.result.items)
  ## prepend "src://" to picture.src property
  #set($picture['src'] = "src://${picture['src']}")
#end
$util.toJson($ctx.result.items)
```

Escolha Criar resolvidor. Você anexou seu primeiro resolvidor de pipeline com sucesso. Na mesma página, adicione as duas funções criadas anteriormente. Na seção de funções, escolha Adicionar uma função e escolha ou digite o nome da primeira função, isFriend. Adicione a segunda função seguindo o mesmo processo para a função getPicturesByProprietário. Certifique-se de que a função isFriend apareça primeiro na lista, seguida pela função getPicturesByOwner. Você pode usar as setas para cima e para baixo para reorganizar a ordem de execução das funções no pipeline.

Agora que o resolvidor de pipeline foi criado e você anexou as funções, vamos testar a API do GraphQL recém-criada.

Teste da API GraphQL

Primeiro, você precisa preencher fotos e amizades executando algumas mutações usando o usuário administrador que você criou. No lado esquerdo do AWS AppSync console, escolha a guia Consultas.

Mutação createPicture

1. No AWS AppSync console, escolha a guia Consultas.
2. Escolha Login com grupos de usuários.
3. No modal, insira o Cognito Sample Client ID que foi criado pela pilha (por exemplo, CloudFormation 37solo6mmhh7k4v63cqdfgdg5d).

4. Insira o nome de usuário que você passou como parâmetro para a CloudFormation pilha. O padrão é nadia.
5. Use a senha temporária que foi enviada para o e-mail que você forneceu como parâmetro para a CloudFormation pilha (por exemplo, UserPoolUserEmail).
6. Escolha Fazer login. Agora você deve ver o botão renomeado para Logout nadia, ou qualquer nome de usuário que você escolheu ao criar a CloudFormation pilha (ou seja,). UserPoolUsername

Vamos enviar algumas mutações createPicture para preencher a tabela de fotos. Execute a seguinte consulta do GraphQL dentro do console:

```
mutation {  
  createPicture(input:{  
    owner: "nadia"  
    src: "nadia.jpg"  
  }) {  
    id  
    owner  
    src  
  }  
}
```

A resposta deve ter a aparência abaixo:

```
{  
  "data": {  
    "createPicture": {  
      "id": "c6fedbbe-57ad-4da3-860a-ffe8d039882a",  
      "owner": "nadia",  
      "src": "nadia.jpg"  
    }  
  }  
}
```

Vamos adicionar mais algumas fotos:

```
mutation {  
  createPicture(input:{  
    owner: "shaggy"
```

```
    src: "shaggy.jpg"
  }) {
    id
    owner
    src
  }
}
```

```
mutation {
  createPicture(input:{
    owner: "rex"
    src: "rex.jpg"
  }) {
    id
    owner
    src
  }
}
```

Você adicionou três fotos usando nadia como usuário administrador.

Mutação createFriendship

Vamos adicionar uma entrada de amizade. Execute as seguintes mutações no console.

Observação: você ainda deve estar conectado como o usuário admin (o usuário admin padrão é nadia).

```
mutation {
  createFriendship(id: "nadia", target: "shaggy")
}
```

A resposta deve ter a seguinte aparência:

```
{
  "data": {
    "createFriendship": true
  }
}
```

nadia e shaggy são amigos. rex não é amigo de ninguém.

getPicturesByConsulta do proprietário

Para esta etapa, faça login como o usuário nadia usando os Grupos de usuários do Cognito, com as credenciais configuradas no início desse tutorial. Como usuário nadia, recupere as fotos de propriedade do usuário shaggy.

```
query {  
  getPicturesByOwner(id: "shaggy") {  
    id  
    owner  
    src  
  }  
}
```

Como nadia e shaggy são amigos, a consulta deve retornar a foto correspondente.

```
{  
  "data": {  
    "getPicturesByOwner": [  
      {  
        "id": "05a16fba-cc29-41ee-a8d5-4e791f4f1079",  
        "owner": "shaggy",  
        "src": "src://shaggy.jpg"  
      }  
    ]  
  }  
}
```

Da mesma forma, se o usuário nadia tentar recuperar suas próprias fotos, ele também terá êxito. O resolvidor de pipeline foi otimizado para evitar a execução da GetItem operação isFriend nesse caso. Tente a seguinte consulta:

```
query {  
  getPicturesByOwner(id: "nadia") {  
    id  
    owner  
    src  
  }  
}
```

Se você habilitar o registro em log em sua API (no painel Configurações), definir o nível de depuração como TODOS e executar a mesma consulta novamente, ele retornará os logs para a execução do campo. Observando os logs, você pode determinar se a função `isFriend` retornou anteriormente no estágio Modelo de mapeamento da solicitação:

```
{
  "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
}
```

A `earlyReturnedValue` representa os dados que foram retornados pela diretiva `#return`.

Finalmente, embora `rex` seja membro do `Viewers Cognito UserPool Group` e porque `rex` não seja amigo de ninguém, ele não poderá acessar nenhuma das fotos de propriedade de `shaggy` ou `nadia`. Se você fizer login como `rex` no console e executar a seguinte consulta:

```
query {
  getPicturesByOwner(id: "nadia") {
```

```
      id
      owner
      src
    }
  }
}
```

Você receberá o seguinte erro não autorizado:

```
{
  "data": {
    "getPicturesByOwner": null
  },
  "errors": [
    {
      "path": [
        "getPicturesByOwner"
      ],
      "data": null,
      "errorType": "Unauthorized",
      "errorInfo": null,
      "locations": [
        {
          "line": 2,
          "column": 9,
          "sourceName": null
        }
      ],
      "message": "Not Authorized to access getPicturesByOwner on type Query"
    }
  ]
}
```

Você implementou com sucesso a autorização complexa usando resolvedores de pipeline.

Usando as operações do Delta Sync em fontes de dados versionadas em AWS AppSync

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Os aplicativos cliente AWS AppSync armazenam dados armazenando em cache as respostas do GraphQL localmente no disco em um aplicativo móvel/web. As fontes de dados e as operações Sync versionadas oferecem aos clientes a capacidade de executar o processo de sincronização usando um único resolvedor. Isso permite que os clientes hidratem o cache local com resultados de uma consulta base que possa ter muitos registros e receba apenas os dados alterados desde a última consulta (as atualizações delta). Ao permitir que os clientes executem a hidratação base do cache com uma solicitação inicial e atualizações adicionais em outra consulta, você pode mover o cálculo do aplicativo cliente para o back-end. Isso é muito mais eficiente para aplicativos cliente que alternam com frequência entre estados online e offline.

Para implementar a Sincronização delta, a consulta Sync usa a operação Sync em uma fonte de dados versionada. Quando uma AWS AppSync mutação altera um item em uma fonte de dados versionada, um registro dessa alteração também será armazenado na tabela Delta. Você pode optar por usar tabelas Delta diferentes (por exemplo, uma por tipo, uma por área de domínio) para outras fontes de dados versionadas ou uma única tabela Delta para sua API. AWS AppSync recomenda não usar uma única tabela Delta para várias APIs a fim de evitar a colisão de chaves primárias.

Além disso, os clientes de Sincronização delta também podem receber uma assinatura como um argumento, e a assinatura de coordenadas do cliente se reconecta e realiza gravações entre transições offline para online. A Sincronização delta realiza isso recuperando automaticamente as assinaturas (incluindo recuo exponencial e nova tentativa com tremulação por meio de diferentes cenários de erro de rede) e armazenando eventos em uma fila. A consulta delta ou base apropriada é executada antes de mesclar quaisquer eventos da fila e, finalmente, processar as assinaturas normalmente.

A documentação das opções de configuração do cliente, incluindo o Amplify DataStore, está disponível no site do [Amplify](#) Framework. Essa documentação descreve como configurar operações

Sync e fontes de dados versionadas do DynamoDB para trabalhar com o cliente de Sincronização delta para obter acesso ideal aos dados.

Configuração com um clique

Para configurar automaticamente o endpoint do GraphQL AWS AppSync com todos os resolvers configurados e os AWS recursos necessários, use este modelo: AWS CloudFormation



Esta pilha cria os seguintes recursos na sua conta:

- 2 Tabelas do DynamoDB (Base e Delta)
- 1 AWS AppSync API com chave de API
- 1 Perfil do IAM com política para tabelas do DynamoDB

Duas tabelas são usadas para particionar suas consultas de sincronização em uma segunda tabela que atua como um diário de eventos que foram perdidos quando os clientes estavam offline. Para manter as consultas eficientes na tabela delta, o [TTLsAmazon](#) DynamoDB é usado para preparar automaticamente os eventos conforme necessário. O tempo TTL é configurável para suas necessidades na fonte de dados (você pode querer que ele tenha 1 hora, 1 dia etc.).

Schema

Para demonstrar o Delta Sync, o aplicativo de amostra cria um esquema Posts apoiado por uma tabela Base e Delta no DynamoDB. AWS AppSync grava automaticamente as mutações nas duas tabelas. A consulta de sincronização obtém registros da tabela Base ou Delta, conforme apropriado, e uma única assinatura é definida para mostrar como os clientes podem aproveitar isso em sua lógica de reconexão.

```
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
}
```

O esquema do GraphQL é padrão, mas vale a pena chamar alguns itens antes de avançar. Primeiro, todas as mutações, automaticamente, primeiro gravam na tabela Base e depois na tabela Delta. A tabela Base é a fonte confiável principal do estado, enquanto a tabela Delta é o diário. Se você não passa no `lastSync: AWSTimestamp`, a consulta `syncPosts` é executada na tabela Base e hidrata o cache, além de ser executada em períodos como um processo de recuperação global para casos de borda quando os clientes ficam off-line por mais tempo que o período de TTL configurado na tabela Delta. Se você passa no `lastSync: AWSTimestamp`, a consulta `syncPosts` é executada em sua tabela Delta e é usada pelos clientes para recuperar eventos alterados desde a última vez que eles ficaram off-line. Amplificar os clientes passa automaticamente o valor `lastSync: AWSTimestamp` e persiste no disco adequadamente.

O campo `_deleted` em Postagens é usado para operações DELETE. Quando os clientes estão off-line e os registros são removidos da tabela Base, esse atributo notifica os clientes que executam a sincronização para remover itens de seu cache local. Nos casos em que os clientes ficam off-line por períodos mais longos e o item foi removido antes que o cliente possa recuperar esse valor com uma consulta de Sincronização delta, o evento de recuperação global na consulta base (configurável no cliente) será executado e remove o item do cache. Esse campo é marcado como opcional porque

retorna um valor apenas ao executar uma consulta de sincronização que tenha itens excluídos presentes.

Mutações

Para todas as mutações, AWS AppSync faz uma Create/Update/Delete operação padrão na tabela Base e também registra a alteração na tabela Delta automaticamente. Você pode reduzir ou aumentar o tempo para manter registros, modificando o valor `DeltaSyncTableTTL` na fonte de dados. Para organizações com alta velocidade de dados, pode ser adequado manter o tempo reduzido. Como alternativa, se seus clientes estiverem off-line por períodos mais longos, convém manter por mais tempo.

Consultas de sincronização

A consulta base é uma operação de sincronização do DynamoDB sem um valor `lastSync` especificado. Para muitas organizações, isso funciona porque a consulta base só é executada na inicialização e periodicamente depois disso.

A consulta delta é uma operação de sincronização do DynamoDB com um valor `lastSync` especificado. A consulta delta é executada sempre que o cliente volta a ficar online de um estado offline (desde que o período da consulta base não tenha sido acionado para execução). Os clientes rastreiam automaticamente a última vez que executaram com êxito uma consulta para sincronizar dados.

Quando uma consulta delta é executada, o resolvedor da consulta usa o `ds_pk` e o `ds_sk` para consultar somente os registros que foram alterados desde a última sincronização executada pelo cliente. O cliente armazena a resposta apropriada do GraphQL.

Para obter mais informações sobre como executar consultas de sincronização, consulte a [documentação da Operação de sincronização](#).

Exemplo

Vamos começar chamando uma mutação `createPost` para criar um item:

```
mutation create {
  createPost(input: {author: "Nadia", title: "My First Post", content: "Hello World"})
  {
    id
  }
}
```

```
    author
    title
    content
    _version
    _lastChangedAt
    _deleted
  }
}
```

O valor de retorno dessa mutação será o seguinte:

```
{
  "data": {
    "createPost": {
      "id": "81d36bbb-1579-4efe-92b8-2e3f679f628b",
      "author": "Nadia",
      "title": "My First Post",
      "content": "Hello World",
      "_version": 1,
      "_lastChangedAt": 1574469356331,
      "_deleted": null
    }
  }
}
```

Se examinar o conteúdo da tabela Base, você verá um registro semelhante a:

```
{
  "_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"  
}  
}
```

Se examinar o conteúdo da tabela Delta, você verá um registro semelhante a:

```
{  
  "_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"  
  }  
}
```

Agora podemos simular uma consulta Base que será executada pelo cliente para hidratar seu armazenamento de dados local usando uma consulta syncPosts como:

```
query baseQuery {  
  syncPosts(limit: 100, lastSync: null, nextToken: null) {  
    items {
```

```
    id
    author
    title
    content
    _version
    _lastChangedAt
  }
  startedAt
  nextToken
}
```

O valor de retorno dessa consulta Base será o seguinte:

```
{
  "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
    }
  }
}
```

Salvaremos o valor `startedAt` posteriormente para simular uma consulta Delta, mas primeiro precisamos fazer uma alteração em nossa tabela. Usaremos a mutação `updatePost` para modificar nossa Postagem existente:

```
mutation updatePost {
  updatePost(input: {id: "81d36bbb-1579-4efe-92b8-2e3f679f628b", _version: 1, title:
    "Actually this is my Second Post"}) {
    id
    author
```

```
    title
    content
    _version
    _lastChangedAt
    _deleted
  }
}
```

O valor de retorno dessa mutação será o seguinte:

```
{
  "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
    }
  }
}
```

Se examinar o conteúdo da tabela Base agora, você verá o item atualizado:

```
{
  "_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"
  }
}
```

Se examinar o conteúdo da tabela Delta agora, você verá dois registros:

1. Um registro quando o item foi criado
2. Um registro de quando o item foi atualizado.

O novo item será semelhante a:

```
{
  "_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"
  }
}
```


Agora podemos simular uma consulta Delta para recuperar modificações que ocorreram quando um cliente estava offline. Usaremos o valor `startedAt` retornado de nossa consulta Base para fazer a solicitação:

```
query delta {
  syncPosts(limit: 100, lastSync: 1574469602238, nextToken: null) {
    items {
      id
      author
      title
      content
      _version
    }
    startedAt
    nextToken
  }
}
```

O valor de retorno dessa consulta Delta será o seguinte:

```
{
  "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
    }
  }
}
```

Configuração e configurações em AWS AppSync

AWS AppSync permite que você:

- Armazene em cache dados que solicitados com frequência, mas que provavelmente não mudarão de uma solicitação para outra. Isso pode reduzir a carga nos seus resolvedores. Para obter mais informações, consulte [the section called “Como configurar o cache do lado do servidor e a compactação da carga útil da API”](#).
- Objetos GraphQL de versionamento para lidar e evitar conflitos entre vários clientes. Para obter mais informações, consulte [the section called “Versionamento, detecção de conflitos e operações de sincronização para DynamoDB”](#).
- Use nomes de domínio personalizados para configurar um domínio único e memorável que funcione tanto para seu GraphQL quanto para tempo real. APIs Para obter mais informações, consulte [Configurar nomes de domínios personalizados](#).
- Permita o acesso ao seu GraphQL APIs por meio de uma VPC. Para obter mais informações, consulte [Usando o AWS AppSync Private APIs](#).
- Compartilhe seu GraphQL APIs por meio de uma integração com o AWS Resource Access Manager Para obter mais informações, consulte [Compartilhando seu AWS AppSync APIs](#).
- Habilite a introspecção e defina o detalhamento da consulta e os limites do resolvedor por consulta. Para obter mais informações, consulte [Configuration limits](#).
- Use variáveis de ambiente para ajustar o comportamento de seus AWS AppSync resolvedores e funções sem atualizar seu código. Para obter mais informações, consulte [Usando variáveis de ambiente em AWS AppSync](#).

Além disso, AWS AppSync inclui as seguintes AWS ferramentas padrão para registro, monitoramento e rastreamento:

- [Fazendo login AWS CloudTrail](#)
- [Monitoramento com a Amazon CloudWatch](#)
- [Rastreando com AWS X-Ray](#)

Configurando o cache do lado do servidor e a compactação da carga útil da API no AWS AppSync

AWS AppSync Os recursos de cache de dados do lado do servidor disponibilizam os dados em um cache na memória de alta velocidade, melhorando a performance e diminuindo a latência. Isso reduz a necessidade de acessar fontes de dados diretamente. O armazenamento em cache está disponível para resolvedores unitários e de pipeline.

AWS AppSync também permite compactar as respostas da API para que o conteúdo da carga seja carregado e baixado mais rapidamente. Isso reduz potencialmente a pressão sobre suas aplicações e, ao mesmo tempo, reduz potencialmente suas cobranças de transferência de dados. O comportamento de compactação pode ser configurado e definido a seu próprio critério.

Consulte esta seção para obter ajuda na definição do comportamento desejado do armazenamento em cache e da compactação do lado do servidor em sua API. AWS AppSync

Tipos de instância

AWS AppSync hospeda instâncias da Amazon ElastiCache (Redis OSS) na mesma AWS conta e AWS região da sua AWS AppSync API.

Os seguintes tipos de instância ElastiCache (Redis OSS) estão disponíveis:

pequeno

1 vCPU, 1,5 GiB de RAM, performance de rede moderada

medium

2 vCPU, 3 GiB de RAM, performance de rede moderada

grande

2 vCPUs, 12,3 GiB de RAM, performance de rede de até 10 Gigabit

xlarge

4 vCPU, 25,05 GiB de RAM, performance de rede de até 10 Gigabit

2xlarge

8 vCPU, 50,47 GiB de RAM, performance de rede de até 10 Gigabit

4xlarge

16 vCPU, 101,38 GiB de RAM, performance de rede de até 10 Gigabit

8xlarge

32 vCPU, 203,26 GiB de RAM, performance de rede de 10 Gigabit (não disponível em todas as regiões)

12xlarge

48 vCPU, 317,77 GiB de RAM, performance de rede de até 10 Gigabit

Note

Historicamente, você especificou um tipo de instância específico (como `t2.medium`). Em julho de 2020, esses tipos de instâncias legados ainda estavam disponíveis, mas seu uso está obsoleto e não é recomendado. Recomendamos que você use os tipos de instância genéricos descritos aqui.

Comportamento de armazenamento em cache

Veja a seguir os comportamentos relacionados ao armazenamento em cache:

Nenhum

Sem armazenamento em cache no lado do servidor.

Armazenamento em cache de solicitação completa

O cache completo de solicitações é um mecanismo que armazena em cache os resultados da execução do resolvedor individualmente. Com essa configuração, armazena em AWS AppSync cache a execução de todos os resolvedores invocados durante uma solicitação, com cada resolvedor armazenado em cache separadamente. Os dados de cada resolvedor são recuperados de sua fonte de dados e preenchem o cache até que o tempo de vida útil (TTL) expire. Para solicitações de API subsequentes, os resultados de cada resolvedor específico são retornados do cache. Isso significa que as fontes de dados não são contatadas diretamente, a menos que o TTL tenha expirado. AWS AppSync usa o conteúdo dos `context.identity` mapas `context.arguments` e como chaves de cache para cada resolvedor.

Armazenamento em cache por resolvedor

Com essa configuração, cada resolvedor precisa ser aceito explicitamente para armazenar as respostas em cache. Um valor de TTL e as chaves de armazenamento em cache podem ser especificados no resolvedor. As chaves de cache que você pode especificar são os mapas `context.arguments` de nível superior e `context.source` os campos `context.identity` de and/or string desses mapas. O valor do TTL é obrigatório, mas as chaves de armazenamento em cache são opcionais. Se você não especificar nenhuma chave de cache, os padrões serão o conteúdo dos mapas `context.arguments`, `context.source` e `context.identity`.

Por exemplo, é possível usar as seguintes combinações:

- `context.arguments` e `context.source`
- `context.arguments` e `context.identity.sub`
- `context.arguments.id` ou `context.arguments. InputType.id`
- `context.source.id` e `context.identity.sub`
- `context.identity.claims.username`

Quando você especifica somente um TTL e nenhuma chave de cache, o comportamento do resolvedor é o mesmo do cache completo da solicitação.

Cache do nível de operação

O cache em nível de operação armazena respostas inteiras da operação de consulta do GraphQL como um todo. Quando ativadas, as respostas de consulta bem-sucedidas são armazenadas em cache até que o TTL expire, com um tamanho máximo de resposta armazenável em cache de 15 MB. Para solicitações de consulta subsequentes com a mesma chave de cache, as respostas serão atendidas diretamente do cache sem executar nenhum resolvedor enquanto o TTL não tiver expirado.

A chave de cache para o armazenamento em cache no nível de operação é gerada usando uma combinação das seguintes opções:

- Alguns atributos da carga JSON da solicitação:
 - A `query` corda
 - A `operationName` corda
 - O `variables` mapa
- O `context.identity` mapa (exceto `context.identity.sourceIp` para solicitações do IAM e do Amazon Cognito)

- O `context.request.headers` mapa (excluindo certos cabeçalhos reservados listados na próxima seção)

O tipo de autorização usado pela solicitação também afetará a chave de cache. Para solicitações autorizadas pelo IAM, a chave de cache também incluirá a lista de recursos permitidos e negados. Para solicitações autorizadas pelo Lambda, a chave de cache também incluirá a lista de campos negados.

A chave de cache considerará todos os cabeçalhos de solicitação encontrados em `context.request.headers`, exceto os seguintes cabeçalhos reservados, que normalmente são exclusivos para solicitações específicas:

- `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`
- `viax-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`

Vida útil do cache

Ele define por quanto tempo as entradas em cache ficarão armazenadas na memória. A TTL máxima é de 3.600 s (1 h), depois desse período as entradas serão automaticamente excluídas.

Criptografia de cache

Quando você usa o recurso AWS AppSync de cache de dados do lado do servidor, a criptografia em repouso e em trânsito está sempre ativada para novos caches e não pode ser desativada.

Para ativar a criptografia em um cache de API existente, exclua o cache e recrie-o.

Para invalidar as entradas do cache, você pode fazer uma chamada de API de cache livre usando o AWS AppSync console ou o AWS Command Line Interface ().AWS CLI

Para obter mais informações, consulte o tipo de [ApiCache](#) dados na Referência AWS AppSync da API.

Remoção de cache

Ao configurar o cache AWS AppSync do lado do servidor, você pode configurar um TTL máximo. Ele define por quanto tempo as entradas em cache serão armazenadas na memória. Em situações em que você precisa remover entradas específicas do seu cache, você pode usar o utilitário AWS AppSync de `evictFromApiCache` extensões na solicitação ou resposta do seu resolvedor. (Por exemplo, os dados em suas fontes de dados mudaram, e agora sua entrada de cache ficou obsoleta.) Para remover um item do cache, você deve saber qual é sua chave. Por esse motivo, se você precisar despejar itens dinamicamente, recomendamos usar o cache por resolvedor e definir explicitamente uma chave para adicionar entradas ao seu cache.

Remover uma entrada de cache

Para remover um item do cache, use o utilitário de extensões `evictFromApiCache`. Especifique o nome do tipo e o nome do campo e, em seguida, forneça um objeto de itens de valor-chave para criar a chave da entrada que você deseja remover. No objeto, cada chave representa uma entrada válida do objeto `context` que é usada na lista de `cachingKey` em cache do resolvedor. Para criar o valor da chave são usados os valores reais. Você deve colocar os itens no objeto seguindo a ordem das chaves de cache na lista em cache de `cachingKey` do resolvedor.

Por exemplo, confira o seguinte esquema:

```
type Note {
  id: ID!
  title: String
  content: String!
}

type Query {
  getNote(id: ID!): Note
}

type Mutation {
  updateNote(id: ID!, content: String!): Note
}
```

Neste exemplo, você pode habilitar o armazenamento em cache por resolvedor e, em seguida, habilitá-lo para a consulta de `getNote`. Em seguida, é possível configurar a chave de cache de modo que ela seja `[context.arguments.id]`.

Quando você tenta obter um `Note`, para criar a chave de cache, AWS AppSync executa uma pesquisa em seu cache do lado do servidor usando o `id` argumento da consulta. `getNote`

Ao atualizar uma `Note`, é preciso remover a entrada da nota específica para garantir que a próxima solicitação realize a busca na fonte de dados do back-end. Para isso, crie um manipulador de solicitações.

O exemplo a seguir mostra um modo de lidar com a remoção usando este método:

```
import { util, Context } from '@aws-appsync/utils';
import { update } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  extensions.evictFromApiCache('Query', 'getNote', { 'ctx.args.id': ctx.args.id });
  return update({ key: { id: ctx.args.id }, update: { context: ctx.args.content } });
}

export const response = (ctx) => ctx.result;
```

Outra opção é tratar da remoção no manipulador de respostas.

Quando a `updateNote` mutação é processada, AWS AppSync tenta expulsar a entrada. Se uma entrada for apagada com sucesso, a resposta vai conter um valor de `apiCacheEntriesDeleted` no objeto `extensions` que mostra quantas entradas foram excluídas:


```
"extensions": { "apiCacheEntriesDeleted": 1}
```

Remover uma entrada de cache com base na identidade

Você pode criar chaves de cache com base em diversos valores do objeto `context`.

Por exemplo, veja o esquema a seguir, que usa grupos de usuários do Amazon Cognito como modo de autenticação padrão e é apoiado por uma fonte de dados do Amazon DynamoDB:

```
type Note {  
  id: ID! # a slug; e.g.: "my-first-note-on-graphql"  
  title: String  
  content: String!  
}  
  
type Query {  
  getNote(id: ID!): Note  
}  
  
type Mutation {  
  updateNote(id: ID!, content: String!): Note  
}
```

Os tipos de objeto `Note` são salvos em uma tabela do DynamoDB. A tabela tem uma chave composta que usa o nome de usuário do Amazon Cognito como chave primária e o `id` (um slug) da `Note` como chave de partição. Esse é um sistema multilocatário que pode hospedar vários usuários e atualizar seus objetos privados `Note`, que nunca são compartilhados.

Como esse é um sistema que exige muita leitura, a `getNote` consulta é armazenada usando o cache por resolvedor, com a chave de cache composta por `[context.identity.username, context.arguments.id]`. Quando a `Note` é atualizada, você pode despejar a entrada específica da `Note`. Você deve adicionar os componentes no objeto seguindo a ordem em que estão especificados na lista de `cachingKeys` do seu resolvedor.

O exemplo a seguir mostra isso.

```
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;
```

Um sistema de back-end também pode atualizar a Note e remover a entrada. Por exemplo, considere esta mutação:

```
type Mutation {
  updateNoteFromBackend(id: ID!, content: String!, username: ID!): Note @aws_iam
}
```

Você pode despejar a entrada, mas adicionar os componentes da chave de cache ao objeto `cachingKeys`.

No exemplo a seguir, a remoção ocorre na resposta do resolvedor:

```
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;
}
```

Nos casos em que seus dados de back-end foram atualizados externamente AWS AppSync, você pode remover um item do cache chamando uma mutação que usa uma fonte de dados. NONE

Compactar respostas da API

AWS AppSync permite que os clientes solicitem cargas comprimidas. Se solicitadas, as respostas da API são compactadas e retornadas em resposta às solicitações que indicam que o conteúdo

compactado tem preferência. As respostas compactadas da API carregam mais rápido, assim como o download do conteúdo, e suas taxas de transferência de dados também podem ser reduzidas.

Note

A compactação está disponível em todos os novos APIs criados após 1º de junho de 2020. AWS AppSync [comprime](#) objetos com base no melhor esforço. Em casos raros, AWS AppSync pode pular a compressão com base em vários fatores, incluindo a capacidade atual.

AWS AppSync pode compactar tamanhos de payload de consulta GraphQL entre 1.000 e 10.000.000 bytes. Para ativar a compactação, o cliente deve enviar o cabeçalho de Accept-Encoding com o valor gzip. A compactação pode ser confirmada ao verificar o valor do cabeçalho Content-Encoding na resposta (gzip).

O explorador de consultas no AWS AppSync console define automaticamente o valor do cabeçalho na solicitação por padrão. Se você executar uma consulta com uma resposta grande o suficiente, a compactação poderá ser confirmada usando as ferramentas de desenvolvedor do seu navegador.

Configurando nomes de domínio personalizados para GraphQL e em tempo real APIs

Com AWS AppSync, você pode usar nomes de domínio personalizados para configurar um domínio único e memorável que funcione tanto para seu GraphQL quanto para tempo real APIs

Em outras palavras, você pode utilizar um endpoint simples e memorável URLs com nomes de domínio de sua escolha criando nomes de domínio personalizados que você associa à AWS AppSync APIs sua conta.

Quando você configura uma AWS AppSync API, dois endpoints são provisionados:

AWS AppSync Ponto final do GraphQL:

```
https://example1234567890000.appsync-api.us-east-1.amazonaws.com/graphql
```

AWS AppSync endpoint em tempo real:

```
wss://example1234567890000.appsync-realtime-api.us-east-1.amazonaws.com/graphql
```

Os nomes de domínio personalizados permitem interagir com os dois endpoints usando um único domínio. Por exemplo, se você configurar `api.example.com` como seu domínio personalizado, poderá interagir com seus endpoints do GraphQL e em tempo real usando estes: URLs

AWS AppSync endpoint GraphQL de domínio personalizado:

```
https://api.example.com/graphql
```

AWS AppSync endpoint em tempo real de domínio personalizado:

```
wss://api.example.com/graphql/realtime
```

Note

AWS AppSync APIs suporta somente TLS 1.2 e TLS 1.3 para nomes de domínio personalizados.

Registrar e configurar um nome de domínio

Para configurar nomes de domínio personalizados para o seu AWS AppSync APIs, você deve ter um nome de domínio da Internet registrado. Se for o caso, é possível registrar um domínio da Internet usando o Amazon Route 53 domain registration ou um registrador de domínios de terceiros de sua escolha. Para obter mais informações, consulte [O que é Amazon Route 53 Resolver?](#) no Guia do desenvolvedor do Amazon Route 53

O nome de domínio personalizado de uma API pode ser o nome de um subdomínio ou do domínio raiz (também conhecido como "ápex de zona") de um domínio da Internet registrado. Depois de criar um nome de domínio personalizado em AWS AppSync, você deve criar ou atualizar o registro de recursos do seu provedor de DNS para mapear para o endpoint da API. Sem esse mapeamento, as solicitações de API vinculadas ao nome de domínio personalizado não podem ser acessadas AWS AppSync.

Criando um nome de domínio personalizado no AWS AppSync

Criar um nome de domínio personalizado para uma AWS AppSync API configura uma Amazon CloudFront distribuição. Você deve configurar um registro DNS para mapear o nome de domínio personalizado para o nome de domínio CloudFront de distribuição. Esse mapeamento é necessário para rotear solicitações de API vinculadas ao nome de domínio personalizado AWS AppSync por

meio da CloudFront distribuição mapeada. Você também deve fornecer um certificado para o nome de domínio personalizado.

Para configurar o nome de domínio personalizado ou atualizar seu certificado, você deve ter permissão para atualizar CloudFront as distribuições e descrever o certificado AWS Certificate Manager (ACM) que você planeja usar. Para conceder essas permissões, anexe a seguinte declaração de política AWS Identity and Access Management (IAM) a um usuário, grupo ou função do IAM em sua conta:

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 oferece suporte a nomes de domínio personalizados utilizando a Indicação de Nome de Servidor (SNI) na CloudFront distribuição. Para obter mais informações sobre o uso de nomes de domínio personalizados em uma CloudFront distribuição, incluindo o formato de certificado necessário e o tamanho máximo da chave do certificado, consulte [Usando HTTPS CloudFront](#) no Amazon CloudFront Developer Guide.

Para configurar um nome de domínio personalizado como nome de host da API, o proprietário da API deve fornecer um certificado ACM público ou importado na us-east-1 AWS região (Leste dos EUA (Norte da Virgínia)) que cubra o nome de domínio personalizado. Para obter mais informações

sobre o ACM, consulte [O que é o AWS Certificate Manager?](#) no Guia do usuário do AWS Certificate Manager .

Nomes de domínio personalizados curinga em AWS AppSync

AWS AppSync suporta nomes de domínio personalizados com caracteres curinga. Para configurar um nome de domínio curinga personalizado, especifique um caractere curinga (*) como o primeiro subdomínio de um domínio personalizado. Isso representa todos os subdomínios possíveis do domínio raiz. Por exemplo, o nome de domínio curinga personalizado *.example.com resulta em subdomínios, como a.example.com, b.example.com e c.example.com. Todos esses subdomínios são roteados para o mesmo domínio.

Para usar um nome de domínio personalizado com caractere curinga AWS AppSync, você deve fornecer um certificado emitido pelo ACM contendo um nome curinga que possa proteger vários sites no mesmo domínio. Para obter mais informações, consulte [Características do certificado do ACM](#) no Guia do usuário do AWS Certificate Manager .

Controle de versão, detecção de conflitos e operações de sincronização para fontes de dados do DynamoDB no AWS AppSync

AWS AppSyncOs recursos avançados de gerenciamento de dados do DynamoDB utilizam três recursos principais: fontes de dados versionadas, detecção e resolução de conflitos e operações de sincronização. Essas ferramentas permitem aplicações robustas e escaláveis que lidam de forma eficiente com modificações e sincronização simultâneas de dados em ambientes distribuídos.

As fontes de dados versionadas formam a base do sistema de gerenciamento AWS AppSync de dados da. Esse recurso aprimora automaticamente os itens do DynamoDB com metadados de controle de versão, registra as alterações feitas AWS AppSync por mutações em uma tabela Delta e mantém “lápides” para itens excluídos. Os desenvolvedores podem configurar períodos de retenção para itens excluídos e logs de alterações, otimizando o armazenamento e garantindo a consistência de dados. As fontes de dados versionadas simplificam a implementação de operações de detecção e sincronização de conflitos, fornecendo uma base sólida para o tratamento avançado de dados.

Os mecanismos de detecção e resolução de conflitos protegem a integridade dos dados quando ocorrem gravações simultâneas. AWS AppSync oferece três estratégias: concorrência otimista, fusão automática e resolução baseada em Lambda. A simultaneidade otimista rejeita mutações

conflitantes, permitindo que os clientes tentem novamente com dados atualizados. O Automerge resolve conflitos automaticamente com base nos tipos de dados, mesclando listas, realizando uniões de conjuntos e preservando valores escalares existentes. A resolução baseada em Lambda permite uma lógica personalizada para cenários de conflitos complexos. Essas opções oferecem aos desenvolvedores flexibilidade para lidar com conflitos de dados, garantindo a consistência em sistemas distribuídos.

As operações de sincronização permitem a recuperação eficiente de dados e atualizações em aplicações cliente. Esse recurso permite que os clientes busquem todos os resultados de uma tabela do DynamoDB e, posteriormente, recuperem somente os dados alterados desde a última consulta. AWS AppSync determina se você deve acessar a tabela Base ou a tabela Delta com base no token de sincronização fornecido, otimizando o desempenho e reduzindo a transferência de dados.

Tópicos

- [Controle de versão de fontes de dados do DynamoDB no AWS AppSync](#)
- [Detecção e resolução de conflitos no AWS AppSync](#)
- [Usando as operações de sincronização do DynamoDB em fontes de dados versionadas no AWS AppSync](#)

Controle de versão de fontes de dados do DynamoDB no AWS AppSync

AWS AppSync atualmente oferece suporte ao controle de versão nas fontes de dados do DynamoDB. As operações de Detecção de conflitos, Resolução de conflitos e Sincronização exigem uma fonte de dados `Versioned`. Quando você ativa o controle de versão em uma fonte de dados, AWS AppSync automaticamente:

- Melhora os itens com metadados de versionamento de objetos.
- Registre as alterações feitas em itens com AWS AppSync mutações em uma tabela Delta.
- Mantém, por um período de tempo configurável, itens excluídos na tabela Base com uma “marca de exclusão”.

Configuração de fonte de dados versionada

Ao habilitar o versionamento em uma fonte de dados do DynamoDB, especifique os seguintes campos:

BaseTableTTL

O número de minutos para reter itens excluídos na tabela Base com uma “marca de exclusão” - um campo de metadados indicando que o item foi excluído. Você pode definir esse valor como 0 se quiser que os itens sejam removidos imediatamente quando excluídos. Este campo é obrigatório.

DeltaSyncTableName

O nome da tabela em que as alterações feitas nos itens com AWS AppSync mutações são armazenadas. Este campo é obrigatório.

DeltaSyncTableTTL

O número de minutos para reter itens na tabela Delta . Este campo é obrigatório.

Registro em log de tabela de sincronização Delta

AWS AppSync atualmente oferece suporte ao Delta Sync Logging para mutações usando `PutItemUpdateItem`, e operações `DeleteItem` do DynamoDB.

Quando uma AWS AppSync mutação altera um item em uma fonte de dados versionada, um registro dessa alteração será armazenado em uma tabela Delta otimizada para atualizações incrementais. Você pode optar por usar tabelas Delta diferentes (por exemplo, uma por tipo, uma por área de domínio) para outras fontes de dados versionadas ou uma única tabela Delta para sua API. AWS AppSync recomenda não usar uma única tabela Delta para várias APIs para evitar a colisão de chaves primárias.

O esquema necessário para esta tabela é o seguinte:

ds_pk

Um valor de string que é usado como chave de partição. Ele é estruturado ao concatenar o nome da fonte de dados Base e o formato ISO8601 da data em que a alteração ocorreu (por exemplo, `Comments:2019-01-01`)

Quando o *customPartitionKey* sinalizador do modelo de mapeamento VTL é definido como o nome da coluna da chave de partição (consulte [Resolver Mapping Template Reference for DynamoDB AWS AppSync no Developer Guide](#)), o formato *ds_pk* das alterações e a string são criados anexando a ela o valor da chave de partição no novo registro na tabela Base. Por

exemplo, se o registro na tabela Base tiver um valor de chave de partição 1a e um valor de chave de classificação de 2b, o novo valor da string será: `Comments:2019-01-01:1a`.

ds_sk

Um valor de string que é usado como chave de classificação. Ele é estruturado ao concatenar o formato ISO 8601 do momento em que a alteração ocorreu, a chave primária e a versão do item. A combinação desses campos garante exclusividade para cada entrada na tabela Delta (por exemplo, para o horário de 09:30:00, um ID 1a e a versão 2, isso seria 09:30:00:1a:2).

Quando o *customPartitionKey* sinalizador do modelo de mapeamento VTL é definido como o nome da coluna da chave de partição (consulte [Resolver Mapping Template Reference for DynamoDB AWS AppSync no Developer Guide](#)), o formato *ds_sk* das alterações e a string são criados substituindo o valor da chave de combinação pelo valor da chave de classificação na tabela base. No exemplo anterior, se o registro na tabela Base tiver um valor de chave de partição de 1a e um valor de chave de classificação de 2b, o novo valor da string será: 09:30:00:2b:3.

_ttl

Um valor numérico que armazena o timestamp, em segundos de epoch, quando um item deve ser removido da tabela Delta. Esse valor é determinado pela adição do valor `DeltaSyncTableTTL` configurado na fonte de dados no momento em que a alteração ocorreu. Esse campo deve ser configurado como o Atributo TTL do DynamoDB.

O perfil do IAM do IAM configurada para uso com a tabela Base também deve conter permissão para operar na tabela Delta. Neste exemplo, a política de permissões para uma tabela Base chamada `Comments` e uma tabela Delta chamada `ChangeLog` é exibida:

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/*"
    ]
}
]
}

```

Metadados de fonte de dados versionada

AWS AppSync gerencia campos de metadados em fontes `Versioned` de dados em seu nome. Modificar esses campos por conta própria pode causar erros no aplicativo ou a perda de dados. Estes campos incluem:

`_version`

Um contador que aumenta monotonicamente e é atualizado sempre que ocorre uma alteração em um item.

`_lastChangedAt`

Um valor numérico que armazena o timestamp, em milésimos de segundos de epoch, quando um item é modificado pela última vez.

`_deleted`

Um valor booleano de “marca de exclusão” que indica que um item foi excluído. Isso pode ser usado por aplicativos para expulsar itens excluídos de armazenamentos de dados locais.

`_ttl`

Um valor numérico que armazena o timestamp, em segundos de epoch, quando um item deve ser removido da fonte de dados subjacente.

`ds_pk`

Um valor de string que é usado como chave de partição para tabelas Delta.

ds_sk

Um valor de string que é usado como chave de classificação para tabelas Delta.

gsi_ds_pk

Um atributo de valor de string gerado para oferecer suporte a um índice secundário global como chave de partição. Ele será incluído somente se os *populateIndexFields* sinalizadores *customPartitionKey* e estiverem habilitados no modelo de mapeamento VTL (consulte [Resolver Mapping Template Reference for DynamoDB no Developer Guide](#)). AWS AppSync Se ativado, o valor será estruturado concatenando o nome da fonte de dados Base e o formato ISO 8601 da data em que a alteração ocorreu (por exemplo, se a tabela Base for chamada de Comentários, esse registro será definido como `Comments:2019-01-01`).

gsi_ds_sk

Um atributo de valor de string gerado para oferecer suporte a um índice secundário global como chave de classificação. Ele será incluído somente se os *populateIndexFields* sinalizadores *customPartitionKey* e estiverem habilitados no modelo de mapeamento VTL (consulte [Resolver Mapping Template Reference for DynamoDB no Developer Guide](#)). AWS AppSync Se ativado, o valor será construído concatenando o formato ISO 8601 da hora em que a alteração ocorreu, a chave de partição do item na tabela Base, a chave de classificação do item na tabela Base e a versão do item (por exemplo, para o horário de 09:30:00, um valor de chave de partição de 1a, um valor de chave de classificação 2b e a versão de 3, isso resultaria em 09:30:00:1a#2b:3).

Esses campos de metadados afetarão o tamanho geral dos itens na fonte de dados subjacente. AWS AppSync recomenda reservar mais de 500 bytes ou mais do tamanho máximo da chave primária de armazenamento para metadados de fonte de dados versionados ao projetar seu aplicativo. Para usar esses metadados em aplicativos cliente, inclua os campos `_version`, `_lastChangedAt` e `_deleted` nos tipos do GraphQL e no conjunto de seleção para mutações.

Detecção e resolução de conflitos no AWS AppSync

Quando ocorrem gravações simultâneas AWS AppSync, você pode configurar estratégias de detecção e resolução de conflitos para lidar com as atualizações de forma adequada. A detecção de conflitos determina se a mutação está em conflito com o item real gravado na fonte de dados. A Detecção de Conflitos é ativada definindo o valor no `conflictDetection` campo para `VERSION`. SyncConfig

A resolução de conflitos é a ação tomada caso um conflito seja detectado. Isso é determinado definindo o campo Conflict Handler no SyncConfig. Existem três estratégias de resolução de conflitos:

- OPTIMISTIC_CONCURRENCY
- AUTOMERGE
- LAMBDA

As versões são incrementadas automaticamente AWS AppSync durante as operações de gravação e não devem ser modificadas pelos clientes ou fora de um resolvedor configurado com uma fonte de dados habilitada para versão. Isso muda o comportamento de consistência do sistema e pode ocasionar perda de dados.

Simultaneidade otimista

A simultaneidade otimista é uma estratégia de resolução de conflitos que AWS AppSync fornece fontes de dados versionadas. Quando o resolvedor de conflitos é definido como Simultaneidade otimista, se uma mutação de entrada é detectada com uma versão diferente da versão real do objeto, o handler de conflitos simplesmente rejeita a solicitação recebida. Dentro da resposta do GraphQL, será fornecido o item existente no servidor que tem a versão mais recente. Espera-se então que o cliente processe esse conflito localmente e repita a mutação com a versão atualizada do item.

Automerges

O Automerge fornece aos desenvolvedores uma maneira fácil de configurar uma estratégia de resolução de conflitos sem gravar lógica no lado do cliente para mesclar manualmente conflitos incapazes de serem resolvidos com outras estratégias. O Automerge adere a um conjunto de regras rigoroso ao mesclar dados para resolver conflitos. Os princípios do Automerge giram em torno do tipo de dados subjacente do campo GraphQL. Eles são os seguintes:

- Conflito em um campo escalar: escalar do GraphQL ou qualquer campo que não seja uma coleção (ou seja, lista, conjunto, mapa). Rejeitar o valor de entrada para o campo escalar e selecionar o valor existente no servidor.
- Conflito em uma lista: o tipo do GraphQL e o tipo do banco de dados são listas. Concatenar a lista de entrada com a lista existente no servidor. Os valores da lista na mutação de entrada serão anexados ao final da lista no servidor. Valores duplicados serão retidos.

- Conflito em um conjunto: o tipo do GraphQL é uma lista e o tipo do banco de dados é um conjunto. Aplique uma união de conjunto usando a entrada do conjunto e o conjunto existente no servidor. Isso adere às propriedades de um conjunto, o que significa que não haverá entradas duplicadas.
- Quando uma mutação de entrada adiciona um novo campo ao item ou é feita em relação a um campo com o valor de `null`, ocorre a mescla no item existente.
- Conflito em um mapa: quando o tipo de dados subjacente no banco de dados é um mapa (ou seja, documento chave/valor), aplique as regras acima à medida que ele analisa e processa cada propriedade do mapa.

O Automerge foi projetado para detectar, mesclar e repetir solicitações automaticamente com uma versão atualizada, isentando o cliente da necessidade de mesclar manualmente quaisquer dados conflitantes.

Para mostrar um exemplo de como o Automerge lida com um conflito em um tipo escalar. Usaremos o seguinte registro como nosso ponto de partida.

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "_version" : 4
}
```

Agora, uma mutação de entrada pode estar tentando atualizar o item, mas com uma versão mais antiga, uma vez que o cliente ainda não efetuou a sincronização com o servidor. O resultado se parece com:

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 55,
  "_version" : 2
}
```

Observe a versão desatualizada do 2 na solicitação recebida. Durante este fluxo, o Automerge irá mesclar os dados rejeitando a atualização de campo 'jersey' para '55', além de manter o valor em '5', resultando na seguinte imagem do item que está sendo salvo no servidor.

```
{
```

```
"id" : 1,
"name" : "Nadia",
"jersey" : 5,
"_version" : 5 # version is incremented every time automerge performs a merge that is
stored on the server.
}
```

Considerando o estado do item mostrado acima na versão 5, vamos supor que uma mutação de entrada tente alterar o item com a seguinte imagem:

```
{
  "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
}
```

Há três pontos de interesse na mutação de entrada. O nome, um escalar, foi alterado, mas dois novos campos “interesses”, um conjunto, e “pontos”, uma lista, foram adicionados. Neste cenário, um conflito será detectado devido à incompatibilidade de versão. O Automerge adere às suas propriedades e rejeita a alteração de nome por ele ser um escalar e adicionar campos não conflitantes. Isso resulta no item que é salvo no servidor de modo a aparecer da seguinte forma.

```
{
  "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
}
```

Com a imagem atualizada do item com a versão 6, vamos supor que uma mutação de entrada (com outra incompatibilidade de versão) tente transformar o item para o seguinte:

```
{
  "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
}
```

Aqui observamos que o campo de entrada para “interesses” tem um valor duplicado que existe no servidor e dois novos valores. Neste caso, como o tipo de dados subjacente é um conjunto, o Automerge combinará os valores existentes no servidor com os da solicitação recebida e eliminará quaisquer duplicatas. Da mesma forma, há um conflito no campo “pontos” onde há um valor duplicado e um novo valor. Entretanto, como o tipo de dados subjacente aqui é uma lista, o Automerge simplesmente acrescentará todos os valores na solicitação de entrada ao final dos valores já existentes no servidor. A imagem mesclada resultante armazenada no servidor aparecerá da seguinte forma:

```
{
  "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
}
```

Agora vamos supor que o item armazenado no servidor aparece da seguinte forma, na versão 8.

```
{
  "id" : 1,
  "name" : "Nadia",
  "jersey" : 5,
  "interests" : ["breakfast", "lunch", "dinner", "brunch"] # underlying data type is a Set
  "points": [24, 30, 27, 30, 35] # underlying data type is a List
  "stats": {
    "ppg": "35.4",
    "apg": "6.3"
  }
  "_version" : 8
}
```

Entretanto, uma solicitação de entrada tenta atualizar o item com a imagem a seguir, mais uma vez com uma incompatibilidade de versão:

```
{
  "id" : 1,
  "name" : "Nadia",
  "stats": {
    "ppg": "25.7",
    "rpg": "6.9"
  }
  "_version" : 3
}
```

Agora, neste cenário, podemos ver que os campos que já existem no servidor estão ausentes (interesses, pontos, jersey). Além disso, o valor para “ppg” dentro do mapa “stats” está sendo editado, um novo valor “rpg” está sendo adicionado e “apg” é omitido. O Automerge preserva os campos que foram omitidos (observação: se os campos tiverem de ser removidos, será preciso tentar a solicitação novamente com a versão correspondente) para que eles não sejam perdidos. Ele também aplicará as mesmas regras aos campos dentro dos mapas e, portanto, a alteração em “ppg” será rejeitada enquanto “apg” será preservada e “rpg”, um novo campo, será adicionado. Agora, o item resultante armazenado no servidor aparecerá como:

```
{
  "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

Há várias estratégias de resolução Lambda para escolher:

- **RESOLVE**: substitui o item existente pelo novo item fornecido na carga útil de resposta. Você só pode repetir a mesma operação em um único item de cada vez. No momento, é compatível com `PutItem` e `UpdateItem` para DynamoDB.
- **REJECT**: rejeita a mutação e retorna um erro com o item existente na resposta do GraphQL. No momento, é compatível com `PutItem`, `UpdateItem` e `DeleteItem` para DynamoDB.
- **REMOVE**: remove o item existente. No momento, é compatível com `DeleteItem` para DynamoDB.

A solicitação de invocação do Lambda

O AWS AppSync resolvedor do DynamoDB invoca a função Lambda especificada no `LambdaConflictHandlerArn`. Ele usa o mesmo `service-role-arn` configurado na fonte de dados. A carga da invocação tem a seguinte estrutura:

```
{
  "newItem": { ... },
  "existingItem": {... },
  "arguments": { ... },
  "resolver": { ... },
  "identity": { ... }
}
```

Os campos são definidos da seguinte forma:

newItem

O item de visualização, se a mutação foi bem-sucedida.

existingItem

O item que reside na tabela do DynamoDB.

arguments

Os argumentos da mutação do GraphQL.

resolver

Informações sobre o AWS AppSync resolvedor.

identity

Informações sobre o chamador. Este campo é definido como nulo, se tiver acesso com a chave de API.

Exemplo de carga:

```
{
  "newItem": {
    "id": "1",
    "author": "Jeff",
    "title": "Foo Bar",
    "rating": 5,
    "comments": ["hello world"],
  },
  "existingItem": {
    "id": "1",
    "author": "Foo",
    "rating": 5,
    "comments": ["old comment"]
  },
  "arguments": {
    "id": "1",
    "author": "Jeff",
    "title": "Foo Bar",
    "comments": ["hello world"]
  },
  "resolver": {
    "tableName": "post-table",
    "awsRegion": "us-west-2",
    "parentType": "Mutation",
    "field": "updatePost"
  },
  "identity": {
    "accountId": "123456789012",
    "sourceIp": "x.x.x.x",
    "username": "AIDAAAAAAAAAAAAAAAAAAAA",
    "userArn": "arn:aws:iam::123456789012:user/appsync"
  }
}
```

A resposta de invocação do Lambda

Para resolução de conflitos PutItem e UpdateItem

RESOLVE a mutação. A resposta deve estar no seguinte formato.

```
{
```

```
"action": "RESOLVE",
"item": { ... }
}
```

O campo `item` representa um objeto que será usado para substituir o item existente na fonte de dados subjacente. A chave primária e os metadados de sincronização serão ignorados se incluídos no `item`.

REJECT a mutação. A resposta deve estar no seguinte formato.

```
{
  "action": "REJECT"
}
```

Para resolução de conflitos `DeleteItem`

REMOVE o item. A resposta deve estar no seguinte formato.

```
{
  "action": "REMOVE"
}
```

REJECT a mutação. A resposta deve estar no seguinte formato.

```
{
  "action": "REJECT"
}
```

O exemplo de função do Lambda abaixo verifica quem faz a chamada e o nome do resolvidor. Se for feito por `jeffTheAdmin`, REMOVE o objeto para `DeletePost` resolvidor ou RESOLVE o conflito com o novo item para `Update/Put` resolvidores. Se não, a mutação é 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;
}
```

Erros

Abaixo está uma lista de possíveis erros que podem ocorrer durante um processo de resolução de conflitos:

ConflictUnhandled

A detecção de conflitos localiza uma incompatibilidade de versão e o handler de conflitos rejeita a mutação.

Exemplo: resolução de conflitos com um handler de conflitos de simultaneidade otimista. Ou, o handler de conflitos do Lambda retornou REJECT.

ConflictError

Ocorre um erro interno ao tentar resolver um conflito.

Exemplo: o handler de conflitos do Lambda retornou uma resposta mal-formada. Ou, não é possível invocar o handler de conflitos do Lambda porque o recurso `LambdaConflictHandlerArn` fornecido não foi encontrado.

MaxConflicts

O máximo de tentativas de repetição foram atingidas para a resolução de conflitos.

Exemplo: excesso de solicitações simultâneas no mesmo objeto. Antes que o conflito seja resolvido, o objeto é atualizado para uma nova versão por outro cliente.

BadRequest

O cliente tenta atualizar campos de metadados (`_version`, `_ttl`, `_lastChangedAt`, `_deleted`).

Exemplo: o cliente tenta atualizar o `_version` de um objeto com uma mutação de atualização.

DeltaSyncWriteError

Falha ao gravar o registro de sincronização delta.

Exemplo: a mutação foi bem-sucedida, mas ocorreu um erro interno ao tentar gravar na tabela de sincronização delta.

InternalFailure

Ocorreu um erro interno.

UnsupportedOperation

Operação não suportada '**X**'. O controle de versão da fonte de dados suporta somente as seguintes operações (TransactGetItems,,, ScanPutItem, QueryBatchGetItem,,,, GetItem SyncDeleteItem). UpdateItem

Exemplo: uso de determinadas operações de transação e lote com conflito detection/resolution ativado. No momento, essas operações não são compatíveis.

CloudWatch Registros

Se uma AWS AppSync API tiver ativado CloudWatch os registros com as configurações de registro definidas como Registros em nível de campo `enabled` e em nível de registro para os registros em nível de campo definidas como `ALL`, AWS AppSync emitirá informações de detecção e resolução de conflitos para o grupo de registros. Para obter informações sobre o formato das mensagens de log, consulte a [documentação de Detecção de conflitos e Registro em log de sincronização](#).

Usando as operações de sincronização do DynamoDB em fontes de dados versionadas no AWS AppSync

As fontes de dados versionadas são compatíveis com as operações Sync que permitem recuperar todos os resultados de uma tabela do DynamoDB e, depois, receber apenas os dados alterados desde a última consulta (as atualizações delta). Quando AWS AppSync recebe uma solicitação para uma Sync operação, ele usa os campos especificados na solicitação para determinar se a tabela Base ou a tabela Delta devem ser acessadas.

- Se o campo `lastSync` não for especificado, será executado um Scan na tabela Base.
- Se o campo `lastSync` for especificado, mas o valor for anterior ao `current moment - DeltaSyncTTL`, será executado um Scan na tabela Base.
- Se o campo `lastSync` for especificado e o valor for igual ou posterior ao `current moment - DeltaSyncTTL`, será executada uma Query na tabela Delta.

AWS AppSync retorna o `startedAt` campo para o modelo de mapeamento de resposta para todas as Sync operações. O campo `startedAt` é o momento, em milésimos de segundos de epoch, no qual a operação Sync foi iniciada e você pode armazenar localmente e usar em outra solicitação. Se um token de paginação foi incluído na solicitação, esse valor será o mesmo que o retornado pela solicitação para a primeira página de resultados.

Para obter informações sobre o formato dos modelos de mapeamento Sync, consulte [a referência do modelo de mapeamento](#).

Usando CloudWatch para monitorar e registrar dados da API GraphQL

Você pode registrar e depurar sua API GraphQL CloudWatch usando métricas CloudWatch e registros. Essas ferramentas permitem que os desenvolvedores monitorem o desempenho, solucionem problemas e otimizem suas operações do GraphQL de forma eficaz.

CloudWatch metrics é uma ferramenta que fornece uma ampla variedade de métricas para monitorar o desempenho e o uso da API. Essas métricas são classificadas em categorias principais:

1. Métricas gerais de API: incluem `4XXError` e `5XXError` para rastrear erros do cliente e do servidor, `Latency` para medir os tempos de resposta, `Requests` para monitorar o total de chamadas de API e `TokensConsumed` para rastrear o uso de recursos.

2. Métricas de assinatura em tempo real: essas métricas se concentram em WebSocket conexões e atividades de assinatura. Elas incluem métricas para solicitações de conexão, conexões bem-sucedidas, registros de assinaturas, publicação de mensagens e conexões e assinaturas ativas.

O guia também apresenta as métricas aprimoradas, que oferecem dados mais granulares sobre desempenho do resolvidor, interações com fontes de dados e operações individuais do GraphQL. Essas métricas fornecem insights mais profundos, mas têm custos adicionais.

CloudWatch Logs é uma ferramenta que habilita recursos de registro para seu GraphQL APIs. Os logs podem ser definidos em dois níveis da API:

1. Logs em nível de solicitação: capturam informações gerais da solicitação, incluindo cabeçalhos HTTP, consultas GraphQL, resumos de operações e registros de assinaturas.
2. Logs em nível de campo: fornecem informações detalhadas sobre resoluções de campo individuais, incluindo mapeamentos de solicitações e respostas e informações de rastreamento para cada campo.

Você pode configurar o registro, interpretar as entradas do registro e usar os dados do registro para solução de problemas e otimização. AWS AppSync fornece vários tipos de log que revelam os dados de execução, análise, validação e resolução de campo da consulta.

Definição e configuração

Para ativar o registro automático em uma API GraphQL, use o AWS AppSync console.

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
2. Na APIspágina, escolha o nome de uma API do GraphQL.
3. Na página inicial da sua API, no painel de navegação, selecione Configurações.
4. Em Registro em log, faça o seguinte:
 - a. Ative a opção Ativar logs.
 - b. Para obter um registro em log detalhado no nível da solicitação, marque a caixa de seleção em Incluir conteúdo detalhado. (opcional)
 - c. Em Nível de registro do resolvidor de campo, escolha seu nível de registro em nível de campo preferido (Nenhum, Erro, Informações, Depuração ou Tudo). (opcional)

- d. Em Criar ou usar uma função existente, escolha Nova função para criar uma nova AWS Identity and Access Management (IAM) que AWS AppSync permita gravar registros CloudWatch. Você também pode selecionar Perfil existente para selecionar o nome do recurso da Amazon (ARN) de um perfil do IAM existente em sua conta da AWS .

5. Selecione Salvar.

Configuração de perfil do IAM manual

Se você optar por usar uma função do IAM existente, a função deverá conceder AWS AppSync as permissões necessárias para gravar registros CloudWatch. Para configurar isso manualmente, você deve fornecer um ARN da função de serviço para que AWS AppSync possa assumir a função ao gravar os registros.

No [console do IAM](#), crie uma nova política com o nome `AWSAppSyncPushToCloudWatchLogsPolicy` que tenha a seguinte definição:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

Em seguida, crie uma nova função com o nome `AWSAppSyncPushToCloudWatchLogsRole` anexe a política recém-criada à função. Edite a relação de confiança desse perfil da seguinte forma:

JSON

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

Copie o ARN da função e use-o ao configurar o registro para uma API GraphQL AWS AppSync .

CloudWatch métricas

Você pode usar CloudWatch métricas para monitorar e fornecer alertas sobre eventos específicos que podem resultar em códigos de status HTTP ou em latência. As seguintes métricas são emitidas:

Lista de métricas

4XXError

Erros resultantes de solicitações que não são válidas devido a uma configuração incorreta do cliente. Normalmente, esses erros acontecem em qualquer lugar fora do processamento do GraphQL. Por exemplo, esses erros podem ocorrer quando a solicitação inclui uma carga JSON incorreta ou uma consulta incorreta, quando o serviço passa por controle de utilização ou quando as configurações de autenticação estão definidas incorretamente.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências desses erros.

5XXError

Erros encontrados durante a execução de uma consulta do GraphQL. Por exemplo, isso pode ocorrer ao invocar uma consulta para um esquema vazio ou incorreto. Também pode ocorrer quando o ID ou a AWS região do grupo de usuários do Amazon Cognito não são válidos. Como alternativa, isso também pode acontecer se houver AWS AppSync um problema durante o processamento de uma solicitação.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências desses erros.

Latency

O tempo entre o momento em que AWS AppSync recebe uma solicitação de um cliente e o momento em que ela retorna uma resposta ao cliente. Isso não inclui a latência da rede encontrada para que uma resposta alcance os dispositivos finais.

Unidade: milissegundo. Use a estatística Média para avaliar as latências esperadas.

Requests

O número de solicitações (consultas + mutações) que todas APIs em sua conta processaram, por região.

Unidade: Contagem. O número de todas as solicitações processadas em uma região específica.

TokensConsumed

Os tokens são alocados para Requests com base na quantidade de recursos (tempo de processamento e memória usada) que uma Request consome. Normalmente, cada Request consome um token. No entanto, tokens adicionais são alocados a uma Request que consome grandes quantidades de recursos, conforme necessário.

Unidade: Contagem. O número de todos os tokens alocados em uma região específica.

NetworkBandwidthOutAllowanceExceeded

Note

No AWS AppSync console, na página de configurações de cache, a opção Cache Health Metrics permite que você ative essa métrica de integridade relacionada ao cache.

Os pacotes de rede foram descartados porque o throughput excedeu o limite de largura de banda agregada. Isso é útil para diagnosticar gargalos em uma configuração de cache. Os dados são registrados para uma API em particular especificando o API_Id na métrica appsyncCacheNetworkBandwidthOutAllowanceExceeded.

Unidade: Contagem. O número de pacotes descartados após exceder o limite de largura de banda de uma API especificada pelo ID.

EngineCPUUtilization

Note

No AWS AppSync console, na página de configurações de cache, a opção Cache Health Metrics permite que você ative essa métrica de integridade relacionada ao cache.

A utilização da CPU (porcentagem) alocada para o processo do Redis OSS. Isso é útil para diagnosticar gargalos em uma configuração de cache. Os dados são registrados para uma API em particular especificando o API_Id na métrica appsyncCacheEngineCPUUtilization.

Unidade: porcentagem. A porcentagem de CPU atualmente em uso pelo processo do Redis OSS para uma API especificada por ID.

Assinaturas em tempo real

Todas as métricas são emitidas em uma dimensão: gráfico QLAPIId. Isso significa que todas as métricas são acopladas à API IDs GraphQL. As métricas a seguir estão relacionadas às assinaturas do GraphQL em vez das puras: WebSockets

Lista de métricas

ConnectRequests

O número de solicitações de WebSocket conexão feitas para AWS AppSync, incluindo tentativas bem-sucedidas e malsucedidas.

Unidade: Contagem. Use a estatística de soma para mostrar o número total de solicitações de conexão.

ConnectSuccess

O número de WebSocket conexões bem-sucedidas com AWS AppSync. É possível ter conexões sem assinaturas.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências das conexões bem-sucedidas.

ConnectClientError

O número de WebSocket conexões que foram rejeitadas por AWS AppSync causa de erros do lado do cliente. Isso pode significar que o serviço está passando por controle de utilização ou que as configurações de autorização estão incorretas.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências dos erros de conexão no lado do cliente.

ConnectServerError

O número de erros originados AWS AppSync durante o processamento de conexões. Isso geralmente acontece quando ocorre um problema inesperado no lado do servidor.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências dos erros de conexão no lado do servidor.

DisconnectSuccess

O número de WebSocket desconexões bem-sucedidas de AWS AppSync.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências das desconexões bem-sucedidas.

DisconnectClientError

O número de erros do cliente originados AWS AppSync durante a desconexão WebSocket das conexões.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências de erros de desconexão.

DisconnectServerError

O número de erros do servidor originados AWS AppSync durante a desconexão WebSocket das conexões.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências de erros de desconexão.

SubscribeSuccess

O número de assinaturas que foram registradas com sucesso por meio de AWS AppSync .
WebSocket É possível ter conexões sem assinaturas, mas não é possível ter assinaturas sem conexões.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências de assinaturas bem-sucedidas.

SubscribeClientError

O número de assinaturas que foram rejeitadas por AWS AppSync causa de erros do lado do cliente. Isso pode ocorrer quando uma carga JSON está incorreta, o serviço é limitado ou as configurações de autorização estão incorretas.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências de erros de assinatura no lado do cliente.

SubscribeServerError

O número de erros originados AWS AppSync durante o processamento de assinaturas. Isso geralmente acontece quando ocorre um problema inesperado no lado do servidor.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências de erros de assinatura no lado do servidor.

UnsubscribeSuccess

O número de solicitações de cancelamento da assinatura que foram processadas com êxito.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências das solicitações de cancelamento de assinatura bem-sucedidas.

UnsubscribeClientError

O número de solicitações de cancelamento de assinatura que foram rejeitadas por AWS AppSync causa de erros do lado do cliente.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências de erros de solicitação de cancelamento de assinatura no lado do cliente.

UnsubscribeServerError

O número de erros originados AWS AppSync durante o processamento de solicitações de cancelamento de assinatura. Isso geralmente acontece quando ocorre um problema inesperado no lado do servidor.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências de erros de solicitação de cancelamento de assinatura no lado do servidor.

PublishDataMessageSuccess

O número de mensagens de evento de assinatura que foram publicadas com êxito.

Unidade: Contagem. Use a estatística Soma para obter o total das mensagens de evento de assinatura publicadas com êxito.

PublishDataMessageClientError

O número de mensagens de evento de assinatura que apresentaram falha na publicação devido a erros no lado do cliente.

Unit: Contagem. Use a estatística Soma para obter o total de ocorrências de erros de eventos de publicação de assinatura no lado do cliente.

PublishDataMessageServerError

O número de erros originados AWS AppSync durante a publicação de mensagens de eventos de assinatura. Isso geralmente acontece quando ocorre um problema inesperado no lado do servidor.

Unidade: Contagem. Use a estatística Soma para obter o total de ocorrências de erros de eventos de publicação de assinatura no lado do servidor.

PublishDataMessageSize

O tamanho das mensagens de evento de assinatura publicadas.

Unidade: Bytes.

ActiveConnections

O número de WebSocket conexões simultâneas de clientes AWS AppSync em 1 minuto.

Unidade: Contagem. Use a estatística Soma para obter o total de conexões abertas.

ActiveSubscriptions

O número de assinaturas simultâneas de clientes em um minuto.

Unidade: Contagem. Use a estatística Soma para obter o total de assinaturas ativas.

ConnectionDuration

A quantidade de tempo em que a conexão permanece aberta.

Unidade: Milissegundos. Use a estatística Média para avaliar a duração da conexão.

OutboundMessages

O número de mensagens medidas publicadas com sucesso. Uma mensagem medida equivale a 5 kB de dados entregues.

Unidade: Contagem. Use a estatística de soma para mostrar o número total de mensagens medidas publicadas.

InboundMessageSuccess

O número de mensagens de entrada processadas com êxito. Cada tipo de assinatura invocado por uma mutação gera uma mensagem de entrada.

Unidade: Contagem. Use a estatística de soma para mostrar o número total de mensagens de entrada processadas.

InboundMessageError

O número de mensagens de entrada que falharam no processamento devido a solicitações de API inválidas, como exceder o limite de tamanho da carga útil da assinatura de 240 kB.

Unidade: Contagem. Use a estatística de soma para mostrar o número total de mensagens de entrada com falhas de processamento relacionadas a API.

InboundMessageFailure

O número de mensagens de entrada que falharam no processamento devido a erros do AWS AppSync.

Unidade: Contagem. Use a estatística Sum para obter o número total de mensagens recebidas com falhas de processamento AWS AppSync relacionadas.

InboundMessageDelayed

O número de mensagens de entrada atrasadas. As mensagens de entrada podem ser atrasadas quando a cota da taxa de mensagens de entrada ou a cota da taxa de mensagens de saída é violada.

Unidade: Contagem. Use a estatística Sum para obter o número total de mensagens de entrada que atrasaram.

InboundMessageDropped

O número de mensagens de entrada descartadas. As mensagens de entrada podem ser descartadas quando a cota da taxa de mensagens de entrada ou a cota da taxa de mensagens de saída é violada.

Unidade: Contagem. Use a estatística Sum para obter o número total de mensagens de entrada que foram descartadas.

InvalidationSuccess

O número de assinaturas invalidadas com sucesso (assinatura cancelada) por uma mutação com `$extensions.invalidateSubscriptions()`.

Unidade: Contagem. Use a estatística Soma para recuperar o número total de assinaturas que foram canceladas com sucesso.

InvalidationRequestSuccess

O número de solicitações de invalidação processadas com êxito.

Unidade: Contagem. Use a estatística de soma para mostrar o número total de solicitações de invalidação processadas.

InvalidationRequestError

O número de solicitações de invalidação que falharam no processamento devido a solicitações de API inválidas.

Unidade: Contagem. Use a estatística de soma para mostrar o número total de solicitações de invalidação com falhas de processamento relacionadas a API.

InvalidationRequestFailure

O número de solicitações de invalidação que falharam no processamento devido a erros do AWS AppSync.

Unidade: Contagem. Use a estatística Sum para obter o número total de solicitações de invalidação com falhas de processamento AWS AppSync relacionadas.

InvalidationRequestDropped

O número de solicitações de invalidação perdidas quando a cota da solicitação de invalidação foi excedida.

Unidade: Contagem. Use a estatística de soma para mostrar o número total de solicitações de invalidação reduzidas.

Comparar mensagens de entrada e de saída

Quando uma mutação é executada, os campos de assinatura com a diretiva `@aws_subscribe` para essa mutação são invocados. Cada invocação de assinatura gera uma mensagem de entrada. Por exemplo, se dois campos de assinatura especificarem a mesma mutação em `@aws_subscribe`, duas mensagens de entrada serão geradas quando essa mutação for chamada.

Uma mensagem de saída equivale a 5 kB de dados entregues aos clientes. WebSocket Por exemplo, enviar 15 kB de dados para 10 clientes gera 30 mensagens de saída (15 kB * 10 clientes/5 kB por mensagem = 30 mensagens).

É possível solicitar aumentos de cota para mensagens de entrada ou de saída. Para obter mais informações, consulte [Endpoints e cotas do AWS AppSync](#) no guia Referência geral da AWS e as instruções de [Solicitação de aumento de cota](#) no Guia do usuário sobre Service Quotas.

Métricas aprimoradas

As métricas aprimoradas emitem dados granulares sobre o uso e o desempenho da API, como contagens de solicitações e erros do AWS AppSync, latência e acertos/erros do cache. Todos os dados métricos aprimorados são enviados para sua CloudWatch conta, e você pode configurar os tipos de dados que serão enviados.

Note

Cobranças adicionais são aplicadas ao usar métricas aprimoradas. Para obter mais informações, consulte os níveis detalhados de preços de monitoramento nos [CloudWatchpreços da Amazon](#).

Essas métricas podem ser encontradas em várias páginas de configurações no AWS AppSync console. Na página de configurações da API, a seção Métricas aprimoradas permite ativar ou desativar os seguintes itens:

Comportamento das métricas de resolvedores: essas opções controlam como métricas adicionais para resolvedores são coletadas. Você pode optar por ativar as métricas de resolvedores de solicitações completas (métricas ativadas para todos os resolvedores de solicitações) ou métricas para cada resolvedor (métricas ativadas somente para resolvedores em que a configuração está definida como ativada). As seguintes opções estão disponíveis:

Lista de comportamento das métricas de resolvidor

GraphQL errors per resolver (GraphQLError)

O número de erros do GraphQL que ocorreram por resolvidor.

Dimensão da métrica: API_Id, Resolver

Unidade: Contagem.

Requests per resolver (Request)

O número de invocações que ocorreram durante uma solicitação. Isso é registrado por resolvidor.

Dimensão da métrica: API_Id, Resolver

Unidade: Contagem.

Latency per resolver (Latency)

O tempo para concluir uma invocação do resolvidor. A latência é medida em milissegundos e registrada por resolvidor.

Dimensão da métrica: API_Id, Resolver

Unidade: milissegundo.

Cache hits per resolver (CacheHit)

O número de acertos do cache durante uma solicitação. Isso só será emitido se um cache for usado. Os acertos ao cache são registrados por resolvidor.

Dimensão da métrica: API_Id, Resolver

Unidade: Contagem.

Cache misses per resolver (CacheMiss)

O número de erros do cache durante uma solicitação. Isso só será emitido se um cache for usado. Os erros do cache são registrados por resolvidor.

Dimensão da métrica: API_Id, Resolver

Unidade: Contagem.

Comportamento das métricas de fontes de dados: essas opções controlam como métricas adicionais para fontes de dados são coletadas. Você pode optar por ativar as métricas de fontes de dados de solicitações completas (métricas ativadas para todas as fontes de dados de solicitações) ou as métricas para cada fonte de dados (métricas ativadas somente para as fontes de dados em que a configuração está definida como ativada). As seguintes opções estão disponíveis:

Lista de comportamento das métricas de fonte de dados

Requests per data source (Request)

O número de invocações que ocorreram durante uma solicitação. As solicitações são registradas por fonte de dados. Se as solicitações completas estiverem habilitadas, cada fonte de dados terá sua própria entrada CloudWatch.

Dimensão da métrica: API_Id, Datasource

Unidade: Contagem.

Latency per data source (Latency)

O tempo para concluir uma invocação de fonte de dados. A latência é registrada por fonte de dados.

Dimensão da métrica: API_Id, Datasource

Unidade: milissegundo.

Errors per data source (GraphQLError)

O número de erros que ocorreram durante uma invocação de fonte de dados.

Dimensão da métrica: API_Id, Datasource

Unidade: Contagem.

Métricas de operação: ative métricas em nível operacional do GraphQL.

Lista de comportamento de métricas de operação

Requests per operation (Request)

O número de vezes que uma operação especificada do GraphQL foi chamada.

Dimensão da métrica: API_Id, Operation

Unidade: Contagem.

GraphQL errors per operation (GraphQLError)

O número de erros do GraphQL que ocorreram durante uma operação especificada do GraphQL.

Dimensão da métrica: API_Id, Operation

Unidade: Contagem.

CloudWatch troncos

Você pode configurar dois tipos de registro em log em qualquer API GraphQL nova ou existente: nível de solicitação e nível de campo.

Logs a nível de solicitação

Quando o registro em log a nível de solicitação (Incluir conteúdo detalhado) é configurado, as seguintes informações são registradas em log:

- O número de tokens consumidos
- Os cabeçalhos HTTP da solicitação e resposta
- A consulta do GraphQL que está sendo executada na solicitação
- O resumo geral da operação
- Assinaturas do GraphQL novas e existentes registradas

Logs a nível de campo

Quando o registro log a nível de campo é configurado, as seguintes informações são registradas em log:

- Mapeamento de solicitação gerado com origem e argumentos para cada campo
- O Mapeamento da resposta transformado para cada campo, que inclui os dados como resultado da resolução desse campo
- Informações de rastreamento para cada campo

Se você ativar o registro, AWS AppSync gerencia os CloudWatch registros. O processo inclui a criação de grupos de log e fluxos de log, além de relatórios aos fluxos de log com esses logs.

Quando você ativa o registro em uma API do GraphQL e faz solicitações, AWS AppSync cria um grupo de registros e fluxos de registros sob o grupo de registros. O grupo de logs é chamado seguindo o formato `/aws/appsync/apis/{graphql_api_id}`. Dentro de cada grupo de logs, os logs são subdivididos em fluxos de log. Esses são ordenados por Hora do último evento conforme os dados registrados em log são reportados.

Cada evento de log é marcado com o `x-amzn-RequestId` dessa solicitação. Isso ajuda você a filtrar eventos de registro CloudWatch para obter todas as informações registradas sobre essa solicitação. Você pode obter o dos cabeçalhos `RequestId` de resposta de cada solicitação do AWS AppSync GraphQL.

O registro em log a nível de campo é configurado com os seguintes níveis de log:

- Nenhum - Nenhum registro em nível de campo é capturado.
- Erro - registra as seguintes informações somente para os campos que estão na categoria de erro:
 - A seção de erro na resposta do servidor
 - Os erros a nível de campo
 - As request/response funções geradas que foram resolvidas para campos de erro
- Informações - registra as seguintes informações somente para os campos que estão nas categorias de informações e erros:
 - Mensagens em nível de informação
 - As mensagens do usuário enviadas por `$util.log.info` e `console.log`
 - Os registros de rastreamento e mapeamento em nível de campo não são mostrados.
 - Se o registro em nível de campo estiver definido como INFO ou superior com conteúdo detalhado incluído, AWS AppSync adicionará as mensagens de registro do modelo de mapeamento transformado. Isso conterá todas as informações adicionadas ao modelo de mapeamento transformado ou à saída do resolvidor ou do JavaScript código executado pela função e não deve ser usado se você planeja enviar informações confidenciais, como senhas ou cabeçalhos de autorização, para fontes de dados downstream e não quiser essas informações em seus registros.
- Depuração - registra as seguintes informações somente para os campos que estão nas categorias de depuração, informações e erro:
 - Mensagens em nível de depuração
 - As mensagens do usuário enviadas por meio de `$util.log.info`, `$util.log.debug`, `console.log`, e `console.debug`

- Os registros de rastreamento e mapeamento em nível de campo não são mostrados.
- Todos – As informações a seguir são registradas em log para todos os campos na consulta:
 - Informações de rastreamento a nível de campo
 - As request/response funções geradas que foram resolvidas para cada campo

Benefícios do monitoramento

É possível usar o registro em log e as métricas para identificar, solucionar problemas e otimizar as consultas do GraphQL. Por exemplo, isso ajudará a depurar problemas de latência usando as informações de rastreamento registradas em log para cada campo na consulta. Para demonstrar isso, suponha que você está usando um ou mais resolvedores aninhados em uma consulta do GraphQL. Um exemplo de operação de campo no CloudWatch Logs pode ser semelhante ao seguinte:

```
{
  "path": [
    "singlePost",
    "authors",
    0,
    "name"
  ],
  "parentType": "Post",
  "returnType": "String!",
  "fieldName": "name",
  "startOffset": 416563350,
  "duration": 11247
}
```

Isso pode corresponder a um esquema do GraphQL, semelhante ao seguinte:

```
type Post {
  id: ID!
  name: String!
  authors: [Author]
}

type Author {
  id: ID!
  name: String!
}
```

```
type Query {  
  singlePost(id:ID!): Post  
}
```

Nos resultados do log acima, o path mostra um único item nos dados retornados da execução de uma consulta chamada `singlePost()`. Nesse exemplo, ele representa o campo nome no primeiro índice (0). O `startOffset` fornece um desvio do início da operação da consulta do GraphQL. A duração é o tempo total para resolver o campo. Esses valores podem ser úteis para entender por que os dados de uma fonte de dados particular podem estar com execução mais lenta que o esperado, ou se um campo específico está atrasando toda a consulta. Por exemplo, é possível aumentar um throughput provisionado para uma tabela do Amazon DynamoDB ou remover um campo específico de uma consulta que está fazendo com que a execução global tenha um desempenho insatisfatório.

A partir de 8 de maio de 2019, AWS AppSync gera eventos de log como JSON totalmente estruturado. Isso pode ajudar você a usar serviços de análise de log, como CloudWatch Logs Insights e Amazon OpenSearch Service, para entender o desempenho de suas solicitações do GraphQL e as características de uso de seus campos de esquema. Por exemplo, é possível identificar facilmente resolvedores com grandes latências que podem ser a causa raiz de um problema de desempenho. Também é possível identificar os campos mais e menos usados no esquema e avaliar o impacto de desativar campos do GraphQL.

Detecção de conflitos e registro em log de sincronização

Se uma AWS AppSync API tiver o registro em CloudWatch registros configurado com o nível de registro do resolvedor de campo definido como Todos, ela AWS AppSync emitirá informações de detecção e resolução de conflitos para o grupo de registros. Isso fornece uma visão granular de como a AWS AppSync API respondeu a um conflito. Para ajudar você a interpretar a resposta, as seguintes informações são fornecidas nos logs:

Lista de métricas

`conflictType`

Detalhes em caso de conflito devido a uma incompatibilidade de versão ou à condição fornecida pelo cliente.

`conflictHandlerConfigured`

Afirma o handler de conflitos configurado no resolvedor no momento da solicitação.

message

Fornece informações sobre como o conflito foi detectado e resolvido.

syncAttempt

O número de tentativas do servidor de sincronizar os dados antes de finalmente rejeitar a solicitação.

data

Se o handler de conflitos configurado foi Automerge, esse campo será preenchido de modo a mostrar a decisão que o Automerge tomou para cada campo. As ações fornecidas podem ser:

- REJEITADO - Quando o Automerge rejeita o valor do campo de entrada em função do valor no servidor.
- ADICIONADO - Quando o Automerge adiciona o campo recebido devido a não haver valor preexistente no servidor.
- ANEXADO - Quando o Automerge anexa os valores de entrada aos valores da Lista que existe no servidor.
- MESCLADO - Quando o Automerge mescla os valores de entrada aos valores do Conjunto que existe no servidor.

Usar contagens de tokens para otimizar suas solicitações

As solicitações que consomem menos ou igual a 1.500 KB por segundo de memória e tempo de vCPU recebem um token. As solicitações com consumo de recursos superior a 1.500 KB por segundo recebem tokens adicionais. Por exemplo, se uma solicitação consumir 3.350 KB por segundo, AWS AppSync aloca três tokens (arredondados para o próximo valor inteiro) para a solicitação. Por padrão, AWS AppSync aloca um máximo de 5.000 ou 10.000 tokens de solicitação por segundo APIs na sua conta, dependendo da AWS região em que está implantado. Se APIs cada um usar uma média de dois tokens por segundo, você estará limitado a 2.500 ou 5.000 solicitações por segundo, respectivamente. Se você precisar de mais tokens por segundo do que o valor alocado, poderá enviar uma solicitação para aumentar a cota padrão da taxa de tokens de solicitação. Para obter mais informações, consulte [AWS AppSync endpoints e cotas](#) no Referência geral da AWS guia e [Solicitando um aumento de cota no Service Quotas User](#) Guide.

Uma alta contagem de tokens por solicitação pode indicar que há uma oportunidade de otimizar suas solicitações e melhorar o desempenho da sua API. Os fatores que podem aumentar sua contagem de tokens por solicitação incluem:

- O tamanho e a complexidade do seu esquema GraphQL.
- A complexidade dos modelos de mapeamento de solicitações e respostas.
- O número de invocações do resolvidor por solicitação.
- A quantidade de dados retornados pelos resolvidores.
- A latência das fontes de dados downstream.
- Projetos de esquemas e consultas que exigem chamadas sucessivas à fonte de dados (em oposição às chamadas paralelas ou em lote).
- Configuração de logs, particularmente conteúdo de log detalhado e em nível de campo.

Note

Além de AWS AppSync métricas e registros, os clientes podem acessar o número de tokens consumidos em uma solicitação por meio do cabeçalho de resposta `x-amzn-appsync-TokensConsumed`.

Limites de tamanho de log

Por padrão, se o registro estiver ativado, AWS AppSync enviará até 1 MB de registros por solicitação. Logs que excederem esse tamanho serão truncados. Para reduzir tamanhos de log, escolha o nível de registro em log de ERROR para logs em nível de campo e desative o registro em log de VERBOSE, ou desative totalmente os logs em nível de campo se não forem necessários. Como alternativa ao nível de ALL log, você pode usar métricas aprimoradas para obter métricas sobre resolvidores, fontes de dados ou operações do GraphQL específicos, ou utilizar os utilitários de registro fornecidos AppSync por para registrar somente as informações necessárias.

Referência de tipo de log

RequestSummary

- `requestId`: identificador exclusivo da solicitação.
- `graphqlAPIId`: ID da API GraphQL que está fazendo a solicitação.
- `statusCode`: resposta do código de status HTTP.
- `latência`: End-to-end latência da solicitação, em nanossegundos, como um número inteiro.

```
{
  "logType": "RequestSummary",
  "requestId": "dbe87af3-c114-4b32-ae79-8af11f3f96f1",
  "graphqlAPIId": "pmo28inf75eepg63qxq4ekoeg4",
  "statusCode": 200,
  "latency": 242000000
}
```

ExecutionSummary

- **requestId**: identificador exclusivo da solicitação.
- **graphqlAPIId**: ID da API GraphQL que está fazendo a solicitação.
- **startTime**: o timestamp de início do processamento do GraphQL para a solicitação, no formato RFC 3339.
- **endTime**: o timestamp de término do processamento do GraphQL para a solicitação, no formato RFC 3339.
- **duration**: o tempo total do processamento do GraphQL, em nanossegundos, como um valor inteiro.
- **versão**: A versão do esquema do ExecutionSummary.
- **parsing**:
 - **startOffset**: o deslocamento inicial para análise, em nanossegundos, em relação ao início da invocação, como um valor inteiro.
 - **duration**: o tempo gasto na análise, em nanossegundos, como um valor inteiro.
- **validation**:
 - **startOffset**: o deslocamento inicial para validação, em nanossegundos, em relação ao início da invocação, como um valor inteiro.
 - **duration**: o tempo gasto na execução da validação, em nanossegundos, como um valor inteiro.

```
{
  "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"
}

```

Rastreamento

- **requestId**: identificador exclusivo da solicitação.
- **graphqlAPIId**: ID da API GraphQL que está fazendo a solicitação.
- **startOffset**: o deslocamento inicial para a resolução do campo, em nanossegundos, em relação ao início da invocação, como um valor inteiro.
- **duration**: o tempo gasto na resolução do campo, em nanossegundos, como um valor inteiro.
- **fieldName**: o nome do campo que está sendo resolvido.
- **parentType**: o tipo pai do campo que está sendo resolvido.
- **returnType**: o tipo de retorno do campo que está sendo resolvido.
- **path**: uma lista de segmentos de caminho, começando pela raiz da resposta e terminando com o campo que está sendo resolvido.
- **resolverArn**: o ARN do resolvidor usado para resolução do campo. Pode não estar presente em campos aninhados.

```

{
    "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"  
}
```

Analizando seus registros com o CloudWatch Logs Insights

Veja a seguir exemplos de consultas que você pode executar para obter insights práticos sobre o desempenho e a integridade das operações do GraphQL. Esses exemplos estão disponíveis como exemplos de consultas no console do CloudWatch Logs Insights. No [CloudWatchconsole](#), escolha Logs Insights, selecione o AWS AppSync grupo de registros para sua API GraphQL e, em seguida, escolha AWS AppSync consultas em Exemplos de consultas.

A consulta a seguir retorna as 10 principais solicitações do GraphQL com o máximo de tokens consumidos:

```
filter @message like "Tokens Consumed"  
| parse @message "* Tokens Consumed: *" as requestId, tokens  
| sort tokens desc  
| display requestId, tokens  
| limit 10
```

A consulta a seguir retorna os 10 principais resolvedores com latência máxima:

```
fields resolverArn, duration  
| filter logType = "Tracing"  
| limit 10  
| sort duration desc
```

A consulta a seguir retorna os resolvedores invocados com mais frequência:

```
fields ispresent(resolverArn) as isRes  
| stats count() as invocationCount by resolverArn  
| filter isRes and logType = "Tracing"  
| limit 10  
| sort invocationCount desc
```

A consulta a seguir retorna resolvedores com a maioria dos erros em modelos de mapeamento:

```
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
```

A consulta a seguir retorna estatísticas de latência do resolvedor:

```
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
```

A consulta a seguir retorna estatísticas de latência de campo:

```
stats min(duration), max(duration), avg(duration) as avg_dur
by concat(parentType, '/', fieldName) as fieldKey
| filter logType = "Tracing"
| limit 10
| sort avg_dur desc
```

Os resultados das consultas do CloudWatch Logs Insights podem ser exportados para CloudWatch painéis.

Analise seus registros com o OpenSearch Service

Você pode pesquisar, analisar e visualizar seus AWS AppSync registros com o Amazon OpenSearch Service para identificar gargalos de desempenho e causas-raiz de problemas operacionais. É possível identificar resolvedores com a latência máxima e os erros. Além disso, você pode usar OpenSearch painéis para criar painéis com visualizações poderosas. OpenSearch O Dashboards é uma ferramenta de visualização e exploração de dados de código aberto disponível no OpenSearch Service. Usando OpenSearch painéis, você pode monitorar continuamente o desempenho e a integridade de suas operações do GraphQL. Por exemplo, você pode criar painéis para visualizar a latência P90 das solicitações do GraphQL e analisar as latências P90 de cada resolvedor.

Ao usar o OpenSearch Serviço, use “cwl*” como padrão de filtro para pesquisar OpenSearch índices. OpenSearch O serviço indexa os registros transmitidos do CloudWatch Logs com o prefixo “cwl -”. Para diferenciar os registros da AWS AppSync API de outros CloudWatch registros

enviados ao OpenSearch Serviço, recomendamos adicionar uma expressão de filtro adicional de `graphqlAPIID.keyword=YourGraphQLAPIID` à sua pesquisa.

Migração do formato de log

Os eventos de registro AWS AppSync gerados em ou após 8 de maio de 2019 são formatados como JSON totalmente estruturado. [Para analisar solicitações do GraphQL anteriores a 8 de maio de 2019, você pode migrar registros antigos para um JSON totalmente estruturado usando um script disponível na amostra. GitHub](#) Se for necessário usar o formato de log antes de 8 de maio de 2019, crie um tíquete de suporte com as seguintes configurações: defina Tipo como Gerenciamento de contas e, depois, defina Categoria como Pergunta geral sobre a conta.

Você também pode usar [filtros métricos](#) CloudWatch para transformar dados de registro em CloudWatch métricas numéricas, para que você possa representar graficamente ou definir um alarme para eles.

Usando AWS X-Ray para rastrear solicitações em AWS AppSync

Você pode usar [AWS X-Ray](#) para rastrear solicitações à medida que elas são executadas em AWS AppSync. Você pode usar o X-Ray AWS AppSync em todas as AWS regiões onde o X-Ray está disponível. O X-Ray oferece uma visão geral detalhada de uma solicitação GraphQL inteira. Isso permite que você analise as latências em seus resolvedores APIs e fontes de dados subjacentes. Você pode usar um mapa do serviço X-Ray para visualizar a latência de uma solicitação, incluindo quaisquer AWS serviços integrados ao X-Ray. Também é possível configurar regras de amostragem para informar ao X-Ray quais solicitações registrar e com quais taxas de amostragem, de acordo com os critérios especificados.

Para obter mais informações sobre amostragem no X-Ray, consulte [Configurando regras de amostragem no console](#). AWS X-Ray

Definição e configuração

Você pode ativar o rastreamento X-Ray para uma API GraphQL por meio do AWS AppSync console.

1. Faça login no AWS AppSync console.
2. Selecione Configurações no painel de navegação.
3. Em X-Ray, ative a opção Habilitar o X-Ray.
4. Escolha Salvar. O rastreamento do X-Ray agora está habilitado para sua API.

Se você estiver usando o AWS CLI ou AWS CloudFormation, você também pode ativar o X-Ray Tracing ao criar uma nova AWS AppSync API ou atualizar uma AWS AppSync API existente, definindo a `xrayEnabled` propriedade como `true`.

Quando o X-Ray Tracing é ativado para uma AWS AppSync API, uma [função AWS Identity and Access Management vinculada ao serviço](#) é criada automaticamente em sua conta com as permissões apropriadas. Isso permite AWS AppSync enviar traços para o X-Ray de forma segura.

Rastreamento da sua API com o X-Ray

Amostragem

Ao usar regras de amostragem, é possível controlar a quantidade de dados gravados no AWS AppSync e modificar o comportamento de amostragem instantaneamente, sem modificar nem reimplantar o código. Por exemplo, esta regra faz amostragem de solicitações para a API GraphQL com o ID da API `3n572shhpcfokwhdnq1ogu59v6`.

- Nome da regra – `test-sample`
- Prioridade – `10`
- Tamanho do reservatório – `10`
- Taxa fixa – `10`
- Nome do serviço – `*`
- Tipo de serviço – `AWS::AppSync::GraphQLAPI`
- Método HTTP – `*`
- ARN do recurso – `arn:aws:appsync:us-west-2:123456789012:apis/3n572shhpcfokwhdnq1ogu59v6`
- Host – `*`

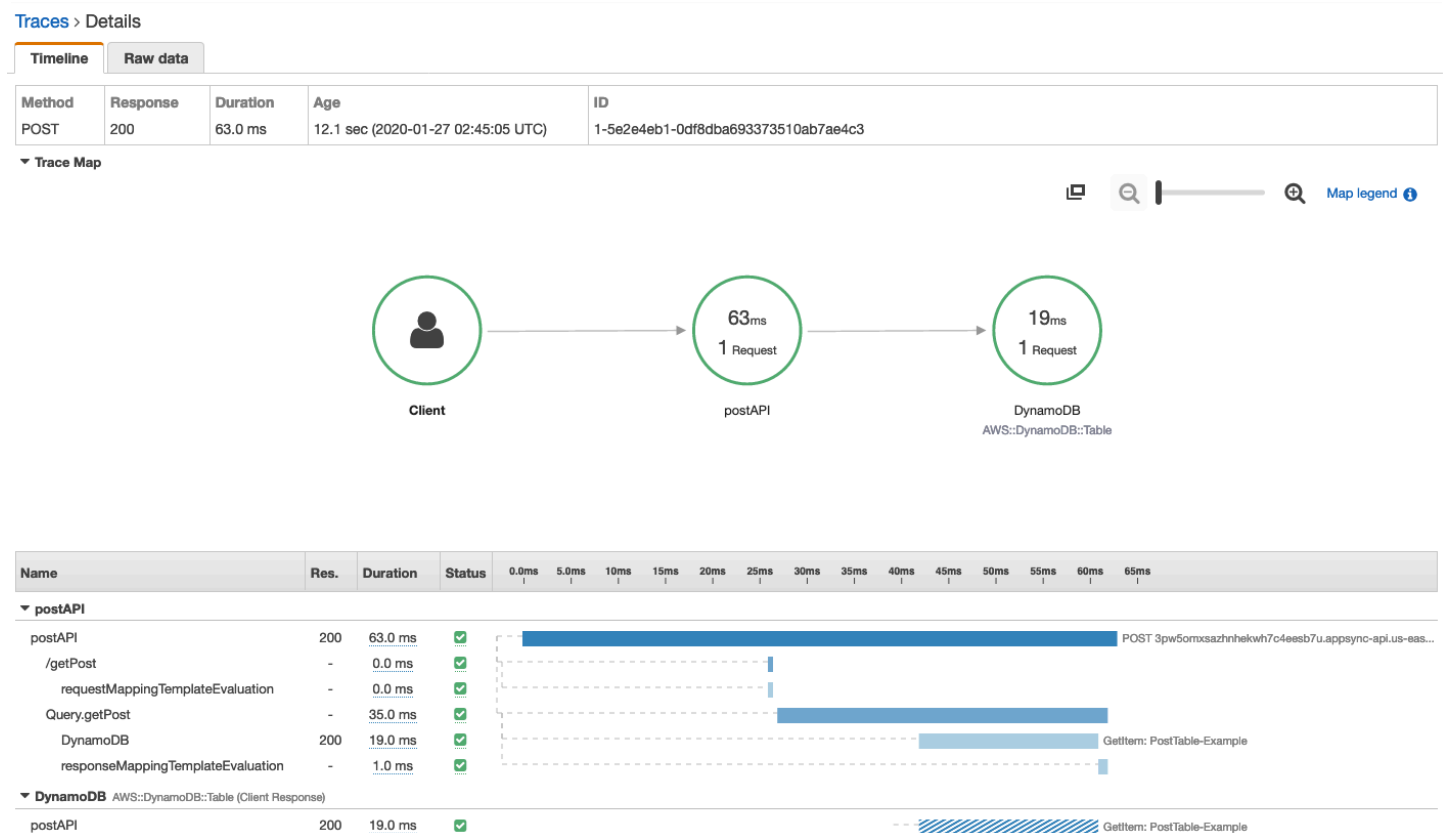
Noções básicas sobre rastreamentos

Ao habilitar o rastreamento do X-Ray para sua API GraphQL, é possível usar a página de detalhes de rastreamento do X-Ray para examinar informações detalhadas de latência sobre solicitações feitas para a API. O exemplo a seguir mostra a exibição de rastreamento junto com o mapa de serviço para essa solicitação específica. A solicitação foi feita para uma API chamada `postAPI` com um tipo `Post`, cujos dados estão contidos em uma tabela do Amazon DynamoDB chamada `PostTable-Example`.

A imagem de rastreamento a seguir corresponde à seguinte consulta do GraphQL:

```
query getPost {
  getPost(id: "1") {
    id
    title
  }
}
```

O resolvidor para a consulta `getPost` usa a fonte de dados subjacente do DynamoDB. A exibição de rastreamento a seguir mostra a chamada para o DynamoDB, bem como as latências de várias partes da execução da consulta:



- Na imagem anterior, `/getPost` representa o caminho completo para o elemento que está sendo resolvido. Nesse caso, como `getPost` é um campo no tipo raiz `Query`, ele é exibido diretamente após a raiz do caminho.
- `requestMappingTemplateEvaluation` representa o tempo gasto pelo AWS AppSync para avaliar o modelo de mapeamento de solicitação para esse elemento na consulta.
- `Query.getPost` representa um tipo e um campo (no formato `Type.field`). Ele pode conter vários subsegmentos, dependendo da estrutura da API e da solicitação que está sendo rastreada.

- `DynamoDB` representa a fonte de dados associada a esse resolvedor. Ele contém a latência para a chamada de rede para o DynamoDB resolver o campo.
- `responseMappingTemplateEvaluation` representa o tempo gasto pelo AWS AppSync para avaliar o modelo de mapeamento de resposta para esse elemento na consulta.

Ao visualizar traços no X-Ray, você pode obter informações contextuais e de metadados adicionais sobre os subsegmentos no AWS AppSync segmento escolhendo os subsegmentos e explorando a exibição detalhada.

Para determinadas consultas complexas ou profundamente aninhadas, observe que o segmento entregue ao X-Ray por AWS AppSync pode ser maior do que o tamanho máximo permitido para documentos do segmento, conforme definido em Documentos do [AWS X-Ray Segmento](#). O X-Ray não exibe segmentos que excedam o limite.

Registrando chamadas de AWS AppSync API usando AWS CloudTrail

AWS AppSync é integrado com AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, função ou AWS serviço em AWS AppSync. CloudTrail captura todas as chamadas de API AWS AppSync como eventos. As chamadas capturadas incluem chamadas do AWS AppSync console e de chamadas de código para AWS AppSync APIs. Você pode usar as informações coletadas por CloudTrail para determinar a solicitação que foi feita AWS AppSync, o endereço IP do solicitante, quem fez a solicitação, quando a solicitação foi feita e detalhes adicionais.

Você pode criar uma trilha para permitir a entrega contínua de CloudTrail eventos para um bucket do Amazon Simple Storage Service (Amazon S3), incluindo eventos para AWS AppSync. Se você não configurar uma trilha, ainda poderá ver os eventos mais recentes no CloudTrail console.

Para obter mais informações sobre CloudTrail, consulte o [Guia AWS CloudTrail do usuário](#).

AWS AppSync informações em CloudTrail

CloudTrail é ativado em sua AWS conta quando você cria a conta. No CloudTrail console, no Histórico de eventos, você pode visualizar, pesquisar e baixar eventos recentes em sua AWS conta. Para obter mais informações, consulte [Visualização de CloudTrail eventos com histórico](#) de eventos no Guia AWS CloudTrail do usuário.

Para um registro contínuo dos eventos em sua AWS conta, incluindo eventos para AWS AppSync, crie uma trilha. Por padrão, quando uma trilha é criada no console, a mesma é aplicada a todas as regiões da AWS. A trilha registra eventos de todas as regiões na AWS partição e entrega os arquivos de log ao bucket do Amazon S3 que você especificar. Além disso, você pode configurar outros AWS serviços para analisar e agir com base nos dados de eventos coletados nos CloudTrail registros. Para mais informações, consulte o seguinte no Guia do usuário do AWS CloudTrail :

- [Criando uma trilha para sua AWS conta](#)
- [AWS Integrações de serviços com registros CloudTrail](#)
- [Configurando notificações do Amazon SNS para CloudTrail](#)
- [Recebendo arquivos de CloudTrail log de várias regiões](#)
- [Recebendo arquivos de CloudTrail log de várias contas](#)

CloudTrail registra todas as operações AWS AppSync da API. Por exemplo, chamadas para o `CreateGraphQLApiCreateDataSource`, e `ListResolvers` APIs geram entradas nos arquivos de CloudTrail log. Essas e outras operações estão documentadas na [Referência da API AWS AppSync](#).

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar:

- Se a solicitação foi feita com credenciais de usuário root ou AWS Identity and Access Management (IAM).
- Se a solicitação foi feita com credenciais de segurança temporárias de uma função ou de um usuário federado.
- Se a solicitação foi feita por outro AWS serviço.

Para obter mais informações, consulte [CloudTrail UserIdentity Element](#) no Guia do AWS CloudTrail usuário.

AWS AppSync eventos de dados em CloudTrail

Os [Eventos de dados](#) fornecem informações sobre as operações de recursos realizadas em um recurso (por exemplo, leitura ou gravação em um objeto do Amazon S3). Também são conhecidas como operações de plano de dados. Eventos de dados geralmente são atividades de alto volume.

Por padrão, CloudTrail não registra eventos de dados. O histórico de CloudTrail eventos não registra eventos de dados.

Há cobranças adicionais para eventos de dados. Para obter mais informações sobre CloudTrail preços, consulte [AWS CloudTrail Preços](#).

Você pode registrar eventos de dados para o tipo de AWS::AppSync::GraphQLApi recurso usando o CloudTrail console ou as operações da CloudTrail API (elas incluem operações de consulta, mutação e assinatura, operações de conexão ao seu endpoint em tempo real, mas não mensagens enviadas por seu WebSocket endpoint em tempo real WebSocket). AWS CLI Para obter mais informações sobre como registrar eventos de dados em log, consulte [Registrar eventos de dados com o AWS Management Console](#) e [Registrar eventos de dados com a AWS Command Line Interface](#) no Guia do usuário do AWS CloudTrail.

A tabela a seguir lista o tipo de AWS AppSync recurso para o qual você pode registrar eventos de dados. A coluna Tipo de evento de dados (console) mostra o valor a ser escolhido na lista Tipo de evento de dados no CloudTrail console. A coluna de valor resources.type mostra o resources.type valor, que você especificaria ao configurar seletores de eventos avançados usando o ou. AWS CLI CloudTrail APIs A CloudTrail coluna Dados APIs registrados em mostra as chamadas de API registradas CloudTrail para o tipo de recurso.

Tipo de evento de dados (console)	valor resources.type	Dados APIs registrados em CloudTrail
AppSync GraphQL	AWS::AppSync::GraphQLApi	GraphQL

É possível configurar seletores de eventos avançados para filtrar os campos eventName, readOnly e resources.ARN para registrar em log somente os eventos que são importantes para você. Para obter mais informações sobre esses campos, consulte [AdvancedFieldSelector](#), na Referência de APIs do 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"
    ]
  }
]
}
]

```

Entendendo as entradas do arquivo de AWS AppSync log

CloudTrail entrega eventos como arquivos de log que contêm uma ou mais entradas de log. Um evento representa uma única solicitação de qualquer origem e inclui informações sobre a operação solicitada, a data e a hora em que ocorreram, os parâmetros da solicitação etc. Os arquivos de log não são um rastreamento de pilha ordenada de chamadas de API pública e, portanto, não são exibidos em uma ordem específica.

Note

`requestID` Não é uma ID exclusiva autorizada para registros emitidos de. AWS AppSync. Eles `requestID` podem ser substituídos pelo cliente. Portanto, você deve ter cuidado ao tomar decisões com base nessas informações.

O exemplo de entrada de CloudTrail registro a seguir demonstra a `CreateApiKey` operação.

```

{
  "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"
}
]
}

```

O exemplo de entrada de CloudTrail registro a seguir demonstra a `ListApiKeys` operação.

```

{
  "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"
  }
]
}

```

O exemplo de entrada de CloudTrail registro a seguir demonstra a DeleteApiKey operação.

```

{
  "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"
  }
]
}

```

O exemplo de entrada de CloudTrail registro a seguir demonstra uma mutação bem-sucedida do GraphQL autorizada com um autorizador de função Lambda personalizado.

```

{
  "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"
}

```

O exemplo de entrada de CloudTrail registro a seguir demonstra uma operação parcialmente bem-sucedida do GraphQL autorizada com um autorizador de função Lambda personalizado. Observe a `fieldAuthorizationResults.deniedFields` propriedade que especifica os campos negados.

```

{
  "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"
  }

```

O exemplo de entrada de CloudTrail registro a seguir demonstra uma falha na operação do GraphQL.

```

{
  "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"
}

```

O exemplo a seguir demonstra uma solicitação bem-sucedida do GraphQL.

```

{
  "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"
    }
  }
}

```

```

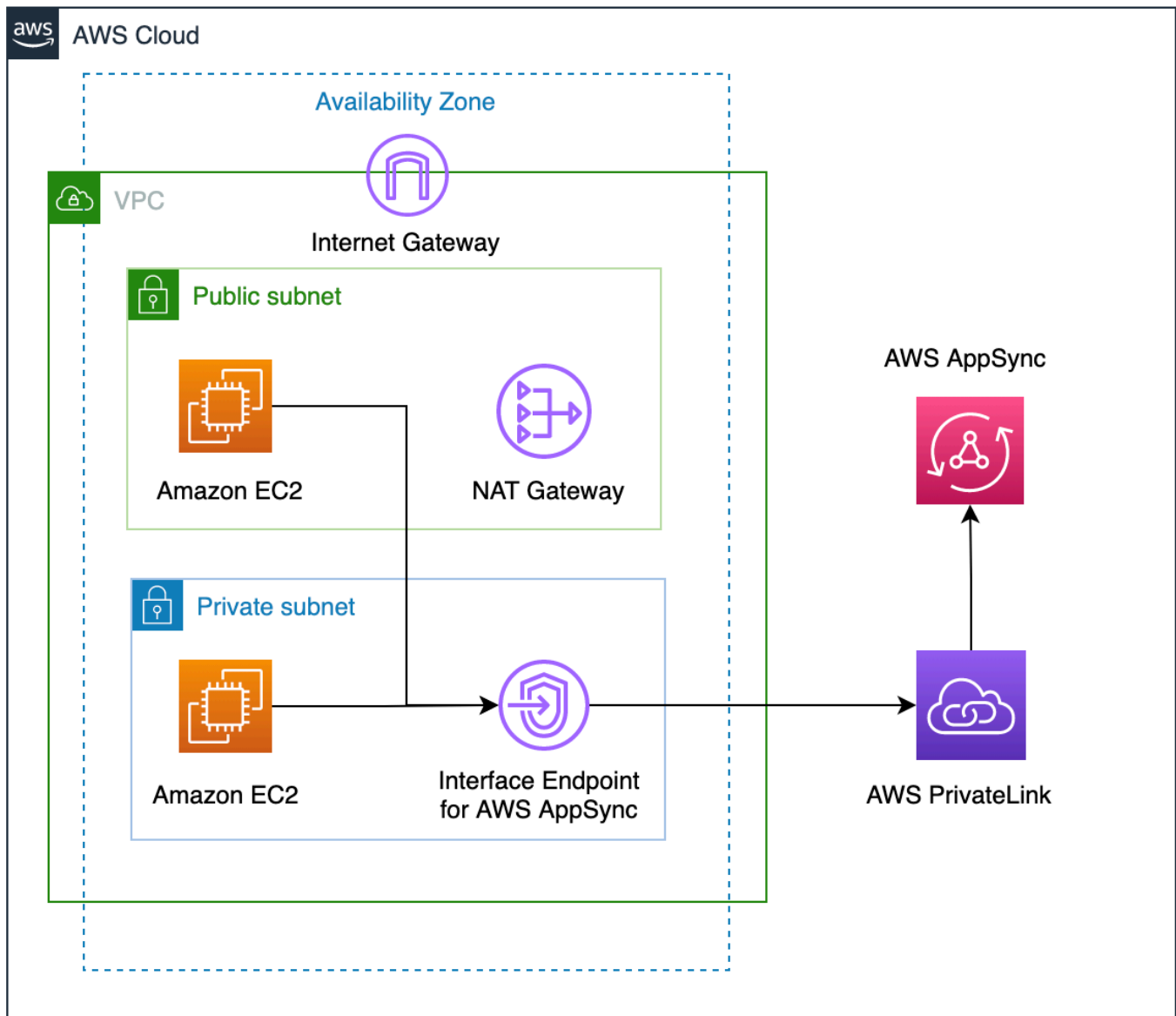
    }
  }
},
"eventTime": "2024-11-06T16:03:43Z",
"eventSource": "appsync.amazonaws.com",
"eventName": "GraphQL",
"awsRegion": "us-west-2",
"sourceIPAddress": "15.248.1.214",
"userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:131.0) Gecko/20100101
Firefox/131.0",
"requestParameters": null,
"responseElements": 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"
}

```

Usando o AWS AppSync Private APIs

Se você usa a Amazon Virtual Private Cloud (Amazon VPC), você pode criar AWS AppSync Private APIs, que só pode ser acessada de uma VPC. APIs Com uma API privada, você pode restringir o acesso da API aos seus aplicativos internos e conectar-se aos endpoints GraphQL e Realtime sem expor os dados publicamente.

Para estabelecer uma conexão privada entre sua VPC e o AWS AppSync serviço, você deve criar uma interface [VPC](#) endpoint. Os endpoints de interface são alimentados por [AWS PrivateLink](#), o que permite que você acesse de forma privada AWS AppSync APIs sem um gateway de internet, dispositivo NAT, conexão VPN ou conexão. AWS Direct Connect As instâncias em sua VPC não precisam de endereços IP públicos para se comunicar. AWS AppSync APIs Tráfego entre sua VPC e o tráfego AWS AppSync que não sai da AWS rede.



Há alguns fatores adicionais a serem considerados antes de ativar os atributos da API privada:

- Configurar endpoints de interface VPC AWS AppSync com recursos de DNS privado habilitados impedirá que recursos na VPC possam invocar outros AWS AppSync públicos usando o URL da API gerado. APIs AWS AppSync Isso se deve ao fato de a solicitação à API pública ser roteada por meio do endpoint da interface, o que não é permitido para o público. APIs Para invocar public APIs nesse cenário, é recomendável configurar nomes de domínio personalizados em public APIs, que podem ser usados por recursos na VPC para invocar a API pública.

- Seu AWS AppSync Private só APIs estará disponível em sua VPC. O editor de consultas do AWS AppSync console só poderá acessar sua API se a configuração de rede do seu navegador puder rotear o tráfego para sua VPC (por exemplo, conexão via VPN ou mais AWS Direct Connect).
- Com um endpoint de interface VPC para AWS AppSync, você pode acessar qualquer API privada na mesma AWS conta e região. Para restringir ainda mais o acesso ao Privado APIs, você pode considerar as seguintes opções:
 - Garantindo que somente os administradores necessários possam criar interfaces de endpoint VPC para. AWS AppSync
 - Usando políticas personalizadas de VPC endpoint para restringir o que APIs pode ser invocado a partir de recursos na VPC.
 - Para recursos na VPC, recomendamos que você use a autorização do IAM para invocar, AWS AppSync APIs garantindo que os recursos recebam funções com escopo reduzido para o. APIs
- Ao criar ou usar políticas que restringem as entidades principais do IAM, você deve definir o `authorizationType` do método como `AWS_IAM` ou `NONE`.

Criação de AWS AppSync dados privados APIs

As etapas a seguir mostram como criar Privado APIs no AWS AppSync serviço.

Warning

Você pode ativar os atributos da API privada somente durante a criação da API. Essa configuração não pode ser modificada em uma AWS AppSync API ou em uma API AWS AppSync privada após sua criação.

1. Faça login no AWS Management Console e abra o [AppSync console](#).
 - No Painel, escolha Criar API.
2. Escolha Criar uma API do zero e, em seguida, escolha Avançar.
3. Na seção API privada, escolha Usar atributos da API privada.
4. Configure o restante das opções, revise os dados da API e escolha Criar.

Antes de usar sua API AWS AppSync privada, você deve configurar um endpoint de interface para AWS AppSync sua VPC. Observe que tanto a API privada quanto a VPC devem estar na mesma AWS conta e região.

Criar um endpoint da interface para o AWS AppSync

Você pode criar um endpoint de interface para AWS AppSync usar o console Amazon VPC ou AWS Command Line Interface o AWS CLI(). Para obter mais informações, consulte [Criar um endpoint de interface](#) no Guia do usuário da Amazon VPC.

Console

1. Faça login AWS Management Console e abra a página [Endpoints](#) do console Amazon VPC.
2. Escolha Criar endpoint.
 - a. No campo Categoria de serviço, verifique se a opção Serviços da AWS está selecionada.
 - b. Na tabela Serviços, escolha `com.amazonaws.{region}.appsync-api`. Verifique se o valor da coluna Tipo é Interface.
 - c. No campo VPC, escolha uma VPC e suas sub-redes.
 - d. Para habilitar os atributos do DNS privado para o endpoint da interface, marque a caixa de seleção Habilitar nome de DNS.
 - e. Em Grupo de segurança, selecione um ou mais grupos de segurança.
3. Escolha Criar endpoint.

CLI

Use o comando [create-vpc-endpoint](#) e especifique o ID da VPC, o tipo de endpoint da VPC (interface), o nome do serviço, as sub-redes que usarão o endpoint e os grupos de segurança associados às interfaces de rede do endpoint. Por exemplo:

```
$ aws ec2 create-vpc-endpoint --vpc-id vpc-ec43eb89 \
--vpc-endpoint-type Interface \
--service-name com.amazonaws.{region}.appsync-api \
--subnet-id subnet-abababab --security-group-id sg-1a2b3c4d
```

Para usar a opção de DNS privado, defina os valores `enableDnsHostnames` e `enableDnsSupportattributes` da VPC. Para obter mais informações, consulte [Ver e atualizar o suporte do DNS para a VPC](#) no Manual do usuário da Amazon VPC. Se você habilitar recursos de DNS privados para o endpoint da interface, poderá fazer solicitações à sua API AWS AppSync GraphQL e ao endpoint em tempo real usando seus endpoints DNS públicos padrão usando o formato abaixo:

```
https://{api_url_identifíer}.appsync-api.{region}.amazonaws.com/graphql
```

Para obter mais informações sobre endpoints de serviço, consulte [Endpoints de serviço e cotas](#) na Referência geral do AWS .

Para obter mais informações sobre interações de serviço com endpoints da interface, consulte [Acessar um serviço por um endpoint de interface](#) no Guia do usuário da Amazon VPC.

Para obter informações sobre como criar e configurar um endpoint usando AWS CloudFormation, consulte o VPC endpoint recurso [AWS::EC2::](#) no Guia do AWS CloudFormation usuário.

Exemplos avançados

Se você habilitar recursos de DNS privados para o endpoint da interface, poderá fazer solicitações à sua API AWS AppSync GraphQL e ao endpoint em tempo real usando seus endpoints DNS públicos padrão usando o formato abaixo:

```
https://{api_url_identifíer}.appsync-api.{region}.amazonaws.com/graphql
```

Usando os nomes de host de DNS público do endpoint da VPC da interface, o URL de base para invocar a API estará neste formato:

```
https://{vpc_endpoint_id}-{endpoint_dns_identifíer}.appsync-api.  
{region}.vpce.amazonaws.com/graphql
```

Você também pode usar o nome de host de DNS específico do AZ se tiver implantado um endpoint no AZ:

```
https://{vpc_endpoint_id}-{endpoint_dns_identifíer}-{az_id}.appsync-api.  
{region}.vpce.amazonaws.com/graphql.
```


Usar o nome DNS público do VPC endpoint exigirá que o nome do host do endpoint da AWS AppSync API seja passado como cabeçalho para a Host solicitação. `x-appsync-domain` Esses exemplos usam um TodoAPI que foi criado no guia [Iniciar esquema de amostra](#):

```
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-{xxxxxxxxxxxxxxxxxxxxxxxxxxxx}" \
-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"}}}'
```

Nos exemplos a seguir, usaremos o aplicativo Todo que é gerado no guia [Iniciar esquema de amostra](#). Para testar a amostra da API Todo, usaremos o DNS privado para invocar a API. Você pode usar qualquer ferramenta de linha de comando de sua escolha; este exemplo usa [curl](#) para enviar consultas e mutações e [wscat](#) para configurar assinaturas. Para emular nosso exemplo, substitua os valores entre colchetes `{ }` nos comandos abaixo pelos valores correspondentes da sua AWS conta.

Operação de mutação de teste — Solicitação do **createTodo**

```
curl https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql \
-H "Content-Type:application/graphql" \
-H "x-api-key:da2-{xxxxxxxxxxxxxxxxxxxxxxxxxxxx}" \
-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"}}}'
```

Operação de mutação de teste — Resposta do **createTodo**

```
{
  "data": {
    "createTodo": {
      "id": "<todo-id>",
      "name": "My first GraphQL task",
      "where": "Day 1",
      "when": "Friday Night",
      "description": "Learn more about GraphQL"
```

```

    }
  }
}
```

Operação de consulta de teste — Solicitação do **listTodos**

```

curl https://{api_url_identifider}.appsync-api.{region}.amazonaws.com/graphql \
-H "Content-Type:application/graphql" \
-H "x-api-key:da2-{xxxxxxxxxxxxxxxxxxxxxxxxxxxx}" \
-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\n GraphQL task","when":"Friday Night","where":"Day 1","description":"Learn more about\n GraphQL"}}}'
```

Operação de consulta de teste — Solicitação do **listTodos**

```

{
  "data": {
    "listTodos": {
      "items": [
        {
          "description": "Learn more about GraphQL",
          "id": "<todo-id>",
          "name": "My first GraphQL task",
          "when": "Friday night",
          "where": "Day 1"
        }
      ]
    }
  }
}
```

Operação de assinatura de teste — Assinatura para a mutação do **createTodo**

Para configurar assinaturas do GraphQL AWS AppSync, consulte [Criação](#) de um cliente em tempo real. WebSocket [Em uma EC2 instância da Amazon em uma VPC, você pode testar seu endpoint de assinatura de API AWS AppSync privada usando o wscat](#). O exemplo abaixo usa um API KEY para autorização.

```

$ header=`echo '{"host":"{api_url_identifider}.appsync-api.{region}.amazonaws.com","x-api-key":"da2-{xxxxxxxxxxxxxxxxxxxxxxxxxxxx}"}' | 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-xxxxxxxxxxxxxxxxxxxxxxxxxxxx"}, "host": "{api_url_identifier}.appsync-api.{region}.amazonaws.com"}}}, {"type": "start"}
< {"id": "f7a49717", "type": "start_ack"}
```

Como alternativa, use o nome de domínio do endpoint da VPC e, ao mesmo tempo, certifique-se de especificar o cabeçalho Host no comando `wscat` para estabelecer o websocket:

```
$ header=`echo '{"host": "{api_url_identifier}.appsync-api.{region}.amazonaws.com", "x-api-key": "da2-xxxxxxxxxxxxxxxxxxxxxxxxxxxx"}' | 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-xxxxxxxxxxxxxxxxxxxxxxxxxxxx"}, "host": "{api_url_identifier}.appsync-api.{region}.amazonaws.com"}}}, {"type": "start"}
< {"id": "f7a49717", "type": "start_ack"}
```

Execute o código de mutação abaixo:

```
curl https://{api_url_identifier}.appsync-api.{region}.amazonaws.com/graphql \
-H "Content-Type:application/graphql" \
-H "x-api-key:da2-xxxxxxxxxxxxxxxxxxxxxxxxxxxx" \
-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"}}}'
```

Depois disso, uma assinatura é acionada, e a mensagem de notificação aparece conforme mostrado abaixo:

```
< {"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"}}}}
```

Usar políticas do IAM para limitar a criação de API pública

AWS AppSync oferece suporte a [Condições de declarações](#) do IAM para uso com Private APIs.

O `visibility` campo pode ser incluído nas declarações de política do IAM para que a `appsync:CreateGraphqlApi` operação controle quais funções e usuários do IAM podem criar privados e públicos APIs. Com isso, o administrador do IAM pode definir uma política do IAM que só permitirá que um usuário crie uma API privada do GraphQL. Um usuário que tentar criar uma API pública receberá uma mensagem informando que a operação não é autorizada.

Por exemplo, um administrador do IAM poderia criar a seguinte declaração de política do IAM para permitir a criação de Private APIs:

```
{
  "Sid": "AllowPrivateAppSyncApis",
  "Effect": "Allow",
  "Action": "appsync:CreateGraphqlApi",
  "Resource": "*",
  "Condition": {
    "ForAnyValue:StringEquals": {
      "appsync:Visibility": "PRIVATE"
    }
  }
}
```

Um administrador do IAM também pode adicionar a seguinte [política de controle de serviços](#) para impedir que todos os usuários de uma AWS organização criem algo AWS AppSync APIs diferente de Privado APIs:

```
{
  "Sid": "BlockNonPrivateAppSyncApis",
  "Effect": "Deny",
  "Action": "appsync:CreateGraphqlApi",
  "Resource": "*",

```

```
"Condition": {
  "ForAnyValue:StringNotEquals": {
    "appsync:Visibility": "PRIVATE"
  }
}
```

Compartilhando o AWS AppSync GraphQL APIs

AWS AppSync integra-se com AWS Resource Access Manager (AWS RAM) para permitir o compartilhamento de recursos. AWS RAM é um serviço que permite que você compartilhe ações de invocação (operações de consulta, mutação e assinatura e conecte solicitações ao seu WebSocket endpoint em tempo real) no GraphQL com outros ou por AWS AppSync meio APIs dele. Contas da AWS AWS Organizations Com AWS RAM, você compartilha recursos de sua propriedade criando um compartilhamento de recursos. Um compartilhamento de atributos especifica os atributos a serem compartilhados, e os consumidores com os quais compartilhá-los. Os consumidores podem incluir o seguinte.

- Específico Contas da AWS dentro ou fora de sua organização em AWS Organizations
- Uma unidade organizacional dentro de sua organização em AWS Organizations
- Uma organização inteira em AWS Organizations

Para obter mais informações sobre AWS RAM, consulte o [Guia AWS Resource Access Manager do usuário](#).

Tópicos

- [Pré-requisitos para compartilhar o GraphQL AWS AppSync APIs](#)
- [Compartilhe AWS AppSync GraphQL APIs](#)
- [Pare de compartilhar o AWS AppSync GraphQL APIs](#)
- [Eventos entre contas](#)

Pré-requisitos para compartilhar o GraphQL AWS AppSync APIs

O compartilhamento do AWS AppSync GraphQL APIs tem os seguintes pré-requisitos.

- Para compartilhar uma API do AWS AppSync GraphQL, você deve possuí-la em sua Conta da AWS. Isso significa que a API AWS AppSync GraphQL deve ser alocada ou provisionada em sua conta.
- Para compartilhar uma API do AWS AppSync GraphQL com sua organização ou unidade organizacional em AWS Organizations, você deve habilitar o compartilhamento com AWS Organizations. Para obter mais informações, consulte [Habilitar o compartilhamento de recursos no AWS Organizations](#) no Guia do usuário do AWS Resource Access Manager.

Compartilhe AWS AppSync GraphQL APIs

Para compartilhar uma API do AWS AppSync GraphQL, comece criando um compartilhamento de recursos usando o AWS Resource Access Manager. Um compartilhamento de recursos especifica os recursos a serem compartilhados, os consumidores com os quais compartilhá-los e quais ações as entidades principais poderão executar. Quando você compartilha uma API AWS AppSync GraphQL que você possui, com outras Contas da AWS, você permite que essas contas chamem essa AWS AppSync API no seu. Conta da AWS.

Se você faz parte de uma organização e o compartilhamento dentro da sua organização está ativado, os consumidores da sua organização recebem automaticamente acesso ao recurso compartilhado. AWS Organizations. Caso contrário, os consumidores receberão um convite para participar do compartilhamento de recursos e acesso ao recurso compartilhado depois de aceitar o convite.

Compartilhando considerações

- Você pode compartilhar somente o AWS AppSync GraphQL APIs, não outros tipos de API, como Event. APIs.
- Você pode compartilhar somente o AWS AppSync GraphQL APIs que tenha AWS_IAM como um dos modos de autorização configurados na API.

Se AWS_IAM for removido da lista de modos de autorização para uma API AppSync GraphQL compartilhada, embora o compartilhamento de recursos ainda exista, ele se tornará ineficaz.

- Você pode compartilhar o AWS AppSync APIs GraphQL público e privado.
- O AWS AppSync GraphQL privado sempre APIs pode ser acessado por meio de VPC endpoints VPCs na origem Conta da AWS, e todos os modos de autorização são suportados, não apenas AWS_IAM.

- Para o AWS AppSync GraphQL compartilhado APIs, as permissões são gerenciadas somente para o recurso da API e não oferecem suporte a permissões refinadas para campo, tipo e recursos de campo. Ao compartilhar uma API, você está compartilhando o ARN da API e o de todos ARNs os seus tipos e campos.

Crie um compartilhamento de recursos de sua propriedade usando o AWS RAM console

Para compartilhar uma API do AWS AppSync GraphQL, use o procedimento descrito em [Criação de um compartilhamento de recursos](#) no Guia do AWS Resource Access Manager usuário, usando o RAM nome da permissão `AWSRAMPermissionAppSyncGraphQLApiInvokeAccess`.

Crie e use uma permissão gerenciada pelo cliente para compartilhar uma API AWS AppSync GraphQL privada usando o console AWS RAM

Para compartilhar uma API AWS AppSync GraphQL privada, crie uma permissão gerenciada pelo cliente usando o procedimento descrito em [Criação e uso de permissões gerenciadas pelo cliente](#) no Guia do AWS Resource Access Manager usuário.

Por exemplo, um proprietário da Conta A deseja conceder aos diretores da Conta B permissão para acessar uma API AWS AppSync GraphQL (A) privada para chamadas feitas via VPCE-B PrivateApi (um VPC Endpoint de propriedade da Conta B). Nesse caso, o proprietário da Conta A precisa criar uma permissão gerenciada pelo AWS RAM cliente da seguinte maneira.

```
{
  "Effect": "Allow",
  "Action": ["appsync:GraphQL"],
  "Condition": {
    "StringEqualsIgnoreCase": {
      "aws:SourceVpce": [
        "VPCE-B"
      ]
    }
  }
}
```

Suponha que essa nova AWS RAM permissão gerenciada pelo cliente tenha um nome `private-api-A-access-via-vpce-b`.

Para permitir o acesso entre contas ao PrivateApiA viaVPCE-B, o cliente pode criar um compartilhamento de AWS RAM recursos com os seguintes parâmetros e a permissão gerenciada pelo cliente no exemplo anterior.

- Tipo de recurso: `appsync:Apis`
- Recurso: `arn:aws:appsync:us-west-2:A:apis/PrivateApiA`
- Permissão: **private-api-A-access-via-vpce-b** (permissão gerenciada pelo cliente)
- Diretor: Account: B

Crie um compartilhamento de recursos de sua propriedade usando o AWS CLI

Para compartilhar uma API do AWS AppSync GraphQL usando o AWS CLI, use o `create-resource-share` comando with `arn:aws:ram::aws:permission/AWSRAMPermissionAppSyncApiInvokeAccess` como valor para o `--permission-arns` switch.

Para obter uma lista completa dos comandos disponíveis para AWS RAM, consulte a referência da [AWS RAM CLI](#).

Pare de compartilhar o AWS AppSync GraphQL APIs

Para parar de compartilhar o AWS AppSync GraphQL APIs que você possui, você deve excluir o compartilhamento de recursos ou atualizar os principais com os quais você compartilhou o recurso. Consulte a documentação nas seções a seguir para saber a ação que você deseja realizar.

Para parar de compartilhar um recurso que você possui usando o AWS RAM console

Consulte [Atualizar um compartilhamento de recursos](#) no Guia do usuário do AWS Resource Access Manager .

Para parar de compartilhar um recurso que você possui usando o AWS CLI

Use o comando [disassociate-resource-share](#).

Para excluir um compartilhamento de recursos que você possui usando o AWS RAM console

Consulte [Excluir um compartilhamento de recursos](#) no Guia do AWS Resource Access Manager usuário.

Para excluir um compartilhamento de recursos que você possui usando o AWS CLI

Use o comando [delete-resource-share](#).

Para obter uma lista completa dos comandos disponíveis para AWS RAM, consulte a referência da [AWS RAM CLI](#).

Eventos entre contas

Você pode optar por registrar eventos de AWS CloudTrail dados para monitorar e auditar a atividade da API AWS AppSync GraphQL em várias contas. DataPlane Para obter mais informações, consulte [Registrar eventos de dados](#), no Guia do usuário do AWS CloudTrail .

Configurar a complexidade de execução, a profundidade das consultas e a introspecção do GraphQL com o AWS AppSync

AWS AppSync permite ativar ou desativar os recursos de introspecção e definir limites para a quantidade de níveis e resolvedores aninhados em uma única consulta.

Usar o recurso de introspecção

Tip

Para obter mais informações sobre introspecção no GraphQL, consulte este artigo no [site de base do GraphQL](#).

Por padrão, o GraphQL permite utilizar a introspecção para consultar o próprio esquema e descobrir tipos, campos, consultas, mutações, assinaturas, etc. É um recurso importante para aprender como os dados são moldados e processados pelo serviço GraphQL. No entanto, há alguns fatores a serem considerados ao lidar com introspecção. É possível ter um caso de uso que se beneficiaria com a desativação da introspecção, como um caso em que os nomes dos campos podem ser confidenciais ou ocultos ou em que o esquema completo da API deve ficar não documentado para os consumidores. Nesses casos, a publicação de dados do esquema por meio de introspecção pode ocasionar o vazamento de dados intencionalmente privados.

Para evitar que isso aconteça, é possível desabilitar a introspecção. Isso evitará que pessoas não autorizadas usem campos de introspecção no esquema. No entanto, é importante observar que a introspecção é útil para que as equipes de desenvolvimento aprendam como os dados nos serviços são processados. Internamente, pode ser útil manter a introspecção habilitada e, ao mesmo tempo, desabilitá-la no código de produção como uma camada extra de segurança. Outra maneira de lidar

com isso é adicionar um método de autorização, que AWS AppSync também fornece. Para ter mais informações, consulte [authorization](#).

AWS AppSync permite que você ative ou desative a introspecção no nível da API. Para habilitar ou desabilitar a introspecção, faça o seguinte:

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
2. Na APIspágina, escolha o nome de uma API do GraphQL.
3. Na página inicial da sua API, no painel de navegação, selecione Configurações.
4. Em Configurações da API, selecione Editar.
5. Em Consultas de introspecção, faça o seguinte:
 - Ative ou desative Habilitar consultas de introspecção.
6. Escolha Salvar.

Quando a introspecção estiver habilitada (o comportamento padrão), o uso do sistema de introspecção funcionará normalmente. Por exemplo, a imagem abaixo mostra um campo `__schema` processando todos os tipos disponíveis no esquema:



Ao desabilitar esse recurso, um erro de validação aparecerá na resposta:



Configurar limites de profundidade de consultas

Há momentos em que convém ter um controle mais granular sobre como a API funciona durante uma operação. Um desses controles é adicionar um limite à quantidade de níveis aninhados que uma consulta pode processar. Por padrão, as consultas podem processar uma quantidade ilimitada de níveis aninhados. Limitar as consultas a uma quantidade específica de níveis aninhados tem implicações potenciais para a performance e a flexibilidade do projeto. Considere a seguinte consulta:

```
query MyQuery {
  L1: nextLayer {
    L2: nextLayer {
      L3: nextLayer {
        L4: value
      }
    }
  }
}
```

O projeto pode exigir a limitação de consultas a L1 ou L2 para alguma finalidade. Por padrão, toda a consulta de L1 a L4 é processada sem nenhuma forma de controlar isso. Ao definir um limite, é possível impedir que as consultas acessem qualquer item além do nível especificado.

Para adicionar um limite de profundidade de consultas, faça o seguinte:

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).

2. Na API página, escolha o nome de uma API do GraphQL.
3. Na página inicial da sua API, no painel de navegação, selecione Configurações.
4. Em Configurações da API, selecione Editar.
5. Em Profundidade da consulta, faça o seguinte:
 - a. Ative ou desative Habilitar profundidade de consulta.
 - b. Em Profundidade máxima, defina o limite de profundidade. Pode ser entre 1 e 75.
6. Escolha Salvar.

Quando um limite for definido, ultrapassar o limite máximo vai gerar um erro `QueryDepthLimitReached`. Por exemplo, a imagem abaixo mostra uma consulta com um limite de profundidade de 2 que ultrapassa o limite até o terceiro (L3) e o quarto (L4) níveis:



Observe que os campos ainda podem ser marcados como anuláveis ou não anuláveis no esquema. Se um campo não anulável receber um erro `QueryDepthLimitReached`, esse erro será lançado para o primeiro campo principal anulável.

Configurar limites de contagem de resolvedores

Também é possível controlar quantos resolvedores cada consulta pode processar. Assim como a profundidade da consulta, é possível definir um limite para esse valor. Faça a seguinte consulta que contém três resolvedores:

```
query MyQuery {
  resolver1: resolver
```

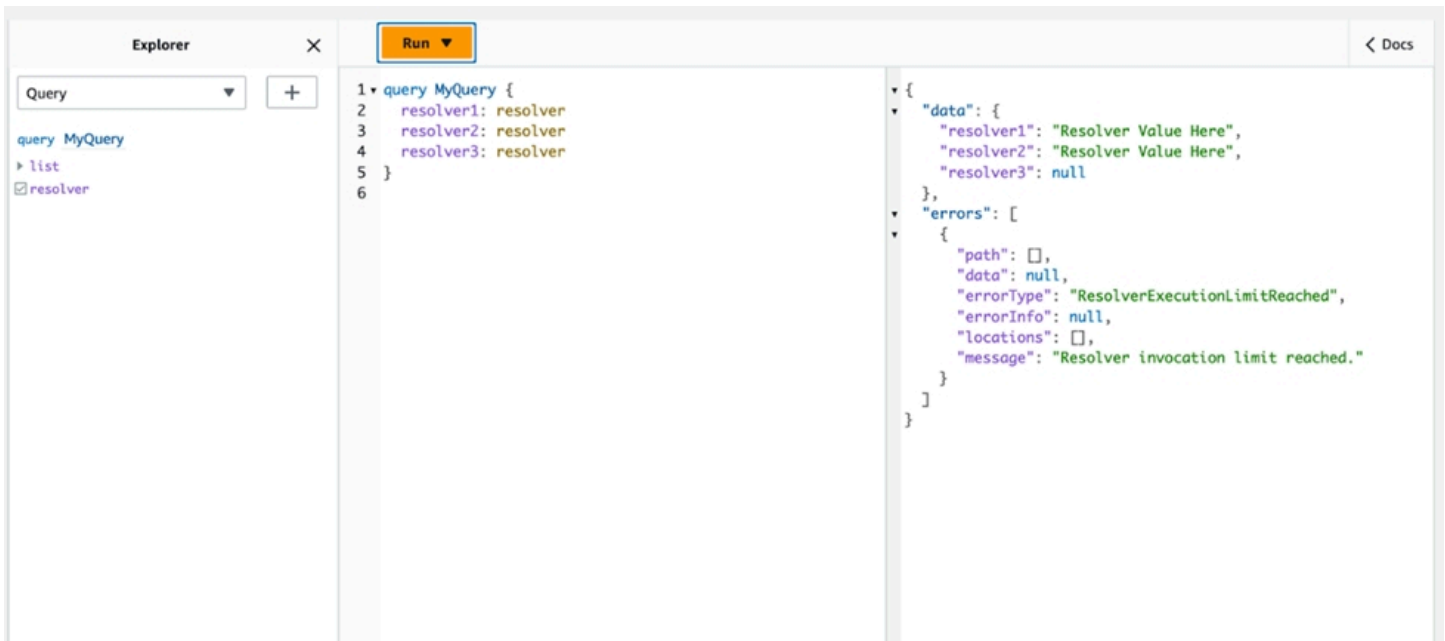
```
resolver2: resolver
resolver3: resolver
}
```

Por padrão, cada consulta pode processar até 10 mil resolvedores. No exemplo acima, `resolver1`, `resolver2` e `resolver3` serão processados. No entanto, o projeto pode exigir a limitação de cada consulta ao tratamento de um ou dois resolvedores no total. Ao definir um limite, é possível fazer com que a consulta não processe nenhum resolvedor além de um número específico, como os primeiros (`resolver1`) ou os segundos (`resolver2`) resolvedores.

Para adicionar um limite de contagem de resolvedores, faça o seguinte:

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
2. Na API página, escolha o nome de uma API do GraphQL.
3. Na página inicial da sua API, no painel de navegação, selecione Configurações.
4. Em Configurações da API, selecione Editar.
5. Em Limite de contagem de resolvedores, faça o seguinte:
 - a. Ative Habilitar contagem de resolvedores.
 - b. Em Contagem máxima de resolvedores, defina o limite de contagem. Pode ser entre 1 e 10000.
6. Escolha Salvar.

Assim como o limite de profundidade da consulta, exceder o limite configurado de resolvedores faz com que a consulta termine com um erro `ResolverExecutionLimitReached` em resolvedores adicionais. Na imagem abaixo, uma consulta com um limite de contagem de resolvedores de 2 tenta processar três resolvedores. Por causa do limite, o terceiro resolvedor gera um erro e não é executado.



Usando variáveis de ambiente em AWS AppSync

Você pode usar variáveis de ambiente para ajustar o comportamento de seus AWS AppSync resolvedores e funções sem atualizar seu código. As variáveis de ambiente são pares de strings armazenadas com sua configuração de API que são disponibilizadas para seus resolvedores e funções para serem aproveitadas em runtime. Elas são particularmente úteis para situações em que você precisa consultar dados de configuração que só estão disponíveis durante a configuração inicial, mas precisam ser usados por seus resolvedores e funções durante a execução. As variáveis de ambiente expõem os dados de configuração em seu código, reduzindo assim a necessidade de codificar esses valores.

Note

Para aumentar a segurança do banco de dados, recomendamos que você use o [Secrets Manager](#) ou o [AWS Systems Manager Parameter Store](#), em vez de variáveis de ambiente, para armazenar credenciais ou informações confidenciais. Para aproveitar esse recurso, consulte Como [invocar AWS serviços com fontes de dados AWS AppSync HTTP](#).

As variáveis de ambiente devem seguir vários comportamentos e regras para funcionar adequadamente:

- Tanto os JavaScript resolvedores/funções quanto os modelos de VTL oferecem suporte a variáveis de ambiente.
- As variáveis de ambiente não são avaliadas antes da invocação da função.
- As variáveis de ambiente só oferecem suporte a valores de string.
- Qualquer valor definido em uma variável de ambiente é considerado um literal de string e não expandido.
- O ideal é que as avaliações de variáveis sejam realizadas no código da função.

Configurar variáveis de ambiente (console)

Você pode configurar variáveis de ambiente para sua API AWS AppSync GraphQL criando a variável e definindo seu par de valores-chave. Seus resolvedores e funções usarão o nome da chave da variável de ambiente para recuperar o valor em runtime. Para definir variáveis de ambiente no AWS AppSync console:

1. Faça login no AWS Management Console e abra o [AppSyncconsole](#).
2. Na APIspágina, escolha o nome de uma API do GraphQL.
3. Na página inicial da sua API, no painel de navegação, selecione Configurações.
4. Em Variáveis de ambiente, escolha Adicionar variáveis de ambiente.
5. Escolha Add environment variable (Adicionar variável de ambiente).
6. Insira um par de chave e valor.
7. Se necessário, repita as etapas 5 e 6 para adicionar mais valores de chave. Se precisar remover um valor de chave, escolha a opção Remove e a chave a ser removida.
8. Selecione Enviar.

Tip

Há algumas regras que você deve seguir ao criar chaves e valores:

- As chaves devem começar com uma letra.
- As chaves devem conter pelo menos dois caracteres.
- As chaves só podem conter letras, números e o caractere de sublinhado (_).
- Os valores podem conter até 512 caracteres.

- Você pode configurar até 50 pares de chave-valor em uma API GraphQL.

Configurar variáveis de ambiente (API)

Para definir uma variável de ambiente usando APIs, você pode usar `PutGraphqlApiEnvironmentVariables`. O comando CLI correspondente é `put-graphql-api-environment-variables`.

Para recuperar uma variável de ambiente usando APIs, você pode usar `GetGraphqlApiEnvironmentVariables`. O comando CLI correspondente é `get-graphql-api-environment-variables`.

O comando deve conter o ID da API e a lista de variáveis de ambiente:

```
aws appsync put-graphql-api-environment-variables \  
  --api-id "<api-id>" \  
  --environment-variables '{"key1":"value1","key2":"value2", ...}'
```

O exemplo a seguir define duas variáveis de ambiente em uma API com o ID `abcdefghijklmnopqrstuvwxyz` usando o comando `put-graphql-api-environment-variables`:

```
aws appsync put-graphql-api-environment-variables \  
  --api-id "abcdefghijklmnopqrstuvwxyz" \  
  --environment-variables '{"USER_TABLE":"users_prod","DEBUG":"true}"
```

Observe que, quando você aplica variáveis de ambiente com o comando `put-graphql-api-environment-variables`, o conteúdo da estrutura das variáveis de ambiente é sobrescrito; isso significa que as variáveis de ambiente existentes serão perdidas. Para reter variáveis de ambiente existentes ao adicionar outras novas, inclua todos os pares de chave-valor existentes junto com os novos em sua solicitação. Usando o exemplo acima, se quiser adicionar `"EMPTY": ""`, você pode fazer o seguinte:

```
aws appsync put-graphql-api-environment-variables \  
  --api-id "abcdefghijklmnopqrstuvwxyz" \  
  --environment-variables '{"USER_TABLE":"users_prod","DEBUG":"true", "EMPTY":""}'
```


Para recuperar a configuração atual, use o comando `get-graphql-api-environment-variables`:

```
aws appsync get-graphql-api-environment-variables --api-id "<api-id>"
```

Usando o exemplo acima, você poderia usar o comando a seguir:

```
aws appsync get-graphql-api-environment-variables --api-id "abcdefghijklmnpqrstuvwxzyz"
```

O resultado mostrará a lista de variáveis de ambiente junto com seus valores de chave:

```
{
  "environmentVariables": {
    "USER_TABLE": "users_prod",
    "DEBUG": "true",
    "EMPTY": ""
  }
}
```

Configuração de variáveis de ambiente (CFN)

Você pode usar o modelo abaixo para criar variáveis de ambiente:

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  GraphQLApiWithEnvVariables:
    Type: "AWS::AppSync::GraphQLApi"
    Properties:
      Name: "MyApiWithEnvVars"
      AuthenticationType: "AWS_IAM"
      EnvironmentVariables:
        EnvKey1: "non-empty"
        EnvKey2: ""
```

variáveis de ambiente e mescladas APIs

As variáveis de ambiente definidas em Source também APIs estão disponíveis em seu Merged APIs. As variáveis de ambiente no Merged APIs são somente para leitura e não podem ser atualizadas. Observe que suas chaves variáveis de ambiente devem ser exclusivas em todo o Source APIs para

que suas mensagens sejam bem-sucedidas; chaves duplicadas sempre resultarão em uma falha na mesclagem.

Recuperar variáveis de ambiente

Para recuperar variáveis de ambiente em seu código de função, recupere o valor do `ctx.env` objeto em seus resolvedores e funções. Veja a seguir alguns exemplos.

Publishing to Amazon SNS

Neste exemplo, nosso resolvedor de HTTP envia uma mensagem para um tópico do Amazon SNS. O ARN do tópico só é conhecido depois que a pilha que define a API GraphQL e o tópico são implantados.

```
/**
 * 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

Neste exemplo, os nomes da tabela do DynamoDB serão diferentes se a API for implantada para preparação ou se já estiver em produção. O código do resolvedor não precisa mudar. Os valores das variáveis de ambiente são atualizados com base no local onde a API é implantada.

```
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
    },
  ],
};
}
```

Configurando autorização e autenticação para proteger seu GraphQL APIs

AWS AppSync oferece os seguintes tipos de autorização para proteger o GraphQL APIs: chaves de API, Lambda, IAM, OpenID Connect e grupos de usuários do Cognito. Cada opção fornece um método diferente de segurança:

1. Autorização de chave de API: controla a limitação de dados não autenticados APIs, fornecendo uma opção de segurança simples.
2. Autorização do Lambda: permite uma lógica de autorização personalizada, explicando as entradas e saídas da função em detalhes.
3. Autorização do IAM: utiliza AWS o processo de assinatura da versão 4 da assinatura, permitindo um controle de acesso refinado por meio de políticas do IAM.
4. Autorização do OpenID Connect: integra-se aos serviços compatíveis com OIDC para autenticação do usuário.
5. Grupos de usuários do Cognito: implementa o controle de acesso baseado em grupo usando os atributos de gerenciamento de usuários do Cognito.

Tipos de autorização

Há cinco maneiras de autorizar aplicativos a interagir com sua API AWS AppSync GraphQL. Você especifica qual tipo de autorização você usa especificando um dos seguintes valores de tipo de autorização em sua chamada de AWS AppSync API ou CLI:

- **API_KEY**

Para usar chaves da API.

- **AWS_LAMBDA**

Para usar uma AWS Lambda função.

- **AWS_IAM**

Para usar permissões AWS Identity and Access Management ([IAM](#)).

- **OPENID_CONNECT**

Para usar o provedor OpenID Connect.

- **AMAZON_COGNITO_USER_POOLS**

Para usar um grupo de usuários do Amazon Cognito.

Esses tipos de autorização básicos funcionam para a maioria dos desenvolvedores. Para casos de uso mais avançados, é possível adicionar modos de autorização adicionais por meio do console, da CLI e do AWS CloudFormation. Para modos de autorização adicionais, AWS AppSync fornece um tipo de autorização que usa os valores listados acima (ou seja, `API_KEY`, `AWS_LAMBDA`, `AWS_IAM`, `OPENID_CONNECT`, e `AMAZON_COGNITO_USER_POOLS`).

Ao especificar `API_KEY`, `AWS_LAMBDA` ou `AWS_IAM` como o tipo de autorização principal ou padrão, não é possível especificá-los novamente como um dos modos de autorização adicionais. Da mesma forma, você não pode duplicar `API_KEY`, `AWS_LAMBDA` ou `AWS_IAM` nos modos de autorização adicionais. É possível usar vários grupos de usuários do Amazon Cognito e provedores do OpenID Connect. No entanto, você não pode usar grupos de usuários do Amazon Cognito nem provedores do OpenID Connect duplicados entre o modo de autorização padrão e qualquer um dos outros modos de autorização. É possível especificar clientes diferentes para o grupo de usuários do Amazon Cognito ou o provedor do OpenID Connect usando a expressão regular de configuração correspondente.

Quando você salva as alterações na configuração da sua API, AWS AppSync começa a propagar as alterações. Até que sua alteração de configuração seja propagada, AWS AppSync continue veiculando seu conteúdo da configuração anterior. Depois que sua alteração de configuração for propagada, AWS AppSync imediatamente começará a veicular seu conteúdo com base na nova configuração. Enquanto AWS AppSync propagamos suas alterações para uma API, não podemos determinar se a API está veiculando seu conteúdo com base na configuração anterior ou na nova configuração.

API_KEY autorização

Os não autenticados APIs exigem uma limitação mais rígida do que os autenticados. APIs Uma maneira de verificar o controle de utilização para endpoints GraphQL não autenticados é por meio do uso de chaves da API. Uma chave da API é um valor codificado no aplicativo que é gerado pelo serviço AWS AppSync ao criar um endpoint GraphQL não autenticado. Você pode rotacionar as chaves de API no console, da CLI ou da [Referência da API do AWS AppSync](#).

Console

1. Faça login no AWS Management Console e abra o [AppSync console](#).
 - a. No APIs painel, escolha sua API GraphQL.
 - b. Na barra lateral, escolha Configurações.
2. Em Modo de autorização padrão, escolha Chave de API.
3. Na tabela chaves de API, escolha Adicionar chave de API.

Uma nova chave de API será gerada na tabela.

- Para excluir uma chave de API antiga, selecione a chave de API na tabela e escolha Excluir.
4. Escolha Salvar na parte inferior da página.

CLI

1. Se você ainda não tiver feito isso, configure seu acesso à AWS CLI. Para obter mais informações, consulte [Noções básicas de configuração](#).
2. Crie um objeto da API GraphQL executando o comando [update-graphql-api](#).

Você precisará digitar dois parâmetros para esse comando específico:

1. O `api-id` da sua API GraphQL.
2. O novo nome da sua API. Você pode usar o mesmo nome.
3. O `authentication-type`, que será `API_KEY`.

Note

Existem outros parâmetros, como `Region`, que devem ser configurados, mas geralmente usam como padrão os valores de configuração da CLI.

Veja um exemplo de comando:

```
aws appsync update-graphql-api --api-id abcdefghijklmnopqrstuvwxyz --name
TestAPI --authentication-type API_KEY
```

Uma saída será retornada na CLI. Aqui está um exemplo em 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"
  }
}
```

As chaves da API são configuráveis para até 365 dias e é possível estender uma data de expiração existente para mais 365 dias a partir dessa data. As chaves da API são recomendadas para fins de desenvolvimento ou casos de uso em que é seguro expor uma API pública.

No cliente, a chave da API é especificada pelo cabeçalho `x-api-key`.

Por exemplo, se o `API_KEY` for `'ABC123'`, envie uma consulta do GraphQL via `curl` da seguinte forma:

```
$ curl -XPOST -H "Content-Type:application/graphql" -H "x-api-key:ABC123" -d '{ "query": "query { movies { id } }" }' https://YOURAPPSYNCENDPOINT/graphql
```

AWS_LAMBDA autorização

Você pode implementar sua própria lógica de autorização de API usando uma AWS Lambda função. Você pode usar uma função do Lambda para seu autorizador primário ou secundário, mas pode haver somente uma função de autorização do Lambda por API. Ao usar funções do Lambda para autorização, o seguinte se aplica:

- Se a API tiver os modos de autorização AWS_IAM e AWS_LAMBDA habilitados, a assinatura do SigV4 não poderá ser usada como token de autorização do AWS_LAMBDA.
- Se a API tiver os modos de autorização AWS_LAMBDA, OPENID_CONNECT ou AMAZON_COGNITO_USER_POOLS habilitados, o token do OIDC não poderá ser usado como token de autorização do AWS_LAMBDA. Observe que o token OIDC pode ser um esquema do Bearer.
- Uma função do Lambda não deve retornar mais de 5 MB de dados contextuais para os resolvedores.

Por exemplo, se o token de autorização for 'ABC123', envie uma consulta do GraphQL via curl da seguinte forma:

```
$ curl -XPOST -H "Content-Type:application/graphql" -H "Authorization:ABC123" -d '{ "query": "query { movies { id } }" }' https://YOURAPPSYNCENDPOINT/graphql
```

As funções do Lambda são chamadas antes de cada consulta ou mutação. O valor de retorno pode ser armazenado em cache com base no ID da API e no token de autenticação. Quando uma resposta do autorizador Lambda tem menos de 1.048.576 bytes, armazena em cache a resposta para solicitações subsequentes. AWS AppSync Se a resposta do autorizador Lambda for igual ou maior que 1.048.576 bytes, AWS AppSync não armazena a resposta em cache e invoca o autorizador Lambda para cada solicitação recebida. Para otimizar o desempenho e minimizar os custos de invocação do Lambda, recomendamos que você limite suas respostas do autorizador do Lambda a 1.048.576 bytes. Por padrão, o armazenamento em cache não está habilitado, mas isso pode ser habilitado no nível da API ou definindo o valor `ttlOverride` no valor de retorno de uma função.

Uma expressão regular que valida os tokens de autorização antes que a função seja chamada pode ser especificada, se desejado. Essas expressões regulares são usadas para validar se um token de autorização está no formato correto antes de sua função ser chamada. Qualquer solicitação que use um token que não corresponda a essa expressão regular será negada automaticamente.

As funções Lambda usadas para autorização exigem que uma política principal seja aplicada `appsync.amazonaws.com` a elas AWS AppSync para permitir sua chamada. Essa ação é feita automaticamente no AWS AppSync console; o AWS AppSync console não remove a política. Para obter mais informações sobre como anexar políticas às funções do Lambda, [consulte Políticas baseadas em recursos](#) no Guia do desenvolvedor. AWS Lambda

A função do Lambda que você especificar receberá um evento com a seguinte forma:

```
{
  "authorizationToken": "ExampleAUTHtoken123123123",
  "requestContext": {
    "apiId": "aaaaaa123123123example123",
    "accountId": "111122223333",
    "requestId": "f4081827-1111-4444-5555-5cf4695f339f",
    "queryString": "mutation CreateEvent {...}\n\nquery MyQuery {...}\n",
    "operationName": "MyQuery",
    "variables": {}
  }
  "requestHeaders": {
    application request headers
  }
}
```

O event objeto contém os cabeçalhos que foram enviados na solicitação do cliente do aplicativo para AWS AppSync.

A função de autorização deve retornar pelo menos `isAuthorized` um booleano indicando se a solicitação foi autorizada. AWS AppSync reconhece as seguintes chaves retornadas das funções de autorização do Lambda:

Note

O valor da operação `operationName` in the `requestContext` for a WebSocket connect é definido AWS AppSync como `"DeepDish:Connect"`.

Lista de funções

`isAuthorized` (booleano, obrigatório)

Um valor booleano indicando se o valor no `authorizationToken` está autorizado a fazer chamadas para a API GraphQL.

Se esse valor for verdadeiro, a execução da API GraphQL continuará. Se esse valor for falso, um `UnauthorizedException` será gerado.

deniedFields (lista de strings, opcional)

Uma lista que é alterada à força para `null`, mesmo que um valor tenha sido retornado de um resolvedor.

Cada item é um ARN de campo totalmente qualificado na forma de `arn:aws:appsync:us-east-1:111122223333:apis/GraphQLApiId/types/TypeName/fields/FieldName` ou uma forma abreviada de `TypeName.FieldName`. O formulário ARN completo deve ser usado quando dois APIs compartilham um autorizador de função Lambda e pode haver ambigüidade entre tipos e campos comuns entre os dois. APIs

resolverContext (objeto JSON, opcional)

Um objeto JSON visível como `$ctx.identity.resolverContext` nos modelos de resolvedor. Por exemplo, se a estrutura a seguir for retornada por um resolvedor:

```
{
  "isAuthorized":true
  "resolverContext": {
    "banana":"very yellow",
    "apple":"very green"
  }
}
```

O valor de `ctx.identity.resolverContext.apple` nos modelos do resolvedor será "very green". O objeto `resolverContext` é compatível apenas com pares de chave-valor. Não há suporte para chaves aninhadas.

Warning

O tamanho total desse objeto JSON não deve exceder 5 MB.

ttlOverride (inteiro, opcional)

O número de segundos que deve levar para a resposta ser armazenada em cache. Se nenhum valor for retornado, o valor da API será usado. Se for 0, a resposta não será armazenada em cache.

Os autorizadores Lambda têm um tempo limite de 10 segundos. Recomendamos criar funções para serem executadas no menor tempo possível para escalar o desempenho da sua API.

Vários AWS AppSync APIs podem compartilhar uma única função Lambda de autenticação. O uso de autorizadores entre contas não é permitido.

Ao compartilhar uma função de autorização entre várias APIs, esteja ciente de que nomes de campo abreviados (*typename.fieldname*) podem ocultar campos inadvertidamente. Para eliminar a ambiguidade de um campo em `deniedFields`, você pode especificar um ARN de campo não ambíguo na forma de `arn:aws:appsync:region:accountId:apis/GraphQLApiId/types/typeName/fields/fieldName`.

Para adicionar uma função do Lambda como o modo de autorização padrão em AWS AppSync:

Console

1. Faça login no AWS AppSync console e navegue até a API que você deseja atualizar.
2. Navegue até a página Configurações da sua API.

Mude a autorização no nível da API para AWS Lambda.

3. Escolha o ARN Região da AWS e o Lambda para autorizar chamadas de API.

Note

A política entidade principal principal apropriada será adicionada automaticamente, permitindo que AWS AppSync chame sua função do Lambda.

4. Opcionalmente, defina o TTL de resposta e a expressão regular de validação de token.

AWS CLI

1. Anexe a seguinte política à função do Lambda que está sendo usada:

```
aws lambda add-permission --function-name "my-function" --statement-id "appsync"
--principal appsync.amazonaws.com --action lambda:InvokeFunction --output text
```

⚠ Important

Se você quiser que a política da função seja bloqueada em uma única API GraphQL, você pode executar este comando:

```
aws lambda add-permission --function-name "my-function" --
statement-id "appsync" --principal appsync.amazonaws.com --action
lambda:InvokeFunction --source-arn "<my AppSync API ARN>" --output text
```

2. Atualize sua AWS AppSync API para usar a função ARN do Lambda fornecida como autorizadora:

```
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"
```

i Note

Você também pode incluir outras opções de configuração, como a expressão regular do token.

O exemplo a seguir descreve uma função do Lambda que demonstra os vários estados de autenticação e falha que uma função do Lambda pode ter quando usada como mecanismo de autorização do AWS AppSync :

```
def handler(event, context):
    # This is the authorization token passed by the client
    token = event.get('authorizationToken')
    # If a lambda authorizer throws an exception, it will be treated as unauthorized.
    if 'Fail' in token:
        raise Exception('Purposefully thrown exception in Lambda Authorizer.')

    if 'Authorized' in token and 'ReturnContext' in token:
        return {
            'isAuthorized': True,
            'resolverContext': {
                'key': 'value'
```

```
    }  
  }  
  
  # Authorized with no f  
  if 'Authorized' in token:  
    return {  
      'isAuthorized': True  
    }  
  # Partial authorization  
  if 'Partial' in token:  
    return {  
      'isAuthorized': True,  
      'deniedFields':['user.favoriteColor']  
    }  
  if 'NeverCache' in token:  
    return {  
      'isAuthorized': True,  
      'ttlOverride': 0  
    }  
  if 'Unauthorized' in token:  
    return {  
      'isAuthorized': False  
    }  
  # if nothing is returned, then the authorization fails.  
  return {}
```

Como contornar as limitações de autorização de tokens do SigV4 e OIDC

Os métodos a seguir podem ser usados para contornar o problema de não ser possível usar sua assinatura do SigV4 ou token do OIDC como seu token de autorização do Lambda quando determinados modos de autorização estão habilitados.

Se você quiser usar a assinatura do SigV4 como token de autorização do Lambda quando os modos de autorização do AWS_IAM e AWS_LAMBDA estiverem habilitados para a API, faça AWS AppSync o seguinte:

- Para criar um novo token de autorização Lambda, adicione and/or prefixos de sufixos aleatórios à assinatura SigV4.
- Para recuperar a assinatura SigV4 original, atualize sua função Lambda removendo os prefixos aleatórios e and/or sufixos do token de autorização do Lambda. Em seguida, use a assinatura original do SigV4 para autenticação.

Se você quiser usar o token OIDC como o token de autorização Lambda quando o modo de autorização ou os modos de OPENID_CONNECT autorização AMAZON_COGNITO_USER_POOLS e de AWS_LAMBDA autorização estiverem habilitados para a API, AWS AppSync faça o seguinte:

- Para criar um novo token de autorização Lambda, adicione and/or prefixos de sufixos aleatórios ao token OIDC. O token de autorização do Lambda não deve conter um prefixo do esquema do Bearer.
- Para recuperar o token OIDC original, atualize sua função Lambda removendo os prefixos aleatórios e and/or sufixos do token de autorização do Lambda. Em seguida, use o token original do OIDC para autenticação.

AWS_IAM autorização

Esse tipo de autorização impõe o [processo de assinatura versão 4 da assinatura AWS](#) na API GraphQL. Associe políticas de acesso do Identity and Access Management ([IAM](#)) com esse tipo de autorização. A aplicação pode aproveitar essa associação ao usar uma chave de acesso (que consiste em um ID de chave de acesso e chave de acesso secreta) ou usando credenciais temporárias de curta duração fornecidas por identidades federadas do Amazon Cognito.

Se quiser um perfil com acesso para realizar todas as operações de dados:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "appsync:GraphQL"
      ],
      "Resource": [
        "arn:aws:appsync:us-west-2:123456789012:apis/YourGraphQLApiId/*"
      ]
    }
  ]
}
```

Você pode encontrar na página principal `YourGraphQLApiId` de listagem de APIs no AppSync console, diretamente abaixo do nome da sua API. Como alternativa, você pode recuperá-lo com a CLI: `aws appsync list-graphql-apis`

Se desejar restringir o acesso somente a determinadas operações do GraphQL, faça isso para os campos `Query`, `Mutation` e `Subscription` raiz.

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>"
      ]
    }
  ]
}
```

Por exemplo, suponha que tenha o seguinte esquema e deseje restringir o acesso à obtenção de todas as postagens:

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

type Query {
  posts:[Post!]!
```

```
}

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

A política do IAM correspondente para uma função (que pode ser anexada a um banco de identidades do Amazon Cognito, por exemplo) seria semelhante ao seguinte:

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 autorização

Esse tipo de autorização impõe tokens do [OpenID Connect](#) (OIDC) fornecidos por um serviço compatível com o OIDC. O aplicativo pode aproveitar os usuários e os privilégios definidos pelo provedor de OIDC para controlar o acesso.

Um URL emissor é o único valor de configuração obrigatório que deve ser fornecido ao AWS AppSync, por exemplo, <https://auth.example.com>. Esse URL deve ser endereçável por HTTPS. AWS AppSync /.well-known/openid-configuration [anexa ao URL do emissor e localiza a configuração do OpenID de acordo com a especificação do https://auth.example.com/.well-known/openid-configuration OpenID Connect Discovery](#). Ele espera recuperar um documento JSON [RFC5785](#) compatível nesse URL. Esse documento JSON

deve conter uma `jwks_uri` chave, que aponta para o documento JSON Web Key Set (JWKS) com as chaves de assinatura. AWS AppSync exige que o JWKS contenha campos JSON de `e`, `kid`

AWS AppSync suporta uma ampla variedade de algoritmos de assinatura.

Algoritmos de assinatura

RS256

RS384

RS512

PS256

PS384

PS512

HS256

HS384

HS512

ES256

ES384

ES512

Recomendamos que você use os algoritmos do RSA. Os tokens emitidos pelo provedor devem incluir a hora em que o token foi emitido (`iat`) e pode incluir a hora em que foi autenticado (`auth_time`). Você pode fornecer valores de TTL para a hora de emissão (`iatTTL`) e a hora de autenticação (`authTTL`) na configuração do OpenID Connect para validação adicional. Se o provedor autoriza vários aplicativos, você também pode fornecer uma expressão regular (`clientId`) usada para autorizar por ID de cliente. Quando o `clientId` está presente em sua configuração do OpenID Connect, AWS AppSync valida a declaração exigindo que a corresponda `clientId` à `azp` reivindicação `aud` ou à reivindicação no token.

Para validar vários clientes, IDs use o operador de pipeline (“|”), que é um “ou” na expressão regular. Por exemplo, se seu aplicativo OIDC tiver quatro clientes com cliente, IDs como 0A1S2D, 1F4G9H, 1J6L4B, 6 MG, para validar somente os três primeiros clientes, você colocaria 1F4G9H|1J6L4B|6 GS5 MG no campo ID do cliente. IDs GS5

Se uma API estiver configurada com vários tipos de autorização, AWS AppSync valida o emissor (declaração iss) presente no token JWT a partir dos cabeçalhos da solicitação comparando-o com o URL do emissor especificado na configuração da API. No entanto, quando uma API é configurada somente com OPENID_CONNECT autorização, AWS AppSync ignora essa etapa de validação do URL do emissor.

Autorização do AMAZON_COGNITO_USER_POOLS

Esse tipo de autorização impõe tokens do OIDC fornecidos por grupos de usuários do Amazon Cognito. Seu aplicativo pode aproveitar os usuários e grupos em seus grupos de usuários e grupos de usuários de outra AWS conta e associá-los aos campos do GraphQL para controlar o acesso.

Ao usar grupos de usuários do Amazon Cognito, crie grupos aos quais os usuários pertencem. Essas informações são codificadas em um token JWT para o qual seu aplicativo envia AWS AppSync em um cabeçalho de autorização ao enviar operações do GraphQL. Use diretivas do GraphQL no esquema para controlar quais grupos podem invocar quais resolvedores em um campo, oferecendo acesso mais controlado aos clientes.

Por exemplo, digamos que tenha o seguinte esquema do GraphQL:

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

type Query {
  posts:[Post!]!
}

type Mutation {
  addPost(id:ID!, title:String!):Post!
}
...
```

Se tiver dois grupos em grupos de usuários do Amazon Cognito (blogueiros e leitores) e quiser restringir os leitores para que não possam adicionar novas entradas, o seu esquema deve ser semelhante ao seguinte:

```
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"])  
}  
...
```

Observe que você pode omitir a `@aws_auth` diretiva se quiser usar como padrão uma `grant-or-deny` estratégia específica de acesso. Você pode especificar a `grant-or-deny` estratégia na configuração do grupo de usuários ao criar sua API GraphQL por meio do console ou do seguinte comando da CLI:

```
$ aws appsync --region us-west-2 create-graphql-api --authentication-  
type AMAZON_COGNITO_USER_POOLS --name userpoolstest --user-pool-config  
'{ "userPoolId":"test", "defaultEffect":"ALLOW", "awsRegion":"us-west-2"}'
```

Usar modos de autorização adicionais

Ao adicionar outros modos de autorização, você pode definir diretamente a configuração de autorização no nível da API AWS AppSync GraphQL (ou seja, o `authenticationType` campo que você pode configurar diretamente no `GraphQLApi` objeto) e ela atua como padrão no esquema. Isso significa que qualquer tipo que não tenha uma diretiva específica deve passar na configuração de autorização no nível da API.

No nível do esquema, é possível especificar modos de autorização adicionais usando diretivas no esquema. Você pode especificar modos de autorização em campos individuais no esquema. Por exemplo, para autorização `API_KEY`, você usaria `@aws_api_key` em definições/campos de tipo de

objeto de esquema. As seguintes diretivas são compatíveis com campos de esquema e definições de tipo de objeto:

- `@aws_api_key` – para especificar que o campo é `API_KEY` autorizado.
- `@aws_iam` – para especificar que o campo é `AWS_IAM` autorizado.
- `@aws_oidc` – para especificar que o campo é `OPENID_CONNECT` autorizado.
- `@aws_cognito_user_pools` – para especificar que o campo é `AMAZON_COGNITO_USER_POOLS` autorizado.
- `@aws_lambda` – para especificar que o campo é `AWS_LAMBDA` autorizado.

Não é possível usar a diretiva `@aws_auth` junto com modos de autorização adicionais. O `@aws_auth` funciona apenas no contexto de autorização `AMAZON_COGNITO_USER_POOLS` sem modos de autorização adicionais. No entanto, é possível usar a diretiva `@aws_cognito_user_pools` no lugar da diretiva `@aws_auth`, usando os mesmos argumentos. A principal diferença entre as duas é que você pode especificar `@aws_cognito_user_pools` em qualquer definição de campo e tipo de objeto.

Para entender como os modos de autorização adicionais funcionam e como eles podem ser especificados em um esquema, vamos dar uma olhada no seguinte esquema:

```
schema {  
  query: Query  
  mutation: Mutation  
}  
  
type Query {  
  getPost(id: ID!): Post  
  getAllPosts(): [Post]  
  @aws_api_key  
}  
  
type Mutation {  
  addPost(  
    id: ID!  
    author: String!  
    title: String!  
    content: String!  
    url: String!  
  ): Post!
```

```
}

type Post @aws_api_key @aws_iam {
  id: ID!
  author: String
  title: String
  content: String
  url: String
  ups: Int!
  downs: Int!
  version: Int!
}

...
```

Para esse esquema, suponha que esse `AWS_IAM` seja o tipo de autorização padrão na API AWS AppSync GraphQL. Isso significa que os campos que não têm uma diretiva são protegidos usando o `AWS_IAM`. Por exemplo, esse é o caso do campo `getPost` no tipo `Query`. As diretivas de esquema permitem que você use mais de um modo de autorização. Por exemplo, você pode ter `API_KEY` configurado como um modo de autorização adicional na API AWS AppSync GraphQL e pode marcar um campo usando a `@aws_api_key` diretiva (por exemplo, `getAllPosts` neste exemplo). As diretivas funcionam no nível do campo, portanto, também é necessário conceder a `API_KEY` acesso ao tipo `Post`. Você pode fazer isso marcando cada campo no tipo `Post` com uma diretiva ou marcando o tipo `Post` com a diretiva `@aws_api_key`.

Para restringir ainda mais o acesso aos campos no tipo `Post`, é possível usar diretivas em relação a campos individuais no tipo `Post`, conforme mostrado a seguir.

Por exemplo, é possível adicionar um campo `restrictedContent` ao tipo `Post` e restringir o acesso a ele usando a diretiva `@aws_iam`. As solicitações `AWS_IAM` autenticadas podem acessar `restrictedContent`, no entanto, as solicitações `API_KEY` não podem acessá-lo.

```
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
}
...
```

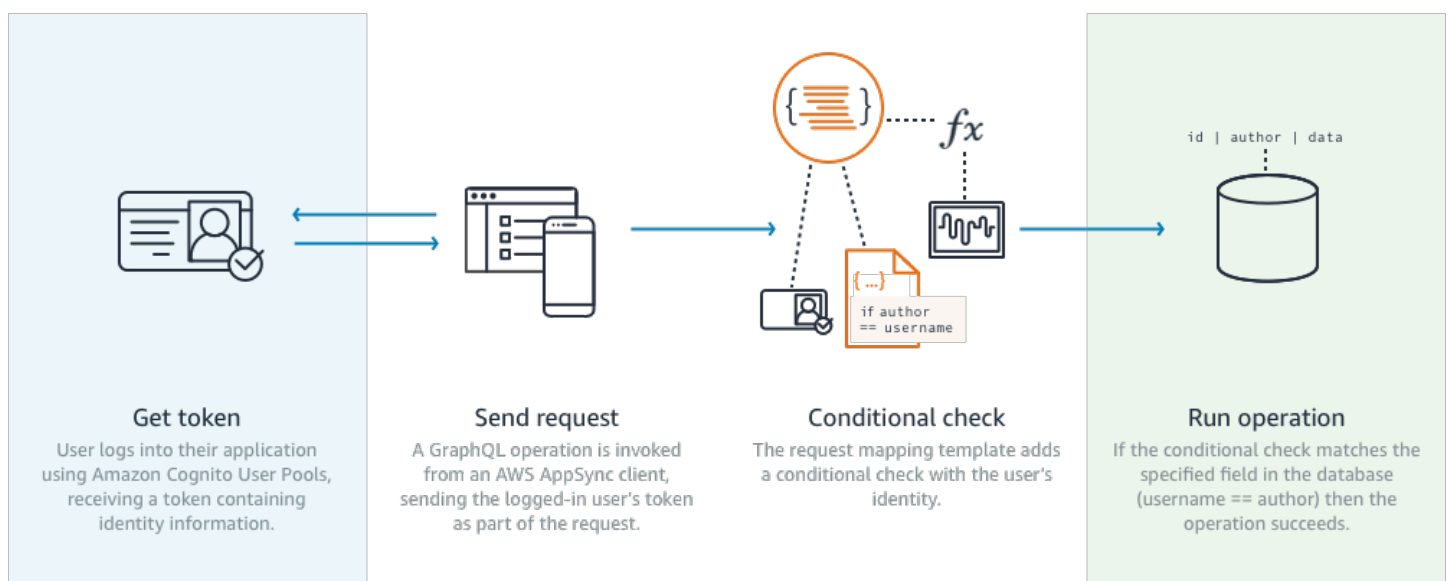
Controle de acesso refinado

As informações anteriores demonstram como restringir ou conceder acesso a determinados campos do GraphQL. Se deseja definir controles de acesso nos dados com base em determinadas condições (por exemplo, com base no usuário que está fazendo uma chamada e se o usuário é proprietário dos dados), você pode usar modelos de mapeamento nos resolvedores. Também é possível executar lógica de negócios mais complexa, descrita em [Filtrar informações](#).

Essa seção mostra como definir controles de acesso nos dados usando um modelo de mapeamento de resolvedor do DynamoDB.

Antes de prosseguir, se você não estiver familiarizado com os modelos de mapeamento em AWS AppSync, talvez queira revisar a referência do modelo de mapeamento do [Resolver e a referência do modelo de mapeamento](#) do [Resolver para o DynamoDB](#).

No exemplo a seguir que usa o DynamoDB, suponha que você esteja usando o esquema de publicação de blog anterior e somente usuários que criaram uma publicação possam editá-lo. O processo de avaliação seria o usuário obter credenciais no seu aplicativo, usando Grupos de usuários do Amazon Cognito por exemplo e, em seguida, enviar essas credenciais como parte de uma operação do GraphQL. Em seguida, o modelo de mapeamento substituirá um valor das credenciais (como o nome do usuário) em uma instrução condicional que então será comparado a um valor do banco de dados.



Para adicionar essa funcionalidade, adicione um campo do GraphQL `editPost` da seguinte forma:

```
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!  
}  
...
```

O modelo de mapeamento do resolvidor `editPost` (mostrado em um exemplo no final dessa seção) precisa executar uma verificação lógica no armazenamento de dados para permitir que apenas o usuário que criou uma publicação possa editá-la. Como essa é uma operação de edição, ela corresponde a um `UpdateItem` no DynamoDB. Execute uma verificação condicional antes de executar essa ação, usando o contexto enviado para a validação de identidade do usuário. Isso é armazenado em um objeto `Identity` com os seguintes valores:

```
{  
  "accountId" : "12321434323",  
  "cognitoIdentityPoolId" : "",  
  "cognitoIdentityId" : "",  
  "sourceIP" : "",  
  "caller" : "ThisistheprincipalARN",  
  "username" : "username",  
  "userArn" : "Sameasabove"  
}
```

Para usar esse objeto em uma chamada do `UpdateItem` do DynamoDB, é necessário armazenar as informações sobre a identidade do usuário na tabela para comparação. Primeiro, a mutação `addPost` precisa armazenar o criador. Em segundo lugar, a mutação `editPost` precisa executar a verificação condicional antes de atualizar.

Veja aqui um exemplo de código de resolvidor para `addPost` que armazena a identidade do usuário como uma coluna `Author`:

```
import { util, Context } from '@aws-appsync/utils';
import { put } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const { id: postId, ...item } = ctx.args;
  return put({
    key: { postId },
    item: { ...item, Author: ctx.identity.username },
    condition: { postId: { attributeExists: false } },
  });
}

export const response = (ctx) => ctx.result;
```

Observe que o atributo `Author` é preenchido a partir do objeto `Identity`, que veio do aplicativo.

Por fim, veja um exemplo do código de resolvedor para `editPost`, que atualizará apenas o conteúdo da publicação de blog se a solicitação vier do usuário que criou a publicação:

```
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;
```

Este exemplo usa um `PutItem` que substitui todos os valores em vez de um `UpdateItem`, mas o mesmo conceito se aplica ao bloco de declarações `condition`.

Filtrar informações

Pode haver casos onde não é possível controlar a resposta da fonte de dados, mas você não deseja enviar informações desnecessárias aos clientes sobre uma gravação ou leitura bem-sucedida da

fonte de dados. Nesses casos, você pode filtrar as informações usando um modelo de mapeamento de resposta.

Por exemplo, suponha que você não tenha um índice apropriado na tabela do DynamoDB da publicação de blog (como um índice em `Author`). É possível usar o seguinte resolvidor:

```
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;
}
```

O manipulador de solicitações busca o item mesmo que o chamador não seja o autor que criou a publicação. Para evitar que isso exiba todos os dados, o manipulador de respostas verifica se o chamador corresponde ao autor do item. Se o chamador não corresponder a essa verificação, apenas uma resposta nula é retornada.

Acesso à fonte de dados

AWS AppSync se comunica com fontes de dados usando funções e políticas de acesso do Identity and Access Management ([IAM](#)). Se você estiver usando uma função existente, uma Política de Confiança precisará ser adicionada AWS AppSync para que você possa assumir a função. A relação de confiança terá a seguinte aparência:

JSON

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

```
        "Service": "appsync.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

É importante restringir a política de acesso no perfil para ter permissões apenas para atuar sobre o conjunto mínimo de recursos necessários. Ao usar o AppSync console para criar uma fonte de dados e criar uma função, isso é feito automaticamente para você. No entanto, ao usar um modelo de exemplo integrado do console do IAM para criar uma função fora do console do AWS AppSync, as permissões não serão restringidas automaticamente em um recurso e você deve executar essa ação antes de mover o aplicativo para produção.

Casos de uso de controle de acesso para proteger solicitações e respostas

Na seção [Segurança](#) você aprendeu sobre os diferentes modos de Autorização para proteger sua API e uma introdução foi dada sobre mecanismos de Autorização refinada para entender os conceitos e o fluxo. Como AWS AppSync permite que você execute operações lógicas completas nos dados por meio do uso dos [modelos de mapeamento](#) do GraphQL Resolver, você pode proteger os dados na leitura ou gravação de uma maneira muito flexível usando uma combinação de identidade do usuário, condicionais e injeção de dados.

Se você não estiver familiarizado com a edição de AWS AppSync resolvers, consulte o [guia de programação](#).

Visão geral

Conceder acesso aos dados em um sistema é tradicionalmente feito por meio de uma [Matriz de controle do acesso](#) em que a interseção de uma linha (recurso) e uma coluna (usuário/função) são as permissões concedidas.

AWS AppSync usa recursos em sua própria conta e insere informações de identidade (usuário/função) na solicitação e resposta do GraphQL como um [objeto de contexto](#), que você pode usar no resolver. Isso significa que as permissões podem ser concedidas adequadamente em operações de leitura ou gravação com base na lógica do resolver. Se essa lógica estiver no nível do recurso, por exemplo, somente determinados usuários ou grupos nomeados podem read/write acessar uma

linha específica do banco de dados, esses “metadados de autorização” deverão ser armazenados. AWS AppSync não armazena nenhum dado, portanto, você deve armazenar esses metadados de autorização com os recursos para que as permissões possam ser calculadas. Os metadados de autorização geralmente são um atributo (coluna) em uma tabela do DynamoDB, como um proprietário ou lista de usuários/grupos. Por exemplo, pode haver atributos Leitores e Gravadores.

Em um alto nível, isso significa que, se estiver lendo um item individual de uma fonte de dados, execute uma instrução `#if ( ) ... #end` condicional no modelo da resposta depois que o resolvidor leu a fonte de dados. A verificação normalmente usará valores de usuário ou grupo em `$context.identity` para verificações de associação nos metadados de autorização retornados de uma operação de leitura. Para vários registros, como listas retornadas de uma tabela Scan ou Query, você enviará a verificação de condição como parte da operação à fonte de dados usando valores de usuário ou grupo semelhantes.

De maneira semelhante à gravação de dados você aplicará uma instrução condicional para a ação (como `PutItem` ou `UpdateItem`) para ver se o usuário ou grupo que faz uma mutação tem permissão. Novamente, o condicional muitas vezes usará um valor em `$context.identity` para comparar nos metadados de autorização daquele recurso. Para ambos os modelos da solicitação e da resposta, você também pode usar cabeçalhos personalizados de clientes para executar verificações de validação.

Leitura de dados

Conforme descrito acima, os metadados de autorização para realizar uma verificação devem ser armazenados com um recurso ou enviados para a solicitação do GraphQL (identidade, cabeçalho, etc.). Para demonstrar isso suponha que você tem a tabela do DynamoDB abaixo:

ID	Data	PeopleCanAccess	GroupsCanAccess	Owner
123	{my: data,...}	[Mary, Joe]	[Admins, Editors]	Nadia

A chave primária é `id` e os dados a serem acessados são `Data`. As outras colunas são exemplos de verificações que podem ser executadas para autorização. `Owner` seria uma `String` enquanto `PeopleCanAccess` e `GroupsCanAccess` seriam `String Sets`, conforme descrito na [Referência do modelo de mapeamento do resolvidor para o DynamoDB](#).

Na [visão geral do modelo de mapeamento do resolvidor](#) o diagrama mostra como o modelo da resposta contém não apenas o objeto de contexto, mas também os resultados da fonte de

dados. Para consultas do GraphQL de itens individuais, você pode usar o modelo da resposta para verificar se o usuário tem permissão para ver esses resultados ou retornar uma mensagem de erro de autorização. Isso geralmente é indicado como um "Filtro de autorização". Para consultas do GraphQL que retornam listas, usando uma Scan ou Query, é mais eficiente realizar a verificação no modelo da solicitação e retornar dados somente se uma condição de autorização for atendida. A implementação é:

1. `GetItem` - verificação de autorização para registros individuais. Feita usando instruções `#if() ... #end`.
2. Operações `Scan/Query` – verificação de autorização é uma instrução `"filter"`: `{"expression":...}`. As verificações comuns são a igualdade (`attribute = :input`) ou verificar se um valor está em uma lista (`contains(attribute, :input)`).

Em #2 o `attribute` em ambas as instruções representa o nome da coluna do registro em uma tabela, como `Owner` no exemplo acima. Você pode transformar isso em alias com um sinal `#` e usar `"expressionNames":{...}`, mas não é obrigatório. O `:input` é uma referência ao valor que você está comparando com o atributo do banco de dados, que será definido em `"expressionValues":{...}`. Você verá esses exemplos abaixo.

Caso de uso: o proprietário pode fazer leitura

Usando a tabela acima, se apenas deseja retornar dados se `Owner == Nadia` para uma única operação de leitura (`GetItem`) o modelo terá a seguinte aparência:

```
#if($context.result["Owner"] == $context.identity.username)
  $utils.toJson($context.result)
#else
  $utils.unauthorized()
#end
```

Algumas coisas devem ser mencionadas aqui, que serão reutilizadas nas seções restantes. Primeiro, a verificação usa `$context.identity.username` que será o nome de login de usuário amigável se grupos de usuários do Amazon Cognito for usado e será a identidade do usuário se o IAM for usado (incluindo as Identidades federadas do Amazon Cognito). Existem outros valores a serem armazenados para um proprietário, como o valor exclusivo "identidade do Amazon Cognito", que é útil ao federar logins de vários locais, e você deve revisar as opções disponíveis na [Referência de contexto do modelo de mapeamento do resolvedor](#).

Segundo, a verificação condicional `else` respondendo com `$util.unauthorized()` é totalmente opcional, mas recomendado como uma das melhores práticas ao projetar sua API GraphQL.

Caso de uso: codificar acesso específico

```
// This checks if the user is part of the Admin group and makes the call
foreach($group in $context.identity.claims.get("cognito:groups"))
  if($group == "Admin")
    set($inCognitoGroup = true)
  end
end
if($inCognitoGroup)
{
  "version" : "2017-02-28",
  "operation" : "UpdateItem",
  "key" : {
    "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
  },
  "attributeValues" : {
    "owner" : $util.dynamodb.toDynamoDBJson($context.identity.username)
    foreach( $entry in $context.arguments.entrySet() )
      , "${entry.key}" : $util.dynamodb.toDynamoDBJson($entry.value)
    end
  }
}
else
  $utils.unauthorized()
end
```

Caso de uso: filtrar uma lista de resultados

No exemplo anterior você pôde executar uma verificação em `$context.result` diretamente pois retornou um único item, no entanto algumas operações como uma verificação retornarão vários itens em `$context.result.items`, onde será necessário executar o filtro de autorização e retornar apenas os resultados que o usuário tem permissão para ver. Digamos que o campo `Owner` tinha o `IdentityID` do Amazon Cognito dessa vez definido no registro, então você poderia usar o seguinte modelo de mapeamento da resposta para filtrar a exibição somente dos registros de propriedade do usuário:

```
#set($myResults = [])
foreach($item in $context.result.items)
  ##For userpools use $context.identity.username instead
```

```

    #if($item.Owner == $context.identity.cognitoIdentityId)
        #set($added = $myResults.add($item))
    #end
#end
$utils.toJson($myResults)

```

Caso de uso: várias pessoas podem fazer leitura

Outra opção de autorização popular é permitir que um grupo de pessoas seja capaz de ler dados. No exemplo abaixo, o `"filter":{"expression":...}` retorna apenas valores de uma verificação de tabela se o usuário que executa a consulta do GraphQL estiver listado no conjunto de `PeopleCanAccess`.

```

{
  "version" : "2017-02-28",
  "operation" : "Scan",
  "limit": #if(${context.arguments.count}) $util.toJson($context.arguments.count)
#else 20 #end,
  "nextToken": #if(${context.arguments.nextToken})
$util.toJson($context.arguments.nextToken) #else null #end,
  "filter":{
    "expression": "contains(#peopleCanAccess, :value)",
    "expressionNames": {
      "#peopleCanAccess": "peopleCanAccess"
    },
    "expressionValues": {
      ":value": $util.dynamodb.toDynamoDBJson($context.identity.username)
    }
  }
}

```

Caso de uso: o grupo pode fazer leitura

Semelhante ao último caso de uso, pode ser que apenas as pessoas em um ou mais grupos tenham direitos para ler determinados itens em um banco de dados. O uso da operação `"expression": "contains()"` é semelhante, no entanto é um OU lógico de todos os grupos dos quais um usuário pode fazer parte, o que precisa ser considerado na associação definida. Nesse caso, acumulamos uma instrução `$expression` abaixo para cada grupo em que o usuário está e, em seguida, enviamos isso ao filtro:

```

#set($expression = "")

```

```

#set($expressionValues = {})
#foreach($group in $context.identity.claims.get("cognito:groups"))
    #set( $expression = "${expression} contains(groupsCanAccess, :var
$foreach.count )" )
    #set( $val = {})
    #set( $test = $val.put("S", $group))
    #set( $values = $expressionValues.put(":var$foreach.count", $val))
    #if ( $foreach.hasNext )
    #set( $expression = "${expression} OR" )
    #end
#end
#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)
    }
}

```

Gravação de dados

A gravação de dados em mutações sempre é controlada no modelo de mapeamento da solicitação. No caso de fontes de dados do DynamoDB, a chave é usar um "condition": {"expression"...} adequado que executa a validação nos metadados de autorização nessa tabela. Em [Segurança](#), fornecemos um exemplo que pode ser usado para verificar o campo Author em uma tabela. Os casos de uso nessa seção exploram mais casos de uso.

Caso de uso: vários proprietários

Usando a tabela de exemplo do diagrama anterior, suponha que a lista PeopleCanAccess

```

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

```

```

    "update" : {
      "expression" : "SET meta = :meta",
      "expressionValues": {
        ":meta" : $util.dynamodb.toDynamoDBJson($ctx.args.meta)
      }
    },
    "condition" : {
      "expression" : "contains(Owner,:expectedOwner)",
      "expressionValues" : {
        ":expectedOwner" :
$util.dynamodb.toDynamoDBJson($context.identity.username)
      }
    }
  }
}

```

Caso de uso: o grupo pode criar um novo registro

```

#set($expression = "")
#set($expressionValues = {})
#foreach($group in $context.identity.claims.get("cognito:groups"))
  #set( $expression = "${expression} contains(groupsCanAccess, :var
$foreach.count )" )
  #set( $val = {})
  #set( $test = $val.put("S", $group))
  #set( $values = $expressionValues.put(":var$foreach.count", $val))
  #if ( $foreach.hasNext )
  #set( $expression = "${expression} OR" )
  #end
#end
{
  "version" : "2017-02-28",
  "operation" : "PutItem",
  "key" : {
    ## If your table's hash key is not named 'id', update it here. **
    "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
    ## If your table has a sort key, add it as an item here. **
  },
  "attributeValues" : {
    ## Add an item for each field you would like to store to Amazon DynamoDB. **
    "title" : $util.dynamodb.toDynamoDBJson($ctx.args.title),
    "content": $util.dynamodb.toDynamoDBJson($ctx.args.content),
    "owner": $util.dynamodb.toDynamoDBJson($context.identity.username)
  },
}

```



```

    "condition" : {
      "expression": $util.toJson("attribute_not_exists(id) AND $expression"),
      "expressionValues": $utils.toJson($expressionValues)
    }
  }
}

```

Caso de uso: o grupo pode atualizar um registro existente

```

#set($expression = "")
#set($expressionValues = {})
#foreach($group in $context.identity.claims.get("cognito:groups"))
  #set( $expression = "${expression} contains(groupsCanAccess, :var
$foreach.count )" )
  #set( $val = {})
  #set( $test = $val.put("S", $group))
  #set( $values = $expressionValues.put(":var$foreach.count", $val))
  #if ( $foreach.hasNext )
  #set( $expression = "${expression} OR" )
  #end
#end
#end
{
  "version" : "2017-02-28",
  "operation" : "UpdateItem",
  "key" : {
    "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
  },
  "update":{
    "expression" : "SET title = :title, content = :content",
    "expressionValues": {
      ":title" : $util.dynamodb.toDynamoDBJson($ctx.args.title),
      ":content" : $util.dynamodb.toDynamoDBJson($ctx.args.content)
    }
  },
  "condition" : {
    "expression": $util.toJson($expression),
    "expressionValues": $utils.toJson($expressionValues)
  }
}

```

Registros públicos e privados

Com os filtros condicionais você também pode optar por marcar os dados como privado, público ou alguma outra verificação Booleana. Isso pode ser combinado como parte de um filtro de autorização dentro do modelo da resposta. Usar essa verificação é uma boa maneira de ocultar temporariamente os dados ou removê-los da visualização sem tentar controlar a associação de grupo.

Por exemplo, suponha que você adicionou um atributo em cada item na tabela do DynamoDB chamada `public` com um valor de `yes` ou `no`. O seguinte modelo da resposta pode ser usado em uma chamada `GetItem` para exibir os dados somente se o usuário estiver em um grupo que tem acesso E se esses dados estiverem marcados como público:

```
#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
```

O código acima também pode usar um OU lógico (`||`) para permitir que as pessoas façam leitura se tiverem permissão para um registro ou se for público:

```
#if($hasPermission || $context.result.public == 'yes')
    $utils.toJson($context.result)
#else
    $utils.unauthorized()
#end
```

No geral, os operadores padrão `==`, `!=`, `&&` e `||` serão úteis ao executar verificações de autorização.

Dados em tempo real

Você pode aplicar Controles de acesso refinados em assinaturas do GraphQL no momento em que um cliente faz uma assinatura, usando as mesmas técnicas descritas anteriormente nessa documentação. Anexe um resolvedor ao campo de assinatura, no momento em que pode consultar dados de uma fonte de dados e realizar a lógica condicional nos modelos de mapeamento da solicitação ou da resposta. Você também pode retornar dados adicionais para o cliente, como os resultados iniciais de uma assinatura, desde que a estrutura de dados corresponda àquela do tipo retornado na assinatura do GraphQL.

Caso de uso: o usuário pode assinar apenas conversas específicas

Um caso de uso comum para dados em tempo real com assinaturas do GraphQL é a criação de um aplicativo mensagens ou bate-papo privado. Ao criar um aplicativo de bate-papo com vários usuários, as conversas podem ocorrer entre duas pessoas ou entre várias pessoas. Elas podem ser agrupadas em "salas", privadas ou públicas. Dessa forma, você deseja autorizar apenas um usuário para assinar uma conversa (que pode ser um a um ou entre um grupo) à qual terão o acesso concedido. Para fins de demonstração, o exemplo abaixo mostra um caso de uso simples de um usuário que envia uma mensagem privada para outro. A configuração tem duas tabelas do Amazon DynamoDB:

- Tabela de mensagens: (chave primária) `toUser`, (chave de classificação) `id`
- Tabela de permissões: (chave primária) `username`

A tabela de Mensagens armazena as mensagens que realmente foram enviadas por meio de uma mutação do GraphQL. A tabela de Permissões é verificada pela assinatura do GraphQL para autorização no momento da conexão do cliente. O exemplo abaixo pressupõe que você está usando o seguinte esquema do GraphQL:

```
input CreateUserPermissionsInput {
  user: String!
  isAuthorizedForSubscriptions: Boolean
}

type Message {
  id: ID
  toUser: String
  fromUser: String
  content: String
}
```

```
}

type MessageConnection {
  items: [Message]
  nextToken: String
}

type Mutation {
  sendMessage(toUser: String!, content: String!): Message
  createUserPermissions(input: CreateUserPermissionsInput!): UserPermissions
  updateUserPermissions(input: UpdateUserPermissionInput!): UserPermissions
}

type Query {
  getMyMessages(first: Int, after: String): MessageConnection
  getUserPermissions(user: String!): UserPermissions
}

type Subscription {
  newMessage(toUser: String!): Message
    @aws_subscribe(mutations: ["sendMessage"])
}

input UpdateUserPermissionInput {
  user: String!
  isAuthorizedForSubscriptions: Boolean
}

type UserPermissions {
  user: String
  isAuthorizedForSubscriptions: Boolean
}

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

Algumas das operações padrão, como `createUserPermissions()`, não são abordadas abaixo para ilustrar os resolvedores de assinatura, mas são implementações padrão de resolvedores do DynamoDB. Em vez disso, vamos nos concentrar nos fluxos de autorização da assinatura com resolvedores. Para enviar uma mensagem de um usuário para outro, anexe um resolvedor ao campo

`sendMessage()` e selecione a fonte de dados da tabela de Mensagens com o seguinte modelo da solicitação:

```
{
  "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),
  }
}
```

Neste exemplo, usamos `$context.identity.username`. Isso retorna informações do usuário AWS Identity and Access Management ou dos usuários do Amazon Cognito. O modelo da resposta é uma simples passagem de `$util.toJson($ctx.result)`. Salvar e voltar à página do esquema. Em seguida, anexe um resolvidor para a assinatura `newMessage()`, usando a tabela de Permissões como uma fonte de dados e o seguinte modelo de mapeamento da solicitação:

```
{
  "version": "2018-05-29",
  "operation": "GetItem",
  "key": {
    "username": $util.dynamodb.toDynamoDBJson($ctx.identity.username),
  },
}
```

Em seguida, use o seguinte modelo de mapeamento da resposta para executar as verificações de autorização usando os dados da tabela de Permissões:

```
#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
```

Nesse caso, você está fazendo três verificações de autorização. A primeira garante que um resultado seja retornado. A segunda garante que o usuário não esteja assinando mensagens destinadas a outra pessoa. A terceira garante que o usuário tenha permissão para assinar qualquer campo, verificando um atributo do DynamoDB de `isAuthorizedForSubscriptions` armazenado como um `BOOL`.

Para testar as coisas, você pode entrar no AWS AppSync console usando grupos de usuários do Amazon Cognito e um usuário chamado “Nadia” e, em seguida, executar a seguinte assinatura do GraphQL:

```
subscription AuthorizedSubscription {
  newMessage(toUser: "Nadia") {
    id
    toUser
    fromUser
    content
  }
}
```

Se, na tabela de Permissões, houver um registro para o atributo chave `username` de Nadia com `isAuthorizedForSubscriptions` definido como `true`, você verá uma resposta bem-sucedida. Se tentar um `username` diferente na consulta `newMessage()` acima, um erro será retornado.

Usando AWS WAF para proteger seu AWS AppSync APIs

AWS WAF é um firewall de aplicativos da web que ajuda a proteger os aplicativos da web e APIs contra ataques. Isso permite configurar um conjunto de regras chamado de lista de controle de acesso à web (ACL da web) que permitem, bloqueiam ou monitoram (contam) solicitações da web com base em regras e condições de segurança da web personalizáveis que você define. Ao integrar sua AWS AppSync API AWS WAF, você ganha mais controle e visibilidade do tráfego HTTP aceito pela sua API. Para saber mais AWS WAF, consulte [Como AWS WAF funciona](#) no Guia do AWS WAF desenvolvedor.

Você pode usar AWS WAF para proteger sua AppSync API contra explorações comuns da Web, como injeção de SQL e ataques de script entre sites (XSS). Isso pode afetar a disponibilidade e a performance da API, comprometer a segurança ou consumir recursos excessivos. Por exemplo,

you can create rules to allow or block requests from IP address ranges, CIDR blocks, requests originating from a specific country or region, requests that contain malicious SQL code, or requests that contain malicious scripts.

You can also create rules that correspond to a specific string or a regular expression in HTTP headers, method, URI, query string, and request body (limited to the first 8 KB). Additionally, you can create rules to block attacks from specific users, bad bots, and content filtering. For example, you can use rules based on IP address ranges to specify the number of requests from the web that are allowed for each IP address of the client at the end of a five-minute update period.

To learn more about the types of rules supported and the additional AWS WAF resources, consult the [AWS WAF Developer Guide](#) and the [AWS WAF API Reference](#).

Important

AWS WAF is your first line of defense against web exploits. When AWS WAF is activated in an API, AWS WAF rules are evaluated before other access control resources, such as API key authorization, IAM policies, OAuth tokens, and Amazon Cognito user groups.

Integrate an AppSync API with AWS WAF

You can integrate an AppSync API with AWS WAF using the AWS Management Console, the AWS CLI, AWS CloudFormation, or any other compatible client.

To integrate an AWS AppSync API with AWS WAF:

1. Create an AWS WAF web ACL. For detailed steps on using the [AWS WAF Console](#), consult [Create a Web ACL](#).
2. Define rules for the web ACL. A rule or rules are defined during the creation of the web ACL. For information on how to structure rules, consult [AWS WAF Rules](#). For useful examples of rules you can define for your AWS AppSync API, consult [Create rules for a web ACL](#).
3. Associate the web ACL with an AWS AppSync API. You can execute this step in the [AWS WAF console](#) or the [AppSync console](#).

- Para associar a ACL da web a uma AWS AppSync API no AWS WAF console, siga as instruções para [associar ou desassociar uma ACL da Web a um AWS recurso no Guia](#) do desenvolvedor. AWS WAF
- Para associar a ACL da web a uma AWS AppSync API no console AWS AppSync
 - a. Faça login no AWS Management Console e abra o [AppSync console](#).
 - b. Escolha a API que deseja associar a uma ACL da Web.
 - c. No painel de navegação, selecione configurações.
 - d. Na seção Firewall da aplicação web, ative Habilitar AWS WAF.
 - e. Na lista suspensa ACL da Web, escolha o nome da ACL da Web para associar à sua API.
 - f. Escolha Salvar para associar a ACL da Web à sua API.

 Note

Depois de criar uma ACL da web no AWS WAF console, pode levar alguns minutos para que a nova ACL da web esteja disponível. Se você não vir uma ACL da Web recém-criada no Firewall de aplicativo web, aguarde alguns minutos e repita as etapas para associar a ACL da Web à sua API.

 Note

AWS WAF a integração só suporta o Subscription registration message evento para endpoints em tempo real. AWS AppSync responderá com uma mensagem de erro em vez de uma start_ack mensagem para qualquer mensagem Subscription registration message bloqueada por AWS WAF.

Depois de associar uma ACL da web a uma AWS AppSync API, você gerenciará a ACL da web usando o. AWS WAF APIs Você não precisa associar novamente a ACL da web à sua AWS AppSync API, a menos que queira associar a AWS AppSync API a uma ACL da web diferente.

Criar regras para uma ACL da Web

As regras definem como inspecionar solicitações da Web e o que fazer quando uma solicitação da Web corresponde aos critérios de inspeção. As regras não existem AWS WAF sozinhas. Você pode acessar uma regra por nome em um grupo de regras ou na ACL da Web onde ela está definida. Para obter mais informações, consulte [Regras do AWS WAF](#). Os exemplos a seguir demonstram como definir e associar regras que são úteis para proteger uma AppSync API.

Exemplo regra ACL da Web para limitar o tamanho do corpo da solicitação

Veja a seguir um exemplo de regra que limita o tamanho do corpo das solicitações. Isso seria inserido no editor Rule JSON ao criar uma ACL da web no AWS WAF console.

```
{
  "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
  }
}
```

Depois de criar sua ACL da web usando a regra do exemplo anterior, você deve associá-la à sua AppSync API. Como alternativa ao uso do AWS Management Console, você pode executar essa etapa no AWS CLI executando o comando a seguir.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Pode levar alguns minutos para que as alterações sejam propagadas, mas, após a execução deste comando, as solicitações que contenham um corpo maior que 1.024 bytes serão rejeitadas pelo AWS AppSync.

Note

Depois de criar uma nova ACL da web no AWS WAF console, pode levar alguns minutos para que a ACL da web esteja disponível para ser associada a uma API. Se você executar o comando da CLI e receber um erro `WAFUnavailableEntityException`, aguarde alguns minutos e tente executar o comando novamente.

Exemplo regra da ACL da Web para limitar as solicitações de um único endereço IP

Veja a seguir um exemplo de uma regra que limita uma AppSync API a 100 solicitações de um único endereço IP. Isso seria inserido no editor de regras JSON ao criar uma ACL da web com uma regra baseada em taxa no console. AWS WAF

```
{
  "Name": "Throttle",
  "Priority": 0,
  "Action": {
    "Block": {}
  },
  "VisibilityConfig": {
    "SampledRequestsEnabled": true,
    "CloudWatchMetricsEnabled": true,
    "MetricName": "Throttle"
  },
  "Statement": {
    "RateBasedStatement": {
      "Limit": 100,
      "AggregateKeyType": "IP"
    }
  }
}
```

```
}  
}
```

Depois de criar sua ACL da web usando a regra do exemplo anterior, você deve associá-la à sua AppSync API. Você pode executar essa etapa no AWS CLI executando o comando a seguir.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Exemplo Regra ACL da Web para evitar consultas de introspecção GraphQL `__schema` para uma API

Veja a seguir um exemplo de uma regra que impede consultas de introspecção do GraphQL `__schema` em uma API. Qualquer corpo HTTP que inclua a string `"__schema"` será bloqueado. Isso seria inserido no editor Rule JSON ao criar uma ACL da web no AWS WAF console.

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

Depois de criar sua ACL da web usando a regra do exemplo anterior, você deve associá-la à sua AppSync API. Você pode executar essa etapa no AWS CLI executando o comando a seguir.

```
aws waf associate-web-acl --web-acl-id waf-web-acl-arn --resource-arn appsync-api-arn
```

Segurança em AWS AppSync

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficia de data centers e arquiteturas de rede criados para atender aos requisitos das organizações mais sensíveis à segurança.

A segurança é uma responsabilidade compartilhada entre você AWS e você. O [modelo de responsabilidade compartilhada](#) descreve isso como segurança da nuvem e segurança na nuvem:

- **Segurança da nuvem** — AWS é responsável por proteger a infraestrutura que executa AWS os serviços na AWS nuvem. AWS também fornece serviços que você pode usar com segurança. Auditores terceirizados testam e verificam regularmente a eficácia de nossa segurança como parte dos Programas de Conformidade Programas de [AWS](#) de . Para saber mais sobre os programas de conformidade que se aplicam AWS AppSync, consulte [AWS Serviços no escopo do programa de conformidade AWS](#) .
- **Segurança na nuvem** — Sua responsabilidade é determinada pelo AWS serviço que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da empresa e as leis e regulamentos aplicáveis.

Esta documentação ajuda você a entender como aplicar o modelo de responsabilidade compartilhada ao usar AWS AppSync. Os tópicos a seguir mostram como configurar para atender AWS AppSync aos seus objetivos de segurança e conformidade. Você também aprenderá a usar outros AWS serviços que ajudam a monitorar e proteger seus AWS AppSync recursos.

Tópicos

- [Proteção de dados em AWS AppSync](#)
- [Validação de conformidade para AWS AppSync](#)
- [Segurança da infraestrutura em AWS AppSync](#)
- [Resiliência em AWS AppSync](#)
- [Gerenciamento de identidade e acesso para AWS AppSync](#)
- [Registrando chamadas de AWS AppSync API com AWS CloudTrail](#)
- [Práticas recomendadas de segurança para AWS AppSync](#)

Proteção de dados em AWS AppSync

O modelo de [responsabilidade AWS compartilhada modelo](#) se aplica à proteção de dados em AWS AppSync. Conforme descrito neste modelo, AWS é responsável por proteger a infraestrutura global que executa todos os Nuvem AWS. Você é responsável por manter o controle sobre o conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para obter mais informações sobre a privacidade de dados, consulte as [Data Privacy FAQ](#). Para obter mais informações sobre a proteção de dados na Europa, consulte a postagem do blog [AWS Shared Responsibility Model and RGPD](#) no Blog de segurança da AWS .

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com AWS IAM Identity Center ou AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com AWS os recursos. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e o registro de atividades do usuário com AWS CloudTrail. Para obter informações sobre o uso de CloudTrail trilhas para capturar AWS atividades, consulte Como [trabalhar com CloudTrail trilhas](#) no Guia AWS CloudTrail do usuário.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sigilosos armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-3 ao acessar AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint FIPS. Para obter mais informações sobre os endpoints FIPS disponíveis, consulte [Federal Information Processing Standard \(FIPS\) 140-3](#).

É altamente recomendável que nunca sejam colocadas informações confidenciais ou sigilosas, como endereços de e-mail de clientes, em tags ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com AWS AppSync ou Serviços da AWS usa o console, a API ou AWS SDKs. AWS CLI Quaisquer dados inseridos em tags ou em campos de texto de formato livre usados

para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, recomendamos fortemente que não sejam incluídas informações de credenciais no URL para validar a solicitação a esse servidor.

Criptografia em movimento

AWS AppSync, como todos os AWS serviços, faz uso de TLS1 .2 e superior para comunicação ao usar o AWS publicado APIs e. SDKs

O uso AWS AppSync com outros AWS serviços, como o Amazon DynamoDB, garante a criptografia em trânsito: AWS todos os serviços usam TLS 1.2 e versões posteriores para se comunicarem entre si, a menos que especificado de outra forma. Para resolvedores que utilizam a Amazon EC2 ou CloudFront, é sua responsabilidade verificar se o TLS (HTTPS) está configurado e seguro. Para obter informações sobre como configurar HTTPS na Amazon EC2, consulte [Configuração no SSL/TLS Amazon Linux 2 no guia](#) EC2 do usuário da Amazon. Para obter informações sobre como configurar HTTPS em CloudFront, consulte [HTTPS na Amazon CloudFront](#) no guia do CloudFront usuário.

Validação de conformidade para AWS AppSync

Audidores terceirizados avaliam a segurança e a conformidade AWS AppSync como parte de vários programas de AWS conformidade. AWS AppSync é compatível com os programas SOC, PCI, HIPAA/HIPAA BAA, IRAP, C5, ENS High, OSPAR e HITRUST CSF.

Para saber se um AWS service (Serviço da AWS) está dentro do escopo de programas de conformidade específicos, consulte [Serviços da AWS Escopo por Programa de Conformidade](#) [Serviços da AWS](#) e escolha o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de [AWS conformidade Programas AWS](#) de .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte [Baixar relatórios em AWS Artifact](#) .

Sua responsabilidade de conformidade ao usar Serviços da AWS é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade de sua empresa e pelas leis e regulamentações aplicáveis. AWS fornece os seguintes recursos para ajudar na conformidade:

- [Governança e conformidade de segurança](#): esses guias de implementação de solução abordam considerações sobre a arquitetura e fornecem etapas para implantar recursos de segurança e conformidade.

- [Referência de serviços qualificados para HIPAA](#): lista os serviços qualificados para HIPAA. Nem todos Serviços da AWS são elegíveis para a HIPAA.
- AWS Recursos de <https://aws.amazon.com/compliance/resources/> de conformidade — Essa coleção de pastas de trabalho e guias pode ser aplicada ao seu setor e local.
- [AWS Guias de conformidade do cliente](#) — Entenda o modelo de responsabilidade compartilhada sob a ótica da conformidade. Os guias resumem as melhores práticas de proteção Serviços da AWS e mapeiam as diretrizes para controles de segurança em várias estruturas (incluindo o Instituto Nacional de Padrões e Tecnologia (NIST), o Conselho de Padrões de Segurança do Setor de Cartões de Pagamento (PCI) e a Organização Internacional de Padronização (ISO)).
- [Avaliação de recursos com regras](#) no Guia do AWS Config desenvolvedor — O AWS Config serviço avalia o quão bem suas configurações de recursos estão em conformidade com as práticas internas, as diretrizes e os regulamentos do setor.
- [AWS Security Hub](#) — Isso AWS service (Serviço da AWS) fornece uma visão abrangente do seu estado de segurança interno AWS. O Security Hub usa controles de segurança para avaliar os recursos da AWS e verificar a conformidade com os padrões e as práticas recomendadas do setor de segurança. Para obter uma lista dos serviços e controles aceitos, consulte a [Referência de controles do Security Hub](#).
- [Amazon GuardDuty](#) — Isso AWS service (Serviço da AWS) detecta possíveis ameaças às suas cargas de trabalho Contas da AWS, contêineres e dados monitorando seu ambiente em busca de atividades suspeitas e maliciosas. GuardDuty pode ajudá-lo a atender a vários requisitos de conformidade, como o PCI DSS, atendendo aos requisitos de detecção de intrusões exigidos por determinadas estruturas de conformidade.
- [AWS Audit Manager](#) — Isso AWS service (Serviço da AWS) ajuda você a auditar continuamente seu AWS uso para simplificar a forma como você gerencia o risco e a conformidade com as regulamentações e os padrões do setor.

Segurança da infraestrutura em AWS AppSync

Como serviço gerenciado, AWS AppSync é protegido pela segurança de rede AWS global. Para obter informações sobre serviços AWS de segurança e como AWS proteger a infraestrutura, consulte [AWS Cloud Security](#). Para projetar seu AWS ambiente usando as melhores práticas de segurança de infraestrutura, consulte [Proteção](#) de infraestrutura no Security Pillar AWS Well-Architected Framework.

Você usa chamadas de API AWS publicadas para acessar AWS AppSync pela rede. Os clientes devem oferecer compatibilidade com:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Conjuntos de criptografia com perfect forward secrecy (PFS) como DHE (Ephemeral Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou é possível usar o [AWS Security Token Service](#) (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Resiliência em AWS AppSync

A infraestrutura AWS global é construída em torno de AWS regiões e zonas de disponibilidade. AWS As regiões fornecem várias zonas de disponibilidade fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e alta redundância. Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que automaticamente executam o failover entre as zonas sem interrupção. As zonas de disponibilidade são altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Para obter mais informações sobre AWS regiões e zonas de disponibilidade, consulte [Infraestrutura AWS global](#).

Além da infraestrutura AWS global, AWS AppSync permite que a maioria dos recursos seja definida usando AWS CloudFormation modelos; para ver um exemplo do uso de AWS CloudFormation modelos para declarar AWS AppSync recursos, consulte [Casos de uso práticos para resolvedores de AWS AppSync pipeline](#) no AWS blog e no Guia do [AWS CloudFormation usuário](#).

Gerenciamento de identidade e acesso para AWS AppSync

AWS Identity and Access Management (IAM) é uma ferramenta AWS service (Serviço da AWS) que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. Os administradores

do IAM controlam quem pode ser autenticado (conectado) e autorizado (tem permissões) a usar AWS AppSync os recursos. O IAM é um AWS service (Serviço da AWS) que você pode usar sem custo adicional.

Tópicos

- [Público](#)
- [Autenticação com identidades](#)
- [Gerenciar o acesso usando políticas](#)
- [Como AWS AppSync funciona com o IAM](#)
- [Políticas baseadas em identidade para o AWS AppSync](#)
- [Solução de problemas AWS AppSync de identidade e acesso](#)

Público

A forma como você usa AWS Identity and Access Management (IAM) difere, dependendo do trabalho que você faz AWS AppSync.

Usuário do serviço — Se você usar o AWS AppSync serviço para realizar seu trabalho, seu administrador fornecerá as credenciais e as permissões de que você precisa. À medida que você usa mais AWS AppSync recursos para fazer seu trabalho, talvez precise de permissões adicionais. Entender como o acesso é gerenciado pode ajudá-lo a solicitar as permissões corretas ao seu administrador. Se não for possível acessar um recurso no AWS AppSync, consulte [Solução de problemas AWS AppSync de identidade e acesso](#).

Administrador de serviços — Se você é responsável pelos AWS AppSync recursos da sua empresa, provavelmente tem acesso total AWS AppSync a. É seu trabalho determinar quais AWS AppSync recursos e recursos seus usuários do serviço devem acessar. Envie as solicitações ao administrador do IAM para alterar as permissões dos usuários de serviço. Revise as informações nesta página para compreender os conceitos básicos do IAM. Para saber mais sobre como sua empresa pode usar o IAM com AWS AppSync, consulte [Como AWS AppSync funciona com o IAM](#).

Administrador do IAM: se você for um administrador do IAM, talvez queira saber detalhes sobre como pode gravar políticas para gerenciar o acesso ao AWS AppSync. Para ver exemplos de políticas AWS AppSync baseadas em identidade que você pode usar no IAM, consulte. [Políticas baseadas em identidade para o AWS AppSync](#)

Autenticação com identidades

A autenticação é como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado (conectado AWS) como o Usuário raiz da conta da AWS, como usuário do IAM ou assumindo uma função do IAM.

Você pode entrar AWS como uma identidade federada usando credenciais fornecidas por meio de uma fonte de identidade. AWS IAM Identity Center Usuários (IAM Identity Center), a autenticação de login único da sua empresa e suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como identidade federada, o administrador já configurou anteriormente a federação de identidades usando perfis do IAM. Ao acessar AWS usando a federação, você está assumindo indiretamente uma função.

Dependendo do tipo de usuário que você é, você pode entrar no AWS Management Console ou no portal de AWS acesso. Para obter mais informações sobre como fazer login em AWS, consulte [Como fazer login Conta da AWS](#) no Guia do Início de Sessão da AWS usuário.

Se você acessar AWS programaticamente, AWS fornece um kit de desenvolvimento de software (SDK) e uma interface de linha de comando (CLI) para assinar criptograficamente suas solicitações usando suas credenciais. Se você não usa AWS ferramentas, você mesmo deve assinar as solicitações. Para obter mais informações sobre como usar o método recomendado para designar solicitações por conta própria, consulte [Versão 4 do AWS Signature para solicitações de API](#) no Guia do usuário do IAM.

Independente do método de autenticação usado, também pode ser necessário fornecer informações adicionais de segurança. Por exemplo, AWS recomenda que você use a autenticação multifator (MFA) para aumentar a segurança da sua conta. Para saber mais, consulte [Autenticação multifator](#) no Guia do usuário do AWS IAM Identity Center e [Usar a autenticação multifator da AWS no IAM](#) no Guia do usuário do IAM.

Conta da AWS usuário root

Ao criar uma Conta da AWS, você começa com uma identidade de login que tem acesso completo a todos Serviços da AWS os recursos da conta. Essa identidade é chamada de usuário Conta da AWS raiz e é acessada fazendo login com o endereço de e-mail e a senha que você usou para criar a conta. É altamente recomendável não usar o usuário-raiz para tarefas diárias. Proteja as credenciais do usuário-raiz e use-as para executar as tarefas que somente ele puder executar. Para obter a lista completa das tarefas que exigem login como usuário-raiz, consulte [Tarefas que exigem credenciais de usuário-raiz](#) no Guia do Usuário do IAM.

Identidade federada

Como prática recomendada, exija que usuários humanos, incluindo usuários que precisam de acesso de administrador, usem a federação com um provedor de identidade para acessar Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário do seu diretório de usuários corporativo, de um provedor de identidade da web AWS Directory Service, do diretório do Identity Center ou de qualquer usuário que acesse usando credenciais fornecidas Serviços da AWS por meio de uma fonte de identidade. Quando as identidades federadas são acessadas Contas da AWS, elas assumem funções, e as funções fornecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, é recomendável usar o AWS IAM Identity Center. Você pode criar usuários e grupos no IAM Identity Center ou pode se conectar e sincronizar com um conjunto de usuários e grupos em sua própria fonte de identidade para uso em todos os seus Contas da AWS aplicativos. Para obter mais informações sobre o Centro de Identidade do IAM, consulte [O que é o Centro de Identidade do IAM?](#) no Guia do Usuário do AWS IAM Identity Center .

Usuários e grupos do IAM

Um [usuário do IAM](#) é uma identidade dentro da sua Conta da AWS que tem permissões específicas para uma única pessoa ou aplicativo. Sempre que possível, é recomendável contar com credenciais temporárias em vez de criar usuários do IAM com credenciais de longo prazo, como senhas e chaves de acesso. No entanto, se você tiver casos de uso específicos que exijam credenciais de longo prazo com usuários do IAM, é recomendável alternar as chaves de acesso. Para obter mais informações, consulte [Alternar as chaves de acesso regularmente para casos de uso que exijam credenciais de longo prazo](#) no Guia do Usuário do IAM.

Um [grupo do IAM](#) é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível usar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um grupo chamado IAMAdminse conceder a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de perfis. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas um perfil pode ser assumido por qualquer pessoa que precisar dele. Os usuários têm credenciais permanentes de longo prazo, mas os perfis fornecem credenciais temporárias. Para saber mais, consulte [Casos de uso para usuários do IAM](#) no Guia do usuário do IAM.

Perfis do IAM

Uma [função do IAM](#) é uma identidade dentro da sua Conta da AWS que tem permissões específicas. Ele é semelhante a um usuário do IAM, mas não está associado a uma pessoa específica. Para assumir temporariamente uma função do IAM no AWS Management Console, você pode [alternar de um usuário para uma função do IAM \(console\)](#). Você pode assumir uma função chamando uma operação de AWS API AWS CLI ou usando uma URL personalizada. Para obter mais informações sobre métodos para usar perfis, consulte [Métodos para assumir um perfil](#) no Guia do usuário do IAM.

Perfis do IAM com credenciais temporárias são úteis nas seguintes situações:

- **Acesso de usuário federado:** para atribuir permissões a identidades federadas, é possível criar um perfil e definir permissões para ele. Quando uma identidade federada é autenticada, essa identidade é associada ao perfil e recebe as permissões definidas por ele. Para ter mais informações sobre perfis para federação, consulte [Criar um perfil para um provedor de identidade de terceiros \(federação\)](#) no Guia do usuário do IAM. Se usar o Centro de Identidade do IAM, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o Centro de Identidade do IAM correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte [Conjuntos de Permissões](#) no Guia do Usuário do AWS IAM Identity Center .
- **Permissões temporárias para usuários do IAM:** um usuário ou um perfil do IAM pode presumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- **Acesso entre contas:** é possível usar um perfil do IAM para permitir que alguém (uma entidade principal confiável) em outra conta acesse recursos em sua conta. Os perfis são a principal forma de conceder acesso entre contas. No entanto, com alguns Serviços da AWS, você pode anexar uma política diretamente a um recurso (em vez de usar uma função como proxy). Para conhecer a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte [Acesso a recursos entre contas no IAM](#) no Guia do usuário do IAM.
- **Acesso entre serviços —** Alguns Serviços da AWS usam recursos em outros Serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicativos na Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões da entidade principal da chamada, usando um perfil de serviço ou um perfil vinculado ao serviço.
 - **Sessões de acesso direto (FAS) —** Quando você usa um usuário ou uma função do IAM para realizar ações AWS, você é considerado principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O FAS usa as permissões do diretor chamando um AWS service (Serviço da AWS), combinadas com a solicitação AWS service

(Serviço da AWS) para fazer solicitações aos serviços posteriores. As solicitações do FAS são feitas somente quando um serviço recebe uma solicitação que requer interações com outros Serviços da AWS ou com recursos para ser concluída. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte [Sessões de acesso direto](#).

- **Perfil de serviço:** um perfil de serviço é um [perfil do IAM](#) que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte [Criar um perfil para delegar permissões a um AWS service \(Serviço da AWS\)](#) no Guia do Usuário do IAM.
- **Função vinculada ao serviço** — Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. AWS service (Serviço da AWS) O serviço pode presumir o perfil para executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para perfis vinculados a serviço.
- **Aplicativos em execução na Amazon EC2** — Você pode usar uma função do IAM para gerenciar credenciais temporárias para aplicativos que estão sendo executados em uma EC2 instância e fazendo solicitações AWS CLI de AWS API. Isso é preferível a armazenar chaves de acesso na EC2 instância. Para atribuir uma AWS função a uma EC2 instância e disponibilizá-la para todos os aplicativos, você cria um perfil de instância anexado à instância. Um perfil de instância contém a função e permite que os programas em execução na EC2 instância recebam credenciais temporárias. Para obter mais informações, consulte [Usar uma função do IAM para conceder permissões a aplicativos executados em EC2 instâncias da Amazon](#) no Guia do usuário do IAM.

Gerenciar o acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política é um objeto AWS que, quando associada a uma identidade ou recurso, define suas permissões. AWS avalia essas políticas quando um principal (usuário, usuário raiz ou sessão de função) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida ou negada. A maioria das políticas é armazenada AWS como documentos JSON. Para obter mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte [Visão geral das políticas JSON](#) no Guia do usuário do IAM.

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

Por padrão, usuários e perfis não têm permissões. Para conceder permissão aos usuários para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis e os usuários podem assumir os perfis.

As políticas do IAM definem permissões para uma ação independentemente do método usado para executar a operação. Por exemplo, suponha que você tenha uma política que permite a ação `iam:GetRole`. Um usuário com essa política pode obter informações de função da AWS Management Console AWS CLI, da ou da AWS API.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário, grupo de usuários ou perfil do IAM. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte [Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente](#) no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas como políticas em linha ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou perfil. As políticas gerenciadas são políticas autônomas que você pode associar a vários usuários, grupos e funções em seu Conta da AWS. As políticas AWS gerenciadas incluem políticas gerenciadas e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte [Escolher entre políticas gerenciadas e políticas em linha](#) no Guia do usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o atributo ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse atributo e em que condições. Você deve [especificar uma entidade principal](#) em uma política baseada em recursos. Os diretores podem incluir contas, usuários, funções, usuários federados ou. Serviços da AWS

Políticas baseadas em recursos são políticas em linha localizadas nesse serviço. Você não pode usar políticas AWS gerenciadas do IAM em uma política baseada em recursos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais diretores (membros da conta, usuários ou funções) têm permissões para acessar um recurso. ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

O Amazon S3 e o AWS WAF Amazon VPC são exemplos de serviços que oferecem suporte. ACLs Para saber mais ACLs, consulte a [visão geral da lista de controle de acesso \(ACL\)](#) no Guia do desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

AWS oferece suporte a tipos de políticas adicionais menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- **Limites de permissões:** um limite de permissões é um recurso avançado no qual você define o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou perfil do IAM). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade de uma entidade com seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou o perfil no campo `Principal` não são limitadas pelo limite de permissões. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações sobre limites de permissões, consulte [Limites de permissões para identidades do IAM](#) no Guia do usuário do IAM.
- **Políticas de controle de serviço (SCPs)** — SCPs são políticas JSON que especificam as permissões máximas para uma organização ou unidade organizacional (OU) em AWS Organizations. AWS Organizations é um serviço para agrupar e gerenciar centralmente várias Contas da AWS que sua empresa possui. Se você habilitar todos os recursos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as suas contas. O SCP limita as permissões para entidades nas contas dos membros, incluindo cada uma Usuário raiz da conta da AWS. Para obter mais informações sobre Organizations e SCPs, consulte [Políticas de controle de serviços](#) no Guia AWS Organizations do Usuário.
- **Políticas de controle de recursos (RCPs)** — RCPs são políticas JSON que você pode usar para definir o máximo de permissões disponíveis para recursos em suas contas sem atualizar as políticas do IAM anexadas a cada recurso que você possui. O RCP limita as permissões para recursos nas contas dos membros e pode afetar as permissões efetivas para identidades, incluindo a Usuário raiz da conta da AWS, independentemente de pertencerem à sua organização. Para obter mais informações sobre Organizations e RCPs, incluindo uma lista Serviços da

AWS desse suporte RCPs, consulte [Políticas de controle de recursos \(RCPs\)](#) no Guia AWS Organizations do usuário.

- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para um perfil ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou do perfil e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em recursos. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte [Políticas de sessão](#) no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como AWS determinar se uma solicitação deve ser permitida quando vários tipos de políticas estão envolvidos, consulte [Lógica de avaliação de políticas](#) no Guia do usuário do IAM.

Como AWS AppSync funciona com o IAM

Antes de usar o IAM para gerenciar o acesso AWS AppSync, saiba com quais recursos do IAM estão disponíveis para uso AWS AppSync.

Recursos do IAM que você pode usar com AWS AppSync

Atributo do IAM	AWS AppSync apoio
Políticas baseadas em identidade	Sim
Políticas baseadas em recurso	Não
Ações de políticas	Sim
Recursos de políticas	Sim
Chaves de condição de políticas	Não
ACLs	Não

Atributo do IAM	AWS AppSync apoio
ABAC (tags em políticas)	Parcial
Credenciais temporárias	Sim
Sessões de acesso direto (FAS)	Parcial
Perfis de serviço	Não
Perfis vinculados ao serviço	Parcial

Para ter uma visão de alto nível de como AWS AppSync e outros AWS serviços funcionam com a maioria dos recursos do IAM, consulte [AWS os serviços que funcionam com o IAM](#) no Guia do usuário do IAM.

Políticas baseadas em identidade para AWS AppSync

Compatível com políticas baseadas em identidade: sim

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário do IAM, grupo de usuários ou perfil. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte [Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente](#) no Guia do Usuário do IAM.

Com as políticas baseadas em identidade do IAM, é possível especificar ações e recursos permitidos ou negados, assim como as condições sob as quais as ações são permitidas ou negadas. Você não pode especificar a entidade principal em uma política baseada em identidade porque ela se aplica ao usuário ou perfil ao qual ela está anexada. Para saber mais sobre todos os elementos que podem ser usados em uma política JSON, consulte [Referência de elemento de política JSON do IAM](#) no Guia do usuário do IAM.

Exemplos de políticas baseadas em identidade para AWS AppSync

Para ver exemplos de políticas AWS AppSync baseadas em identidade, consulte. [Políticas baseadas em identidade para o AWS AppSync](#)

Políticas baseadas em recursos dentro AWS AppSync

Compatibilidade com políticas baseadas em recursos: não

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o atributo ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse atributo e em que condições. Você deve [especificar uma entidade principal](#) em uma política baseada em recursos. Os diretores podem incluir contas, usuários, funções, usuários federados ou. Serviços da AWS

Para permitir o acesso entre contas, você pode especificar uma conta inteira ou as entidades do IAM em outra conta como a entidade principal em uma política baseada em recursos. Adicionar uma entidade principal entre contas à política baseada em recurso é apenas metade da tarefa de estabelecimento da relação de confiança. Quando o principal e o recurso são diferentes Contas da AWS, um administrador do IAM na conta confiável também deve conceder permissão à entidade principal (usuário ou função) para acessar o recurso. Eles concedem permissão ao anexar uma política baseada em identidade para a entidade. No entanto, se uma política baseada em recurso conceder acesso a uma entidade principal na mesma conta, nenhuma política baseada em identidade adicional será necessária. Consulte mais informações em [Acesso a recursos entre contas no IAM](#) no Guia do usuário do IAM.

Ações políticas para AWS AppSync

Compatível com ações de políticas: sim

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Action` de uma política JSON descreve as ações que podem ser usadas para permitir ou negar acesso em uma política. As ações de política geralmente têm o mesmo nome da operação de AWS API associada. Existem algumas exceções, como ações somente de permissão, que não têm uma operação de API correspondente. Algumas operações também exigem várias ações em uma política. Essas ações adicionais são chamadas de ações dependentes.

Incluem ações em uma política para conceder permissões para executar a operação associada.

Para ver uma lista de AWS AppSync ações, consulte [Ações definidas por AWS AppSync](#) na Referência de Autorização de Serviço.

As ações de política AWS AppSync usam o seguinte prefixo antes da ação:

```
appsync
```

Para especificar várias ações em uma única declaração, separe-as com vírgulas.

```
"Action": [  
    "appsync:action1",  
    "appsync:action2"  
]
```

Para ver exemplos de políticas AWS AppSync baseadas em identidade, consulte. [Políticas baseadas em identidade para o AWS AppSync](#)

Recursos políticos para AWS AppSync

Compatível com recursos de políticas: sim

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento de política JSON `Resource` especifica o objeto ou os objetos aos quais a ação se aplica. As instruções devem incluir um elemento `Resource` ou `NotResource`. Como prática recomendada, especifique um recurso usando seu [nome do recurso da Amazon \(ARN\)](#). Isso pode ser feito para ações que oferecem compatibilidade com um tipo de recurso específico, conhecido como permissões em nível de recurso.

Para ações que não oferecem compatibilidade com permissões em nível de recurso, como operações de listagem, use um curinga (*) para indicar que a instrução se aplica a todos os recursos.

```
"Resource": "*"
```

Para ver uma lista dos tipos de AWS AppSync recursos e seus ARNs, consulte [Recursos definidos por AWS AppSync](#) na Referência de Autorização de Serviço. Para saber com quais ações é possível especificar o ARN de cada atributo, consulte [Ações definidas pelo AWS AppSync](#).

Para ver exemplos de políticas AWS AppSync baseadas em identidade, consulte [Políticas baseadas em identidade para o AWS AppSync](#)

Chaves de condição de política para AWS AppSync

Compatível com chaves de condição de política específicas de serviço: não

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Condition` (ou bloco `Condition`) permite que você especifique condições nas quais uma instrução estiver em vigor. O elemento `Condition` é opcional. É possível criar expressões condicionais que usem [agentes de condição](#), como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação.

Se você especificar vários elementos de `Condition` em uma declaração ou várias chaves em um único elemento de `Condition`, a AWS os avaliará usando uma operação lógica AND. Se você especificar vários valores para uma única chave de condição, AWS avalia a condição usando uma OR operação lógica. Todas as condições devem ser atendidas antes que as permissões da instrução sejam concedidas.

Você também pode usar variáveis de espaço reservado ao especificar condições. Por exemplo, é possível conceder a um usuário do IAM permissão para acessar um recurso somente se ele estiver marcado com seu nome de usuário do IAM. Para obter mais informações, consulte [Elementos da política do IAM: variáveis e tags](#) no Guia do usuário do IAM.

AWS suporta chaves de condição globais e chaves de condição específicas do serviço. Para ver todas as chaves de condição AWS globais, consulte as [chaves de contexto de condição AWS global](#) no Guia do usuário do IAM.

Para ver uma lista de chaves de AWS AppSync condição, consulte [Chaves de condição AWS AppSync](#) na Referência de autorização de serviço. Para saber com quais ações e recursos você pode usar uma chave de condição, consulte [Ações definidas por AWS AppSync](#).

Para ver exemplos de políticas AWS AppSync baseadas em identidade, consulte [Políticas baseadas em identidade para o AWS AppSync](#)

Listas de controle de acesso (ACLs) em AWS AppSync

Suportes ACLs: Não

As listas de controle de acesso (ACLs) controlam quais diretores (membros da conta, usuários ou funções) têm permissões para acessar um recurso. ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

Controle de acesso baseado em atributos (ABAC) com AWS AppSync

Compatível com ABAC (tags em políticas): parcial

O controle de acesso por atributo (ABAC) é uma estratégia de autorização que define as permissões com base em atributos. Em AWS, esses atributos são chamados de tags. Você pode anexar tags a entidades do IAM (usuários ou funções) e a vários AWS recursos. Marcar de entidades e atributos é a primeira etapa do ABAC. Em seguida, você cria políticas de ABAC para permitir operações quando a tag da entidade principal corresponder à tag do recurso que ela estiver tentando acessar.

O ABAC é útil em ambientes que estão crescendo rapidamente e ajuda em situações em que o gerenciamento de políticas se torna um problema.

Para controlar o acesso baseado em tags, forneça informações sobre as tags no [elemento de condição](#) de uma política usando as `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou chaves de condição `aws:TagKeys`.

Se um serviço for compatível com as três chaves de condição para cada tipo de recurso, o valor será Sim para o serviço. Se um serviço for compatível com as três chaves de condição somente para alguns tipos de recursos, o valor será Parcial

Para obter mais informações sobre o ABAC, consulte [Definir permissões com autorização do ABAC](#) no Guia do usuário do IAM. Para visualizar um tutorial com etapas para configurar o ABAC, consulte [Usar controle de acesso baseado em atributos \(ABAC\)](#) no Guia do usuário do IAM.

Usando credenciais temporárias com AWS AppSync

Compatível com credenciais temporárias: sim

Alguns Serviços da AWS não funcionam quando você faz login usando credenciais temporárias. Para obter informações adicionais, incluindo quais Serviços da AWS funcionam com credenciais temporárias, consulte Serviços da AWS “[Trabalhe com o IAM](#)” no Guia do usuário do IAM.

Você está usando credenciais temporárias se fizer login AWS Management Console usando qualquer método, exceto um nome de usuário e senha. Por exemplo, quando você acessa AWS usando o link de login único (SSO) da sua empresa, esse processo cria automaticamente credenciais temporárias. Você também cria automaticamente credenciais temporárias quando faz login no

console como usuário e, em seguida, alterna perfis. Para obter mais informações sobre como alternar funções, consulte [Alternar para um perfil do IAM \(console\)](#) no Guia do usuário do IAM.

Você pode criar manualmente credenciais temporárias usando a AWS API AWS CLI ou. Em seguida, você pode usar essas credenciais temporárias para acessar AWS. AWS recomenda que você gere credenciais temporárias dinamicamente em vez de usar chaves de acesso de longo prazo. Para obter mais informações, consulte [Credenciais de segurança temporárias no IAM](#).

Sessões de acesso direto para AWS AppSync

Suporte ao recurso de encaminhamento de sessões de acesso (FAS): parcial

Quando você usa um usuário ou uma função do IAM para realizar ações AWS, você é considerado um principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O FAS usa as permissões do diretor chamando um AWS service (Serviço da AWS), combinadas com a solicitação AWS service (Serviço da AWS) para fazer solicitações aos serviços posteriores. As solicitações do FAS são feitas somente quando um serviço recebe uma solicitação que requer interações com outros Serviços da AWS ou com recursos para ser concluída. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte [Sessões de acesso direto](#).

Funções de serviço para AWS AppSync

Compatível com perfis de serviço: não

O perfil de serviço é um [perfil do IAM](#) que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte [Criar um perfil para delegar permissões a um AWS service \(Serviço da AWS\)](#) no Guia do Usuário do IAM.

Warning

Alterar as permissões de uma função de serviço pode interromper AWS AppSync a funcionalidade. Edite as funções de serviço somente quando AWS AppSync fornecer orientação para fazer isso.

Funções vinculadas a serviços para AWS AppSync

Compatível com perfis vinculados ao serviço: parcialmente

Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. AWS service (Serviço da AWS) O serviço pode presumir o perfil para executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar as permissões para perfis vinculados ao serviço.

Para obter detalhes sobre como criar ou gerenciar perfis vinculadas a serviços, consulte [AWS Serviços que funcionam com o IAM](#) no Guia do usuário do IAM.. Encontre um serviço na tabela que inclua um Yes na coluna Perfil vinculado ao serviço. Escolha o link Sim para visualizar a documentação do perfil vinculado a serviço desse serviço.

Políticas baseadas em identidade para o AWS AppSync

Por padrão, usuários e perfis não têm permissão para criar ou modificar recursos do AWS AppSync. Eles também não podem realizar tarefas usando a AWS API AWS Management Console, AWS Command Line Interface (AWS CLI) ou. Para conceder permissão aos usuários para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis e os usuários podem assumir os perfis.

Para aprender a criar uma política baseada em identidade do IAM ao usar esses documentos de política em JSON de exemplo, consulte [Criar políticas do IAM \(console\)](#) no Guia do usuário do IAM.

Para obter detalhes sobre ações e tipos de recursos definidos por AWS AppSync, incluindo o formato do ARNs para cada um dos tipos de recursos, consulte [Ações, recursos e chaves de condição AWS AppSync na Referência de Autorização de Serviço](#).

Para conhecer as melhores práticas para criar e configurar políticas baseadas em identidade do IAM, consulte [the section called “Práticas recomendadas das políticas do IAM”](#).

Para obter uma lista de políticas baseadas em identidade do IAM para AWS AppSync, consulte. [AWS políticas gerenciadas para AWS AppSync](#)

Tópicos

- [Usar o console do AWS AppSync](#)
- [Permitir que os usuários visualizem suas próprias permissões](#)
- [Acessar um bucket do Amazon S3](#)
- [Visualização AWS AppSync widgets com base em tags](#)

- [AWS políticas gerenciadas para AWS AppSync](#)

Usar o console do AWS AppSync

Para acessar o AWS AppSync console, você deve ter um conjunto mínimo de permissões. Essas permissões devem permitir que você liste e visualize detalhes sobre os AWS AppSync recursos em seu Conta da AWS. Caso crie uma política baseada em identidade mais restritiva que as permissões mínimas necessárias, o console não funcionará como pretendido para entidades (usuários ou perfis) com essa política.

Você não precisa permitir permissões mínimas do console para usuários que estão fazendo chamadas somente para a API AWS CLI ou para a AWS API. Em vez disso, permita o acesso somente a ações que correspondam à operação de API que estiverem tentando executar.

Para garantir que os usuários e funções do IAM ainda possam usar o AWS AppSync console, anexe também a política AWS AppSync ConsoleAccess ou a política ReadOnly AWS gerenciada às entidades. Para obter informações, consulte [Adicionar permissões a um usuário](#) no Guia do usuário do IAM.

Permitir que os usuários visualizem suas próprias permissões

Este exemplo mostra como criar uma política que permita que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou programaticamente usando a API AWS CLI ou 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": "*"
    }
  ]
}

```

Acessar um bucket do Amazon S3

Neste exemplo, você deseja conceder a um usuário do IAM em sua AWS conta acesso a um dos seus buckets do Amazon S3, `examplebucket`. Você também deseja permitir que o usuário adicione, atualize e exclua objetos.

Além de conceder as permissões `s3:PutObject`, `s3:GetObject` e `s3:DeleteObject` ao usuário, a política também concede as permissões `s3:ListAllMyBuckets`, `s3:GetBucketLocation` e `s3:ListBucket`. Estas são permissões adicionais, exigidas pelo console. As ações `s3:PutObjectAcl` e `s3:GetObjectAcl` também são necessárias para copiar, recortar e colar objetos no console. Para obter uma demonstração de exemplo que concede permissões aos usuários e testa-os ao usar o console, consulte [Demonstração de exemplo: Usar políticas de usuário para controlar o acesso a seu 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/*"
  }
]
}

```

Visualização AWS AppSync *widgets* com base em tags

É possível utilizar condições na política baseada em identidade para controlar o acesso aos recursos do AWS AppSync com base em tags. Este exemplo mostra como você pode criar uma política que permite visualizar um *widget*. No entanto, a permissão é concedida somente se a tag `Owner` de *widget* tiver o valor do nome desse usuário. Essa política também concede as permissões necessárias concluir essa ação no console.

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}"}
        }
      }
    ]
  }
}

```

É possível anexar essa política aos usuários do IAM na sua conta. Se um usuário chamado richard-roe tentar visualizar um *widget* do AWS AppSync, o *widget* deverá estar marcado como Owner=richard-roe ou owner=richard-roe. Caso contrário, ele terá o acesso negado. A chave da tag de condição Owner corresponde a Owner e a owner porque os nomes das chaves de condição não fazem distinção entre maiúsculas e minúsculas. Para obter mais informações, consulte [Elementos de política JSON do IAM: condição](#) no Guia do usuário do IAM.

AWS políticas gerenciadas para AWS AppSync

Para adicionar permissões a usuários, grupos e funções, é mais fácil usar políticas AWS gerenciadas do que escrever políticas você mesmo. É necessário tempo e experiência para [criar políticas do IAM gerenciadas pelo cliente](#) que fornecem à sua equipe apenas as permissões necessárias. Para começar rapidamente, você pode usar nossas políticas AWS gerenciadas. Essas políticas abrangem casos de uso comuns e estão disponíveis na sua Conta da AWS. Para obter mais informações sobre políticas AWS gerenciadas, consulte [políticas AWS gerenciadas](#) no Guia do usuário do IAM.

AWS os serviços mantêm e atualizam as políticas AWS gerenciadas. Você não pode alterar as permissões nas políticas AWS gerenciadas. Ocasionalmente, os serviços adicionam permissões adicionais a uma política AWS gerenciada para oferecer suporte a novos recursos. Esse tipo

de atualização afeta todas as identidades (usuários, grupos e funções) em que a política está anexada. É mais provável que os serviços atualizem uma política AWS gerenciada quando um novo recurso é lançado ou quando novas operações são disponibilizadas. Os serviços não removem as permissões de uma política AWS gerenciada, portanto, as atualizações de políticas não violarão suas permissões existentes.

Além disso, AWS oferece suporte a políticas gerenciadas para funções de trabalho que abrangem vários serviços. Por exemplo, a política `ReadOnlyAccess` AWS gerenciada fornece acesso somente de leitura a todos os AWS serviços e recursos. Quando um serviço lança um novo recurso, AWS adiciona permissões somente de leitura para novas operações e recursos. Para obter uma lista e descrições das políticas de perfis de trabalho, consulte [Políticas gerenciadas pela AWS para perfis de trabalho](#) no Guia do usuário do IAM.

AWS política gerenciada: `AWSAppSyncInvokeFullAccess`

Use a política `AWSAppSyncInvokeFullAccess` AWS gerenciada para permitir que seus administradores acessem o AWS AppSync serviço por meio do console ou de forma independente.

É possível anexar a política `AWSAppSyncInvokeFullAccess` às identidades do IAM.

Detalhes das permissões

Esta política inclui as seguintes permissões.

- **AWS AppSync**— Permite acesso administrativo total a todos os recursos em AWS AppSync

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "appsync:GraphQL",
        "appsync:GetGraphqlApi",
        "appsync:ListGraphqlApis",
```

```

        "appsync:ListApiKeys"
      ],
      "Resource": "*"
    }
  ]
}

```

AWS política gerenciada: `AWSAppSyncSchemaAuthor`

Use a política `AWSAppSyncSchemaAuthor` AWS gerenciada para permitir que os usuários do IAM acessem para criar, atualizar e consultar seus esquemas do GraphQL. Para obter informações sobre o que os usuários podem fazer com essas permissões, consulte [Projetando o GraphQL APIs com AWS AppSync](#).

É possível anexar a política `AWSAppSyncSchemaAuthor` às identidades do IAM.

Detalhes das permissões

Esta política inclui as seguintes permissões.

- AWS AppSync – Permite realizar as ações a seguir:
 - Criar esquemas do GraphQL
 - Permitir a criação, modificação e exclusão de tipos, resolvedores e funções do GraphQL
 - Avaliar a lógica do modelo de solicitação e resposta
 - Avaliar o código com um runtime e contexto
 - Enviando consultas do GraphQL para o GraphQL APIs
 - Recuperar dados do GraphQL

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 política gerenciada: `AWSApp SyncPushToCloudWatchLogs`

AWS AppSync usa CloudWatch a Amazon para monitorar o desempenho do seu aplicativo gerando registros que você pode usar para solucionar problemas e otimizar suas solicitações do GraphQL.

Para obter mais informações, consulte [Usando CloudWatch para monitorar e registrar dados da API GraphQL](#).

Use a política `AWSAppSyncPushToCloudWatchLogs` AWS gerenciada AWS AppSync para permitir o envio de registros para a CloudWatch conta de um usuário do IAM.

É possível anexar a política `AWSAppSyncPushToCloudWatchLogs` às identidades do IAM.

Detalhes das permissões

Esta política inclui as seguintes permissões.

- `CloudWatch Logs`— Permite AWS AppSync criar grupos de registros e fluxos com nomes especificados. AWS AppSync envia eventos de log para o fluxo de log especificado.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS política gerenciada: `AWSAppSyncAdministrator`

Use a política `AWSAppSyncAdministrator` AWS gerenciada para permitir que seus administradores acessem tudo AWS AppSync, exceto o AWS console.

Você pode anexar `AWSAppSyncAdministrator` às suas entidades do IAM. AWS AppSync também anexa essa política a uma função de serviço que permite que ela execute ações em seu nome.

Detalhes das permissões

Esta política inclui as seguintes permissões.

- **AWS AppSync**— Permite acesso administrativo total a todos os recursos em AWS AppSync
- **IAM** – Permite realizar as ações a seguir:
 - Criação de funções vinculadas a serviços para permitir AWS AppSync a análise de recursos em outros serviços em seu nome
 - Excluir perfis vinculadas ao serviço
 - Transferir funções vinculadas a serviços para outros AWS serviços para assumir a função posteriormente e realizar ações em seu nome

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 política gerenciada: AWSApp SyncServiceRolePolicy

Use a política AWSAppSyncServiceRolePolicy AWS gerenciada para permitir o acesso aos AWS serviços e recursos que AWS AppSync usa ou gerencia.

Não é possível anexar a AWSAppSyncServiceRolePolicy às entidades do IAM. Essa política está vinculada a uma função vinculada ao serviço que permite AWS AppSync realizar ações em seu nome. Para obter mais informações, consulte [Funções vinculadas a serviços para AWS AppSync](#).

Detalhes das permissões

Esta política inclui as seguintes permissões.

- X-Ray— AWS AppSync usa AWS X-Ray para coletar dados sobre solicitações feitas em seu aplicativo. Para obter mais informações, consulte [Usando AWS X-Ray para rastrear solicitações em AWS AppSync](#).

Essa política permite as seguintes ações:

- Recuperar regras de amostragem e os resultados
- Enviar dados de rastreamento para o daemon do X-Ray.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "xray:PutTraceSegments",
        "xray:PutTelemetryRecords",
        "xray:GetSamplingTargets",
        "xray:GetSamplingRules",
        "xray:GetSamplingStatisticSummaries"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Atualizações do AWS AppSync para políticas gerenciadas pela AWS

Veja detalhes sobre as atualizações das políticas AWS gerenciadas AWS AppSync desde que esse serviço começou a rastrear essas alterações. Para receber alertas automáticos sobre alterações nessa página, assine o feed RSS na página Histórico do AWS AppSync documento.

Alteração	Descrição	Data
AWSAppSyncSchemaAuthor - Atualização em uma política existente	Foi adicionada uma ação da política EvaluateCode para permitir que os usuários avaliem o código com um runtime e contexto.	7 de fevereiro de 2023
AWSAppSyncSchemaAuthor - Atualização em uma política existente	Ações de política adicionadas para permitir as funções de listar, obter, criar, atualizar e excluir de uma API. Foi adicionada uma ação de política EvaluateModelingTemplate para permitir que os usuários avaliem a lógica do modelo de mapeamento do resolutor de solicitações e respostas. Ações de políticas adicionadas para permitir a marcação de recursos.	25 de agosto de 2022
AWS AppSync começou a rastrear as alterações	AWS AppSync começou a rastrear as mudanças em suas políticas AWS gerenciadas.	25 de agosto de 2022

Solução de problemas AWS AppSync de identidade e acesso

Use as informações a seguir para ajudá-lo a diagnosticar e corrigir problemas comuns que você pode encontrar ao trabalhar com AWS AppSync um IAM.

Não tenho autorização para executar uma ação no AWS AppSync

Se isso AWS Management Console indicar que você não está autorizado a realizar uma ação, entre em contato com o administrador para obter ajuda. O administrador é a pessoa que forneceu o seu nome de usuário e senha.

O erro do exemplo a seguir ocorre quando o usuário do IAM `mateojackson` tenta usar o console para visualizar detalhes sobre um recurso fictício do *my-example-widget*, mas não tem as permissões fictícias do appsync: *GetWidget*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
appsync:GetWidget on resource: my-example-widget
```

Neste caso, Mateo pede ao administrador para atualizar suas políticas para permitir a ele o acesso ao recurso *my-example-widget* usando a ação appsync: *GetWidget*.

Não estou autorizado a realizar iam: PassRole

Se você receber uma mensagem de erro informando que não está autorizado a executar a ação `iam:PassRole`, as suas políticas devem ser atualizadas para permitir que você passe uma função para o AWS AppSync.

Alguns Serviços da AWS permitem que você passe uma função existente para esse serviço em vez de criar uma nova função de serviço ou uma função vinculada ao serviço. Para fazê-lo, você deve ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta utilizar o console para executar uma ação no AWS AppSync. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero visualizar minhas chaves de acesso

Depois de criar suas chaves de acesso de usuário do IAM, é possível visualizar seu ID da chave de acesso a qualquer momento. No entanto, você não pode visualizar sua chave de acesso secreta novamente. Se você perder sua chave secreta, crie um novo par de chaves de acesso.

As chaves de acesso consistem em duas partes: um ID de chave de acesso (por exemplo, AKIAIOSFODNN7EXAMPLE) e uma chave de acesso secreta (por exemplo, wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY). Como um nome de usuário e uma senha, você deve usar o ID da chave de acesso e a chave de acesso secreta em conjunto para autenticar suas solicitações. Gerencie suas chaves de acesso de forma tão segura quanto você gerencia seu nome de usuário e sua senha.

Important

Não forneça as chaves de acesso a terceiros, mesmo que seja para ajudar a [encontrar o ID de usuário canônico](#). Ao fazer isso, você pode dar a alguém acesso permanente ao seu Conta da AWS.

Ao criar um par de chaves de acesso, você é solicitado a guardar o ID da chave de acesso e a chave de acesso secreta em um local seguro. A chave de acesso secreta só está disponível no momento em que é criada. Se você perder sua chave de acesso secreta, será necessário adicionar novas chaves de acesso para seu usuário do IAM. Você pode ter no máximo duas chaves de acesso. Se você já tiver duas, você deverá excluir um par de chaves para poder criar um novo. Para visualizar as instruções, consulte [Gerenciar chaves de acesso](#) no Guia do usuário do IAM.

Sou administrador e quero permitir que outras pessoas acessem AWS AppSync

Para permitir que outras pessoas acessem AWS AppSync, você deve conceder permissão às pessoas ou aplicativos que precisam de acesso. Se você estiver usando o AWS IAM Identity Center para gerenciar pessoas e aplicações, atribua conjuntos de permissões a usuários ou grupos para definir o nível de acesso. Os conjuntos de permissões criam e atribuem automaticamente políticas do IAM aos perfis do IAM associados à pessoa ou aplicação. Para ter mais informações, consulte [Conjuntos de permissões](#) no Guia do usuário do AWS IAM Identity Center .

Se você não estiver usando o Centro de Identidade do IAM, deverá criar entidades do IAM (usuários ou perfis) para as pessoas ou aplicações que precisam de acesso. Você deve anexar uma

política à entidade que concede a eles as permissões corretas no AWS AppSync. Depois que as permissões forem concedidas, forneça as credenciais ao usuário ou desenvolvedor da aplicação. Eles usarão essas credenciais para acessar AWS. Para saber mais sobre como criar grupos, políticas, permissões e usuários do IAM, consulte [Identidades do IAM](#) e [Políticas e permissões no IAM](#) no Guia do usuário do IAM.

Quero permitir que pessoas fora da minha AWS conta acessem meus AWS AppSync recursos

Você pode criar um perfil que os usuários de outras contas ou pessoas fora da organização podem usar para acessar seus recursos. É possível especificar quem é confiável para assumir o perfil. Para serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se é AWS AppSync compatível com esses recursos, consulte [Como AWS AppSync funciona com o IAM](#).
- Para saber como fornecer acesso aos seus recursos em todas as Contas da AWS que você possui, consulte Como [fornecer acesso a um usuário do IAM em outra Conta da AWS que você possui](#) no Guia do usuário do IAM.
- Para saber como fornecer acesso aos seus recursos a terceiros Contas da AWS, consulte Como [fornecer acesso Contas da AWS a terceiros](#) no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte [Conceder acesso a usuários autenticados externamente \(federação de identidades\)](#) no Guia do usuário do IAM.
- Para saber a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte [Acesso a recursos entre contas no IAM](#) no Guia do usuário do IAM.

Registrando chamadas de AWS AppSync API com AWS CloudTrail

AWS AppSync é integrado com AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, função ou AWS serviço em AWS AppSync. CloudTrail captura chamadas de API AWS AppSync como eventos. As chamadas capturadas incluem chamadas do AWS AppSync console e chamadas de código para as operações AWS AppSync da API. Se você criar uma trilha, poderá habilitar a entrega contínua de CloudTrail eventos para um bucket do Amazon S3, incluindo

eventos para. AWS AppSync Se você não configurar uma trilha, ainda poderá ver os eventos mais recentes no CloudTrail console no Histórico de eventos. Usando as informações coletadas por CloudTrail, você pode determinar a solicitação que foi feita AWS AppSync, o endereço IP do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e detalhes adicionais.

Para saber mais CloudTrail, consulte o [Guia AWS CloudTrail do usuário](#).

AWS AppSync informações em CloudTrail

CloudTrail é ativado em sua AWS conta quando você cria a conta. Quando a atividade ocorre em AWS AppSync, essa atividade é registrada em um CloudTrail evento junto com outros eventos AWS de serviço no histórico de eventos. Você pode visualizar, pesquisar e baixar eventos recentes em sua AWS conta. Para obter mais informações, consulte [Visualização de eventos com histórico de CloudTrail eventos](#).

Para um registro contínuo dos eventos em sua AWS conta, incluindo eventos para AWS AppSync, crie uma trilha. Uma trilha permite CloudTrail entregar arquivos de log para um bucket do Amazon S3. Por padrão, quando você cria uma trilha no console, a trilha se aplica a todas as AWS regiões. A trilha registra eventos de todas as regiões na AWS partição e entrega os arquivos de log ao bucket do Amazon S3 que você especificar. Além disso, você pode configurar outros AWS serviços para analisar e agir com base nos dados de eventos coletados nos CloudTrail registros. Para obter mais informações, consulte:

- [Visão geral da criação de uma trilha](#)
- [CloudTrail serviços e integrações suportados](#)
- [Configurando notificações do Amazon SNS para CloudTrail](#)
- [Recebendo arquivos de CloudTrail log de várias regiões](#) e [Recebendo arquivos de CloudTrail log de várias contas](#)

AWS AppSync suporta o registro de chamadas feitas por meio da AWS AppSync API. No momento, as chamadas para o seu APIs, assim como as chamadas feitas para os resolvedores, não são registradas pelo AWS AppSync . CloudTrail

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar o seguinte:

- Se a solicitação foi feita com credenciais de usuário root ou AWS Identity and Access Management (IAM).

- Se a solicitação foi feita com credenciais de segurança temporárias de uma função ou de um usuário federado.
- Se a solicitação foi feita por outro AWS serviço.

Para obter mais informações, consulte [Elemento `userIdentity` do CloudTrail](#).

Entendendo as entradas do arquivo de AWS AppSync log

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log para um bucket do Amazon S3 que você especificar. CloudTrail os arquivos de log contêm uma ou mais entradas de log. Um evento representa uma única solicitação de qualquer fonte e inclui informações sobre a ação solicitada, a data e a hora da ação, os parâmetros da solicitação e assim por diante. CloudTrail os arquivos de log não são um rastreamento de pilha ordenado das chamadas públicas de API, portanto, eles não aparecem em nenhuma ordem específica.

O exemplo a seguir mostra uma entrada de CloudTrail registro que demonstra a `GetGraphQLApi` ação realizada por meio do AWS AppSync console:

```
{
  "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"
  }
}

```

O exemplo a seguir mostra uma entrada de CloudTrail registro que demonstra a CreateApiKey ação realizada por meio do 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"
}
```

Práticas recomendadas de segurança para AWS AppSync

Proteger AWS AppSync é mais do que simplesmente ativar algumas alavancas ou configurar o registro. As seções a seguir abordam as práticas de segurança recomendadas que variam dependendo de como você usa o serviço.

Entenda os métodos de autenticação

AWS AppSync fornece várias maneiras de autenticar seus usuários no seu GraphQL APIs. Cada método tem vantagens e desvantagens para segurança, auditabilidade e usabilidade.

Os métodos de autenticação de registro a seguir estão disponíveis:

- Os grupos de usuários do Amazon Cognito permitem que sua API GraphQL use atributos do usuário para controle de acesso e filtragem refinados.
- Os tokens de API têm uma vida útil limitada e são apropriados para sistemas automatizados, como sistemas de integração contínua e integração com sistemas externos APIs.
- AWS Identity and Access Management (IAM) é apropriado para aplicativos internos gerenciados em seu Conta da AWS.
- O OpenID Connect permite controlar e federar o acesso com o protocolo OpenID Connect.

Para obter mais informações sobre autenticação e autorização em AWS AppSync, consulte [Configurando autorização e autenticação para proteger seu GraphQL APIs](#).

Entenda como as alterações na configuração da API se propagam

Quando você salva as alterações na configuração da sua API, AWS AppSync começa a propagar as alterações. Até que sua alteração de configuração seja propagada, AWS AppSync continue veiculando seu conteúdo da configuração anterior. Depois que sua alteração de configuração for propagada, AWS AppSync imediatamente começará a veicular seu conteúdo com base na nova configuração. Enquanto AWS AppSync propagamos suas alterações para uma API, não podemos determinar se a API está veiculando seu conteúdo com base na configuração anterior ou na nova configuração.

Usar o TLS para resolvedores HTTP

Ao usar resolvedores HTTP, é preciso utilizar conexões protegidas pelo TLS (HTTPS) sempre que possível. Para obter uma lista completa dos certificados TLS AWS AppSync confiáveis, consulte.

[Autoridades de certificação \(CA\) reconhecidas AWS AppSync por quatro endpoints HTTPS](#)

Usar perfis com o mínimo de permissões possível

Ao usar resolvedores como o [resolvedor do DynamoDB](#), utilize perfis que forneçam a visão mais restritiva dos seus recursos, como as tabelas do Amazon DynamoDB.

Práticas recomendadas das políticas do IAM

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir AWS AppSync recursos em sua conta. Essas ações podem incorrer em custos para sua Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas AWS gerenciadas e avance para as permissões de privilégios mínimos — Para começar a conceder permissões aos seus usuários e cargas de trabalho, use as políticas AWS gerenciadas que concedem permissões para muitos casos de uso comuns. Eles estão disponíveis no seu Conta da AWS. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo AWS cliente que sejam específicas para seus casos de uso. Para obter mais informações, consulte [Políticas gerenciadas pela AWS](#) ou [Políticas gerenciadas pela AWS para funções de trabalho](#) no Guia do usuário do IAM.
- Aplique permissões de privilégio mínimo: ao definir permissões com as políticas do IAM, conceda apenas as permissões necessárias para executar uma tarefa. Você faz isso definindo as ações que podem ser executadas em recursos específicos sob condições específicas, também conhecidas como permissões de privilégio mínimo. Para obter mais informações sobre como usar o IAM para aplicar permissões, consulte [Políticas e permissões no IAM](#) no Guia do usuário do IAM.

- Use condições nas políticas do IAM para restringir ainda mais o acesso: você pode adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, você pode escrever uma condição de política para especificar que todas as solicitações devem ser enviadas usando SSL. Você também pode usar condições para conceder acesso às ações de serviço se elas forem usadas por meio de uma ação específica AWS service (Serviço da AWS), como AWS CloudFormation. Para obter mais informações, consulte [Elementos da política JSON do IAM: condição](#) no Guia do usuário do IAM.
- Use o IAM Access Analyzer para validar suas políticas do IAM a fim de garantir permissões seguras e funcionais: o IAM Access Analyzer valida as políticas novas e existentes para que elas sigam a linguagem de política do IAM (JSON) e as práticas recomendadas do IAM. O IAM Access Analyzer oferece mais de cem verificações de política e recomendações práticas para ajudar a criar políticas seguras e funcionais. Para obter mais informações, consulte [Validação de políticas do IAM Access Analyzer](#) no Guia do Usuário do IAM.
- Exigir autenticação multifator (MFA) — Se você tiver um cenário que exija usuários do IAM ou um usuário root, ative Conta da AWS a MFA para obter segurança adicional. Para exigir MFA quando as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para obter mais informações, consulte [Configuração de acesso à API protegido por MFA](#) no Guia do Usuário do IAM.

Para obter mais informações sobre as práticas recomendadas do IAM, consulte [Práticas recomendadas de segurança no IAM](#) no Guia do usuário do IAM.

AWS AppSync referência do resolvidor (JavaScript)

As seções a seguir contêm a referência do APPSYNC_JS tempo de execução e do JavaScript resolvidor:

- [JavaScript visão geral dos resolvidores](#) - Saiba mais sobre como os resolvidores funcionam em. AWS AppSync
- [Referência do objeto de contexto do resolvidor](#): saiba mais sobre o objeto de contexto e como ele é usado em resolvidores.
- [JavaScript recursos de tempo de execução para resolvidores e funções](#) - Saiba mais sobre os recursos de tempo de execução compatíveis e o uso de utilitários para simplificar o código.
- [JavaScript referência da função de resolução para o DynamoDB](#) - Saiba mais sobre como os resolvidores interagem com o DynamoDB.
- [JavaScript referência da função de resolução para OpenSearch](#) - Saiba mais sobre a estrutura de solicitações e respostas do resolvidor e as interações com o OpenSearch Serviço.
- [JavaScript Referência da função de resolução para Lambda](#) - Saiba mais sobre a estrutura de solicitações e respostas do resolvidor e as interações com o Lambda.
- [JavaScript referência da função de resolução para fonte de EventBridge dados](#) - Saiba mais sobre a estrutura de solicitação e resposta do resolvidor e as interações com EventBridge o.
- [JavaScript referência da função de resolução para nenhuma fonte de dados](#) - Saiba mais sobre a estrutura de solicitação e resposta do resolvidor e as interações com nenhuma fonte de dados.
- [JavaScript referência da função de resolução para HTTP](#) - Saiba mais sobre a estrutura de solicitação e resposta do resolvidor e as interações com endpoints HTTP.
- [JavaScript referência da função de resolução para Amazon RDS](#) - Saiba mais sobre a estrutura do resolvidor e as interações com o RDS.
- [JavaScript referência da função de resolução para o Amazon Bedrock](#) - Saiba mais sobre a estrutura do resolvidor e as interações com o Amazon Bedrock.

AWS AppSync JavaScript visão geral dos resolvidores

AWS AppSync permite que você responda às solicitações do GraphQL executando operações em suas fontes de dados. Para cada campo do GraphQL em que você deseja executar uma consulta, mutação ou assinatura, um resolvidor deve ser anexado.

Os resolvedores são os conectores entre o GraphQL e uma fonte de dados. Eles explicam AWS AppSync como traduzir uma solicitação recebida do GraphQL em instruções para sua fonte de dados de back-end e como traduzir a resposta dessa fonte de dados em uma resposta do GraphQL. Com AWS AppSync, você pode escrever seus resolvedores usando JavaScript e executá-los no ambiente AWS AppSync (APPSYNC_JS).

AWS AppSync permite escrever resolvedores de unidades ou resolvedores de pipeline compostos por várias AWS AppSync funções em um pipeline.

Atributos compatíveis de runtime

O AWS AppSync JavaScript tempo de execução fornece um subconjunto de JavaScript bibliotecas, utilitários e recursos. Para obter uma lista completa dos recursos e funcionalidades suportados pelo tempo de APPSYNC_JS execução, consulte [recursos JavaScript de tempo de execução para resolvedores e funções](#).

Resolvedores de unidade

Um resolvedor de unidade é composto de código que define um único manipulador de solicitação e resposta que é executado em uma fonte de dados. O manipulador da solicitação usa um objeto de contexto como argumento e retorna o payload da solicitação usado para chamar sua fonte de dados. O manipulador de respostas recebe uma payload da fonte de dados com o resultado da solicitação executada. O manipulador de resposta transforma o payload em uma resposta do GraphQL para resolver o campo GraphQL. No exemplo abaixo, um resolvedor recupera um item de uma fonte de dados do DynamoDB:

```
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
  return ddb.get({ key: { id: ctx.args.id } });
}

export const response = (ctx) => ctx.result;
```

Anatomia de um resolvedor de JavaScript pipeline

Um resolvedor de pipeline é composto de código que define um manipulador de solicitação e resposta e uma lista de funções. Cada função possui um manipulador de solicitação e resposta que

é executado em uma fonte de dados. Como um resolvidor de pipeline delega a execução a uma lista de funções, ele não está vinculado a nenhuma fonte de dados. Os resolvidores de unidade e funções que executam a operação mediante fontes de dados são primitivos.

Manipulador de solicitações do resolvidor de pipeline

O manipulador de solicitação de um resolvidor de pipeline, ou etapa Anterior, permite executar uma lógica de preparação antes de executar as funções definidas.

Lista de funções

A lista de funções que um resolvidor de pipeline executará em sequência. O resultado do manipulador da solicitação do resolvidor de pipeline é disponibilizado para a primeira função como `ctx.prev.result`. Cada resultado da avaliação da função está disponível para a próxima função como `ctx.prev.result`.

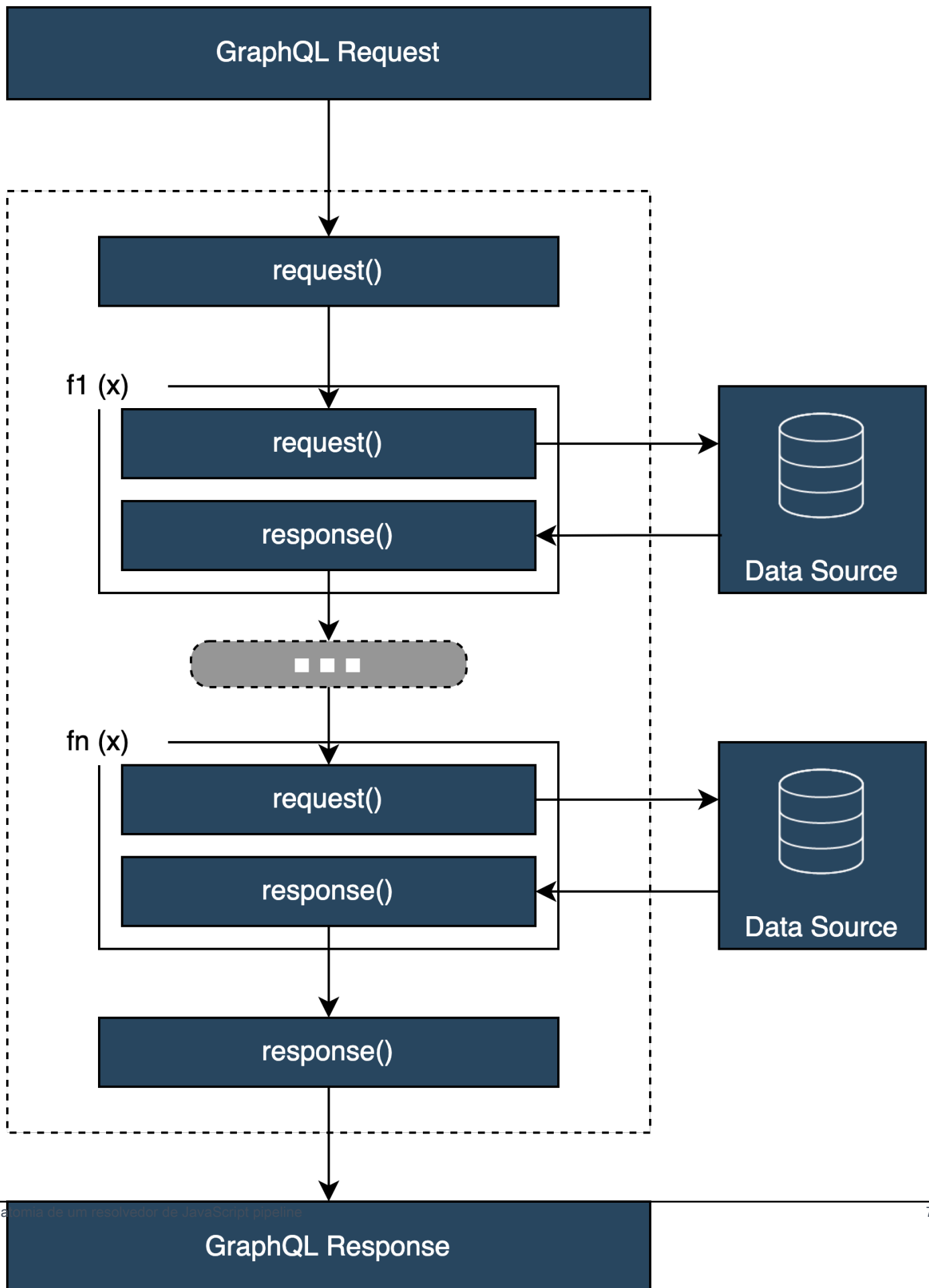
Manipulador de resposta do resolvidor de pipeline

O modelo de resposta de um resolvidor de pipeline permite executar uma lógica final na saída da última função para o tipo de campo do GraphQL esperado. A saída da última função na lista de funções está disponível no manipulador de resposta do resolvidor de pipeline, como `ctx.prev.result` ou `ctx.result`.

Fluxo de execução

Considerando um resolvidor de pipeline composto de duas funções, a lista abaixo representa o fluxo de execução quando o resolvidor é invocado:

1. Manipulador de solicitações do resolvidor de pipeline
2. Função 1: manipulador de solicitação de função
3. Função 1: invocação da fonte de dados
4. Função 1: manipulador de resposta de função
5. Função 2: manipulador de solicitação de função
6. Função 2: invocação da fonte de dados
7. Função 2: manipulador de resposta de função
8. Manipulador de resposta do resolvidor de pipeline



Utilitários integrados úteis do runtime **APPSYNC_JS**

Os utilitários a seguir podem ajudá-lo quando você estiver trabalhando com resolvedores de pipeline.

`ctx.stash`

O `stash` é um objeto disponibilizado dentro de cada manipulador de resposta e solicitação de resolvedor e função. A mesma instância `stash` passa por uma única execução do resolvedor. Isso significa que é possível usar o `stash` para enviar dados arbitrários entre os manipuladores de solicitações e respostas e entre as funções em um resolvedor de pipeline. Você pode testar o estoque como um JavaScript objeto normal.

`ctx.prev.result`

O `ctx.prev.result` representa o resultado da operação anterior que foi executada no pipeline. Se a operação anterior foi o manipulador de solicitações do resolvedor de pipeline, `ctx.prev.result` será disponibilizado para a primeira função no encadeamento. Se a operação anterior foi a primeira função, `ctx.prev.result` representa a saída da primeira função e será disponibilizado para a segunda função no pipeline. Se a operação anterior foi a última função, `ctx.prev.result` representa a saída da última função e será disponibilizado para o manipulador de resposta do resolvedor de pipeline.

`util.error`

O utilitário `util.error` é útil para gerar um erro de campo. Usar `util.error` dentro de um manipulador de solicitação ou resposta de função gera um erro de campo imediatamente, o que impede que funções subsequentes sejam executadas. Para obter mais detalhes e outras `util.error` assinaturas, visite [recursos de JavaScript tempo de execução para resolvedores e funções](#).

`util.appendError`

O `util.appendError` é semelhante a `util.error()`, com a principal distinção de que ele não interrompe a avaliação do manipulador. Em vez disso, ele sinaliza que ocorreu um erro com o campo, mas permite que o manipulador seja avaliado e, conseqüentemente, retorne dados. Usar `util.appendError` dentro de uma função não interromperá o fluxo de execução do pipeline. Para obter mais detalhes e outras `util.error` assinaturas, visite os [recursos de JavaScript tempo de execução para resolvedores e funções](#).

`runtime.earlyReturn`

A função `runtime.earlyReturn` permite que você gere resultados prematuramente para qualquer função de solicitação. Usar `runtime.earlyReturn` em um manipulador de solicitações do resolvidor retornará resultados do resolvidor. Chamá-lo em um manipulador de solicitação de função AWS AppSync retornará resultados a partir da função e continuará a execução até a próxima função no pipeline ou o manipulador de resposta do resolvidor.

Escrever resolvedores de pipeline

Um resolvidor de pipeline também tem um manipulador de solicitação e resposta para a execução das funções no pipeline: o manipulador de solicitação é executado antes da solicitação da primeira função, e o manipulador de resposta é executado após a resposta da última função. O manipulador de solicitação do resolvidor pode configurar dados para serem usados pelas funções no pipeline. O manipulador de resposta do resolvidor é responsável por retornar dados que são mapeados para o tipo de saída do campo GraphQL. No exemplo abaixo, um manipulador de solicitação do resolvidor define `allowedGroups`; os dados retornados devem pertencer a um desses grupos. Esse valor pode ser usado pelas funções do resolvidor para solicitar dados. O manipulador de resposta do resolvidor realiza uma verificação final e filtra o resultado para garantir que somente os itens que pertencem aos grupos permitidos sejam retornados.

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

/**
 * Called before the request function of the first AppSync function in the pipeline.
 * @param ctx the context object holds contextual information about the function
 * invocation.
 */
export function request(ctx) {
  ctx.stash.allowedGroups = ['admin'];
  ctx.stash.startedAt = util.time.nowISO8601();
  return {};
}

/**
 * Called after the response function of the last AppSync function in the pipeline.
 * @param ctx the context object holds contextual information about the function
 * invocation.
 */
export function response(ctx) {
  const result = [];
  for (const item of ctx.prev.result) {
```

```
    if (ctx.stash.allowedGroups.indexOf(item.group) > -1) result.push(item);  
  }  
  return result;  
}
```

AWS AppSync Funções de escrita

AWS AppSync as funções permitem que você escreva uma lógica comum que você pode reutilizar em vários resolvedores em seu esquema. Por exemplo, você pode ter uma AWS AppSync função chamada `QUERY_ITEMS` responsável por consultar itens de uma fonte de dados do Amazon DynamoDB. Para resolvedores com os quais você gostaria de consultar itens, basta adicionar a função ao pipeline do resolvedor e fornecer o índice de consulta a ser usado. A lógica não precisa ser reimplementada.

Tópicos complementares

Tópicos

- [Exemplo de resolvedor de pipeline com o Amazon DynamoDB](#)
- [Configuração de utilitários para o runtime do APPSYNC_JS](#)
- [Agrupamento e mapas TypeScript de origem para o tempo de execução APPSYNC_JS](#)
- [Como testar seu resolvedor e manipuladores de função](#)
- [Migrando da VTL para JavaScript](#)
- [Escolha entre acesso direto à fonte de dados e proxy por meio de uma fonte de dados do Lambda](#)

Exemplo de resolvedor de pipeline com o Amazon DynamoDB

Digamos que você queira anexar um resolvedor de pipeline em um campo chamado `getPost(id:ID!)` que retorna o tipo `Post` de uma fonte de dados do Amazon DynamoDB com a seguinte consulta do GraphQL:

```
getPost(id:1){  
  id  
  title  
  content  
}
```

Primeiro, anexe um resolvidor simples a `Query.getPost` com o código abaixo. Este é um exemplo de código de resolvidor simples. Não há lógica definida no manipulador de solicitação, e o manipulador de resposta simplesmente retorna o resultado da última função.

```
/**
 * 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
}
```

Em seguida, defina a função `GET_ITEM` que recupera um postItem da sua fonte de dados:

```
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
}
```

Se houver um erro durante a solicitação, o manipulador de resposta da função anexará no final um erro que será retornado ao cliente chamador na resposta do GraphQL. Adicione a função `GET_ITEM` à sua lista de funções do resolvidor. Quando você executa a consulta, o manipulador de solicitações da `GET_ITEM` função usa os utilitários fornecidos pelo módulo AWS AppSync DynamoDB para criar uma `DynamoDBGetItem` solicitação usando o `id` como chave. `id` `ddb.get({ key: { id } })` gera a `GetItem` operação apropriada:

```
{
  "operation" : "GetItem",
  "key" : {
    "id" : { "S" : "1" }
  }
}
```

AWS AppSync usa a solicitação para buscar os dados do Amazon DynamoDB. Depois que os dados são retornados, eles são tratados pelo manipulador de resposta da função `GET_ITEM`, que verifica erros e retorna o resultado.

```
{
  "result" : {
    "id": 1,
    "title": "hello world",
    "content": "<long story>"
  }
}
```

Finalmente, o manipulador de resposta do resolvidor retorna o resultado diretamente.

Como trabalhar com eventos

Se ocorrer um erro na sua função durante uma solicitação, ele será disponibilizado no manipulador de resposta da função em `ctx.error`. Você pode acrescentar o erro no final da sua resposta do GraphQL usando o utilitário `util.appendError`. É possível disponibilizar o erro para outras funções no pipeline usando o `stash`. Veja o exemplo abaixo:

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

Configuração de utilitários para o runtime do **APPSYNC_JS**

AWS AppSync fornece duas bibliotecas que auxiliam no desenvolvimento de resolvedores com o APPSYNC_JS tempo de execução:

- `@aws-appsync/eslint-plugin`: detecta e corrige problemas rapidamente durante o desenvolvimento.
- `@aws-appsync/utils`: fornece validação de tipo e preenchimento automático em editores de código.

Configurar o plug-in eslint

[ESLint](#) é uma ferramenta que analisa estaticamente seu código para encontrar problemas rapidamente. Você pode executar ESLint como parte do seu pipeline de integração contínua. `@aws-appsync/eslint-plugin` é um ESLint plug-in que captura a sintaxe inválida em seu código ao aproveitar o tempo de execução. APPSYNC_JS O plug-in permite que você receba rapidamente

feedback sobre seu código durante o desenvolvimento sem precisar enviar suas alterações para a nuvem.

`@aws-appsync/eslint-plugin` fornece dois conjuntos de regras que você pode usar durante o desenvolvimento.

“`plugin:@aws-appsync/base`” configura um conjunto básico de regras que você pode aproveitar no seu projeto:

Rule	Descrição
no-async	Promessas e processos assíncronos não são compatíveis.
no-await	Promessas e processos assíncronos não são compatíveis.
no-classes	Classes não são compatíveis.
no-for	<code>for</code> não é compatível (exceto para <code>for-in</code> e <code>for-of</code> , que são aceitos)
no-continue	Não há suporte ao <code>continue</code> .
no-generators	Geradores não são compatíveis.
no-yield	Não há suporte ao <code>yield</code> .
no-labels	Rótulos não são compatíveis.
no-this	A palavra-chave <code>this</code> não é compatível.
no-try	A estrutura <code>try/catch</code> não é compatível.
no-while	Loops <code>While</code> não são compatíveis.
no-disallowed-unary-operators	Operadores unários <code>++</code> , <code>--</code> e <code>~</code> não são compatíveis.
no-disallowed-binary-operators	O operador <code>instanceof</code> não é compatível.

Rule	Descrição
no-promise	Promessas e processos assíncronos não são compatíveis.

“plugin: @aws -appsync/recommended” fornece algumas regras adicionais, mas também exige que você adicione TypeScript configurações ao seu projeto.

Rule	Descrição
no-recursion	Chamadas de função recursivas não são compatíveis
no-disallowed-methods	Alguns métodos não são compatíveis. Consulte a referência para obter um conjunto completo de funções integradas compatíveis.
no-function-passing	Não é permitido enviar funções como argumentos de função para funções.
no-function-reassign	As funções não podem ser reatribuídas.
no-function-return	As funções não podem ser o valor de retorno das funções.

Para adicionar o plug-in ao seu projeto, siga as etapas de instalação e uso em [Introdução ao ESLint](#). Em seguida, instale o [plug-in](#) no seu projeto usando o gerenciador de pacotes do projeto (por exemplo, npm, yarn ou pnpm):

```
$ npm install @aws-appsync/eslint-plugin
```

No seu arquivo `.eslintrc.{js,yml,json}`, adicione “plugin:@aws-appsync/base” ou “plugin:@aws-appsync/recommended” à propriedade `extends`. O trecho abaixo é um exemplo básico de `.eslintrc` configuração para: JavaScript

```
{
  "extends": ["plugin:@aws-appsync/base"]
}
```

```
}
```

Para usar o conjunto de regras “plugin:@aws-appsync/recommended”, instale a dependência necessária:

```
$ npm install -D @typescript-eslint/parser
```

Depois, crie um arquivo `.eslintrc.js`:

```
{
  "parser": "@typescript-eslint/parser",
  "parserOptions": {
    "ecmaVersion": 2018,
    "project": "./tsconfig.json"
  },
  "extends": ["plugin:@aws-appsync/recommended"]
}
```

Agrupamento e mapas TypeScript de origem para o tempo de execução APPSYNC_JS

TypeScript aprimora AWS AppSync o desenvolvimento fornecendo segurança de tipo e detecção precoce de erros. Você pode escrever TypeScript código localmente e transpilá-lo JavaScript antes de usá-lo com o APPSYNC_JS tempo de execução. O processo começa com a instalação TypeScript e configuração do `tsconfig.json` para o ambiente. APPSYNC_JS Em seguida, é possível utilizar ferramentas de empacotamento, como a `esbuild`, para compilar e empacotar o código. A CLI do Amplify gerará tipos do esquema GraphQL, permitindo que você use esses tipos no código do resolvedor.

No seu resolvedor e código de função, você pode aproveitar bibliotecas personalizadas e externas, desde que estejam em conformidade com os requisitos do APPSYNC_JS. As ferramentas de agrupamento combinam código em um único arquivo para uso em AWS AppSync. Os mapas de origem podem ser incluídos para ajudar na depuração.

Aproveitar as bibliotecas e empacotar seu código

No seu resolvedor e código de função, você pode aproveitar bibliotecas personalizadas e externas, desde que estejam em conformidade com os requisitos de APPSYNC_JS. Isso possibilita a reutilização do código existente na sua aplicação. Para usar bibliotecas definidas por vários arquivos,

you must use a bundling tool, such as [esbuild](#), to combine your code into a single file that can be saved in your AWS AppSync resolver or function.

When packaging the code, remember the following:

- APPSYNC_JS supports only ECMAScript modules (ESM).
- The modules `@aws-appsync/*` are integrated into APPSYNC_JS and must not be packaged with your code.
- The APPSYNC_JS runtime environment is similar to NodeJS, as the code is not executed in a browser environment.
- You can include an optional source map. However, do not include the original content.

To learn more about source maps, consult [Using source maps](#).

For example, to package the code of the resolver located in `src/appsync/getPost.resolver.js`, it is possible to use the following CLI esbuild command:

```
$ 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
```

Criando seu código e trabalhando com TypeScript

[TypeScript](#) is a programming language developed by Microsoft that provides all the JavaScript features along with the TypeScript type system. You can use TypeScript to write type-safe code and catch errors and bugs at the compilation stage before saving your code. In AWS AppSync, the `@aws-appsync/utils` package is fully integrated.

The APPSYNC_JS runtime environment does not provide direct TypeScript support. First, you must transpile your TypeScript code into JavaScript code compatible with the APPSYNC_JS runtime environment before saving it. In AWS AppSync, you can use TypeScript to write your code in your local integrated development environment (IDE), but observe that you cannot create TypeScript code in the AWS AppSync console.

Para começar, verifique se você [TypeScript](#) instalou em seu projeto. Em seguida, defina suas configurações de TypeScript transcompilação para trabalhar com o APPSYNC_JS tempo de execução usando. [TSConfig](#) Aqui está um exemplo de um arquivo básico `tsconfig.json` que você pode usar:

```
// tsconfig.json
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
    "noEmit": true,
    "moduleResolution": "node",
  }
}
```

Em seguida, é possível utilizar uma ferramenta de empacotamento como `esbuild` para compilar e agrupar seu código. Por exemplo, em um projeto com seu AWS AppSync código localizado em `src/appsync`, você pode usar o comando a seguir para compilar e agrupar seu código:

```
$ esbuild --bundle \
--sourcemap=inline \
--sources-content=false \
--target=esnext \
--platform=node \
--format=esm \
--external:@aws-appsync/utils \
--outdir=out/appsync \
src/appsync/**/*.ts
```

Usar o codegen Amplify

Você pode usar a [CLI do Amplify](#) para gerar os tipos para seu esquema. No diretório em que seu arquivo `schema.graphql` está localizado, execute o comando a seguir e revise os prompts para configurar seu codegen:

```
$ npx @aws-amplify/cli codegen add
```

Para regenerar seu codegen em determinadas circunstâncias (por exemplo, quando seu esquema é atualizado), execute o seguinte comando:

```
$ npx @aws-amplify/cli codegen
```

Em seguida, é possível usar os tipos gerados no código do resolvidor. Por exemplo, considerando o seguinte esquema:

```
type Todo {
  id: ID!
  title: String!
  description: String
}

type Mutation {
  createTodo(title: String!, description: String): Todo
}

type Query {
  listTodos: Todo
}
```

Você pode usar os tipos gerados na seguinte AWS AppSync função de exemplo:

```
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
}
```

Usando genéricos em TypeScript

Você pode usar genéricos com vários dos tipos fornecidos. Por exemplo, o trecho abaixo é do tipo `Todo`:

```
export type Todo = {
  __typename: "Todo",
  id: string,
```

```
title: string,  
description?: string | null,  
};
```

Você pode escrever um resolvedor para uma assinatura que use `Todo`. No seu IDE, as definições de tipo e as dicas de preenchimento automático orientarão você a usar corretamente o utilitário de transformação `toSubscriptionFilter`:

```
import { util, Context, extensions } from '@aws-appsync/utils'  
import { Todo } from './API'  
  
export function request(ctx: Context) {  
  return {}  
}  
  
export function response(ctx: Context) {  
  const filter = util.transform.toSubscriptionFilter<Todo>({  
    title: { beginsWith: 'hello' },  
    description: { contains: 'created' },  
  })  
  extensions.setSubscriptionFilter(filter)  
  return null  
}
```

Usar Lint nos seus pacotes

Você pode filtrar automaticamente seus pacotes importando o plug-in `esbuild-plugin-eslint`. Em seguida, você pode ativá-lo fornecendo um valor `plugins` que ative os recursos do `eslint`. Abaixo está um trecho que usa a JavaScript API `esbuild` em um arquivo chamado: `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 })],
})
```

Em particular, as opções `sourcesContent` e `sourcemap` especificam que um mapa de origem deve ser adicionado in-line no final de cada compilação, mas não deve incluir o conteúdo de origem. Como convenção, recomendamos não incluir o conteúdo de origem em `sourcemap`. É possível desativar isso no esbuild definindo `sources-content` como `false`.

Para ilustrar como os mapas de origem funcionam, revise o exemplo a seguir em que um código de resolvidor faz referência a funções de ajuda de uma biblioteca de ajuda. O código contém instruções de log no código do resolvidor e na biblioteca de ajuda:

`./src/default.resolver.ts` (seu resolvidor)

```
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` (um arquivo auxiliar)

```
export const logit = (...rest: any[]) => {
  // a special logger
  console.log('[logger]', ...rest.map((r) => `<${r}>`))
}

export const hello = () => {
  // This just returns a simple sentence, but it could do more.
  console.log('i just say hello..')
}
```


Ao criar e empacotar o arquivo resolvedor, seu código do resolvedor incluirá um mapa de origem in-line. Quando seu resolvedor é executado, as seguintes entradas aparecem nos CloudWatch registros:

```
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:
```

Observando as entradas no CloudWatch registro, você notará que a funcionalidade dos dois arquivos foi agrupada e está sendo executada simultaneamente. O nome do arquivo original de cada arquivo também é claramente refletido nos registros.

Testando seu resolvedor e manipuladores de funções em AWS AppSync

Você pode usar o comando da API `EvaluateCode` para testar remotamente seu resolvedor e manipuladores de funções com dados simulados antes mesmo de salvar seu código em um resolvedor ou função. Para começar a usar o comando, certifique-se de ter adicionado a permissão `appsync:evaluatecode` à sua política. Por exemplo:

JSON

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

Você pode aproveitar o comando usando a [AWS CLI](#) ou [AWS SDKs](#). Por exemplo, para testar seu código usando a CLI, basta apontar para o arquivo, fornecer um contexto e especificar o manipulador que você deseja avaliar:

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

A resposta tem um `evaluationResult` contendo o payload retornada pelo seu manipulador. Também contém um objeto `logs` com a lista de logs que foram gerados pelo seu manipulador durante a avaliação. Isso facilita a depuração da execução do código e a visualização de informações sobre sua avaliação para ajudar na solução de problemas. Por exemplo:

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

O resultado da avaliação pode ser analisado como JSON, que fornece:

```
{
  "operation": "PutItem",
  "key": {
    "id": {
      "S": "record-id"
    }
  },
  "attributeValues": {
    "owner": {
      "S": "John doe"
    },
    "expectedVersion": {
      "N": 2
    },
    "authorId": {
      "S": "Sammy Davis"
    }
  }
}
```

```
}
```

Usando o SDK, você pode incorporar facilmente testes do seu conjunto de testes para validar o comportamento do código. Este exemplo usa o [Jest Testing Framework](#), mas qualquer conjunto de testes funciona. O trecho a seguir mostra uma execução de validação hipotética. Esperamos que a resposta de avaliação seja um JSON válido, por isso, usamos `JSON.parse` para recuperar o JSON da resposta da string:

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

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

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

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

Isso produz o seguinte resultado:

```
Ran all test suites.
> jest

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

Migrando da VTL para a JavaScript AWS AppSync

AWS AppSync permite que você escreva sua lógica de negócios para seus resolvedores e funções usando VTL ou JavaScript. Com as duas linguagens, você escreve uma lógica que instrui o AWS

AppSync serviço sobre como interagir com suas fontes de dados. Com o VTL, você grava modelos de mapeamento que devem ser avaliados como uma string válida codificada em JSON. Com JavaScript, você escreve manipuladores de solicitações e respostas que retornam objetos. Você não retorna uma string codificada em JSON.

Por exemplo, use o seguinte modelo de mapeamento de VTL para obter um item do Amazon DynamoDB:

```
{
  "operation": "GetItem",
  "key": {
    "id": $util.dynamodb.toDynamoDBJson($ctx.args.id),
  }
}
```

O utilitário `$util.dynamodb.toDynamoDBJson` retorna uma string codificada em JSON. Se `$ctx.args.id` estiver definido como `<id>`, o modelo será avaliado como uma string válida codificada em JSON:

```
{
  "operation": "GetItem",
  "key": {
    "id": {"S": "<id>"},
  }
}
```

Ao trabalhar com JavaScript, você não precisa imprimir strings brutas codificadas em JSON em seu código, e usar um utilitário como esse não é necessário. `toDynamoDBJson` Um exemplo equivalente do modelo de mapeamento acima é:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'GetItem',
    key: {id: util.dynamodb.toDynamoDB(ctx.args.id)}
  };
}
```

Uma alternativa é usar `util.dynamodb.toMapValues`, que é a abordagem recomendada para manipular um objeto de valores:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'GetItem',
    key: util.dynamodb.toMapValues({ id: ctx.args.id }),
  };
}
```

É avaliado como:

```
{
  "operation": "GetItem",
  "key": {
    "id": {
      "S": "<id>"
    }
  }
}
```

Note

Recomendamos usar o módulo do DynamoDB com as fontes de dados do DynamoDB:

```
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
  ddb.get({ key: { id: ctx.args.id } })
}
```

Por exemplo, use o seguinte modelo de mapeamento para obter um item em uma fonte de dados do Amazon DynamoDB:

```
{
  "operation" : "PutItem",
  "key" : {
    "id": $util.dynamodb.toDynamoDBJson($util.autoId()),
  },
  "attributeValues" : $util.dynamodb.toMapValuesJson($ctx.args)
}
```

Quando avaliada, essa string do modelo de mapeamento deve produzir uma string válida codificada em JSON. Ao usar JavaScript, seu código retorna o objeto de solicitação diretamente:

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { id = util.autoId(), ...values } = ctx.args;
  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({ id }),
    attributeValues: util.dynamodb.toMapValues(values),
  };
}
```

que é avaliado como:

```
{
  "operation": "PutItem",
  "key": {
    "id": { "S": "2bff3f05-ff8c-4ed8-92b4-767e29fc4e63" }
  },
  "attributeValues": {
    "firstname": { "S": "Shaggy" },
    "age": { "N": 4 }
  }
}
```

Note

Recomendamos usar o módulo do DynamoDB com as fontes de dados do DynamoDB:

```
import { util } from '@aws-appsync/utils'
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
  const { id = util.autoId(), ...item } = ctx.args
  return ddb.put({ key: { id }, item })
}
```

Escolha entre acesso direto à fonte de dados e proxy por meio de uma fonte de dados do Lambda

Com AWS AppSync e o `APPSYNC_JS` tempo de execução, você pode escrever seu próprio código que implemente sua lógica de negócios personalizada usando AWS AppSync funções para acessar suas fontes de dados. Isso facilita a interação direta com fontes de dados como Amazon DynamoDB, Aurora OpenSearch Serverless, Service APIs, HTTP AWS e outros serviços sem precisar implantar serviços computacionais ou infraestrutura adicionais. AWS AppSync também facilita a interação com uma AWS Lambda função configurando uma fonte de dados Lambda. As fontes de dados Lambda permitem que você execute uma lógica comercial complexa usando os recursos completos AWS Lambda do conjunto para resolver uma solicitação do GraphQL. Na maioria dos casos, uma AWS AppSync função conectada diretamente à fonte de dados de destino fornecerá todas as funcionalidades de que você precisa. Em situações em que você precisa implementar uma lógica de negócios complexa que não é suportada pelo runtime `APPSYNC_JS`, você pode usar uma fonte de dados do Lambda como proxy para interagir com sua fonte de dados de destino.

	Integração direta de fontes de dados	Fonte de dados Lambda como proxy
Caso de uso	AWS AppSync as funções interagem diretamente com as fontes de dados da API.	AWS AppSync funções chamadas de Lambdas que interagem com fontes de dados da API.
Runtime	<code>APPSYNC_JS</code> (JavaScript)	Qualquer tempo de execução Lambda compatível
Tamanho máximo do código	32.000 caracteres por função AWS AppSync	50 MB (compactado, para upload direto) por Lambda
Módulos externos	Limitado - somente recursos compatíveis com o <code>APPSYNC_JS</code>	Sim
Ligue para qualquer AWS serviço	Sim - Usando a fonte de dados AWS AppSync HTTP	Sim - Usando o AWS SDK

Acesso ao cabeçalho da solicitação	Sim	Sim
Acesso à rede	Não	Sim
Acesso ao sistema de arquivos	Não	Sim
Registro e métricas	Sim	Sim
Crie e teste inteiramente dentro AppSync	Sim	Não
Arranque a frio	Não	Não - Com simultaneidade provisionada
Ajuste de escala automático	Sim - de forma transparente por AWS AppSync	Sim - conforme configurado no Lambda
Preços	Sem custo adicional	Cobrado pelo uso do Lambda

AWS AppSync funções que se integram diretamente à fonte de dados de destino são ideais para casos de uso como os seguintes:

- Interagindo com o Amazon DynamoDB, o Aurora Serverless e o Service OpenSearch
- Interagindo com HTTP APIs e passando cabeçalhos de entrada
- Interagindo com AWS serviços usando fontes de dados HTTP (com solicitações de assinatura AWS AppSync automática com a função de fonte de dados fornecida)
- Implementar o controle de acesso antes de acessar as fontes de dados
- Implementar a filtragem dos dados recuperados antes de atender a uma solicitação
- Implementação de orquestração simples com execução sequencial de AWS AppSync funções em um pipeline de resolução
- Controlar conexões de cache e assinatura em consultas e mutações.

AWS AppSync funções que usam uma fonte de dados Lambda como proxy são ideais para casos de uso como os seguintes:

- Usando uma linguagem diferente JavaScript da Velocity Template Language (VTL)
- Ajustar e controlar a CPU ou a memória para otimizar a performance
- Importar bibliotecas de terceiros ou exigir atributos não suportados no APPSYNC_JS
- Fazer várias solicitações de rede and/or obtendo acesso ao sistema de arquivos para atender a uma consulta
- Fazer solicitações em lote usando a [configuração em lote](#)

AWS AppSync JavaScript referência de objeto de contexto do resolvidor

AWS AppSync define um conjunto de variáveis e funções para trabalhar com manipuladores de solicitações e respostas. Isso facilita as operações lógicas em dados com o GraphQL. Este documento descreve essas funções e fornece exemplos.

Acesso ao **context**

O argumento `context` de um manipulador de solicitações e respostas é um objeto que contém todas as informações contextuais para a invocação do seu resolvidor. Ela tem a seguinte estrutura:

```
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

Você descobrirá que o objeto context é chamado de ctx.

Cada campo no objeto context é definido da seguinte forma:

Campos de **context**

arguments

Um mapa que contém todos os argumentos do GraphQL para este campo.

identity

Um objeto que contém informações sobre o chamador. Para obter mais informações sobre a estrutura desse campo, consulte [Identidade](#).

source

Um mapa que contém a resolução do campo pai.

stash

O stash é um objeto disponibilizado dentro de cada manipulador de resolvedor e função. O mesmo objeto stash subsiste por meio de uma única execução de resolvedor. Isso significa que é possível usar o stash para passar dados arbitrários entre os manipuladores de solicitações e respostas e entre as funções em um resolvedor de pipeline.

Note

Não é possível excluir ou substituir o stash inteiro, mas você pode adicionar, atualizar, excluir e ler as propriedades dele.

Você pode adicionar itens ao stash modificando um dos exemplos de código abaixo:

```
//Example 1
ctx.stash.newItem = { key: "something" }

//Example 2
```

```
Object.assign(ctx.stash, {key1: value1, key2: value})
```

Você pode remover itens do stash modificando o código abaixo:

```
delete ctx.stash.key
```

result

Um contêiner para os resultados desse resolvedor. Esse campo só está disponível para manipuladores de respostas.

Por exemplo, se estiver resolvendo o campo `author` da seguinte consulta:

```
query {  
  getPost(id: 1234) {  
    postId  
    title  
    content  
    author {  
      id  
      name  
    }  
  }  
}
```

Então, a variável `context` completa estará disponível quando um manipulador de respostas for avaliado:

```
{  
  "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" : "AIDAAAAAAAAAAAAAAAAAAAA"
  }
}
```

prev.result

O resultado de qualquer operação anterior executada em um resolvedor de pipeline.

Se a operação anterior foi o manipulador de solicitações do resolvedor de pipeline, `ctx.prev.result` representa o resultado da avaliação e será disponibilizado para a primeira função no pipeline.

Se a operação anterior foi a primeira função, `ctx.prev.result` representa o resultado da avaliação do manipulador de respostas da primeira função e será disponibilizado para a segunda função no pipeline.

Se a operação anterior foi a última função, `ctx.prev.result` representa o resultado da avaliação da última função e será disponibilizado para o manipulador de respostas do resolvedor do pipeline.

info

Um objeto que contém informações sobre a solicitação do GraphQL. Para ver a estrutura desse campo, consulte [Informações](#).

Identidade

A seção `identity` contém informações sobre o chamador. O formato dessa seção depende do tipo de autorização da sua AWS AppSync API.

Para obter mais informações sobre as opções de AWS AppSync segurança, consulte [Autorização e autenticação](#).

API_KEY authorization

O campo `identity` não está preenchido.

AWS_LAMBDA authorization

O `identity` tem o seguinte formato:

```
type AppSyncIdentityLambda = {  
  resolverContext: any;  
};
```

`identity` inclui a chave `resolverContext` que tem o mesmo conteúdo de `resolverContext` retornado pela função do Lambda que autorizou a solicitação.

AWS_IAM authorization

O `identity` tem o seguinte formato:

```
type AppSyncIdentityIAM = {  
  accountId: string;  
  cognitoIdentityPoolId: string;  
  cognitoIdentityId: string;  
  sourceIp: string[];  
  username: string;  
  userArn: string;  
  cognitoIdentityAuthType: string;  
  cognitoIdentityAuthProvider: string;  
};
```

AMAZON_COGNITO_USER_POOLS authorization

O `identity` tem o seguinte formato:

```
type AppSyncIdentityCognito = {  
  sourceIp: string[];  
  username: string;  
  groups: string[] | null;  
  sub: string;  
  issuer: string;  
  claims: any;  
  defaultAuthStrategy: string;  
};
```

Cada campo é definido da seguinte forma:

accountId

O ID da AWS conta do chamador.

claims

As reivindicações do usuário.

cognitoIdentityAuthType

Autenticado ou não autenticado com base no tipo de identidade.

cognitoIdentityAuthProvider

Uma lista separada por vírgulas das informações do provedor de identidade externo usada na obtenção das credenciais usadas para assinar a solicitação.

cognitoIdentityId

O ID de identidade do Amazon Cognito do chamador.

cognitoIdentityPoolId

O ID do banco de identidades do Amazon Cognito associado ao chamador.

defaultAuthStrategy

A estratégia de autorização padrão para este chamador (ALLOW ou DENY).

issuer

O emissor do token.

sourceIp

O endereço IP de origem do chamador que AWS AppSync recebe. Se a solicitação não incluir o cabeçalho `x-forwarded-for`, o valor do IP de origem conterá apenas um único endereço IP da conexão TCP. Se a solicitação inclui um cabeçalho `x-forwarded-for`, o IP de origem será uma lista de endereços IP do cabeçalho `x-forwarded-for`, além do endereço IP da conexão TCP.

sub

O UUID do usuário autenticado.

user

O usuário do IAM.

userArn

O nome do recurso da Amazon (ARN) do usuário do IAM.

username

O nome do usuário autenticado. Em caso de autorização `AMAZON_COGNITO_USER_POOLS`, o valor do nome de usuário é o valor de `username` é o valor do atributo `cognito:username`. No caso de `AWS_IAM` autorização, o valor do nome de usuário é o valor do AWS usuário principal. Se você estiver usando a autorização do IAM com credenciais fornecidas por bancos de identidades do Amazon Cognito, recomendamos utilizar `cognitoIdentityId`.

Cabeçalhos de solicitação de acesso

AWS AppSync suporta passar cabeçalhos personalizados de clientes e acessá-los em seus resolvedores GraphQL usando `ctx.request.headers`. Em seguida, você pode usar valores de cabeçalho para ações como inserir dados em uma fonte de dados ou até mesmo verificações de autorização. É possível utilizar cabeçalhos de solicitação por meio de `$curl` com uma chave da API na linha de comando conforme mostrado nos exemplos a seguir:

Exemplo de cabeçalho único

Digamos que você defina um cabeçalho custom com um valor de `nadia` da seguinte forma:

```
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
```

Isso pode ser acessado com `ctx.request.headers.custom`. Por exemplo, ele pode estar no seguinte código para o DynamoDB:

```
"custom": util.dynamodb.toDynamoDB(ctx.request.headers.custom)
```

Exemplo de vários cabeçalhos

Você também pode enviar vários cabeçalhos em uma única solicitação e acessá-los no manipulador do resolvedor. Por exemplo, se o cabeçalho custom foi definido com dois valores:

```
curl -XPOST -H "Content-Type:application/graphql" -H "custom:bailey" -H "custom:nadia" -H "x-api-key:<API-KEY-VALUE>" -d '{"query":"mutation { createEvent(name: \"demo
```

```
\", when: \"Next Friday!\", where: \"Here!\") {id name when where description}}\"}'  
https://<ENDPOINT>/graphql
```

Em seguida, você pode acessá-los como uma matriz, como `ctx.request.headers.custom[1]`.

Note

AWS AppSync não expõe o cabeçalho do cookie em `ctx.request.headers`.

Acessar o nome de domínio personalizado da solicitação

AWS AppSync suporta a configuração de um domínio personalizado que você pode usar para acessar seu GraphQL e endpoints em tempo real para seu. APIs Ao fazer uma solicitação com um nome de domínio personalizado, você pode obter o nome de domínio usando `ctx.request.domainName`.

Ao usar o nome de domínio padrão do endpoint do GraphQL, o valor será `null`.

Informações

A seção `info` contém informações sobre a solicitação do GraphQL. Esta seção tem o seguinte formato:

```
type Info = {  
  fieldName: string;  
  parentTypeName: string;  
  variables: any;  
  selectionSetList: string[];  
  selectionSetGraphQL: string;  
};
```

Cada campo é definido da seguinte forma:

fieldName

O nome do campo que está sendo resolvido no momento.

parentTypeName

O nome do tipo pai para o campo que está sendo resolvido no momento.

variables

Um mapa que contém todas as variáveis que são passadas para a solicitação do GraphQL.

selectionSetList

Uma representação de lista dos campos no conjunto de seleções do GraphQL. Os campos com alias serão referenciados somente pelo nome do alias, não pelo nome do campo. O exemplo a seguir mostra isso em detalhes.

selectionSetGraphQL

Uma representação de string do conjunto de seleções, formatada como linguagem de definição de esquema (SDL) do GraphQL. Embora os fragmentos não sejam mesclados no conjunto de seleções, os fragmentos inline são preservados, conforme mostrado no exemplo a seguir.

Note

`JSON.stringify` não incluirá `selectionSetGraphQL` e `selectionSetList` na serialização da string. Você deve referenciar essas propriedades diretamente.

Por exemplo, se estiver resolvendo o campo `getPost` da seguinte consulta:

```
query {  
  getPost(id: $postId) {  
    postId  
    title  
    secondTitle: title  
    content  
    author(id: $authorId) {  
      authorId  
      name  
    }  
    secondAuthor(id: "789") {  
      authorId  
    }  
    ... on Post {  
      inlineFrag: comments: {  
        id  
      }  
    }  
  }  
}
```

```

    ... postFrag
  }
}

fragment postFrag on Post {
  postFrag: comments: {
    id
  }
}

```

Depois, a variável `ctx.info` completa que está disponível ao processar o manipulador pode ser:

```

{
  "fieldName": "getPost",
  "parentTypeName": "Query",
  "variables": {
    "postId": "123",
    "authorId": "456"
  },
  "selectionSetList": [
    "postId",
    "title",
    "secondTitle",
    "content",
    "author",
    "author/authorId",
    "author/name",
    "secondAuthor",
    "secondAuthor/authorId",
    "inlineFragComments",
    "inlineFragComments/id",
    "postFragComments",
    "postFragComments/id"
  ],
  "selectionSetGraphQL": "{\n  getPost(id: $postId) {\n    postId\n    title\n    secondTitle: title\n    content\n    author(id: $authorId) {\n      authorId\n      name\n    }\n    secondAuthor(id: \"789\") {\n      authorId\n    }\n    ... on Post\n    {\n      inlineFrag: comments {\n        id\n      }\n    }\n    ... postFrag\n  }\n}"
}

```

`selectionSetList` expõe somente campos que pertencem ao tipo atual. Se o tipo atual for uma interface ou união, somente os campos selecionados que pertencem à interface serão expostos. Por exemplo, considerando o seguinte esquema:

```
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  
}
```

E a seguinte consulta:

```
query {  
  node(id: "post1") {  
    id  
    ... on Post {  
      title  
    }  
  
    ... on Blog {  
      title  
    }  
  }  
}
```

Ao chamar `ctx.info.selectionSetList` na resolução do campo `Query.node`, somente `id` é exposto:

```
"selectionSetList": [  
  "id"  
]
```

AWS AppSync JavaScript recursos de tempo de execução para resolvedores e funções

O ambiente APPSYNC_JS de tempo de execução fornece funcionalidade semelhante à [ECMAScript \(ES\) versão 6.0](#). Ele suporta um subconjunto dos atributos e fornece alguns métodos adicionais (utilitários) que não fazem parte das especificações ES. Os tópicos a seguir listam todos os atributos de idioma compatíveis:

- [Recursos de runtime compatíveis](#): saiba mais sobre os principais atributos, objetos primitivos, objetos e funções integrados e outros elementos compatíveis.
- [Utilitários integrados](#): a variável `util` contém métodos utilitários gerais para ajudar você a trabalhar com dados. A menos que especificado o contrário, todos os utilitários usam o conjunto de caracteres UTF-8.
- [Módulos integrados](#) - Saiba mais sobre como os módulos integrados podem ajudar a escrever JavaScript resolvedores e funções.
- [Utilitários de runtime](#): a biblioteca de runtime fornece utilitários para controlar ou modificar as propriedades de runtime dos seus resolvedores e funções.
- [Auxiliares de tempo em `util.time`](#): a variável `util.time` contém métodos de data e hora para ajudar a gerar timestamps, converter entre formatos de data e hora e analisar strings de data e hora. A sintaxe dos formatos de data e hora é baseada em [DateTimeFormatter](#), que você pode consultar para obter mais documentação.
- [Auxiliares do DynamoDB em `util.dynamodb`](#): `util.dynamodb` contém métodos auxiliares que tornam mais fácil gravar e ler dados no Amazon DynamoDB, como mapeamento e formatação do tipo automático.
- [Auxiliares HTTP em `util.http`](#): o utilitário `util.http` fornece métodos auxiliares que podem ser usados para gerenciar parâmetros de solicitação HTTP e adicionar cabeçalhos de resposta.
- [Auxiliares de transformação em `util.transform`](#): `util.transform` contém métodos auxiliares que facilitam a execução de operações complexas em fontes de dados.
- [Auxiliares de string em `util.str`](#): `util.str` contém métodos para ajudar com operações comuns de string.
- [Extensões](#): `extensions` contém um conjunto de métodos para realizar ações adicionais em seus resolvedores.
- [Auxiliares XML em `util.xml`](#): `util.xml` contém métodos para ajudar na conversão de strings XML.

 Note

Atualmente, essa referência só se aplica à versão 1.0.0 do runtime.

Atributos compatíveis de runtime

As seções abaixo descrevem o conjunto de atributos compatível com o runtime do APPSYNC_JS.

Atributos principais

Os seguintes atributos principais são compatíveis.

Types

Os seguintes tipos são compatíveis:

- números
- strings
- booleanos
- objects
- arrays
- funções

Operators

Os operadores são suportados, incluindo:

- Operadores matemáticos padrão (+, -, /, %, *, etc.)
- Operador de coalescência nula (??)
- Encadeamento opcional (?.)
- Operadores bitwise
- Operador void e typeof
- Operadores de distribuição (...)

Os operadores a seguir não são aceitos:

- Operadores unários (`++`, `--` e `~`)
- Operador `in`

 Note

Use o operador `Object.hasOwnProperty` para verificar se a propriedade especificada está no objeto especificado.

Statements

As seguintes instruções são compatíveis:

- `const`
- `let`
- `var`
- `break`
- `else`
- `for-in`
- `for-of`
- `if`
- `return`
- `switch`
- sintaxe de propagação

Não há suporte para o seguinte:

- `catch`
- `continue`
- `do-while`
- `finally`
- `for(initialization; condition; afterthought)`

 Note

As exceções são expressões `for-in` e `for-of`, que são suportadas.

- `throw`
- `try`
- `while`
- Instruções rotuladas

Literals

As seguintes [literais do modelo](#) ES 6 são compatíveis:

- Strings de várias linhas
- Interpolação de expressão
- Modelos de aninhamento

Functions

A sintaxe da função a seguir também tem suporte:

- As declarações de função são suportadas.
- As funções de seta ES 6 são suportadas.
- A sintaxe de parâmetro rest ES 6 é compatível.

Strict mode

As funções operam no modo estrito por padrão, então você não precisa adicionar uma instrução `use_strict` ao seu código de função. Elas não podem ser alteradas.

Objetos primitivos

Os seguintes objetos primitivos de ES e suas funções são compatíveis.

Object

Há suporte para os seguintes objetos:

- `Object.assign()`
- `Object.entries()`
- `Object.hasOwn()`
- `Object.keys()`
- `Object.values()`
- `delete`

String

As seguintes strings são compatíveis:

- `String.prototype.length()`
- `String.prototype.charAt()`
- `String.prototype.concat()`
- `String.prototype.endsWith()`
- `String.prototype.indexOf()`
- `String.prototype.lastIndexOf()`
- `String.raw()`
- `String.prototype.replace()`

Note

Não há suporte para expressões regulares.

No entanto, constructos de expressões regulares estilo Java são compatíveis no parâmetro fornecido. Para obter mais informações, consulte [Padrão](#).

- `String.prototype.replaceAll()`

Note

Não há suporte para expressões regulares.

No entanto, constructos de expressões regulares estilo Java são compatíveis no parâmetro fornecido. Para obter mais informações, consulte [Padrão](#).

- `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

Os seguintes números são compatíveis:

- `Number.isFinite`
- `Number.isNaN`

Objetos e funções integrados

As funções e objetos a seguir são compatíveis.

Math

As seguintes funções matemáticas são compatíveis:

- `Math.random()`
- `Math.min()`
- `Math.max()`
- `Math.round()`
- `Math.floor()`
- `Math.ceil()`

Array

Os seguintes métodos de matriz são compatíveis:

- `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()` não é compatível com argumentos.

- `Array.prototype.splice()`
- `Array.prototype.unshift()`
- `Array.prototype.forEach()`
- `Array.prototype.map()`
- `Array.prototype.flatMap()`
- `Array.prototype.filter()`
- `Array.prototype.reduce()`
- `Array.prototype.reduceRight()`
- `Array.prototype.find()`

- `Array.prototype.some()`
- `Array.prototype.every()`
- `Array.prototype.findIndex()`
- `Array.prototype.findLast()`
- `Array.prototype.findLastIndex()`
- `delete`

Console

O objeto do console está disponível para depuração. Durante a execução da consulta em tempo real, log/error as instruções do console são enviadas para o Amazon CloudWatch Logs (se o registro estiver ativado). Durante a avaliação do código com `evaluateCode`, as instruções de log são retornadas na resposta do comando.

- `console.error()`
- `console.log()`

Function

- Os métodos `apply`, `bind` e `call` não são compatíveis.
- Os construtores de função não são compatíveis.
- Não há suporte para passar uma função como argumento.
- Chamadas de funções recursivas não são compatíveis

JSON

Os seguintes métodos JSON são compatíveis:

- `JSON.parse()`



Note

Retornará uma string em branco se a string analisada não for um JSON válido.

- `JSON.stringify()`

Promises

Promessas e processos assíncronos não são compatíveis.

Note

O acesso à rede e ao sistema de arquivos não é suportado no APPSYNC_JS tempo de execução do AWS AppSync. AWS AppSync lida com todas as operações de E/S com base nas solicitações feitas pelo AWS AppSync resolvedor ou pela AWS AppSync função.

Variáveis globais

As seguintes restrições globais são válidas:

- NaN
- Infinity
- undefined
- [util](#)
- [extensions](#)
- [runtime](#)

Tipos de erro

Não é possível lançar erros com `throw`. Você pode retornar um erro usando a função `util.error()`. Você pode incluir um erro na sua resposta do GraphQL usando a função `util.appendError`.

Para obter mais informações, consulte [Utilitários de erro](#).

Utilitários integrados

A variável `util` contém métodos utilitários gerais para ajudar você a trabalhar com dados. A menos que especificado o contrário, todos os utilitários usam o conjunto de caracteres UTF-8.

Utilitários de codificação

Lista de utilitários de codificação

util.urlEncode(String)

Retorna a string de entrada como uma string codificada `application/x-www-form-urlencoded`.

util.urlDecode(String)

Decodifica uma string codificada `application/x-www-form-urlencoded` de volta ao seu formato não codificado.

util.base64Encode(string) : string

Codifica a entrada em uma string codificada em base64.

util.base64Decode(string) : string

Decodifica os dados de uma string codificada em base64.

Utilitários de geração de ID

Lista de utilitários de geração de ID

util.autoId()

Retorna um UUID de 128 bits gerado aleatoriamente.

util.autoUlid()

Retorna um identificador lexicograficamente classificável universalmente exclusivo (ULID) de 128 bits gerado aleatoriamente.

util.autoKsuid()

Retorna um identificador exclusivo classificável por K (KSUID) de 128 bits gerado aleatoriamente, codificado em base62, como uma string com comprimento de 27.

Utilitários de erro

Lista de utilitários de erro

`util.error(String, String?, Object?, Object?)`

Lança um erro personalizado. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Além disso, é possível especificar os campos `errorType`, `data` e `errorInfo`. O valor `data` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL.

Note

`data` será filtrado com base no conjunto de seleção da consulta. O valor `errorInfo` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL. `errorInfo` não será filtrado com base no conjunto de seleção da consulta.

`util.appendError(String, String?, Object?, Object?)`

Anexa um erro personalizado no final. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Além disso, é possível especificar os campos `errorType`, `data` e `errorInfo`. Diferente de `util.error(String, String?, Object?, Object?)`, a avaliação do modelo não será interrompida para que os dados possam ser retornados ao chamador. O valor `data` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL.

Note

`data` será filtrado com base no conjunto de seleção da consulta. O valor `errorInfo` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL. `errorInfo` não será filtrado com base no conjunto de seleção da consulta.

Utilitários de correspondência de tipos e padrões

Lista de utilitários de correspondência de tipos e padrões

util.matches(String, String) : Boolean

Retorna verdadeiro se o padrão especificado no primeiro argumento corresponde aos dados fornecidos no segundo argumento. O padrão deve ser uma expressão regular, como `util.matches("a*b", "aaaaab")`. A funcionalidade se baseia em [Padrão](#), que você pode consultar para obter documentação adicional.

util.authType()

Retorna uma String que descreve o tipo de autenticação múltipla que está sendo usado por uma solicitação, retornando "Autorização do IAM", "Autorização de grupo de usuários", "Autorização do Open ID Connect" ou "Autorização de chave de API".

Utilitários de comportamento do valor de retorno

Lista de utilitários de comportamento de valor de retorno

util.escapeJavaScript(String)

Retorna a string de entrada como uma string JavaScript de escape.

Utilitários de autorização do resolvidor

Lista de utilitários de autorização do resolvidor

util.unauthorized()

Lança `Unauthorized` para o campo a ser resolvido. Use em modelos de mapeamento de solicitação ou resposta para determinar se é preciso ou não permitir que o chamador resolva o campo.

Módulos integrados

Os módulos fazem parte do `APPSYNC_JS` tempo de execução e fornecem utilitários para ajudar a escrever JavaScript resolvidores e funções. Para amostras e exemplos, consulte o [aws-appsync-resolver-samples](#) GitHub repositório.

Funções do módulo DynamoDB

As funções do módulo DynamoDB oferecem uma experiência aprimorada ao interagir com fontes de dados do DynamoDB. Você pode fazer solicitações para suas fontes de dados do DynamoDB usando as funções e sem adicionar mapeamento de tipos.

Os módulos são importados usando `@aws-appsync/utils/dynamodb`:

```
// Modules are imported using @aws-appsync/utils/dynamodb
import * as ddb from '@aws-appsync/utils/dynamodb';
```

Funções

Lista de funções

get<T>(payload: GetInput): DynamoDBGetItemRequest

Tip

Consulte [the section called “Entradas”](#) para obter informações sobre GetInput.

Gera um `DynamoDBGetItemRequest` objeto para fazer uma [GetItem](#) solicitação ao DynamoDB.

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

export function request(ctx) {
  return get({ key: { id: ctx.args.id } });
}
```

put<T>(payload): DynamoDBPutItemRequest

Gera um `DynamoDBPutItemRequest` objeto para fazer uma [PutItem](#) solicitação ao DynamoDB.

```
import * as ddb from '@aws-appsync/utils/dynamodb'

export function request(ctx) {
  return ddb.put({ key: { id: util.autoId() }, item: ctx.args });
}
```


remove<T>(payload): DynamoDBDeleteItemRequest

Gera um `DynamoDBDeleteItemRequest` objeto para fazer uma [DeleteItem](#) solicitação ao DynamoDB.

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  return ddb.remove({ key: { id: ctx.args.id } });
}
```

scan<T>(payload): DynamoDBScanRequest

Gera um objeto `DynamoDBScanRequest` para fazer uma solicitação [Scan](#) ao DynamoDB.

```
import * as ddb from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const { limit = 10, nextToken } = ctx.args;
  return ddb.scan({ limit, nextToken });
}
```

sync<T>(payload): DynamoDBSyncRequest

Gera um objeto `DynamoDBSyncRequest` para fazer uma solicitação [Sync](#). A solicitação só recebe os dados alterados desde a última consulta (atualizações delta). As solicitações só podem ser feitas para fontes de dados versionadas do DynamoDB.

```
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

Gera um `DynamoDBUpdateItemRequest` objeto para fazer uma [UpdateItem](#) solicitação ao DynamoDB.

Operações

Os auxiliares de operação permitem que você execute ações específicas em partes dos seus dados durante as atualizações. Para começar, importe `operations` de `@aws-appsync/utils/dynamodb`:

```
// Modules are imported using operations
import {operations} from '@aws-appsync/utils/dynamodb';
```

Operações de lista

add<T>(payload)

Uma função auxiliar que adiciona um novo item de atributo ao atualizar o DynamoDB.

Exemplo

Para adicionar um endereço (rua, cidade e código postal) a um item existente do DynamoDB usando o valor do ID:

```
import { update, operations } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const updateObj = {
    address: operations.add({
      street1: '123 Main St',
      city: 'New York',
      zip: '10001',
    }),
  };
  return update({ key: { id: 1 }, update: updateObj });
}
```

append <T>(payload)

Uma função auxiliar que anexa um payload no final da lista existente no DynamoDB.

Exemplo

Para acrescentar amigo recém-adicionado IDs (`newFriendIds`) a uma lista de amigos existente (`friendsIds`) durante uma atualização:

```
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?)

Uma função auxiliar que diminui o valor do atributo existente no item ao atualizar o DynamoDB.

Exemplo

Para diminuir o contador de amigos (friendsCount) em 10:

```
import { update, operations } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const updateObj = {
    friendsCount: operations.decrement(10),
  };
  return update({ key: { id: 1 }, update: updateObj });
}
```

increment (by?)

Uma função auxiliar que aumenta o valor do atributo existente no item ao atualizar o DynamoDB.

Exemplo

Para aumentar o contador de amigos (friendsCount) em 10:

```
import { update, operations } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const updateObj = {
    friendsCount: operations.increment(10),
  };
  return update({ key: { id: 1 }, update: updateObj });
}
```

```
}
```

prepend <T>(payload)

Uma função auxiliar que anexa no início da lista existente no DynamoDB.

Exemplo

Para acrescentar amigo IDs (`newFriendIds`) recém-adicionado a uma lista de amigos existente (`friendsIds`) durante uma atualização:

```
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)

Uma função auxiliar que substitui o atributo existente no item ao atualizar um item no DynamoDB. Isso é útil quando você deseja atualizar todo o objeto ou subobjeto no atributo e não apenas as chaves no payload.

Exemplo

Para substituir um endereço (rua, cidade e CEP) em um objeto `info`:

```
import { update, operations } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const updateObj = {
    info: {
      address: operations.replace({
        street1: '123 Main St',
        city: 'New York',
        zip: '10001',
      }),
    },
  };
}
```

```
return update({ key: { id: 1 }, update: updateObj });
}
```

updateListItem <T>(payload, index)

Uma função auxiliar que substitui um item em uma lista.

Exemplo

No escopo do `update (newFriendIds)`, este exemplo é usado `updateListItem` para atualizar os valores de ID do segundo item (índice: 1, novo ID: 102) e do terceiro item (índice: 2, novo ID: 112) em uma lista (`friendsIds`).

```
import { update, operations as ops } from '@aws-appsync/utils/dynamodb';

export function request(ctx) {
  const newFriendIds = [
    ops.updateListItem('102', 1), ops.updateListItem('112', 2)
  ];
  const updateObj = { friendsIds: newFriendIds };
  return update({ key: { id: 1 }, update: updateObj });
}
```

Entradas

Lista de entradas

Type `GetInput<T>`

```
GetInput<T>: {
  consistentRead?: boolean;
  key: DynamoDBKey<T>;
}
```

Declaração de tipo

- `consistentRead?: boolean` (opcional)

Um booleano opcional para especificar se você deseja realizar uma leitura altamente consistente com o DynamoDB.

- `key: DynamoDBKey<T>` (obrigatório)

Um parâmetro obrigatório que especifica a chave do item no DynamoDB. Os itens do DynamoDB podem ter uma única chave de hash ou chaves de hash e de classificação.

Type **PutInput<T>**

```
PutInput<T>: {  
  _version?: number;  
  condition?: DynamoDBFilterObject<T> | null;  
  customPartitionKey?: string;  
  item: Partial<T>;  
  key: DynamoDBKey<T>;  
  populateIndexFields?: boolean;  
}
```

Declaração de tipo

- `_version?: number` (opcional)
- `condition?: DynamoDBFilterObject<T> | null` (opcional)

Quando você insere um objeto em uma tabela do DynamoDB, pode especificar uma expressão condicional que controla se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB antes que a operação seja realizada.

- `customPartitionKey?: string` (opcional)

Quando ativado, esse valor de string modifica o formato dos registros `ds_sk` e `ds_pk` usados pela tabela de sincronização delta quando o versionamento é ativado. Quando ativado, o processamento da entrada `populateIndexFields` também é ativado.

- `item: Partial<T>` (obrigatório)

O restante dos atributos do item a ser colocado no DynamoDB.

- `key: DynamoDBKey<T>` (obrigatório)

Um parâmetro obrigatório que especifica a chave do item no DynamoDB em que a inserção será feita. Os itens do DynamoDB podem ter uma única chave de hash ou chaves de hash e de classificação.

- `populateIndexFields?: boolean` (opcional)

Um valor booleano que, quando ativado com `customPartitionKey`, cria novas entradas para cada registro na tabela de sincronização delta, especificamente nas colunas `gsi_ds_pk`

e `gsi_ds_sk`. Para obter mais informações, consulte [Detecção e sincronização de conflitos](#) no Guia do desenvolvedor do AWS AppSync .

Type `QueryInput<T>`

```
QueryInput<T>: ScanInput<T> & {  
  query: DynamoDBKeyCondition<Required<T>>;  
}
```

Declaração de tipo

- `query: DynamoDBKeyCondition<Required<T>>`(obrigatório)

Especifica uma condição de chave que descreve os itens a serem consultados. Para um determinado índice, a condição para uma chave de partição deve ser uma igualdade e para a chave de classificação, uma comparação ou `beginsWith` (quando for uma string). Somente tipos de números e strings de caracteres são suportados para chaves de partição e de classificação.

Exemplo

Veja o tipo `User` abaixo:

```
type User = {  
  id: string;  
  name: string;  
  age: number;  
  isVerified: boolean;  
  friendsIds: string[]  
}
```

A consulta só pode incluir os seguintes campos: `id`, `name` e `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;
}
```

Declaração de tipo

- `_version?: number` (opcional)
- `condition?: DynamoDBFilterObject<T>` (opcional)

Quando você remove um objeto no DynamoDB, pode especificar uma expressão condicional que controla se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB antes que a operação seja realizada.

Exemplo

O exemplo a seguir é uma expressão `DeleteItem` contendo uma condição que permitirá que a operação seja bem-sucedida somente se o proprietário do documento corresponder ao usuário que fez a solicitação.

```
type Task = {
  id: string;
  title: string;
  description: string;
  owner: string;
  isComplete: boolean;
}
const condition: DynamoDBFilterObject<Task> = {
  owner: { eq: 'XXXXXXXXXXXXXXXXXX' },
}

remove<Task>({
  key: {
    id: 'XXXXXXXXXXXXXXXXXX',
  },
  condition,
});
```

- `customPartitionKey?: string` (opcional)

Quando ativado, esse valor `customPartitionKey` modifica o formato dos registros `ds_sk` e `ds_pk` usados pela tabela de sincronização delta quando o versionamento é ativado. Quando ativado, o processamento da entrada `populateIndexFields` também é ativado.

- `key: DynamoDBKey<T>` (obrigatório)

Um parâmetro obrigatório que especifica a chave do item no DynamoDB que está sendo removido. Os itens do DynamoDB podem ter uma única chave de hash ou chaves de hash e de classificação.

Exemplo

Se `User` tiver apenas a chave de hash com um usuário `id`, a chave ficará assim:

```
type User = {
  id: number
  name: string
  age: number
  isVerified: boolean
}
const key: DynamoDBKey<User> = {
  id: 1,
}
```

Se o usuário da tabela tiver a chave de hash (`id`) e a chave de classificação (`name`), a chave ficará assim:

```
type User = {
  id: number
  name: string
  age: number
  isVerified: boolean
  friendsIds: string[]
}
const key: DynamoDBKey<User> = {
  id: 1,
  name: 'XXXXXXXXXX',
}
```

- `populateIndexFields?: boolean` (opcional)

Um valor booleano que, quando ativado com `customPartitionKey`, cria novas entradas para cada registro na tabela de sincronização delta, especificamente nas colunas `gsi_ds_pk` e `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;  
}
```

Declaração de tipo

- `consistentRead?: boolean | null` (opcional)

Um booleano opcional que indica leituras consistentes ao consultar o DynamoDB. O valor padrão é `false`.

- `filter?: DynamoDBFilterObject<T> | null` (opcional)

Um filtro opcional para aplicar aos resultados depois de recuperá-los da tabela.

- `index?: string | null` (opcional)

Um nome opcional do índice para digitalizar.

- `limit?: number | null` (opcional)

O número máximo opcional de resultados a serem retornados.

- `nextToken?: string | null` (opcional)

O token de paginação opcional para continuar uma consulta anterior. Isso seria obtido de uma consulta anterior.

- `scanIndexForward?: boolean | null` (opcional)

Um booleano opcional para indicar se a consulta é executada em ordem crescente ou decrescente. Por padrão, esse valor é definido como `true`.

- `segment?: number` (opcional)
- `select?: DynamoDBSelectAttributes` (opcional)

Atributos a serem retornados do DynamoDB. Por padrão, o AWS AppSync resolvedor do DynamoDB retorna somente atributos que são projetados no índice. Os valores compatíveis são:

- `ALL_ATTRIBUTES`

Retorna todos os atributos de item da tabela ou índice especificado. Se você consultar um índice secundário local, o DynamoDB buscará todo o item da tabela pai para cada item correspondente no índice. Se o índice estiver configurado para projetar todos os atributos de item, todos os dados podem ser obtidos no índice secundário local, e nenhuma busca será necessária.

- `ALL_PROJECTED_ATTRIBUTES`

Recupera todos os atributos que foram projetados no índice. Se o índice estiver configurado para projetar todos os atributos, esse valor de retorno é equivalente a especificar `ALL_ATTRIBUTES`.

- `SPECIFIC_ATTRIBUTES`

Retorna somente os atributos listados em `ProjectionExpression`. Esse valor de retorno é equivalente a especificar `ProjectionExpression` sem especificar nenhum valor para `AttributesToGet`.

- `totalSegments?: number` (opcional)

Type `DynamoDBSyncInput<T>`

```
DynamoDBSyncInput<T>: {  
  basePartitionKey?: string;  
  deltaIndexName?: string;  
  filter?: DynamoDBFilterObject<T> | null;  
  lastSync?: number;  
  limit?: number | null;  
  nextToken?: string | null;  
}
```

Declaração de tipo

- `basePartitionKey?: string` (opcional)

A chave de partição da tabela Base usada ao realizar uma operação Sync. Esse campo permite que uma operação Sync seja executada quando a tabela utiliza uma chave de partição personalizada.

- `deltaIndexName?: string` (opcional)

O índice usado para a operação Sync. Esse índice é necessário para habilitar uma operação de sincronização em toda a tabela de armazenamento delta quando a tabela usa uma chave de partição personalizada. A operação de sincronização será executada no GSI (criado em `gsi_ds_pk` e `gsi_ds_sk`).

- `filter?: DynamoDBFilterObject<T> | null` (opcional)

Um filtro opcional para aplicar aos resultados depois de recuperá-los da tabela.

- `lastSync?: number` (opcional)

O momento, em milésimos de segundos de epoch, no qual a última operação de sincronização bem-sucedida foi iniciada. Se especificado, somente os itens que foram alterados após `lastSync` serão retornados. Este campo deve ser preenchido somente depois de recuperar todas as páginas de uma operação inicial de sincronização. Se omitidos, os resultados da tabela base serão retornados. Caso contrário, os resultados da tabela delta serão retornados.

- `limit?: number | null` (opcional)

O número máximo opcional de itens a serem avaliados ao mesmo tempo. Se omitido, o limite padrão será definido como 100 itens. O valor máximo para esse campo é de 1000 itens.

- `nextToken?: string | null` (opcional)

Type `DynamoDBUpdateInput<T>`

```
DynamoDBUpdateInput<T>: {  
  _version?: number;  
  condition?: DynamoDBFilterObject<T>;  
  customPartitionKey?: string;  
  key: DynamoDBKey<T>;  
  populateIndexFields?: boolean;  
  update: DynamoDBUpdateObject<T>;  
}
```

Declaração de tipo

- `_version?: number` (opcional)
- `condition?: DynamoDBFilterObject<T>` (opcional)

Quando você atualiza um objeto no DynamoDB, pode especificar uma expressão condicional que controla se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB antes que a operação seja realizada.

- `customPartitionKey?: string` (opcional)

Quando ativado, esse valor `customPartitionKey` modifica o formato dos registros `ds_sk` e `ds_pk` usados pela tabela de sincronização delta quando o versionamento é ativado. Quando ativado, o processamento da entrada `populateIndexFields` também é ativado.

- `key: DynamoDBKey<T>` (obrigatório)

Um parâmetro obrigatório que especifica a chave do item no DynamoDB que está sendo atualizado. Os itens do DynamoDB podem ter uma única chave de hash ou chaves de hash e de classificação.

- `populateIndexFields?: boolean` (opcional)

Um valor booleano que, quando ativado com `customPartitionKey`, cria novas entradas para cada registro na tabela de sincronização delta, especificamente nas colunas `gsi_ds_pk` e `gsi_ds_sk`.

- `update: DynamoDBUpdateObject<T>`

Um objeto que especifica os atributos a serem atualizados junto com os novos valores para eles. O objeto de atualização pode ser usado com `add`, `remove`, `replace`, `increment`, `decrement`, `append`, `prepend`, `updateListItem`.

Funções do módulo do Amazon RDS

As funções do módulo do Amazon RDS oferecem uma experiência aprimorada ao interagir com bancos de dados configurados com a API de dados do Amazon RDS. O módulo é importado usando `@aws-appsync/utils/rds`:

```
import * as rds from '@aws-appsync/utils/rds';
```

As funções também podem ser importadas individualmente. Por exemplo, a importação abaixo usa `sql`:

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

Funções

Você pode usar os auxiliares utilitários do módulo AWS AppSync RDS para interagir com seu banco de dados.

Selecionar

O utilitário `select` cria uma declaração `SELECT` para consultar o banco de dados relacional.

Uso básico

Na forma básica, é possível especificar a tabela que deseja consultar:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

  // Generates statement:
  // "SELECT * FROM "persons"
  return createPgStatement(select({table: 'persons'}));
}
```

Observe também que é possível especificar o esquema no identificador da tabela:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {

  // Generates statement:
  // SELECT * FROM "private"."persons"
  return createPgStatement(select({table: 'private.persons'}));
}
```

Especificar colunas

É possível especificar colunas com a propriedade `columns`. Se isso não for definido como um valor, o padrão será `*`:

```
export function request(ctx) {

  // Generates statement:
  // SELECT "id", "name"
  // FROM "persons"
  return createPgStatement(select({
    table: 'persons',
    columns: ['id', 'name']
  }));
}
```

Também é possível especificar a tabela de uma coluna:

```
export function request(ctx) {

  // Generates statement:
  // SELECT "id", "persons"."name"
  // FROM "persons"
  return createPgStatement(select({
    table: 'persons',
    columns: ['id', 'persons.name']
  }));
}
```

Limites e deslocamentos:

É possível aplicar `limit` e `offset` à consulta:

```
export function request(ctx) {

  // Generates statement:
  // SELECT "id", "name"
  // FROM "persons"
  // LIMIT :limit
  // OFFSET :offset
  return createPgStatement(select({
    table: 'persons',
    columns: ['id', 'name'],
    limit: 10,
    offset: 40
  }));
}
```

Ordenar por

É possível classificar os resultados com a propriedade `orderBy`. Forneça uma matriz de objetos especificando a coluna e uma propriedade `dir` opcional:

```
export function request(ctx) {  
  
  // Generates statement:  
  // SELECT "id", "name" FROM "persons"  
  // ORDER BY "name", "id" DESC  
  return createPgStatement(select({  
    table: 'persons',  
    columns: ['id', 'name'],  
    orderBy: [{column: 'name'}, {column: 'id', dir: 'DESC'}]  
  }));  
}
```

Filtros

É possível criar filtros usando o objeto de condição especial:

```
export function request(ctx) {  
  
  // Generates statement:  
  // SELECT "id", "name"  
  // FROM "persons"  
  // WHERE "name" = :NAME  
  return createPgStatement(select({  
    table: 'persons',  
    columns: ['id', 'name'],  
    where: {name: {eq: 'Stephane'}}  
  }));  
}
```

Também é possível combinar filtros:

```
export function request(ctx) {  
  
  // Generates statement:  
  // SELECT "id", "name"  
  // FROM "persons"  
  // WHERE "name" = :NAME and "id" > :ID
```



```
return createPgStatement(select({
  table: 'persons',
  columns: ['id', 'name'],
  where: {name: {eq: 'Stephane'}, id: {gt: 10}}
}));
}
```

Também é possível criar declarações OR:

```
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 } }
    ]}
  }));
}
```

Também é possível negar uma condição com 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 } }
    ]}
  }));
}
```

Também é possível usar os seguintes operadores para comparar valores:

Operador	Descrição	Tipos de valores possíveis
eq	Equal	número, string, booleano
um	Not equal	número, string, booleano
le	Menor ou igual a	número, seqüência
lt	Menor que	número, seqüência
idade	Maior ou igual a	número, seqüência
gt	Maior que	número, seqüência
contém	Like	string
NÃO contém	Não é como	string
Começa com	Começa com o prefixo	string
entre	Entre dois valores	número, seqüência
O atributo existe	O atributo não é nulo	número, string, booleano
size	verifica o comprimento do elemento	string

Inserir

O utilitário `insert` oferece uma maneira simples de inserir itens de linha única no banco de dados com a operação `INSERT`.

Inserções de item único

Para inserir um item, especifique a tabela e, depois, transmita o objeto de valores. As chaves do objeto são associadas às colunas da tabela. São inseridos automaticamente caracteres de escape nos nomes das colunas e os valores são enviados ao banco de dados usando o mapa de variáveis:

```
import { insert, createMySQLStatement } from '@aws-appsync/utils/rds';
```

```
export function request(ctx) {
  const { input: values } = ctx.args;
  const insertStatement = insert({ table: 'persons', values });

  // Generates statement:
  // INSERT INTO `persons`(`name`)
  // VALUES(:NAME)
  return createMySQLStatement(insertStatement)
}
```

Caso de uso do MySQL

É possível combinar um insert seguido por um select para recuperar a linha inserida:

```
import { insert, select, createMySQLStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
  const { input: values } = ctx.args;
  const insertStatement = insert({ table: 'persons', values });
  const selectStatement = select({
    table: 'persons',
    columns: '*',
    where: { id: { eq: values.id } },
    limit: 1,
  });

  // Generates statement:
  // INSERT INTO `persons`(`name`)
  // VALUES(:NAME)
  // and
  // SELECT *
  // FROM `persons`
  // WHERE `id` = :ID
  return createMySQLStatement(insertStatement, selectStatement)
}
```

Caso de uso do Postgres

Com o Postgres, é possível usar [returning](#) para obter dados da linha que você inseriu. Ele aceita * ou uma matriz de nomes de colunas:

```
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)
}
```

Atualizar

O utilitário `update` permite atualizar as linhas existentes. É possível usar o objeto de condição para aplicar alterações às colunas especificadas em todas as linhas que atendam à condição. Por exemplo, digamos que temos um esquema que nos permita fazer essa mutação. Queremos atualizar o nome de `Person` com o valor `id` de 3, mas somente se os conhecermos (`known_since`) desde o ano 2000:

```
mutation Update {
  updatePerson(
    input: {id: 3, name: "Jon"},
    condition: {known_since: {ge: "2000"}}
  ) {
    id
    name
  }
}
```

Nosso resolvedor de atualização é semelhante a:

```
import { update, createPgStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
  const { input: { id, ...values }, condition } = ctx.args;
  const where = {
    ...condition,
```

```

        id: { eq: id },
    };
    const updateStatement = update({
        table: 'persons',
        values,
        where,
        returning: ['id', 'name'],
    });

    // Generates statement:
    // UPDATE "persons"
    // SET "name" = :NAME, "birthday" = :BDAY, "country" = :COUNTRY
    // WHERE "id" = :ID
    // RETURNING "id", "name"
    return createPgStatement(updateStatement)
}

```

Podemos adicionar uma verificação à nossa condição para garantir que somente a linha que tem a chave primária `id` igual a 3 seja atualizada. Da mesma forma, para Postgres inserts, é possível usar `returning` para exibir os dados modificados.

Remover

O utilitário `remove` permite excluir as linhas existentes. É possível usar o objeto de condição em todas as linhas que atendam à condição. Observe que `delete` é uma palavra-chave reservada em JavaScript. `remove` deve ser usado em vez disso:

```

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(deleteStatement)
}

```

```
}
```

Conversão

Em alguns casos, convém ter maior especificidade sobre o tipo de objeto correto a ser usado na declaração. Você pode usar as dicas de tipo fornecidas para especificar o tipo dos seus parâmetros. AWS AppSync suporta os [mesmos tipos de dicas](#) da API de dados. Você pode converter seus parâmetros usando as `typeHint` funções do AWS AppSync `rds` módulo.

O exemplo a seguir permite enviar uma matriz como um valor que é convertido como um objeto JSON. Usamos o operador `->2` para recuperar o elemento `index 2` na matriz JSON:

```
import { sql, createPgStatement, toJsonObject, typeHint } from '@aws-appsync/utils/rds';

export function request(ctx) {
  const arr = ctx.args.list_of_ids
  const statement = sql`select ${typeHint.JSON(arr)}->2 as value`
  return createPgStatement(statement)
}

export function response(ctx) {
  return toJsonObject(ctx.result)[0][0].value
}
```

A conversão também é útil ao processar e comparar `DATE`, `TIME` e `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)
}
```

Veja outro exemplo de como enviar a data e a hora atuais:

```
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)}`)
}
```

Dicas de tipo disponíveis

- `typeHint.DATE`: o parâmetro correspondente é enviado como objeto do tipo `DATE` ao banco de dados. O formato aceito é `YYYY-MM-DD`.
- `typeHint.DECIMAL`: o parâmetro correspondente é enviado como objeto do tipo `DECIMAL` ao banco de dados.
- `typeHint.JSON`: o parâmetro correspondente é enviado como objeto do tipo `JSON` ao banco de dados.
- `typeHint.TIME`: o valor de parâmetro de string correspondente é enviado como objeto do tipo `TIME` ao banco de dados. O formato aceito é `HH:MM:SS[.FFF]`.
- `typeHint.TIMESTAMP`: o valor de parâmetro de string correspondente é enviado como objeto do tipo `TIMESTAMP` ao banco de dados. O formato aceito é `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `typeHint.UUID`: o valor de parâmetro de string correspondente é enviado como objeto do tipo `UUID` ao banco de dados.

Utilitários Runtime

A biblioteca `runtime` fornece utilitários para controlar ou modificar as propriedades de runtime dos seus resolvedores e funções.

Lista de utilitários de runtime

`runtime.earlyReturn(obj?: unknown, returnOptions?: {skipTo: 'END' | 'NEXT'}): never`

Invocar essa função interromperá a execução do manipulador, AWS AppSync função ou resolvidor atual (Unit ou Pipeline Resolver), dependendo do contexto atual. O objeto especificado é retornado como resultado.

- Quando chamado em um manipulador de solicitação de AWS AppSync função, a fonte de dados e o manipulador de resposta são ignorados e o próximo manipulador de solicitação de função (ou o manipulador de resposta do resolvidor de pipeline, se essa for a última função) é chamado. AWS AppSync

- Quando chamado em um manipulador de solicitações do resolvidor de AWS AppSync pipeline, a execução do pipeline é ignorada e o manipulador de resposta do resolvidor de pipeline é chamado imediatamente.
- Quando `returnOptions` é fornecido com `skipTo` definido como “END”, a execução do pipeline é ignorada e o manipulador de resposta do resolvidor do pipeline é chamado imediatamente.
- Quando `returnOptions` é fornecido com o `skipTo` valor definido como “NEXT”, a execução da função é ignorada e o próximo manipulador de pipeline é chamado.

Exemplo

```
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
}
```

Auxiliares de tempo de `util.time`

A variável `util.time` contém métodos de data e hora para ajudar a gerar timestamps, converter entre formatos de data e hora e analisar strings de data e hora. A sintaxe dos formatos de data e hora é baseada na [DateTimeFormatter](#) qual você pode consultar para obter mais documentação.

Lista de utilitários de tempo

`util.time.nowISO8601()`

Retorna uma representação String do UTC no [formato ISO86 01](#).

`util.time.nowEpochSeconds()`

Retorna o número de segundos desde epoch do 1970-01-01T00:00:00Z até agora.

util.time.nowEpochMilliseconds()

Retorna o número de milissegundos desde epoch do 1970-01-01T00:00:00Z até agora.

util.time.nowFormatted(String)

Retorna uma string do timestamp atual em UTC usando o formato especificado a partir de um tipo de entrada String.

util.time.nowFormatted(String, String)

Retorna uma string do timestamp atual para um fuso horário usando o formato especificado e o fuso horário a partir de tipos de entrada String.

util.time.parseFormattedToEpochMilliseconds(String, String)

Analisa um carimbo de data/hora passado como uma string junto com um formato contendo um fuso horário e, em seguida, retorna o carimbo de data/hora em milissegundos desde a época.

util.time.parseFormattedToEpochMilliseconds(String, String, String)

Analisa um timestamp enviado como uma string junto com um formato e fuso horário e retorna o timestamp como milissegundos desde o epoch.

util.time.parseISO8601ToEpochMilliseconds(String)

Analisa um carimbo de data/hora ISO86 01 passado como uma string e, em seguida, retorna o carimbo de data/hora em milissegundos desde a época.

util.time.epochMillisecondsToSeconds(long)

Converte um timestamp em milissegundos epoch em um timestamp em segundos epoch.

util.time.epochMillisecondsToISO8601(long)

Converte um carimbo de data/hora de milissegundos de época em um carimbo de data/hora de 0,1. ISO86

util.time.epochMillisecondsToFormatted(long, String)

Converte um timestamp em milissegundos epoch enviado como long em um timestamp formatado de acordo com o formato em UTC.

util.time.epochMillisecondsToFormatted(long, String, String)

Converte um timestamp em milissegundos epoch enviado como um long em um timestamp formatado de acordo com o formato no fuso horário fornecido.

Auxiliares do DynamoDB em `util.dynamodb`

`util.dynamodb` contém os métodos auxiliares que facilitam gravar e ler dados no Amazon DynamoDB, como mapeamento e formatação do tipo automático.

`toDynamoDB`

Lista de utilitários do `toDynamoDB`

`util.dynamodb.toDynamoDB(Object)`

Ferramenta de conversão de objetos gerais do DynamoDB que converte objetos de entrada para a representação adequada do DynamoDB. Ela tem opiniões fortes sobre como representa alguns tipos: por exemplo, usará listas ("L") em vez de conjuntos ("SS", "NS", "BS"). Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

Exemplo de string

```
Input:      util.dynamodb.toDynamoDB("foo")
Output:     { "S" : "foo" }
```

Exemplo de número

```
Input:      util.dynamodb.toDynamoDB(12345)
Output:     { "N" : 12345 }
```

Exemplo de booleano

```
Input:      util.dynamodb.toDynamoDB(true)
Output:     { "BOOL" : true }
```

Exemplo de lista

```
Input:      util.dynamodb.toDynamoDB([ "foo", 123, { "bar" : "baz" } ])
Output:     {
      "L" : [
        { "S" : "foo" },
        { "N" : 123 },
        {
          "M" : {
            "bar" : { "S" : "baz" }
          }
        }
      ]
    }
```

```

    }
  }
}

```

Exemplo de mapa

```

Input:      util.dynamodb.toDynamoDB({ "foo": "bar", "baz" : 1234, "beep":
[ "boop" ] })
Output:     {
    "M" : {
      "foo" : { "S" : "bar" },
      "baz" : { "N" : 1234 },
      "beep" : {
        "L" : [
          { "S" : "boop" }
        ]
      }
    }
  }
}

```

Utilitários toString

Lista de utilitários toString

util.dynamodb.toString(String)

Converte uma string de entrada para o formato de string do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```

Input:      util.dynamodb.toString("foo")
Output:     { "S" : "foo" }

```

util.dynamodb.toStringSet(List<String>)

Converte uma lista com strings para o formato de conjunto de strings do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```

Input:      util.dynamodb.toStringSet([ "foo", "bar", "baz" ])
Output:     { "SS" : [ "foo", "bar", "baz" ] }

```

Utilitários toNumber

Lista de utilitários toNumber

util.dynamodb.toNumber(Number)

Converte um número para o formato de número do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      util.dynamodb.toNumber(12345)
Output:     { "N" : 12345 }
```

util.dynamodb.toNumberSet(List<Number>)

Converte uma lista de números para o formato de conjunto de números do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      util.dynamodb.toNumberSet([ 1, 23, 4.56 ])
Output:     { "NS" : [ 1, 23, 4.56 ] }
```

Utilitários toBinary

Lista de utilitários toBinary

util.dynamodb.toBinary(String)

Converte dados binários codificados como uma string em base64 para o formato binário do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      util.dynamodb.toBinary("foo")
Output:     { "B" : "foo" }
```

util.dynamodb.toBinarySet(List<String>)

Converte uma lista de dados binários codificados como strings em base64 para o formato de conjunto de binários do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      util.dynamodb.toBinarySet([ "foo", "bar", "baz" ])
Output:     { "BS" : [ "foo", "bar", "baz" ] }
```

Utilitários toBoolean

Lista de utilitários toBoolean

util.dynamodb.toBoolean(Boolean)

Converte um booleano para o formato booleano adequado do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      util.dynamodb.toBoolean(true)
Output:     { "BOOL" : true }
```

Utilitários toNull

Lista de utilitários toNull

util.dynamodb.toNull()

Retorna um valor nulo no formato nulo do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      util.dynamodb.toNull()
Output:     { "NULL" : null }
```

Utilitários toList

Lista de utilitários toList

util.dynamodb.toList(List)

Converte uma lista de objetos no formato de lista do DynamoDB. Cada item na lista também é convertido para o formato adequado do DynamoDB. Ela tem opiniões fortes sobre como representa alguns dos objetos aninhados: por exemplo, usará listas ("L") em vez de conjuntos ("SS", "NS", "BS"). Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      util.dynamodb.toList([ "foo", 123, { "bar" : "baz" } ])
Output:     {
        "L" : [
            { "S" : "foo" },
```

```

        { "N" : 123 },
        {
            "M" : {
                "bar" : { "S" : "baz" }
            }
        }
    ]
}

```

Utilitários toMap

Lista de utilitários toMap

util.dynamodb.toMap(Map)

Converte um mapa para o formato de mapa do DynamoDB. Cada valor no mapa também é convertido para o formato adequado do DynamoDB. Ela tem opiniões fortes sobre como representa alguns dos objetos aninhados: por exemplo, usará listas ("L") em vez de conjuntos ("SS", "NS", "BS"). Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```

Input:      util.dynamodb.toMap({ "foo": "bar", "baz" : 1234, "beep": [ "boop" ] })
Output:     {
              "M" : {
                  "foo"  : { "S" : "bar" },
                  "baz"  : { "N" : 1234 },
                  "beep" : {
                      "L" : [
                          { "S" : "boop" }
                      ]
                  }
              }
          }

```

util.dynamodb.toMapValues(Map)

Cria uma cópia do mapa onde cada valor foi convertido para o formato adequado do DynamoDB. Ela tem opiniões fortes sobre como representa alguns dos objetos aninhados: por exemplo, usará listas ("L") em vez de conjuntos ("SS", "NS", "BS").

```

Input:      util.dynamodb.toMapValues({ "foo": "bar", "baz" : 1234, "beep":
          [ "boop" ] })

```

```
Output:  {
    "foo" : { "S" : "bar" },
    "baz" : { "N" : 1234 },
    "beep" : {
        "L" : [
            { "S" : "boop" }
        ]
    }
}
```

Note

Observação: isso é um pouco diferente de `util.dynamodb.toMap(Map)`, uma vez que retorna somente o conteúdo do valor do atributo do DynamoDB, mas não todo o valor do atributo em si. Por exemplo, as seguintes instruções são exatamente as mesmas:

```
util.dynamodb.toMapValues(<map>)
util.dynamodb.toMap(<map>)("M")
```

Utilitários S3Object

Lista de utilitários do S3Object

`util.dynamodb.toS3Object(String key, String bucket, String region)`

Converte a chave, bucket e região na representação de Objeto do S3 do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      util.dynamodb.toS3Object("foo", "bar", region = "baz")
Output:     { "S" : "{ \"s3\" : { \"key\" : \"foo\", \"bucket\" : \"bar\", \"region\" : \"baz\" } }" }
```

`util.dynamodb.toS3Object(String key, String bucket, String region, String version)`

Converte a chave, o bucket, a região e a versão opcional na representação de objeto do S3 do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      util.dynamodb.toS3Object("foo", "bar", "baz", "beep")
```

```
Output:      { "S" : "{ \"s3\" : { \"key\" : \"foo\", \"bucket\" : \"bar\", \"region  
\" : \"baz\", \"version\" = \"beep\" } }" }
```

util.dynamodb.fromS3ObjectJson(String)

Aceita o valor de string de um objeto do S3 do DynamoDB e retorna um mapa que contém a chave, o bucket, a região e a versão opcional.

```
Input:      util.dynamodb.fromS3ObjectJson({ "S" : "{ \"s3\" : { \"key\" : \"foo\",  
\"bucket\" : \"bar\", \"region\" : \"baz\", \"version\" = \"beep\" } }" })  
Output:     { "key" : "foo", "bucket" : "bar", "region" : "baz", "version" :  
"beep" }
```

Auxiliares HTTP em util.http

O utilitário `util.http` fornece métodos auxiliares que podem ser usados para gerenciar parâmetros de solicitação HTTP e adicionar cabeçalhos de resposta.

Lista de utilitários util.http

util.http.copyHeaders(headers)

Copia os cabeçalhos do mapa, excluindo os seguintes cabeçalhos HTTP restritos:

- transfer-encoding
- connection
- host
- expect
- keep-alive
- upgrade
- proxy-authenticate
- proxy-authorization
- te
- content-length

util.http.addResponseHeader(String, Object)

Adiciona um único cabeçalho personalizado com o nome (`String`) e o valor (`Object`) da resposta. As limitações a seguir se aplicam a:

- Além da lista de cabeçalhos restritos `paracopyHeaders(headers)`, os nomes dos cabeçalhos não podem corresponder a nenhuma das seguintes opções:
 - `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`
- Os nomes dos cabeçalhos não podem começar com os prefixos restritos `x-amzn-` ou `amz-`.
- O tamanho dos cabeçalhos de resposta personalizados não pode exceder 4 KB. Isso inclui nomes e valores de cabeçalho.
- Você deve definir cada cabeçalho de resposta uma vez por operação do GraphQL. No entanto, se você definir um cabeçalho personalizado com o mesmo nome várias vezes, a definição mais recente aparecerá na resposta. Todos os cabeçalhos são contabilizados para o limite de tamanho do cabeçalho, independentemente do nome.
- Cabeçalhos com um nome vazio (`String`) ou restrito ou um valor nulo (`Object`) serão ignorados e gerarão um `ResponseHeaderError` erro que será adicionado à saída da `errors` operação.

```
export function request(ctx) {
  util.http.addResponseHeader('itemsCount', 7)
  util.http.addResponseHeader('render', ctx.args.render)
  return {}
}
```

`util.http.addResponseHeaders(Map)`

Adiciona vários cabeçalhos de resposta à resposta do mapa especificado de nomes (`String`) e valores (`Object`). As mesmas limitações listadas para o método `addResponseHeader(String, Object)` também se aplicam a esse método.

```
export function request(ctx) {
  const headers = {
    headerInt: 12,
```

```
    headerString: 'stringValue',
    headerObject: {
      field1: 7,
      field2: 'string'
    }
  }
  util.http.addResponseHeaders(headers)
  return {}
}
```

Auxiliares de transformação em util.transform

`util.transform` contém métodos auxiliares que facilitam a execução de operações complexas em fontes de dados.

Lista de utilitários auxiliares de transformação

`util.transform.toDynamoDBFilterExpression(filterObject: DynamoDBFilterObject) : string`

Converte uma string de entrada em uma expressão de filtro para o uso com o DynamoDB.

Recomendamos o uso `toDynamoDBFilterExpression` com [funções de módulo integradas](#).

`util.transform.toElasticsearchQueryDSL(object: OpenSearchQueryObject) : string`

Converte a entrada fornecida em sua expressão DSL de OpenSearch consulta equivalente, retornando-a como uma string JSON.

Exemplo de entrada:

```
util.transform.toElasticsearchQueryDSL({
  "upvotes":{
    "ne":15,
    "range":[
      10,
      20
    ]
  },
  "title":{
    "eq":"hihihi",
    "wildcard":"h*i"
  }
})
```

```
}  
}))
```

Exemplos de resultado:

```
{  
  "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

Presume-se que o operador padrão seja AND.

```
util.transform.toSubscriptionFilter(objFilter, ignoredFields?, rules?):  
SubscriptionFilter
```

Converte um objeto de entrada Map em um objeto de expressão SubscriptionFilter. O método `util.transform.toSubscriptionFilter` é usado como entrada para a extensão `extensions.setSubscriptionFilter()`. Para obter mais informações, consulte [Extensões](#).

 Note

Os parâmetros e a declaração de retorno estão listados abaixo:

Parâmetros

- objFilter: SubscriptionFilterObject

Um objeto de entrada Map que é convertido no objeto de expressão SubscriptionFilter.

- ignoredFields: SubscriptionFilterExcludeKeysType (opcional)

Um List dos nomes de campo no primeiro objeto que serão ignorados.

- `rules: SubscriptionFilterRuleObject` (opcional)

Um objeto Map de entrada com regras rígidas que é incluído quando você está criando o objeto de expressão SubscriptionFilter. Essas regras estritas são incluídas no objeto de expressão SubscriptionFilter de forma que pelo menos uma das regras seja satisfeita para passar pelo filtro de assinatura.

Resposta

Retorna um [SubscriptionFilter](#).

util.transform.toSubscriptionFilter(Map, List)

Converte um objeto de entrada Map em um objeto de expressão SubscriptionFilter. O método util.transform.toSubscriptionFilter é usado como entrada para a extensão extensions.setSubscriptionFilter(). Para obter mais informações, consulte [Extensões](#).

O primeiro argumento é o objeto de entrada Map que é convertido no objeto de expressão SubscriptionFilter. O segundo argumento é uma List de nomes de campos que são ignorados no primeiro objeto de entrada Map durante a criação da estrutura do objeto de expressão SubscriptionFilter.

util.transform.toSubscriptionFilter(Map, List, Map)

Converte um objeto de entrada Map em um objeto de expressão SubscriptionFilter. O método util.transform.toSubscriptionFilter é usado como entrada para a extensão extensions.setSubscriptionFilter(). Para obter mais informações, consulte [Extensões](#).

util.transform.toDynamoDBConditionExpression(conditionObject)

Cria uma expressão de condição do DynamoDB.

Argumentos do filtro de assinatura

A tabela a seguir explica como os argumentos dos seguintes utilitários são definidos:

- Util.transform.toSubscriptionFilter(objFilter, ignoredFields?, rules?): SubscriptionFilter

Argument 1: Map

O argumento 1 é um objeto Map com os seguintes valores-chave:

- nomes de campos
- "and"
- "or"

Para nomes de campos como chaves, as condições nas entradas desses campos estão no formato "operator" : "value".

O exemplo a seguir mostra como entradas podem ser adicionadas ao Map:

```
"field_name" : {  
    "operator1" : value  
}  
  
## We can have multiple conditions for the same field_name:  
  
"field_name" : {  
    "operator1" : value  
    "operator2" : value  
    .  
    .  
    .  
}
```

Quando um campo contém duas ou mais condições, todas essas condições são consideradas para usar a operação OR.

A entrada Map também pode ter "and" e "or" como chaves, o que implica que todas as entradas dentro dessas devem ser unidas usando a lógica AND ou OR dependendo da chave. Os valores-chave "and" e "or" esperam uma série de condições.

```
"and" : [  
    {  
        "field_name1" : {  
            "operator1" : value  
        }  
    },  
    {  
        "field_name2" : {  
            "operator1" : value  
        }  
    },  
    .  
    .  
].
```

Observe que você pode aninhar "and" e "or". Ou seja, você pode ter aninhado "and"/"or" em outro bloco "and"/"or". No entanto, isso não funciona em campos simples.

```
"and" : [
  {
    "field_name1" : {
      "operator" : value
    }
  },
  {
    "or" : [
      {
        "field_name2" : {
          "operator" : value
        }
      },
      {
        "field_name3" : {
          "operator" : value
        }
      }
    ]
  }
].
```

O exemplo a seguir mostra uma entrada do argumento 1 usando `util.transform.toSubscriptionFilter(Map) : Map`.

Entrada(s)

Argumento 1: mapa:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    }
  ]
}
```

```
    }
  },
  {
    "downvotes": {
      "gt": 2000
    }
  }
],
"or": [
  {
    "author": {
      "eq": "Admin"
    }
  },
  {
    "isPublished": {
      "eq": false
    }
  }
]
}
```

Saída

O resultado é um objeto Map:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "percentageUp",
          "operator": "lte",
          "value": 50
        },
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 2000
        }
      ]
    }
  ]
}
```



```
    },
    {
      "fieldName": "author",
      "operator": "eq",
      "value": "Admin"
    }
  ]
},
{
  "filters": [
    {
      "fieldName": "percentageUp",
      "operator": "lte",
      "value": 50
    },
    {
      "fieldName": "title",
      "operator": "ne",
      "value": "Book1"
    },
    {
      "fieldName": "downvotes",
      "operator": "gt",
      "value": 2000
    },
    {
      "fieldName": "isPublished",
      "operator": "eq",
      "value": false
    }
  ]
},
{
  "filters": [
    {
      "fieldName": "percentageUp",
      "operator": "gte",
      "value": 20
    },
    {
      "fieldName": "title",
      "operator": "ne",
      "value": "Book1"
    }
  ],
```

```
{
  {
    "fieldName": "downvotes",
    "operator": "gt",
    "value": 2000
  },
  {
    "fieldName": "author",
    "operator": "eq",
    "value": "Admin"
  }
]
},
{
  "filters": [
    {
      "fieldName": "percentageUp",
      "operator": "gte",
      "value": 20
    },
    {
      "fieldName": "title",
      "operator": "ne",
      "value": "Book1"
    },
    {
      "fieldName": "downvotes",
      "operator": "gt",
      "value": 2000
    },
    {
      "fieldName": "isPublished",
      "operator": "eq",
      "value": false
    }
  ]
}
]
```

Argument 2: List

O argumento 2 contém uma `List` de nomes de campos que não devem ser considerados na entrada `Map` (argumento 1) durante a criação da estrutura do objeto de expressão `SubscriptionFilter`. A `List` também pode estar vazia.

O exemplo a seguir mostra uma entrada do argumento 1 e argumento 2 usando `util.transform.toSubscriptionFilter(Map, List) : Map`.

Entrada(s)

Argumento 1: mapa:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Argumento 2: lista:

```
["percentageUp", "author"]
```

Saída

O resultado é um objeto Map:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        },
        {
          "fieldName": "isPublished",
          "operator": "eq",
          "value": false
        }
      ]
    }
  ]
}
```

Argument 3: Map

O argumento 3 é um objeto Map que tem nomes de campo como valores-chave (não pode ter "and" ou "or"). Para nomes de campos como chaves, as condições nesses campos são entradas no formato "operator" : "value". Diferentemente do argumento 1, o argumento 3 não pode ter várias condições na mesma chave. Além disso, o argumento 3 não possui uma cláusula "and" ou "or"; portanto, também não há aninhamento envolvido.

O argumento 3 representa uma lista de regras estritas, que são adicionadas ao objeto de expressão `SubscriptionFilter` para que pelo menos uma dessas condições seja atendida para passar pelo filtro.

```
{
  "fieldname1": {
    "operator": value
  },
  "fieldname2": {
    "operator": value
  }
}
.
.
.
```

O exemplo a seguir mostra as entradas de argumento 1, argumento 2 e argumento 3 usando `util.transform.toSubscriptionFilter(Map, List, Map) : Map`.

Entrada(s)

Argumento 1: mapa:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "lt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
```

```
      "eq": "Admin"
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Argumento 2: lista:

```
["percentageUp", "author"]
```

Argumento 3: mapa:

```
{
  "upvotes": {
    "gte": 250
  },
  "author": {
    "eq": "Person1"
  }
}
```

Saída

O resultado é um objeto Map:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "title",
          "operator": "ne",
          "value": "Book1"
        },
        {
          "fieldName": "downvotes",
          "operator": "gt",
          "value": 20
        }
      ]
    }
  ]
}
```

```
    },
    {
      "fieldName": "isPublished",
      "operator": "eq",
      "value": false
    },
    {
      "fieldName": "upvotes",
      "operator": "gte",
      "value": 250
    }
  ]
},
{
  "filters": [
    {
      "fieldName": "title",
      "operator": "ne",
      "value": "Book1"
    },
    {
      "fieldName": "downvotes",
      "operator": "gt",
      "value": 20
    },
    {
      "fieldName": "isPublished",
      "operator": "eq",
      "value": false
    },
    {
      "fieldName": "author",
      "operator": "eq",
      "value": "Person1"
    }
  ]
}
]
```

Auxiliares de string em util.str

`util.str` contém métodos para ajudar com operações comuns de string.

Lista de utilitários util.str

`util.str.normalize(String, String)`

Normaliza uma string usando uma das quatro formas de normalização unicode: NFC, NFD, NFKC ou NFKD. O primeiro argumento é a string a ser normalizada. O segundo argumento é “nfc”, “nfd”, “nfkc” ou “nfkd”, especificando o tipo de normalização a ser usado no processo de normalização.

Extensões

`extensions` contém um conjunto de métodos para realizar ações adicionais nos seus resolvedores.

Extensões do cache

`extensions.evictFromApiCache(typeName: string, fieldName: string, keyValuePair: Record<string, string>) : Object`

Elimina um item do cache do lado do AWS AppSync servidor. O primeiro argumento é o nome do tipo. O segundo argumento é o nome do campo. O terceiro argumento é um objeto contendo itens do par de chave/valor que especificam o valor da chave de armazenamento em cache. Você deve colocar os itens no objeto na mesma ordem das chaves de cache em `cachingKey` do resolvedor em cache. Para obter mais informações sobre armazenamento em cache, consulte [Comportamento de cache](#).

Exemplo 1:

Este exemplo remove os itens que foram armazenados em cache para um resolvedor chamado `Query.allClasses` no qual uma chave de cache chamada `context.arguments.semester` foi usada. Quando a mutação é chamada e o resolvedor é executado, se uma entrada for limpa com sucesso, a resposta conterá um valor `apiCacheEntriesDeleted` no objeto de extensões que mostra quantas entradas foram excluídas.

```
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

Essa função funciona somente para mutações, não para consultas.

Extensões de assinatura

`extensions.setSubscriptionFilter(filterJsonObject)`

Define filtros de assinatura aprimorados. Cada evento de notificação de assinatura é avaliado em relação aos filtros de assinatura fornecidos e envia notificações aos clientes se todos os filtros forem avaliados como `true`. O argumento é `filterJsonObject` (Mais informações sobre esse argumento podem ser encontradas abaixo na `filterJsonObject` seção Argumento:). Consulte [Filtragem de assinatura avançada](#).

 Note

Você pode usar esse método de extensão somente no manipulador de resposta de um resolvidor de assinatura. Além disso, recomendamos usar `util.transform.toSubscriptionFilter` para criar seu filtro.

`extensions.setSubscriptionInvalidationFilter(filterJsonObject)`

Define os filtros de invalidação da assinatura. Os filtros de assinatura são avaliados em relação à carga de invalidação e, em seguida, invalidam determinada assinatura se os filtros forem avaliados como `true`. O argumento é `filterJsonObject` (Mais informações sobre esse argumento podem ser encontradas abaixo na `filterJsonObject` seção Argumento:). Consulte [Filtragem de assinatura avançada](#).

Note

Você pode usar esse método de extensão somente no manipulador de resposta de um resolvidor de assinatura. Além disso, recomendamos usar `util.transform.toSubscriptionFilter` para criar seu filtro.

`extensions.invalidateSubscriptions(invalidationJsonObject)`

Usado para iniciar uma invalidação de assinatura a partir de uma mutação. O argumento é `invalidationJsonObject` (Mais informações sobre esse argumento podem ser encontradas abaixo na `invalidationJsonObject` seção Argumento:).

Note

Essa extensão pode ser usada somente nos modelos de mapeamento de resposta dos resolvidores de mutação.

Você só pode usar no máximo cinco chamadas de método `extensions.invalidateSubscriptions()` exclusivas em uma única solicitação. Se você exceder esse limite, receberá um erro do GraphQL.

Argumento: `filterJsonObject`

O objeto JSON define filtros de assinatura ou de invalidação. É uma série de filtros em um `filterGroup`. Cada filtro é uma coleção de filtros individuais.

```
{
  "filterGroup": [
    {
      "filters" : [
        {
          "fieldName" : "userId",
          "operator" : "eq",
          "value" : 1
        }
      ]
    },
    {
```

```
      "filters" : [
        {
          "fieldName" : "group",
          "operator" : "in",
          "value" : ["Admin", "Developer"]
        }
      ]
    }
  ]
}
```

Cada filtro tem três atributos:

- `fieldName` – O campo do esquema GraphQL.
- `operator` – O tipo de operador.
- `value` – Os valores a serem comparados com o valor `fieldName` da notificação de assinatura.

Veja a seguir um exemplo de atribuição desses atributos:

```
{
  "fieldName" : "severity",
  "operator" : "le",
  "value" : context.result.severity
}
```

Argumento: `invalidationJsonObject`

O `invalidationJsonObject` define o seguinte:

- `subscriptionField` – A assinatura do esquema GraphQL a ser invalidada. Uma única assinatura, definida como uma string em `subscriptionField`, é considerada invalidada.
- `payload` – Uma lista de pares de valores-chave que é usada como entrada para invalidar assinaturas se o filtro de invalidação for avaliado como `true` em relação aos seus valores.

O exemplo a seguir invalida clientes inscritos e conectados usando a assinatura de `onUserDelete` quando o filtro de invalidação definido no resolvidor de assinatura é avaliado como `true` em relação ao valor `payload`.

```
export const request = (ctx) => ({ payload: null });
```

```
export function response(ctx) {
  extensions.invalidateSubscriptions({
    subscriptionField: 'onUserDelete',
    payload: { group: 'Developer', type: 'Full-Time' },
  });
  return ctx.result;
}
```

Auxiliares XML em util.xml

`util.xml` contém métodos para ajudar na conversão de strings XML.

Lista de utilitários util.xml

`util.xml.toMap(String)` : Object

Converte uma string XML para um dicionário.

Exemplo 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"
    }
  }
}
```

Exemplo 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

Converte uma string XML para uma string JSON. Isso é semelhante a `toMap`, exceto que a saída é uma string. Isso é útil se quiser converter e retornar diretamente a resposta XML de um objeto HTTP para JSON. Você pode definir um parâmetro booleano opcional para determinar se deseja codificar o JSON em string.

AWS AppSync JavaScript referência da função de resolução para o DynamoDB

A função AWS AppSync DynamoDB permite que você use o [GraphQL](#) para armazenar e recuperar dados em tabelas existentes do Amazon DynamoDB em sua conta mapeando uma solicitação de entrada do GraphQL em uma chamada do DynamoDB e, em seguida, mapeando a resposta do DynamoDB de volta para o GraphQL. Esta seção descreve os manipuladores de solicitação e resposta para as operações do DynamoDB compatíveis:

- [GetItem](#)- A GetItem solicitação permite que você instrua a função do DynamoDB a fazer uma GetItem solicitação ao DynamoDB e permite especificar a chave do item no DynamoDB e se deve usar uma leitura consistente ou não.
- [PutItem](#) - O documento de mapeamento de solicitações permite que você PutItem solicite à função do DynamoDB que faça uma PutItem solicitação ao DynamoDB e especifique a chave do item no DynamoDB, o conteúdo completo do item (composto por valores de chave e atributos) e condições para que a operação seja bem-sucedida.
- [UpdateItem](#) - A UpdateItem solicitação permite que você peça à função do DynamoDB que faça uma UpdateItem solicitação ao DynamoDB e especifique a chave do item no DynamoDB, uma expressão de atualização descrevendo como atualizar o item no DynamoDB e as condições para que a operação seja bem-sucedida.
- [DeleteItem](#) - A DeleteItem solicitação permite que você solicite à função do DynamoDB que faça uma DeleteItem solicitação ao DynamoDB e especifique a chave do item no DynamoDB e as condições para que a operação seja bem-sucedida.
- [Query](#): o objeto de solicitação Query permite orientar o resolvedor do DynamoDB a fazer uma solicitação Query ao DynamoDB, além de especificar a expressão de chave, qual índice usar, filtros adicionais, quantos itens retornar, se deve usar leituras consistentes, direção da consulta (para a frente ou para trás) e tokens de paginação.
- [Scan](#): a solicitação Scan permite orientar a função do DynamoDB a fazer uma solicitação Scan ao DynamoDB, além de especificar um filtro para excluir resultados, qual índice usar, quantos itens retornar, se deve usar leituras consistentes, tokens de paginação e verificações paralelas.
- [Sync](#): o objeto Sync permite recuperar todos os resultados de uma tabela do DynamoDB e, depois, receber apenas os dados alterados desde a última consulta (as atualizações delta). As solicitações Sync só podem ser feitas a fontes de dados versionadas do DynamoDB. Você pode especificar um filtro para excluir resultados, quantos itens devolver, tokens de paginação e quando sua última operação de sincronização foi iniciada.

- [BatchGetItem](#) - O objeto de BatchGetItem solicitação permite que você peça à função do DynamoDB que faça uma BatchGetItem solicitação ao DynamoDB para recuperar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar os nomes das tabelas das quais recuperar os itens e as chaves dos itens a serem recuperados de cada tabela.
- [BatchDeleteItem](#) - O objeto de BatchDeleteItem solicitação permite que você peça à função do DynamoDB que faça uma BatchWriteItem solicitação ao DynamoDB para excluir vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar os nomes das tabelas das quais excluir os itens e as chaves dos itens a serem excluídos de cada tabela.
- [BatchPutItem](#) - O objeto de BatchPutItem solicitação permite que você peça à função do DynamoDB que faça uma BatchWriteItem solicitação ao DynamoDB para colocar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar os nomes das tabelas nas quais colocar os itens e os itens completos a serem colocados em cada tabela.
- [TransactGetItems](#) - O objeto de TransactGetItems solicitação permite que você peça à função do DynamoDB que faça uma TransactGetItems solicitação ao DynamoDB para recuperar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar o nome da tabela de cada item de solicitação do qual recuperar o item e a chave de cada item a ser recuperado de cada tabela.
- [TransactWriteItems](#) - O objeto de TransactWriteItems solicitação permite que você peça à função do DynamoDB que faça uma TransactWriteItems solicitação ao DynamoDB para gravar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar o nome da tabela de destino de cada item de solicitação, a operação de cada item a ser executada e a chave de cada item a ser gravada.
- [Sistema de tipos \(mapeamento de solicitações\)](#) - Saiba mais sobre como a digitação do DynamoDB é integrada às solicitações. AWS AppSync
- [Sistema de tipos \(mapeamento de respostas\)](#): saiba mais sobre como os tipos do DynamoDB são convertidos automaticamente em GraphQL ou JSON em uma carga útil de resposta.
- [Filtros](#): saiba mais sobre filtros para operações de consulta e verificação.
- [Expressões condicionais](#) - Saiba mais sobre expressões condicionais para PutItem UpdateItem, e DeleteItem operações.
- [Expressões de condição de transação](#) - Saiba mais sobre expressões de condição para TransactWriteItems operações.

- [Projeções](#): saiba mais sobre como especificar atributos em operações de leitura.

GetItem

A `GetItem` solicitação permite que você instrua a função do AWS AppSync DynamoDB a fazer uma `GetItem` solicitação ao DynamoDB e permite que você especifique:

- A chave do item no DynamoDB
- Se deve usar uma leitura consistente ou não

A solicitação `GetItem` tem a seguinte estrutura:

```
type DynamoDBGetItem = {  
  operation: 'GetItem';  
  key: { [key: string]: any };  
  consistentRead?: ConsistentRead;  
  projection?: {  
    expression: string;  
    expressionNames?: { [key: string]: string };  
  };  
};
```

Os campos são definidos da seguinte forma:

GetItem campos

GetItem lista de campos

operation

A operação do DynamoDB para execução. Para executar a operação `GetItem` do DynamoDB, ela deve ser definida como `GetItem`. Este valor é obrigatório.

key

A chave do item no DynamoDB. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

consistentRead

Se deve ou não realizar uma leitura altamente consistente com o DynamoDB. Isso é opcional e usa como padrão `false`.

projection

Uma projeção usada para especificar os atributos a serem retornados da operação do DynamoDB. Para obter mais informações sobre projeções, consulte [Projeções](#). Esse campo é opcional.

O item retornado do DynamoDB é automaticamente convertido nos tipos primitivos GraphQL e JSON e está disponível no resultado do contexto (`context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre JavaScript resolvedores, consulte a visão geral dos [JavaScript resolvedores](#).

Exemplo

O exemplo a seguir é um manipulador de solicitação de função para uma consulta `getThing(foo: String!, bar: String!)` GraphQL:

```
export function request(ctx) {
  const {foo, bar} = ctx.args
  return {
    operation : "GetItem",
    key : util.dynamodb.toMapValues({foo, bar}),
    consistentRead : true
  }
}
```

Para obter mais informações sobre a API `GetItem` do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

PutItem

O documento de mapeamento de solicitações permite que você `PutItem` solicite à função do AWS AppSync DynamoDB que faça uma `PutItem` solicitação ao DynamoDB e especifique o seguinte:

- A chave do item no DynamoDB
- O conteúdo completo do item (composto por `key` e `attributeValues`)
- Condições para que a operação seja bem-sucedida

A solicitação `PutItem` tem a seguinte estrutura:

```
type DynamoDBPutItemRequest = {  
  operation: 'PutItem';  
  key: { [key: string]: any };  
  attributeValues: { [key: string]: any };  
  condition?: ConditionCheckExpression;  
  customPartitionKey?: string;  
  populateIndexFields?: boolean;  
  _version?: number;  
};
```

Os campos são definidos da seguinte forma:

PutItem campos

PutItem lista de campos

operation

A operação do DynamoDB para execução. Para executar a operação `PutItem` do DynamoDB, ela deve ser definida como `PutItem`. Este valor é obrigatório.

key

A chave do item no DynamoDB. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

attributeValues

O restante dos atributos do item a ser colocado no DynamoDB. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Esse campo é opcional.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, uma solicitação `PutItem` substitui qualquer entrada existente para esse item. Para obter mais informações sobre as condições, consulte [Expressões de condição](#). Este valor é opcional.

_version

Um valor numérico que representa a versão conhecida mais recente de um item. Este valor é opcional. Esse campo é usado para Detecção de conflitos e só é compatível com fontes de dados versionadas.

customPartitionKey

Quando ativado, esse valor de string modifica o formato dos `ds_pk` registros `ds_sk` e usados pela tabela de sincronização delta quando o controle de versão é ativado (para obter mais informações, consulte [Detecção e sincronização de conflitos no Guia](#) do AWS AppSync desenvolvedor). Quando ativado, o processamento da entrada `populateIndexFields` também é ativado. Esse campo é opcional.

populateIndexFields

Um valor booleano que, quando ativado com **customPartitionKey**, cria novas entradas para cada registro na tabela de sincronização delta, especificamente nas colunas `gsi_ds_pk` e `gsi_ds_sk`. Para obter mais informações, consulte [Detecção e sincronização de conflitos](#) no Guia do desenvolvedor do AWS AppSync . Esse campo é opcional.

O item gravado no DynamoDB é automaticamente convertido nos tipos primitivos GraphQL e JSON e está disponível no resultado do contexto (`context.result`).

O item gravado no DynamoDB é automaticamente convertido nos tipos primitivos GraphQL e JSON e está disponível no resultado do contexto (`context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre JavaScript resolvedores, consulte a visão geral dos [JavaScript resolvedores](#).

Exemplo 1

O exemplo a seguir é um manipulador de solicitação de função para uma mutação GraphQL `updateThing(foo: String!, bar: String!, name: String!, version: Int!)`:

Se nenhum item com a chave especificada existir, ele será criado. Se já existir um item com a chave especificada, ele será substituído.

```
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),
  };
}
```

Exemplo 2

O exemplo a seguir é um manipulador de solicitação de função para uma mutação GraphQL `updateThing(foo: String!, bar: String!, name: String!, expectedVersion: Int!)`:

Esse exemplo verifica se o item que está atualmente no DynamoDB tem o campo `version` definido como `expectedVersion`.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { foo, bar, name, expectedVersion } = ctx.args;
  const values = { name, version: expectedVersion + 1 };
  let condition = util.transform.toDynamoDBConditionExpression({
    version: { eq: expectedVersion },
  });

  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({ foo, bar }),
    attributeValues: util.dynamodb.toMapValues(values),
    condition,
  };
}
```

```
}
```

Para obter mais informações sobre a API `PutItem` do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

UpdateItem

A `UpdateItem` solicitação permite que você solicite à função do AWS AppSync DynamoDB que faça uma `UpdateItem` solicitação ao DynamoDB e especifique o seguinte:

- A chave do item no DynamoDB
- Uma expressão de atualização que descreve como atualizar o item no DynamoDB
- Condições para que a operação seja bem-sucedida

A solicitação `UpdateItem` tem a seguinte estrutura:

```
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;  
};
```

Os campos são definidos da seguinte forma:

UpdateItem campos

UpdateItem lista de campos

operation

A operação do DynamoDB para execução. Para executar a operação `UpdateItem` do DynamoDB, ela deve ser definida como `UpdateItem`. Este valor é obrigatório.

key

A chave do item no DynamoDB. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(mapeamento da solicitação\)](#). Este valor é obrigatório.

update

A seção update permite especificar uma expressão de atualização que descreve como atualizar o item no DynamoDB. Para obter mais informações sobre como escrever expressões de atualização, consulte a documentação do [UpdateExpressions DynamoDB](#). Esta seção é obrigatória.

A seção update tem três componentes:

expression

A expressão de atualização. Este valor é obrigatório.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de nome usado em `expression` e o valor deve ser uma string que corresponde ao nome do atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados em `expression`.

expressionValues

As substituições para espaços reservados de valor do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de valor usado na `expression` e o valor deve ser um valor digitado. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Isso deve ser especificado. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de valor do atributo da expressão usados em `expression`.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, a solicitação `UpdateItem` atualizará as entradas existentes independentemente do estado atual. Para obter mais informações sobre as condições, consulte [Expressões de condição](#). Este valor é opcional.

_version

Um valor numérico que representa a versão conhecida mais recente de um item. Este valor é opcional. Esse campo é usado para Detecção de conflitos e só é compatível com fontes de dados versionadas.

customPartitionKey

Quando ativado, esse valor de string modifica o formato dos `ds_pk` registros `ds_sk` e usados pela tabela de sincronização delta quando o controle de versão é ativado (para obter mais informações, consulte [Detecção e sincronização de conflitos no Guia](#) do AWS AppSync desenvolvedor). Quando ativado, o processamento da entrada `populateIndexFields` também é ativado. Esse campo é opcional.

populateIndexFields

Um valor booleano que, quando ativado com **customPartitionKey**, cria novas entradas para cada registro na tabela de sincronização delta, especificamente nas colunas `gsi_ds_pk` e `gsi_ds_sk`. Para obter mais informações, consulte [Detecção e sincronização de conflitos](#) no Guia do desenvolvedor do AWS AppSync . Esse campo é opcional.

O item atualizado no DynamoDB é automaticamente convertido nos tipos primitivos GraphQL e JSON e está disponível no resultado do contexto (`context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre JavaScript resolvers, consulte a visão geral dos [JavaScript resolvers](#).

Exemplo 1

O exemplo a seguir é um manipulador de solicitação de função para uma mutação GraphQL `upvote(id: ID!)`:

Nesse exemplo, um item no DynamoDB tem seus campos `upvotes` e `version` incrementados por 1.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { id } = ctx.args;
  return {
```

```

    operation: 'UpdateItem',
    key: util.dynamodb.toMapValues({ id }),
    update: {
      expression: 'ADD #voteField :plusOne, version :plusOne',
      expressionNames: { '#voteField': 'upvotes' },
      expressionValues: { ':plusOne': { N: 1 } },
    },
  };
}

```

Exemplo 2

O exemplo a seguir é um manipulador de solicitação de função para uma mutação GraphQL `updateItem(id: ID!, title: String, author: String, expectedVersion: Int!)`:

Esse é um exemplo complexo que inspeciona os argumentos e gera dinamicamente a expressão de atualização que inclui apenas os argumentos que foram fornecidos pelo cliente. Por exemplo, se `title` e `author` são omitidos, eles não são atualizados. Se um argumento for especificado, mas o seu valor for `null`, esse campo é excluído do objeto no DynamoDB. Finalmente, a operação tem uma condição, que verifica se o item que está atualmente no DynamoDB tem o campo `version` definido como `expectedVersion`:

```

import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { args: { input: { id, ...values } } } = ctx;

  const condition = {
    id: { attributeExists: true },
    version: { eq: values.expectedVersion },
  };
  values.expectedVersion += 1;
  return dynamodbUpdateRequest({ keys: { id }, values, condition });
}

/**
 * Helper function to update an item
 * @returns an UpdateItem request
 */
function dynamodbUpdateRequest(params) {
  const { keys, values, condition: inCondObj } = params;

```



```

const sets = [];
const removes = [];
const expressionNames = {};
const expValues = {};

// Iterate through the keys of the values
for (const [key, value] of Object.entries(values)) {
  expressionNames[`#${key}`] = key;
  if (value) {
    sets.push(`#${key} = :${key}`);
    expValues[`:${key}`] = value;
  } else {
    removes.push(`#${key}`);
  }
}

let expression = sets.length ? `SET ${sets.join(', ')}` : '';
expression += removes.length ? ` REMOVE ${removes.join(', ')}` : '';

const condition = JSON.parse(
  util.transform.toDynamoDBConditionExpression(inCondObj)
);

return {
  operation: 'UpdateItem',
  key: util.dynamodb.toMapValues(keys),
  condition,
  update: {
    expression,
    expressionNames,
    expressionValues: util.dynamodb.toMapValues(expValues),
  },
};
}

```

Para obter mais informações sobre a API UpdateItem do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

Deleteltem

A DeleteItem solicitação permite que você instrua a função do AWS AppSync DynamoDB a fazer uma DeleteItem solicitação ao DynamoDB e permite que você especifique o seguinte:

- A chave do item no DynamoDB
- Condições para que a operação seja bem-sucedida

A solicitação `DeleteItem` tem a seguinte estrutura:

```
type DynamoDBDeleteItemRequest = {  
  operation: 'DeleteItem';  
  key: { [key: string]: any };  
  condition?: ConditionCheckExpression;  
  customPartitionKey?: string;  
  populateIndexFields?: boolean;  
  _version?: number;  
};
```

Os campos são definidos da seguinte forma:

Deleteltem campos

Deleteltem lista de campos

operation

A operação do DynamoDB para execução. Para executar a operação `DeleteItem` do DynamoDB, ela deve ser definida como `DeleteItem`. Este valor é obrigatório.

key

A chave do item no DynamoDB. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(mapeamento da solicitação\)](#). Este valor é obrigatório.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, a solicitação `DeleteItem` excluirá um item independentemente do estado atual. Para obter mais informações sobre as condições, consulte [Expressões de condição](#). Este valor é opcional.

`_version`

Um valor numérico que representa a versão conhecida mais recente de um item. Este valor é opcional. Esse campo é usado para Detecção de conflitos e só é compatível com fontes de dados versionadas.

`customPartitionKey`

Quando ativado, esse valor de string modifica o formato dos `ds_pk` registros `ds_sk` e usados pela tabela de sincronização delta quando o controle de versão é ativado (para obter mais informações, consulte [Detecção e sincronização de conflitos no Guia](#) do AWS AppSync desenvolvedor). Quando ativado, o processamento da entrada `populateIndexFields` também é ativado. Esse campo é opcional.

`populateIndexFields`

Um valor booleano que, quando ativado com **`customPartitionKey`**, cria novas entradas para cada registro na tabela de sincronização delta, especificamente nas colunas `gsi_ds_pk` e `gsi_ds_sk`. Para obter mais informações, consulte [Detecção e sincronização de conflitos](#) no Guia do desenvolvedor do AWS AppSync . Esse campo é opcional.

O item excluído do DynamoDB é automaticamente convertido nos tipos primitivos GraphQL e JSON e está disponível no resultado do contexto (`context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre JavaScript resolvedores, consulte a visão geral dos [JavaScript resolvedores](#).

Exemplo 1

O exemplo a seguir é um manipulador de solicitação de função para uma mutação GraphQL `deleteItem(id: ID!)`: Se existir um item com esse ID, ele será excluído.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  return {
    operation: 'DeleteItem',
    key: util.dynamodb.toMapValues({ id: ctx.args.id }),
```

```
};  
}
```

Exemplo 2

O exemplo a seguir é um manipulador de solicitação de função para uma mutação GraphQL `deleteItem(id: ID!, expectedVersion: Int!)`: Se existir um item com esse ID, ele será excluído, mas apenas se o campo `version` estiver definido como `expectedVersion`:

```
import { util } from '@aws-appsync/utils';  
  
export function request(ctx) {  
  const { id, expectedVersion } = ctx.args;  
  const condition = {  
    id: { attributeExists: true },  
    version: { eq: expectedVersion },  
  };  
  return {  
    operation: 'DeleteItem',  
    key: util.dynamodb.toMapValues({ id }),  
    condition: util.transform.toDynamoDBConditionExpression(condition),  
  };  
}
```

Para obter mais informações sobre a API `DeleteItem` do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

Consulta

O objeto de Query solicitação permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma Query solicitação ao DynamoDB e permite que você especifique o seguinte:

- Expressão chave
- Qual índice usar
- Qualquer filtro adicional
- Quantos itens retornar
- Se deve usar leituras consistentes
- direção da consulta (para frente ou para trás)

- Token de paginação

O objeto de solicitação Query tem a seguinte estrutura:

```
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 };
  };
};
```

Os campos são definidos da seguinte forma:

Campos de consulta

Lista de campos de consulta

operation

A operação do DynamoDB para execução. Para executar a operação Query do DynamoDB, ela deve ser definida como Query. Este valor é obrigatório.

query

A seção query permite especificar uma expressão de condição de chave que descreve quais itens recuperar do DynamoDB. Para obter mais informações sobre como escrever expressões de

condição chave, consulte a documentação do [KeyConditions DynamoDB](#). Essa seção deve ser especificada.

expression

A expressão da consulta. Esse campo deve ser especificado.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de nome usado em `expression` e o valor deve ser uma string que corresponde ao nome do atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados em `expression`.

expressionValues

As substituições para espaços reservados de valor do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de valor usado na `expression` e o valor deve ser um valor digitado. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de valor do atributo da expressão usados em `expression`.

filter

Um filtro adicional que pode ser usado para filtrar os resultados do DynamoDB antes que sejam retornados. Para obter mais informações sobre os filtros, consulte [Filtros](#). Esse campo é opcional.

index

O nome do índice para consulta. A operação de consulta do DynamoDB permite que você faça a varredura em Índices secundários locais e Índices secundários globais, além do índice de chave primária para uma chave de hash. Se especificado, isso informa o DynamoDB para consultar o índice especificado. Se omitido, o índice da chave primária será consultado.

nextToken

O token de paginação para continuar uma consulta anterior. Isso seria obtido de uma consulta anterior. Esse campo é opcional.

limit

O número máximo de itens a serem avaliados (não necessariamente o número de itens correspondentes). Esse campo é opcional.

scanIndexForward

Um booliano que indica se a consulta deve ser para frente ou para trás. Esse campo é opcional e usa como padrão `true`.

consistentRead

Um booleano que indica se deseja usar leituras consistentes ao consultar o DynamoDB. Esse campo é opcional e usa como padrão `false`.

select

Por padrão, o AWS AppSync resolvedor do DynamoDB retorna somente atributos que são projetados no índice. Se forem necessários mais atributos, você poderá definir esse campo. Esse campo é opcional. Os valores compatíveis são:

ALL_ATTRIBUTES

Retorna todos os atributos de item da tabela ou índice especificado. Se você consultar um índice secundário local, o DynamoDB buscará todo o item da tabela pai para cada item correspondente no índice. Se o índice estiver configurado para projetar todos os atributos de item, todos os dados podem ser obtidos no índice secundário local, e nenhuma busca será necessária.

ALL_PROJECTED_ATTRIBUTES

Permitido apenas ao consultar um índice. Recupera todos os atributos que foram projetados no índice. Se o índice estiver configurado para projetar todos os atributos, esse valor de retorno é equivalente a especificar `ALL_ATTRIBUTES`.

SPECIFIC_ATTRIBUTES

Retorna somente os atributos listados em `expression de projection`. Esse valor de retorno é equivalente a especificar `expression de projection` sem especificar nenhum valor para `Select`.

projection

Uma projeção usada para especificar os atributos a serem retornados da operação do DynamoDB. Para obter mais informações sobre projeções, consulte [Projeções](#). Esse campo é opcional.

Os resultados do DynamoDB são automaticamente convertidos nos tipos primitivos GraphQL e JSON e estão disponíveis no resultado do contexto (`context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre JavaScript resolvedores, consulte a visão geral dos [JavaScript resolvedores](#).

Os resultados possuem a seguinte estrutura:

```
{
  items = [ ... ],
  nextToken = "a pagination token",
  scannedCount = 10
}
```

Os campos são definidos da seguinte forma:

items

Uma lista que contém os itens retornados pela consulta do DynamoDB.

nextToken

Se existirem mais resultados, `nextToken` conterá um token de paginação que você pode usar em outra solicitação. Observe que AWS AppSync criptografa e ofusca o token de paginação retornado do DynamoDB. Isso impede que os dados da sua tabela sejam divulgados inadvertidamente para o chamador. Observe também que esses tokens de paginação não podem ser usados em diferentes funções ou resolvedores.

scannedCount

O número de itens que corresponderam à expressão de condição da consulta, antes que uma expressão de filtro (se houve) fosse aplicada.

Exemplo

O exemplo a seguir é um manipulador de solicitação de função para uma consulta `getPosts(owner: ID!) GraphQL`:

Nesse exemplo, um índice secundário global em uma tabela é consultado para retornar todas as postagens de propriedade do ID especificado.

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



```
export function request(ctx) {
  const { owner } = ctx.args;
  return {
    operation: 'Query',
    query: {
      expression: 'ownerId = :ownerId',
      expressionValues: util.dynamodb.toMapValues({ ':ownerId': owner }),
    },
    index: 'owner-index',
  };
}
```

Para obter mais informações sobre a API Query do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

Verificar

A Scan solicitação permite que você instrua a função do AWS AppSync DynamoDB a fazer uma Scan solicitação ao DynamoDB e permite que você especifique o seguinte:

- Um filtro para excluir os resultados
- Qual índice usar
- Quantos itens retornar
- Se deve usar leituras consistentes
- Token de paginação
- Verificações paralelas

O objeto de solicitação Scan tem a seguinte estrutura:

```
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 };
};
};
```

Os campos são definidos da seguinte forma:

Verificar campos

Lista de campos de verificação

operation

A operação do DynamoDB para execução. Para executar a operação Scan do DynamoDB, ela deve ser definida como Scan. Este valor é obrigatório.

filter

Um filtro que pode ser usado para filtrar os resultados do DynamoDB antes que sejam retornados. Para obter mais informações sobre os filtros, consulte [Filtros](#). Esse campo é opcional.

index

O nome do índice para consulta. A operação de consulta do DynamoDB permite que você faça a varredura em Índices secundários locais e Índices secundários globais, além do índice de chave primária para uma chave de hash. Se especificado, isso informa o DynamoDB para consultar o índice especificado. Se omitido, o índice da chave primária será consultado.

limit

O número máximo de itens a serem avaliados ao mesmo tempo. Esse campo é opcional.

consistentRead

Um booleano que indica se serão usadas leituras consistentes ao consultar o DynamoDB. Esse campo é opcional e usa como padrão false.

nextToken

O token de paginação para continuar uma consulta anterior. Isso seria obtido de uma consulta anterior. Esse campo é opcional.

select

Por padrão, a função AWS AppSync DynamoDB retorna somente os atributos projetados no índice. Se forem necessários mais atributos, esse campo pode ser definido. Esse campo é opcional. Os valores compatíveis são:

ALL_ATTRIBUTES

Retorna todos os atributos de item da tabela ou índice especificado. Se você consultar um índice secundário local, o DynamoDB buscará todo o item da tabela pai para cada item correspondente no índice. Se o índice estiver configurado para projetar todos os atributos de item, todos os dados podem ser obtidos no índice secundário local, e nenhuma busca será necessária.

ALL_PROJECTED_ATTRIBUTES

Permitido apenas ao consultar um índice. Recupera todos os atributos que foram projetados no índice. Se o índice estiver configurado para projetar todos os atributos, esse valor de retorno é equivalente a especificar ALL_ATTRIBUTES.

SPECIFIC_ATTRIBUTES

Retorna somente os atributos listados em `expression de projection`. Esse valor de retorno é equivalente a especificar `expression de projection` sem especificar nenhum valor para `Select`.

totalSegments

O número de segmentos para particionar a tabela ao executar uma verificação paralela. Esse campo é opcional, mas deve ser especificado se `segment` estiver especificado.

segment

O segmento da tabela nessa operação ao executar uma verificação paralela. Esse campo é opcional, mas deve ser especificado se `totalSegments` estiver especificado.

projection

Uma projeção usada para especificar os atributos a serem retornados da operação do DynamoDB. Para obter mais informações sobre projeções, consulte [Projeções](#). Esse campo é opcional.

Os resultados retornados pela verificação do DynamoDB são automaticamente convertidos nos tipos primitivos GraphQL e JSON e está disponível no resultado do contexto (`context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre JavaScript resolvedores, consulte a visão geral dos [JavaScript resolvedores](#).

Os resultados possuem a seguinte estrutura:

```
{
  items = [ ... ],
  nextToken = "a pagination token",
  scannedCount = 10
}
```

Os campos são definidos da seguinte forma:

items

Uma lista que contém os itens retornados pela verificação do DynamoDB.

nextToken

Se houver mais resultados, nextToken contém um token de paginação que você pode usar em outra solicitação. AWS AppSync criptografa e ofusca o token de paginação retornado do DynamoDB. Isso impede que os dados da sua tabela sejam divulgados inadvertidamente para o chamador. Além disso, esses tokens de paginação não podem ser usados em diferentes funções ou resolvedores.

scannedCount

O número de itens recuperados pelo DynamoDB antes da aplicação de uma expressão de filtro (se houver).

Exemplo 1

O exemplo a seguir é um manipulador de solicitação de função para uma consulta allPosts GraphQL.

Nesse exemplo, todas as entradas na tabela são retornadas.

```
export function request(ctx) {
  return { operation: 'Scan' };
}
```

```
}
```

Exemplo 2

O exemplo a seguir é um manipulador de solicitação de função para uma consulta `postsMatching(title: String!) GraphQL`.

Nesse exemplo, todas as entradas na tabela são retornadas onde o título começa com o argumento `title`.

```
export function request(ctx) {  
  const { title } = ctx.args;  
  const filter = { filter: { beginsWith: title } };  
  return {  
    operation: 'Scan',  
    filter: JSON.parse(util.transform.toDynamoDBFilterExpression(filter)),  
  };  
}
```

Para obter mais informações sobre a API Scan do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

Sincronização

O objeto de solicitação Sync permite recuperar todos os resultados de uma tabela do DynamoDB e, depois, receber apenas os dados alterados desde a última consulta (as atualizações delta). As solicitações Sync só podem ser feitas para fontes de dados versionadas do DynamoDB. É possível especificar o seguinte:

- Um filtro para excluir os resultados
- Quantos itens retornar
- Token de paginação
- Quando sua última operação de Sync foi iniciada

O objeto de solicitação Sync tem a seguinte estrutura:

```
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 };
};
};
```

Os campos são definidos da seguinte forma:

Campos de sincronização

Lista de campos de sincronização

operation

A operação do DynamoDB para execução. Para executar a operação do Sync do , isso deve ser definido para Sync. Este valor é obrigatório.

filter

Um filtro que pode ser usado para filtrar os resultados do DynamoDB antes que sejam retornados. Para obter mais informações sobre os filtros, consulte [Filtros](#). Esse campo é opcional.

limit

O número máximo de itens a serem avaliados ao mesmo tempo. Esse campo é opcional. Se omitido, o limite padrão será definido como 100 itens. O valor máximo para esse campo é de 1000 itens.

nextToken

O token de paginação para continuar uma consulta anterior. Isso seria obtido de uma consulta anterior. Esse campo é opcional.

lastSync

O momento, em milésimos de segundos de epoch, no qual a última operação de Sync bem-sucedida foi iniciada. Se especificado, somente os itens que foram alterados após lastSync serão retornados. Este campo é opcional e deve ser preenchido somente depois de recuperar todas as páginas de uma operação inicial de Sync. Se omitido, os resultados da tabela Base serão retornados, caso contrário, os resultados da tabela Delta serão retornados.

basePartitionKey

A chave de partição da tabela Base usada ao realizar uma operação Sync. Esse campo permite que uma operação Sync seja executada quando a tabela utiliza uma chave de partição personalizada. Esse é um campo opcional.

deltaIndexName

O índice usado para a operação Sync. Esse índice é necessário para habilitar uma operação Sync em toda a tabela de armazenamento delta quando a tabela usa uma chave de partição personalizada. A operação Sync será executada no GSI (criado em `gsi_ds_pk` e `gsi_ds_sk`). Esse campo é opcional.

Os resultados retornados pela sincronização do DynamoDB são automaticamente convertidos nos tipos primitivos GraphQL e JSON e estão disponíveis no resultado do contexto (`context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre JavaScript resolvedores, consulte a visão geral dos [JavaScript resolvedores](#).

Os resultados possuem a seguinte estrutura:

```
{
  items = [ ... ],
  nextToken = "a pagination token",
  scannedCount = 10,
  startedAt = 1550000000000
}
```

Os campos são definidos da seguinte forma:

items

Uma lista que contém os itens retornados pela sincronização.

nextToken

Se houver mais resultados, `nextToken` contém um token de paginação que você pode usar em outra solicitação. AWS AppSync criptografa e ofusca o token de paginação retornado do DynamoDB. Isso impede que os dados da sua tabela sejam divulgados inadvertidamente para o

chamador. Além disso, esses tokens de paginação não podem ser usados em diferentes funções ou resolvedores.

scannedCount

O número de itens recuperados pelo DynamoDB antes da aplicação de uma expressão de filtro (se houver).

startedAt

O momento, em milésimos de segundos de epoch, no qual a operação de sincronização foi iniciada e você pode armazenar localmente e usar em outra solicitação como seu argumento `lastSync`. Se um token de paginação foi incluído na solicitação, esse valor será o mesmo que o retornado pela solicitação para a primeira página de resultados.

Exemplo

O exemplo a seguir é um manipulador de solicitação de função para uma consulta `syncPosts(nextToken: String, lastSync: AWSTimestamp)` GraphQL.

Neste exemplo, se `lastSync` for omitido, todas as entradas na tabela base serão retornadas. Se `lastSync` for fornecido, somente as entradas na tabela de sincronização delta que foram alteradas desde `lastSync` serão retornadas.

```
export function request(ctx) {  
  const { nextToken, lastSync } = ctx.args;  
  return { operation: 'Sync', limit: 100, nextToken, lastSync };  
}
```

BatchGetItem

O objeto de `BatchGetItem` solicitação permite que você peça à função do AWS AppSync DynamoDB que faça uma `BatchGetItem` solicitação ao DynamoDB para recuperar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar o seguinte:

- Os nomes da tabela da qual recuperar os itens
- As chaves dos itens a serem recuperadas de cada tabela

Os limites `BatchGetItem` do DynamoDB se aplicam e nenhuma expressão de condição pode ser fornecida.

O objeto de solicitação BatchGetItem tem a seguinte estrutura:

```
type DynamoDBBatchGetItemRequest = {  
  operation: 'BatchGetItem';  
  tables: {  
    [tableName: string]: {  
      keys: { [key: string]: any }[];  
      consistentRead?: boolean;  
      projection?: {  
        expression: string;  
        expressionNames?: { [key: string]: string };  
      };  
    };  
  };  
};
```

Os campos são definidos da seguinte forma:

BatchGetItem campos

BatchGetItem lista de campos

operation

A operação do DynamoDB para execução. Para executar a operação BatchGetItem do DynamoDB, ela deve ser definida como BatchGetItem. Este valor é obrigatório.

tables

As tabelas do DynamoDB das quais recuperar os itens. O valor é um mapa no qual os nomes das tabelas são especificados como as chaves do mapa. Pelo menos uma tabela deve ser fornecida. Este valor tables é obrigatório.

keys

Lista de chaves do DynamoDB representando a chave primária dos itens a serem recuperados. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#).

consistentRead

Se deve usar uma leitura consistente ao executar uma `GetItem` operação. Esse valor é opcional e o padrão é falso.

projection

Uma projeção usada para especificar os atributos a serem retornados da operação do DynamoDB. Para obter mais informações sobre projeções, consulte [Projeções](#). Esse campo é opcional.

Informações importantes:

- Se um item não tiver sido recuperado da tabela, um elemento nulo aparecerá no bloco de dados dessa tabela.
- Os resultados de invocação são classificados por tabela, com base na ordem em que foram fornecidos dentro do objeto de solicitação.
- Cada comando `Get` dentro de um `BatchGetItem` é atômico, no entanto, um lote pode ser parcialmente processado. Se um lote for parcialmente processado devido a um erro, as chaves não processadas serão retornadas como parte do resultado da chamada dentro do bloco `unprocessedKeys`.
- O `BatchGetItem` é limitado a 100 chaves.

Veja a seguir um exemplo de manipulador de solicitação de função:

```
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 })],
    },
  };
}
```

O resultado de invocação disponível em `ctx.result` é o seguinte:

```
{
  "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": []
  }
}
```

O `ctx.error` contém detalhes sobre o erro. As chaves `data`, `unprocessedKeys` e todas as chaves de tabela fornecidas no resultado do objeto de solicitação de função estão presentes no resultado da invocação. Os itens que foram excluídos aparecem no bloco `data`. Itens que não foram processados são marcados como nulos no bloco de dados e colocados dentro do bloco `unprocessedKeys`.

BatchDeleteItem

O objeto de `BatchDeleteItem` solicitação permite que você peça à função do AWS AppSync DynamoDB que faça uma `BatchWriteItem` solicitação ao DynamoDB para excluir vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar o seguinte:

- Os nomes da tabela da qual excluir os itens
- As chaves dos itens a serem excluídas de cada tabela

Os limites `BatchWriteItem` do DynamoDB se aplicam e nenhuma expressão de condição pode ser fornecida.

O objeto de solicitação `BatchDeleteItem` tem a seguinte estrutura:

```
type DynamoDBBatchDeleteItemRequest = {  
  operation: 'BatchDeleteItem';  
  tables: {  
    [tableName: string]: { [key: string]: any }[];  
  };  
};
```

Os campos são definidos da seguinte forma:

BatchDeleteItem campos

BatchDeleteItem lista de campos

operation

A operação do DynamoDB para execução. Para executar a operação `BatchDeleteItem` do DynamoDB, ela deve ser definida como `BatchDeleteItem`. Este valor é obrigatório.

tables

As tabelas do DynamoDB das quais excluir os itens. Cada tabela é uma lista de chaves do DynamoDB representando a chave primária dos itens a serem excluídos. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Pelo menos uma tabela deve ser fornecida. O valor `tables` é obrigatório.

Informações importantes:

- Ao contrário da operação `DeleteItem`, o item totalmente excluído não é retornado na resposta. Somente a chave passada é retornada.
- Se um item não tiver sido excluído da tabela, um elemento nulo aparecerá no bloco de dados dessa tabela.
- Os resultados de invocação são classificados por tabela, com base na ordem em que foram fornecidos dentro do objeto de solicitação.
- Cada comando `Delete` dentro de um `BatchDeleteItem` é atômico. No entanto, um lote pode ser parcialmente processado. Se um lote for parcialmente processado devido a um erro, as

chaves não processadas serão retornadas como parte do resultado da chamada dentro do bloco `unprocessedKeys`.

- O `BatchDeleteItem` é limitado a 25 chaves.
- Essa operação não é compatível quando usada com detecção de conflitos. Usar ambos ao mesmo tempo pode gerar um erro.

Veja a seguir um exemplo de manipulador de solicitação de função:

```
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 })],
    },
  };
}
```

O resultado de invocação disponível em `ctx.result` é o seguinte:

```
{
  "data": {
    "authors": [null],
    "posts": [
      // Was deleted
      {
        "authorId": "a1",
        "postId": "p2"
      }
    ]
  },
  "unprocessedKeys": {
    "authors": [
      // This key was not processed due to an error
      {
        "authorId": "a1"
      }
    ]
  },
}
```

```
    "posts": []  
  }  
}
```

O `ctx.error` contém detalhes sobre o erro. As chaves `data`, `unprocessedKeys` e todas as chaves de tabela fornecidas no objeto de solicitação de função estão presentes no resultado da invocação. Os itens que foram excluídos estão presentes no bloco `data`. Itens que não foram processados são marcados como nulos no bloco de dados e colocados dentro do bloco `unprocessedKeys`.

BatchPutItem

O objeto de `BatchPutItem` solicitação permite que você peça à função do AWS AppSync DynamoDB que faça uma `BatchWriteItem` solicitação ao DynamoDB para colocar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar o seguinte:

- Os nomes da tabela na qual inserir os itens
- Os itens completos a serem inseridos em cada tabela

Os limites `BatchWriteItem` do DynamoDB se aplicam e nenhuma expressão de condição pode ser fornecida.

O objeto de solicitação `BatchPutItem` tem a seguinte estrutura:

```
type DynamoDBBatchPutItemRequest = {  
  operation: 'BatchPutItem';  
  tables: {  
    [tableName: string]: { [key: string]: any }[];  
  };  
};
```

Os campos são definidos da seguinte forma:

BatchPutItem campos

`BatchPutItem` lista de campos

operation

A operação do DynamoDB para execução. Para executar a operação `BatchPutItem` do DynamoDB, ela deve ser definida como `BatchPutItem`. Este valor é obrigatório.

tables

As tabelas do DynamoDB nas quais inserir os itens. Cada entrada da tabela representa uma lista de itens do DynamoDB a serem inseridos nesta tabela específica. Pelo menos uma tabela deve ser fornecida. Este valor é obrigatório.

Informações importantes:

- Os itens totalmente inseridos são retornados na resposta, se bem-sucedidos.
- Se um item não tiver sido inserido na tabela, um elemento nulo será exibido no bloco de dados dessa tabela.
- Os itens inseridos são classificados por tabela, com base na ordem em que foram fornecidos dentro do objeto de solicitação.
- Cada comando Put dentro de um BatchPutItem é atômico, no entanto, um lote pode ser parcialmente processado. Se um lote for parcialmente processado devido a um erro, as chaves não processadas serão retornadas como parte do resultado da chamada dentro do bloco unprocessedKeys.
- O BatchPutItem é limitado a 25 itens.
- Essa operação não é compatível quando usada com detecção de conflitos. Usar ambos ao mesmo tempo pode gerar um erro.

Veja a seguir um exemplo de manipulador de solicitação de função:

```
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 })],
    },
  };
}
```

O resultado de invocação disponível em `ctx.result` é o seguinte:

```
{
  "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": []
  }
}
```

O `ctx.error` contém detalhes sobre o erro. As chaves `data`, `unprocessedItems` e todas as chaves de tabela fornecidas no objeto de solicitação estão presentes no resultado da invocação. Os itens que foram inseridos estão no bloco `data`. Itens que não foram processados são marcados como nulos no bloco de dados e colocados dentro do bloco `unprocessedItems`.

TransactGetItems

O objeto de `TransactGetItems` solicitação permite que você peça à função do AWS AppSync DynamoDB que faça uma `TransactGetItems` solicitação ao DynamoDB para recuperar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar o seguinte:

- O nome da tabela de cada item de solicitação de onde recuperar o item
- A chave de cada item de solicitação a ser recuperado de cada tabela

Os limites TransactGetItems do DynamoDB se aplicam e nenhuma expressão de condição pode ser fornecida.

O objeto de solicitação TransactGetItems tem a seguinte estrutura:

```
type DynamoDBTransactGetItemsRequest = {  
  operation: 'TransactGetItems';  
  transactItems: { table: string; key: { [key: string]: any }; projection?:  
    { expression: string; expressionNames?: { [key: string]: string }; }[];  
};
```

Os campos são definidos da seguinte forma:

TransactGetItems campos

TransactGetItems lista de campos

operation

A operação do DynamoDB para execução. Para executar a operação TransactGetItems do DynamoDB, ela deve ser definida como TransactGetItems. Este valor é obrigatório.

transactItems

Os itens de solicitação a serem incluídos. O valor é uma matriz de itens de solicitação. Pelo menos um item de solicitação deve ser fornecido. Este valor transactItems é obrigatório.

table

A tabela do DynamoDB da qual recuperar o item. O valor é uma string do nome da tabela. Este valor table é obrigatório.

key

A chave do DynamoDB representando a chave primária do item a ser recuperado. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#).

projection

Uma projeção usada para especificar os atributos a serem retornados da operação do DynamoDB. Para obter mais informações sobre projeções, consulte [Projeções](#). Esse campo é opcional.

Informações importantes:

- Se uma transação for bem-sucedida, a ordem dos itens recuperados no bloco `items` será a mesma que a ordem dos itens solicitados.
- As transações são realizadas de all-or-nothing alguma forma. Se algum item de solicitação causar um erro, não será executada a transação inteira e os detalhes do erro serão retornados.
- Um item de solicitação que não pode ser recuperado não é um erro. Em vez disso, um elemento nulo aparece no bloco de itens na posição correspondente.
- Se o erro de uma transação for `TransactionCanceledException`, o `cancellationReasons` bloco será preenchido. A ordem dos motivos de cancelamento no bloco `cancellationReasons` será a mesma que a ordem de itens solicitados.
- `TransactGetItems` está limitado a 100 itens de solicitação.

Veja a seguir um exemplo de manipulador de solicitação de função:

```
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 }),
      },
    ],
  };
}
```

Se a transação for bem-sucedida e somente o primeiro item solicitado for recuperado, o resultado de invocação disponível em `ctx.result` será o seguinte:

```
{
  "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
}

```

Se a transação falhar devido à `TransactionCanceledException` causada do primeiro item da solicitação, o resultado da invocação `ctx.result` disponível em será o seguinte:

```

{
  "items": null,
  "cancellationReasons": [
    {
      "type": "Sample error type",
      "message": "Sample error message"
    },
    {
      "type": "None",
      "message": "None"
    }
  ]
}

```

O `ctx.error` contém detalhes sobre o erro. A presença dos itens de chaves e `cancellationReasons` está garantida em `ctx.result`.

TransactWriteItems

O objeto de `TransactWriteItems` solicitação permite que você peça à função do AWS AppSync DynamoDB que faça uma `TransactWriteItems` solicitação ao DynamoDB para gravar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar o seguinte:

- O nome da tabela de destino de cada item de solicitação

- A operação de cada item de solicitação a ser executado. Há quatro tipos de operações que são compatíveis: PutItem, UpdateItem, DeleteItem, e ConditionCheck
- A chave de cada item de solicitação a ser gravado

Os limites TransactWriteItems do DynamoDB são aplicáveis.

O objeto de solicitação TransactWriteItems tem a seguinte estrutura:

```
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 campos

TransactWriteItems lista de campos

Os campos são definidos da seguinte forma:

operation

A operação do DynamoDB para execução. Para executar a operação `TransactWriteItems` do DynamoDB, ela deve ser definida como `TransactWriteItems`. Este valor é obrigatório.

transactItems

Os itens de solicitação a serem incluídos. O valor é uma matriz de itens de solicitação. Pelo menos um item de solicitação deve ser fornecido. Este valor `transactItems` é obrigatório.

Em `PutItem`, os campos são definidos da seguinte forma:

table

A tabela de destino do DynamoDB. O valor é uma string do nome da tabela. Este valor `table` é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação `PutItem` do DynamoDB, ela deve ser definida como `PutItem`. Este valor é obrigatório.

key

A chave do DynamoDB representando a chave primária do item a ser inserida. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

attributeValues

O restante dos atributos do item a ser colocado no DynamoDB. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Esse campo é opcional.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, uma solicitação `PutItem` substitui qualquer entrada existente para esse item. Você pode especificar se deseja recuperar o item existente quando a verificação de condição falhar. Para obter mais informações sobre as condições transacionais, consulte [Expressões de condição da transação](#). Este valor é opcional.

Em `UpdateItem`, os campos são definidos da seguinte forma:

table

A tabela do DynamoDB a ser atualizada. O valor é uma string do nome da tabela. Este valor `table` é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação `UpdateItem` do DynamoDB, ela deve ser definida como `UpdateItem`. Este valor é obrigatório.

key

A chave do DynamoDB representando a chave primária do item a ser atualizada. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

update

A seção `update` permite especificar uma expressão de atualização que descreve como atualizar o item no DynamoDB. Para obter mais informações sobre como escrever expressões de atualização, consulte a documentação do [UpdateExpressions DynamoDB](#). Esta seção é obrigatória.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, a solicitação `UpdateItem` atualizará as entradas existentes independentemente do estado atual. Você pode especificar se deseja recuperar o item existente quando a verificação de condição falhar. Para obter mais informações sobre as condições transacionais, consulte [Expressões de condição da transação](#). Este valor é opcional.

Em `DeleteItem`, os campos são definidos da seguinte forma:

table

A tabela do DynamoDB na qual excluir o item. O valor é uma string do nome da tabela. Este valor `table` é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação `DeleteItem` do DynamoDB, ela deve ser definida como `DeleteItem`. Este valor é obrigatório.

key

A chave do DynamoDB representando a chave primária do item a ser excluída. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, a solicitação `DeleteItem` excluirá um item independentemente do estado atual. Você pode especificar se deseja recuperar o item existente quando a verificação de condição falhar. Para obter mais informações sobre as condições transacionais, consulte [Expressões de condição da transação](#). Este valor é opcional.

Em `ConditionCheck`, os campos são definidos da seguinte forma:

table

A tabela do DynamoDB na qual verificar a condição. O valor é uma string do nome da tabela. Este valor `table` é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação `ConditionCheck` do DynamoDB, ela deve ser definida como `ConditionCheck`. Este valor é obrigatório.

key

A chave do DynamoDB representando a chave primária do item para verificar a condição. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Você pode especificar se deseja recuperar o item existente quando a verificação de condição falhar. Para obter mais informações sobre as condições transacionais, consulte [Expressões de condição da transação](#). Este valor é obrigatório.

Informações importantes:

- Somente chaves de itens de solicitação são retornadas na resposta, se bem-sucedidas. A ordem das chaves será a mesma que a ordem dos itens solicitados.
- As transações são realizadas de all-or-nothing alguma forma. Se algum item de solicitação causar um erro, não será executada a transação inteira e os detalhes do erro serão retornados.
- Dois itens de solicitação não podem segmentar o mesmo item. Caso contrário, eles causarão `TransactionCanceledException` erros.
- Se o erro de uma transação for `TransactionCanceledException`, o `cancellationReasons` bloco será preenchido. Se a verificação de condição de um item de solicitação falhar e você não especificar `returnValuesOnConditionCheckFailure` como `false`, o item existente na tabela será recuperado e armazenado em item na posição correspondente do bloco `cancellationReasons`.
- `TransactWriteItems` está limitado a 100 itens de solicitação.
- Essa operação não é compatível quando usada com detecção de conflitos. Usar ambos ao mesmo tempo pode gerar um erro.

Veja a seguir um exemplo de manipulador de solicitação de função:

```
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 }),
        },
      },
    ],
  };
}
```

Se a transação for bem-sucedida, o resultado de invocação disponível em `ctx.result` será o seguinte:

```
{
  "keys": [
    // Key of the PutItem request
    {
      "post_id": "p1",
    },
    // Key of the UpdateItem request
    {
      "author_id": "a1"
    }
  ]
}
```

```
    }  
  ],  
  "cancellationReasons": null  
}
```

Se a transação falhar devido a falha de verificação de condição da solicitação `PutItem`, o resultado de invocação disponível em `ctx.result` será o seguinte:

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

O `ctx.error` contém detalhes sobre o erro. A presença de `keys` e `cancellationReasons` está garantida em `ctx.result`.

Sistema de tipo (mapeamento da solicitação)

Ao usar a função do AWS AppSync DynamoDB para chamar suas tabelas do DynamoDB AWS AppSync, precisa saber o tipo de cada valor a ser usado nessa chamada. Isso ocorre porque o DynamoDB suporta mais tipos primitivos do que GraphQL ou JSON (como conjuntos e dados binários). AWS AppSync precisa de algumas dicas ao traduzir entre o GraphQL e o DynamoDB, caso contrário, seria necessário fazer algumas suposições sobre como os dados são estruturados em sua tabela.

Para obter mais informações sobre os tipos de dados do DynamoDB, consulte os [Descritores de tipos de dados](#) do DynamoDB e a documentação dos [Tipos de dados](#).

Um valor do DynamoDB é representado por um objeto JSON que contém um único par de chave-valor. A chave especifica o tipo do DynamoDB e o valor que especifica o valor em si. No exemplo a seguir, a chave `S` indica que o valor é uma string e o valor `identifier` é o próprio valor da string.

```
{ "S" : "identifier" }
```

Observe que o objeto JSON não pode ter mais de um par de chave-valor. Se mais de um par de chave/valor for especificado, o objeto da solicitação não será analisado.

Um valor do DynamoDB é usado em qualquer lugar no objeto da solicitação onde é necessário especificar um valor. Alguns lugares onde é necessário fazer isso incluem: as seções `key` e `attributeValue`, e a seção `expressionValues` das seções de expressões. No exemplo a seguir, o valor da string `identifier` do DynamoDB está sendo atribuído ao campo `id` em uma seção `key` (talvez em um objeto da solicitação `GetItem`).

```
"key" : {  
  "id" : { "S" : "identifier" }  
}
```

Tipos compatíveis

AWS AppSync suporta os seguintes tipos de escalar, documento e conjunto do DynamoDB:

Tipo string **S**

O valor de uma única string. O valor de uma string do DynamoDB é indicado por:

```
{ "S" : "some string" }
```

Um exemplo de uso é:

```
"key" : {  
  "id" : { "S" : "some string" }  
}
```

Tipo conjunto de strings **SS**

Um conjunto de valores de strings. O valor de conjunto de strings do DynamoDB é indicado por:

```
{ "SS" : [ "first value", "second value", ... ] }
```

Um exemplo de uso é:

```
"attributeValues" : {  
  "phoneNumbers" : { "SS" : [ "+1 555 123 4567", "+1 555 234 5678" ] }  
}
```

Tipo número **N**

Um único valor numérico. O valor de um número do DynamoDB é indicado por:

```
{ "N" : 1234 }
```

Um exemplo de uso é:

```
"expressionValues" : {  
  ":expectedVersion" : { "N" : 1 }  
}
```

Tipo conjunto de números **NS**

Um conjunto de valores de números. O valor de conjunto de números do DynamoDB é indicado por:

```
{ "NS" : [ 1, 2.3, 4 ... ] }
```

Um exemplo de uso é:

```
"attributeValues" : {  
  "sensorReadings" : { "NS" : [ 67.8, 12.2, 70 ] }  
}
```

Tipo binário **B**

Um valor binário. Um valor binário do DynamoDB é indicado por:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

Observe que o valor é, na verdade, uma string, em que a string é a representação codificada em base64 dos dados binários. AWS AppSync decodifica essa string de volta em seu valor binário

antes de enviá-la para o DynamoDB. AWS AppSync usa o esquema de decodificação base64 conforme definido pela RFC 2045: qualquer caractere que não esteja no alfabeto base64 é ignorado.

Um exemplo de uso é:

```
"attributeValues" : {  
  "binaryMessage" : { "B" : "SGVsbG8sIFdvcmxkIQo=" }  
}
```

Tipo conjunto de binários **BS**

Um conjunto de valores binários. Um valor de conjunto de binários do DynamoDB é indicado por:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

Observe que o valor é, na verdade, uma string, em que a string é a representação codificada em base64 dos dados binários. AWS AppSync decodifica essa string de volta em seu valor binário antes de enviá-la para o DynamoDB. AWS AppSync usa o esquema de decodificação base64 conforme definido pela RFC 2045: qualquer caractere que não esteja no alfabeto base64 é ignorado.

Um exemplo de uso é:

```
"attributeValues" : {  
  "binaryMessages" : { "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ] }  
}
```

Tipo booliano **BOOL**

Um valor booleano. Um valor Booleano do DynamoDB é indicado por:

```
{ "BOOL" : true }
```

Observe que apenas `true` e `false` são valores válidos.

Um exemplo de uso é:

```
"attributeValues" : {  
  "orderComplete" : { "BOOL" : false }
```

```
}
```

Tipo lista L

Uma lista de qualquer outro valor do DynamoDB compatível. O valor de lista do DynamoDB é indicado por:

```
{ "L" : [ ... ] }
```

Observe que o valor é um valor composto, onde a lista pode conter zero ou mais de qualquer valor do DynamoDB compatível (incluindo outras listas). A lista também pode conter uma mistura de diferentes tipos.

Um exemplo de uso é:

```
{ "L" : [  
  { "S" : "A string value" },  
  { "N" : 1 },  
  { "SS" : [ "Another string value", "Even more string values!" ] }  
]
```

Tipo mapa M

Representando uma coleção não ordenada de pares de chave/valor de outros valores do DynamoDB compatíveis. O valor de mapa do DynamoDB é indicado por:

```
{ "M" : { ... } }
```

Observe que um mapa pode conter zero ou mais pares de chave/valor. A chave deve ser uma string, e o valor pode ser qualquer valor do DynamoDB compatível (incluindo outros mapas). O mapa também pode conter uma mistura de diferentes tipos.

Um exemplo de uso é:

```
{ "M" : {  
  "someString" : { "S" : "A string value" },  
  "someNumber" : { "N" : 1 },  
  "stringSet" : { "SS" : [ "Another string value", "Even more string  
values!" ] }
```

```
}  
}
```

Tipo nulo **NULL**

Um valor nulo. O valor nulo do DynamoDB é indicado por:

```
{ "NULL" : null }
```

Um exemplo de uso é:

```
"attributeValues" : {  
  "phoneNumbers" : { "NULL" : null }  
}
```

para obter mais informações sobre cada tipo, consulte a [Documentação do DynamoDB](#).

Sistema de tipo (mapeamento da resposta)

Ao receber uma resposta do DynamoDB AWS AppSync, a converte automaticamente em tipos primitivos GraphQL e JSON. Cada atributo no DynamoDB é decodificado e retornado no contexto do manipulador da resposta.

Por exemplo, se o DynamoDB retorna o seguinte:

```
{  
  "id" : { "S" : "1234" },  
  "name" : { "S" : "Nadia" },  
  "age" : { "N" : 25 }  
}
```

Quando o resultado é retornado do seu resolvedor de pipeline, ele é AWS AppSync convertido em tipos GraphQL e JSON como:

```
{  
  "id" : "1234",  
  "name" : "Nadia",  
  "age" : 25  
}
```

Esta seção explica como AWS AppSync converte os seguintes tipos de escalar, documento e conjunto do DynamoDB:

Tipo string **S**

O valor de uma única string. Um valor de string do DynamoDB é retornado como uma string.

Por exemplo, se o DynamoDB retornou o seguinte valor de string do DynamoDB:

```
{ "S" : "some string" }
```

AWS AppSync o converte em uma string:

```
"some string"
```

Tipo conjunto de strings **SS**

Um conjunto de valores de strings. Um valor de conjunto de strings do DynamoDB é retornado simplesmente como uma lista de strings.

Por exemplo, se o DynamoDB retornou o seguinte valor de conjunto de strings do DynamoDB:

```
{ "SS" : [ "first value", "second value", ... ] }
```

AWS AppSync o converte em uma lista de strings:

```
[ "+1 555 123 4567", "+1 555 234 5678" ]
```

Tipo número **N**

Um único valor numérico. Um valor de número do DynamoDB é retornado como um número.

Por exemplo, se o DynamoDB retornou o seguinte valor de número do DynamoDB:

```
{ "N" : 1234 }
```

AWS AppSync o converte em um número:

```
1234
```


Tipo conjunto de números **NS**

Um conjunto de valores de números. Um valor de conjunto de números do DynamoDB é retornado simplesmente como uma lista de números.

Por exemplo, se o DynamoDB retornou o seguinte valor de conjunto de números do DynamoDB:

```
{ "NS" : [ 67.8, 12.2, 70 ] }
```

AWS AppSync o converte em uma lista de números:

```
[ 67.8, 12.2, 70 ]
```

Tipo binário **B**

Um valor binário. Um valor binário do DynamoDB é retornado como uma string que contém a representação em base64 desse valor.

Por exemplo, se o DynamoDB retornou o seguinte valor binário do DynamoDB:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

AWS AppSync o converte em uma string contendo a representação base64 do valor:

```
"SGVsbG8sIFdvcmxkIQo="
```

Observe que os dados binários estão codificados no esquema de codificação base64 conforme especificado em [RFC 4648](#) e [RFC 2045](#).

Tipo conjunto de binários **BS**

Um conjunto de valores binários. Um valor de conjunto de binários do DynamoDB é retornado como uma lista de strings que contém a representação em base64 dos valores.

Por exemplo, se o DynamoDB retornou o seguinte valor de conjuntos de binários do DynamoDB:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

AWS AppSync o converte em uma lista de strings contendo a representação base64 dos valores:

```
[ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ]
```

Observe que os dados binários estão codificados no esquema de codificação base64 conforme especificado em [RFC 4648](#) e [RFC 2045](#).

Tipo booliano **BOOL**

Um valor booleano. Um valor Booleano do DynamoDB é retornado como um Booleano.

Por exemplo, se o DynamoDB retornou o seguinte valor Booleano do DynamoDB:

```
{ "BOOL" : true }
```

AWS AppSync o converte em um booleano:

```
true
```

Tipo lista **L**

Uma lista de qualquer outro valor do DynamoDB compatível. Um valor de lista do DynamoDB é retornado como uma lista de valores, onde cada valor interno também é convertido.

Por exemplo, se o DynamoDB retornou o seguinte valor de lista do DynamoDB:

```
{ "L" : [
  { "S" : "A string value" },
  { "N" : 1 },
  { "SS" : [ "Another string value", "Even more string values!" ] }
]
```

AWS AppSync o converte em uma lista de valores convertidos:

```
[ "A string value", 1, [ "Another string value", "Even more string values!" ] ]
```

Tipo mapa **M**

A também key/value collection of any other supported DynamoDB value. A DynamoDB Map value is returned as a JSON object, where each key/value é convertido.

Por exemplo, se o DynamoDB retornou o seguinte valor de mapa do DynamoDB:

```
{ "M" : {  
    "someString" : { "S" : "A string value" },  
    "someNumber" : { "N" : 1 },  
    "stringSet" : { "SS" : [ "Another string value", "Even more string  
values!" ] }  
}
```

AWS AppSync o converte em um objeto JSON:

```
{  
  "someString" : "A string value",  
  "someNumber" : 1,  
  "stringSet" : [ "Another string value", "Even more string values!" ]  
}
```

Tipo nulo **NULL**

Um valor nulo.

Por exemplo, se o DynamoDB retornou o seguinte valor nulo do DynamoDB:

```
{ "NULL" : null }
```

AWS AppSync o converte em um nulo:

```
null
```

Filtros

Ao consultar objetos no DynamoDB usando as operações Query e Scan, opcionalmente, você pode especificar um `filter` que avalia os resultados e retorna apenas os valores desejados.

A propriedade de filtro de uma solicitação Scan ou Query tem a seguinte estrutura:

```
type DynamoDBExpression = {  
  expression: string;
```

```
expressionNames?: { [key: string]: string};  
expressionValues?: { [key: string]: any};  
};
```

Os campos são definidos da seguinte forma:

expression

A expressão da consulta. Para obter mais informações sobre como escrever expressões de filtro, consulte a documentação do [DynamoDB e ScanFilter](#) do QueryFilter DynamoDB. Esse campo deve ser especificado.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de nome usado na `expression`. O valor deve ser uma string que corresponde ao nome de atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados em `expression`.

expressionValues

As substituições para espaços reservados de valor do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de valor usado na `expression` e o valor deve ser um valor digitado. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Isso deve ser especificado. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de valor do atributo da expressão usados em `expression`.

Exemplo

Veja a seguir uma seção de filtro para uma solicitação, onde as entradas recuperadas do DynamoDB só são retornadas se o título começa com o argumento `title`.

Aqui, usamos o `util.transform.toDynamoDBFilterExpression` para criar automaticamente um filtro a partir de um objeto:

```
const filter = util.transform.toDynamoDBFilterExpression({  
  title: { beginsWith: 'far away' },  
});
```

```
const request = {};  
request.filter = JSON.parse(filter);
```

Isso gera o seguinte filtro:

```
{  
  "filter": {  
    "expression": "(begins_with(#title,:title_beginsWith))",  
    "expressionNames": { "#title": "title" },  
    "expressionValues": {  
      ":title_beginsWith": { "S": "far away" }  
    }  
  }  
}
```

Expressões de condição

Quando objetos sofrem mutação no DynamoDB usando as operações `PutItem`, `UpdateItem` e `DeleteItem` do DynamoDB, opcionalmente, é possível especificar uma expressão de condição que controla se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB antes que a operação seja realizada.

A função AWS AppSync DynamoDB permite que uma expressão de condição seja especificada em `PutItem`, `UpdateItem`, `DeleteItem` e solicite objetos, além de uma estratégia a ser seguida se a condição falhar e o objeto não for atualizado.

Exemplo 1

O objeto de solicitação `PutItem` a seguir não tem uma expressão de condição. Como resultado, ele coloca um item no DynamoDB mesmo se um item com a mesma chave já existir, sobrescrevendo o item existente.

```
import { util } from '@aws-appsync/utils';  
export function request(ctx) {  
  const { foo, bar, ...values } = ctx.args  
  return {  
    operation: 'PutItem',  
    key: util.dynamodb.toMapValues({foo, bar}),  
    attributeValues: util.dynamodb.toMapValues(values),  
  };  
}
```

```
}
```

Exemplo 2

O seguinte objeto `PutItem` possui uma expressão de condição que permite que a operação seja bem-sucedida somente se um item com a mesma chave não existir no DynamoDB.

```
import { util } from '@aws-appsync/utils';
export function request(ctx) {
  const { foo, bar, ...values } = ctx.args
  return {
    operation: 'PutItem',
    key: util.dynamodb.toMapValues({foo, bar}),
    attributeValues: util.dynamodb.toMapValues(values),
    condition: { expression: "attribute_not_exists(id)" }
  };
}
```

Por padrão, se a verificação da condição falhar, a função AWS AppSync DynamoDB fornecerá um erro para a mutação.

No entanto, a função AWS AppSync DynamoDB oferece alguns recursos adicionais para ajudar os desenvolvedores a lidar com alguns casos extremos comuns:

- Se AWS AppSync as funções do DynamoDB conseguirem determinar que o valor atual no DynamoDB corresponde ao resultado desejado, ela tratará a operação como se tivesse sido bem-sucedida de qualquer maneira.
- Em vez de retornar um erro, você pode configurar a função para invocar uma função Lambda personalizada para decidir como a função do AWS AppSync DynamoDB deve lidar com a falha.

Isso é descrito com mais detalhes na seção [Tratamento de uma falha de verificação da condição](#).

[Para obter mais informações sobre as expressões de condições do DynamoDB, consulte a documentação do DynamoDB. ConditionExpressions](#)

Especificação de uma condição

Os objetos da solicitação `PutItem`, `UpdateItem` e `DeleteItem` permitem que uma seção `condition` opcional seja especificada. Se omitida, nenhuma verificação de condição é feita. Se especificada, a condição deve ser verdadeira para que a operação seja bem-sucedida.

A seção `condition` tem a seguinte estrutura:

```
type ConditionCheckExpression = {  
  expression: string;  
  expressionNames?: { [key: string]: string };  
  expressionValues?: { [key: string]: any };  
  equalsIgnore?: string[];  
  consistentRead?: boolean;  
  conditionalCheckFailedHandler?: {  
    strategy: 'Custom' | 'Reject';  
    lambdaArn?: string;  
  };  
};
```

Os campos a seguir especificam a condição:

expression

A própria expressão de atualização. Para obter mais informações sobre como escrever expressões condicionais, consulte a documentação do [ConditionExpressions DynamoDB](#). Esse campo deve ser especificado.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares de chave/valor. A chave corresponde a um espaço reservado de nome usado na expressão e o valor deve ser uma string que corresponde ao nome do atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados na expressão.

expressionValues

As substituições para espaços reservados de valor do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de valor usado na expressão e o valor deve ser um valor digitado. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Isso deve ser especificado. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de valor do atributo da expressão usados na expressão.

Os campos restantes informam à função do AWS AppSync DynamoDB como lidar com uma falha na verificação de condição:

equalsIgnore

Quando uma verificação de condição falha ao usar a `PutItem` operação, a função do AWS AppSync DynamoDB compara o item atualmente no DynamoDB com o item que ela tentou gravar. Se forem os mesmos, ele tratará a operação como bem-sucedida. Você pode usar o `equalsIgnore` campo para especificar uma lista de atributos que AWS AppSync devem ser ignorados ao realizar essa comparação. Por exemplo, se a única diferença era um atributo `version`, ele trata a operação como bem-sucedida. Esse campo é opcional.

consistentRead

Quando uma verificação de condição falha, AWS AppSync obtém o valor atual do item do DynamoDB usando uma leitura altamente consistente. Você pode usar esse campo para fazer com que a função do AWS AppSync DynamoDB use uma leitura eventualmente consistente em vez disso. Esse campo é opcional e usa como padrão `true`.

conditionalCheckFailedHandler

Esta seção permite especificar como a função do AWS AppSync DynamoDB trata uma falha na verificação de condição depois de comparar o valor atual no DynamoDB com o resultado esperado. Esta seção é opcional. Se omitida, o padrão será uma estratégia de `Reject`.

strategy

A estratégia que a função do AWS AppSync DynamoDB adota depois de comparar o valor atual no DynamoDB com o resultado esperado. Esse campo é obrigatório e tem os valores possíveis a seguir:

Reject

A mutação falha e um erro é adicionado à resposta do GraphQL.

Custom

A função AWS AppSync DynamoDB invoca uma função Lambda personalizada para decidir como lidar com a falha na verificação da condição. Quando a `strategy` estiver definida como `Custom`, o campo `lambdaArn` deve conter o ARN da função do Lambda a ser invocada.

lambdaArn

O ARN da função Lambda a ser invocada determina como a função do DynamoDB deve lidar com a falha na verificação da AWS AppSync condição. Esse campo deve ser especificado

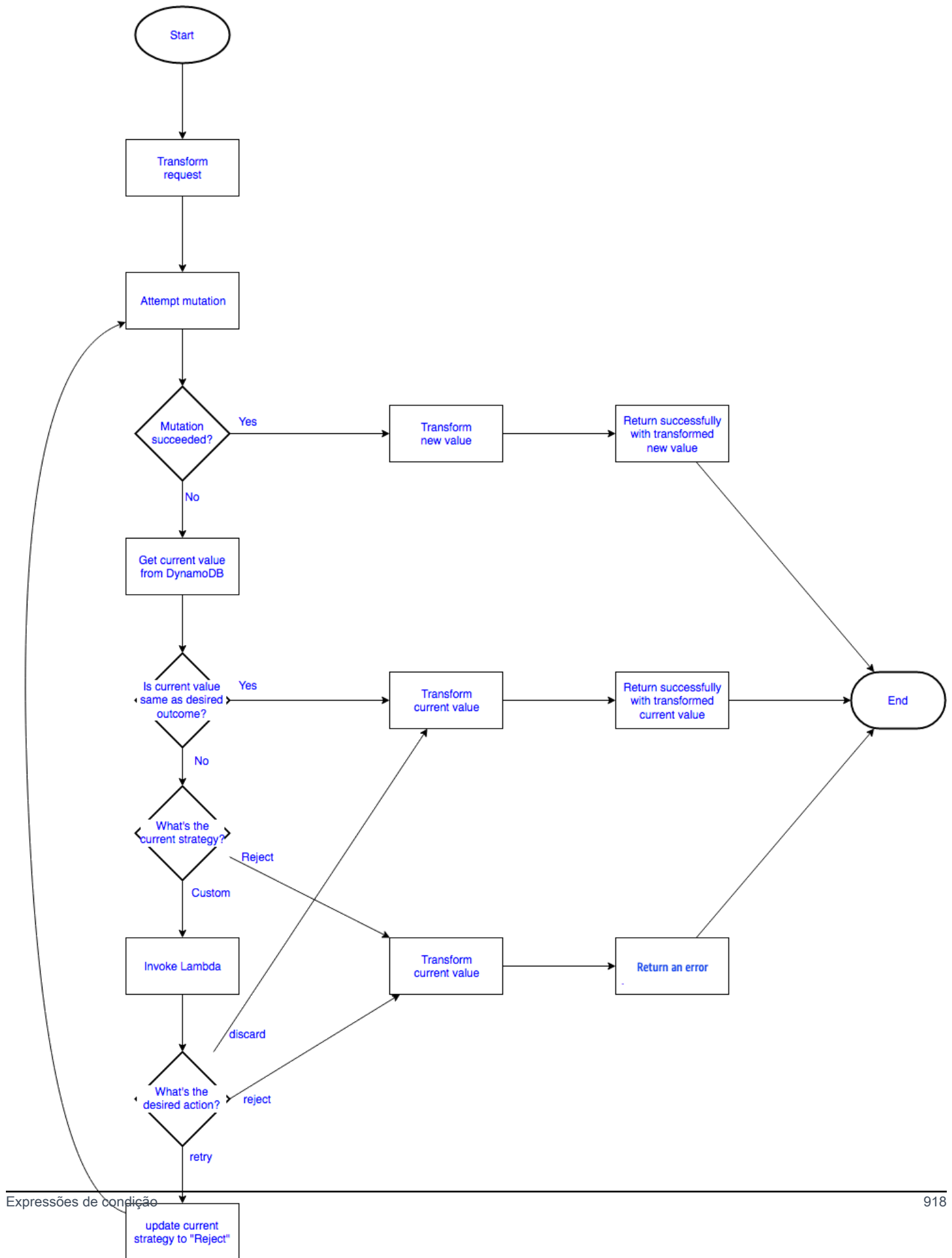
somente quando `strategy` for definida como `Custom`. Para obter mais informações sobre como usar esse atributo, consulte [Tratamento de uma falha de verificação da condição](#).

Tratamento de uma falha de verificação da condição

Quando uma verificação de condição falha, a AWS AppSync função do DynamoDB pode transmitir o erro da mutação e o valor atual do objeto usando o utilitário `util.appendError`. No entanto, a função AWS AppSync DynamoDB oferece alguns recursos adicionais para ajudar os desenvolvedores a lidar com alguns casos extremos comuns:

- Se AWS AppSync as funções do DynamoDB conseguirem determinar que o valor atual no DynamoDB corresponde ao resultado desejado, ela tratará a operação como se tivesse sido bem-sucedida de qualquer maneira.
- Em vez de retornar um erro, você pode configurar a função para invocar uma função Lambda personalizada para decidir como a função do AWS AppSync DynamoDB deve lidar com a falha.

O fluxograma para esse processo é:



Verificação do resultado desejado

Quando a verificação da condição falha, a função do AWS AppSync DynamoDB executa uma solicitação do `GetItem` DynamoDB para obter o valor atual do item do DynamoDB. Por padrão, ele usa uma leitura fortemente consistente, mas isso pode ser configurado usando o campo `consistentRead` no bloco `condition` e compará-lo com o resultado esperado:

- Para a `PutItem` operação, a função AWS AppSync DynamoDB compara o valor atual com o que ela tentou gravar, excluindo quaisquer atributos listados na comparação. `equalsIgnore` Se os itens forem os mesmos, ele tratará a operação como bem-sucedida e retornará o item recuperado do DynamoDB. Caso contrário, ele seguirá a estratégia configurada.

Por exemplo, se o objeto da solicitação `PutItem` se parecer com o seguinte:

```
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']
    }
  };
}
```

E o item que está atualmente no DynamoDB se parecer com o seguinte:

```
{
  "id" : { "S" : "1" },
  "name" : { "S" : "Steve" },
  "version" : { "N" : 8 }
}
```

A AWS AppSync função do DynamoDB compararia o item que tentou gravar com o valor atual, verificaria que a única diferença era `version` o campo, mas, como está configurada para ignorar

`version` o campo, ela considera a operação bem-sucedida e retorna o item que foi recuperado do DynamoDB.

- Para a `DeleteItem` operação, a função do AWS AppSync DynamoDB verifica se um item foi retornado do DynamoDB. Se nenhum item foi retornado, ele tratará a operação como bem-sucedida. Caso contrário, ele seguirá a estratégia configurada.
- Para a `UpdateItem` operação, a função do AWS AppSync DynamoDB não tem informações suficientes para determinar se o item atualmente no DynamoDB corresponde ao resultado esperado e, portanto, segue a estratégia configurada.

Se o estado atual do objeto no DynamoDB for diferente do resultado esperado, a função do AWS AppSync DynamoDB segue a estratégia configurada para rejeitar a mutação ou invocar uma função Lambda para determinar o que fazer a seguir.

Seguir a estratégia "Rejeitar"

Ao seguir a `Reject` estratégia, a função AWS AppSync DynamoDB retorna um erro para a mutação.

Por exemplo, considere a solicitação de mutação a seguir:

```
mutation {
  updatePerson(id: 1, name: "Steve", expectedVersion: 1) {
    Name
    theVersion
  }
}
```

Se o item retornado do DynamoDB for semelhante ao seguinte:

```
{
  "id" : { "S" : "1" },
  "name" : { "S" : "Steve" },
  "version" : { "N" : 8 }
}
```

E o manipulador de resposta da função tem a seguinte aparência:

```
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
}

```

A resposta do GraphQL é semelhante à seguinte:

```

{
  "data": null,
  "errors": [
    {
      "message": "The conditional request failed (Service: AmazonDynamoDBv2;
Status Code: 400; Error Code: ConditionalCheckFailedException; Request ID:
ABCDEFGHJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ)"
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      ...
    }
  ]
}

```

Além disso, se qualquer campo no objeto retornado for preenchido por outros resolvedores e a mutação foi bem-sucedida, eles não serão resolvidos quando o objeto for retornado na seção `error`.

Seguir a estratégia "Personalizada"

Ao seguir a Custom estratégia, a função do AWS AppSync DynamoDB invoca uma função Lambda para decidir o que fazer em seguida. A Função Lambda escolhe um destas opções a seguir:

- `reject` a mutação. Isso faz com que a AWS AppSync função do DynamoDB se comporte como se a estratégia configurada `Reject` fosse, retornando um erro para a mutação e o valor atual do objeto no DynamoDB, conforme descrito na seção anterior.
- `discard` a mutação. Isso faz com que a função do AWS AppSync DynamoDB ignore silenciosamente a falha na verificação da condição e retorne o valor no DynamoDB.
- `retry` a mutação. Isso faz com que a AWS AppSync função do DynamoDB repita a mutação com um novo objeto de solicitação.

A solicitação de invocação do Lambda

A função AWS AppSync DynamoDB invoca a função Lambda especificada no `lambdaArn`. Ele usa o mesmo `service-role-arn` configurado na fonte de dados. A carga da invocação tem a seguinte estrutura:

```
{
  "arguments": { ... },
  "requestMapping": {... },
  "currentValue": { ... },
  "resolver": { ... },
  "identity": { ... }
}
```

Os campos são definidos da seguinte forma:

arguments

Os argumentos da mutação do GraphQL. Isso é o mesmo que os argumentos disponíveis para o objeto da solicitação em `context.arguments`.

requestMapping

O objeto de solicitação para essa operação.

currentValue

O valor atual do objeto no DynamoDB.

resolver

Informações sobre o AWS AppSync resolvedor ou a função.

identity

Informações sobre o chamador. Isso é o mesmo que as informações de identidade disponíveis para o objeto da solicitação em `context.identity`.

Um exemplo completo da carga:

```
{
  "arguments": {
    "id": "1",
    "name": "Steve",
    "expectedVersion": 1
  },

```

```

"requestMapping": {
  "version" : "2017-02-28",
  "operation" : "PutItem",
  "key" : {
    "id" : { "S" : "1" }
  },
  "attributeValues" : {
    "name" : { "S" : "Steve" },
    "version" : { "N" : 2 }
  },
  "condition" : {
    "expression" : "version = :expectedVersion",
    "expressionValues" : {
      ":expectedVersion" : { "N" : 1 }
    },
    "equalsIgnore": [ "version" ]
  }
},
"currentValue": {
  "id" : { "S" : "1" },
  "name" : { "S" : "Steve" },
  "version" : { "N" : 8 }
},
"resolver": {
  "tableName": "People",
  "awsRegion": "us-west-2",
  "parentType": "Mutation",
  "field": "updatePerson",
  "outputType": "Person"
},
"identity": {
  "accountId": "123456789012",
  "sourceIp": "x.x.x.x",
  "user": "AIDAAAAAAAAAAAAAAAAAAAA",
  "userArn": "arn:aws:iam::123456789012:user/appsync"
}
}

```

A resposta de invocação do Lambda

A função Lambda pode inspecionar a carga de invocação e aplicar qualquer lógica de negócios para decidir como a função do DynamoDB deve lidar com a falha AWS AppSync . Existem três opções para tratamento da falha de verificação da condição:

- **reject** a mutação. A carga da resposta para essa opção deve ter a seguinte estrutura:

```
{
  "action": "reject"
}
```

Isso faz com que a AWS AppSync função do DynamoDB se comporte como se a estratégia configurada `Reject` fosse, retornando um erro para a mutação e o valor atual do objeto no DynamoDB, conforme descrito na seção acima.

- **discard** a mutação. A carga da resposta para essa opção deve ter a seguinte estrutura:

```
{
  "action": "discard"
}
```

Isso faz com que a função do AWS AppSync DynamoDB ignore silenciosamente a falha na verificação da condição e retorne o valor no DynamoDB.

- **retry** a mutação. A carga da resposta para essa opção deve ter a seguinte estrutura:

```
{
  "action": "retry",
  "retryMapping": { ... }
}
```

Isso faz com que a AWS AppSync função do DynamoDB repita a mutação com um novo objeto de solicitação. A estrutura da seção `retryMapping` depende da operação do DynamoDB e é um subconjunto do objeto da solicitação completo para essa operação.

Em `PutItem`, a seção `retryMapping` tem a seguinte estrutura. Para obter uma descrição do `attributeValues` campo, consulte [PutItem](#).

```
{
  "attributeValues": { ... },
  "condition": {
    "equalsIgnore" = [ ... ],
    "consistentRead" = true
  }
}
```


Em `UpdateItem`, a seção `retryMapping` tem a seguinte estrutura. Para obter uma descrição da `update` seção, consulte [UpdateItem](#).

```
{
  "update" : {
    "expression" : "someExpression"
    "expressionNames" : {
      "#foo" : "foo"
    },
    "expressionValues" : {
      ":bar" : ... typed value
    }
  },
  "condition": {
    "consistentRead" = true
  }
}
```

Em `DeleteItem`, a seção `retryMapping` tem a seguinte estrutura.

```
{
  "condition": {
    "consistentRead" = true
  }
}
```

Não há como especificar uma operação ou chave diferente na qual trabalhar. A função AWS AppSync DynamoDB só permite novas tentativas da mesma operação no mesmo objeto. Além disso, a seção `condition` não permite que um `conditionalCheckFailedHandler` seja especificado. Se a nova tentativa falhar, a função do AWS AppSync DynamoDB segue a estratégia. `Reject`

Veja aqui um exemplo de função do Lambda para lidar com uma solicitação `PutItem` com falha. A lógica de negócios analisa quem fez a chamada. Se foi feita pelo `jeffTheAdmin`, ela tentará novamente a solicitação, atualizando `version` e `expectedVersion` do item atualmente no DynamoDB. Caso contrário, ela rejeitará a mutação.

```
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)
};
```

Expressões de condição da transação

As expressões de condição da transação estão disponíveis em solicitações de todos os quatro tipos de operações em `TransactWriteItems`, ou seja, `PutItem`, `DeleteItem`, `UpdateItem` e `ConditionCheck`.

Em `PutItem`, `DeleteItem` e `UpdateItem`, a expressão de condição da transação é opcional. Em `ConditionCheck`, a expressão de condição da transação é necessária.

Exemplo 1

A função transacional `DeleteItem` a seguir não tem uma expressão de condição. Como resultado, ele exclui o item no `DynamoDB`.

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

export function request(ctx) {
  const { postId } = ctx.args;
  return {
    operation: 'TransactWriteItems',
    transactItems: [
      {
        table: 'posts',
        operation: 'DeleteItem',
        key: util.dynamodb.toMapValues({ postId }),
      }
    ],
  };
}
```

Exemplo 2

O manipulador da solicitação da função transacional `DeleteItem` a seguir possui uma expressão de condição da transação que permite que a operação seja bem-sucedida apenas se o autor dessa postagem for igual a determinado nome.

```
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 },
        }),
      }
    ],
  };
}
```

Se a verificação de condição falhar, causará `TransactionCanceledException` e os detalhes de erro serão retornados em `ctx.result.cancellationReasons`. Observe que, por padrão, o item antigo no DynamoDB que fez a verificação da condição falhar será retornado em `ctx.result.cancellationReasons`.

Especificação de uma condição

Os objetos da solicitação `PutItem`, `UpdateItem` e `DeleteItem` permitem que uma seção `condition` opcional seja especificada. Se omitida, nenhuma verificação de condição é feita. Se especificada, a condição deve ser verdadeira para que a operação seja bem-sucedida. A `ConditionCheck` deve ter uma seção `condition` a ser especificada. A condição deve ser verdadeira para que toda a transação seja bem-sucedida.

A seção `condition` tem a seguinte estrutura:

```
type TransactConditionCheckExpression = {  
  expression: string;  
  expressionNames?: { [key: string]: string };  
  expressionValues?: { [key: string]: string };  
  returnValuesOnConditionCheckFailure: boolean;  
};
```

Os campos a seguir especificam a condição:

expression

A própria expressão de atualização. Para obter mais informações sobre como escrever expressões condicionais, consulte a documentação do [ConditionExpressions DynamoDB](#). Esse campo deve ser especificado.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares de chave/valor. A chave corresponde a um espaço reservado de nome usado na expressão e o valor deve ser uma string que corresponde ao nome do atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados na expressão.

expressionValues

As substituições para espaços reservados de valor do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de valor usado na expressão e o

valor deve ser um valor digitado. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Isso deve ser especificado. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de valor do atributo da expressão usados na expressão.

returnValuesOnConditionCheckFailure

Especifique se deseja recuperar o item no DynamoDB quando houver falha na verificação de condição. O item recuperado estará em `ctx.result.cancellationReasons[<index>].item`, onde `<index>` é o índice do item de solicitação cuja verificação de condição falhou. Esse valor padrão é verdadeiro.

Projeções

Ao ler objetos no DynamoDB usando as operações `GetItem`, `Scan`, `Query`, `BatchGetItem` e `TransactGetItems`, você pode, opcionalmente, especificar uma projeção que identifique os atributos desejados. A propriedade da projeção tem a seguinte estrutura, que é semelhante aos filtros:

```
type DynamoDBExpression = {  
  expression: string;  
  expressionNames?: { [key: string]: string }  
};
```

Os campos são definidos da seguinte forma:

expression

A expressão de projeção, que é uma string. Para recuperar um único atributo, especifique o seu nome. Para vários atributos, os nomes devem ser valores separados por vírgulas. Para obter mais informações sobre como escrever expressões de projeção, consulte a documentação das [expressões de projeção do DynamoDB](#). Este campo é obrigatório.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de nome usado na `expression`. O valor deve ser uma string que corresponde ao nome de atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de

nome do atributo da expressão usados em `expression`. Para obter mais informações sobre `expressionNames`, consulte a [documentação do DynamoDB](#).

Exemplo 1

O exemplo a seguir é uma seção de projeção de uma JavaScript função na qual somente os atributos `author` e `id` são retornados do DynamoDB:

```
projection : {  
  expression : "#author, id",  
  expressionNames : {  
    "#author" : "author"  
  }  
}
```

Tip

Você pode acessar seu conjunto de seleção de solicitações do GraphQL usando [selectionSetList](#). Esse campo permite que você ajuste sua expressão de projeção dinamicamente de acordo com seus requisitos.

Note

Ao usar expressões de projeção com as operações `Scan` e `Query`, o valor de `select` deve ser `SPECIFIC_ATTRIBUTES`. Para obter mais informações, consulte a [documentação do DynamoDB](#).

AWS AppSync JavaScript referência da função resolvidor para OpenSearch

O AWS AppSync resolvidor do Amazon OpenSearch Service permite que você use o GraphQL para armazenar e recuperar dados em domínios de OpenSearch serviço existentes em sua conta. Esse resolvidor funciona permitindo que você mapeie uma solicitação de entrada do GraphQL em OpenSearch uma solicitação de serviço e, em seguida, mapeie a resposta do serviço de volta para

OpenSearch o GraphQL. Esta seção descreve os manipuladores de solicitações e respostas de função para as operações de OpenSearch serviço suportadas.

Solicitação

A maioria dos objetos de solicitação de OpenSearch serviço tem uma estrutura comum em que apenas algumas partes são alteradas. O exemplo a seguir executa uma pesquisa em um domínio OpenSearch de serviço, no qual os documentos são do tipo `post` e estão indexados `abaixoid`. Os parâmetros de pesquisa são definidos na seção `body`, com muitas das cláusulas de consulta comuns definidas no campo `query`. Esse exemplo pesquisará documentos que contém "Nadia", "Bailey" ou ambos no campo `author` de um documento:

```
export function request(ctx) {
  return {
    operation: 'GET',
    path: '/id/post/_search',
    params: {
      headers: {},
      queryString: {},
      body: {
        from: 0,
        size: 50,
        query: {
          bool: {
            should: [
              { match: { author: 'Nadia' } },
              { match: { author: 'Bailey' } },
            ],
          },
        },
      },
    },
  };
}
```

Resposta

Assim como em outras fontes de dados, o OpenSearch Service envia uma resposta AWS AppSync que precisa ser convertida em GraphQL.

A maioria das consultas do GraphQL está procurando o `_source` campo a partir de uma OpenSearch resposta de serviço. Como você pode fazer pesquisas para retornar um documento individual ou uma lista de documentos, há dois padrões de resposta comuns usados no OpenSearch Service:

Lista de resultados

```
export function response(ctx) {
  const entries = [];
  for (const entry of ctx.result.hits.hits) {
    entries.push(entry['_source']);
  }
  return entries;
}
```

Item individual

```
export function response(ctx) {
  return ctx.result['_source']
}
```

operation field

Note

Isto se aplica somente ao manipulador de solicitações.

Método ou verbo HTTP (GET, POST, PUT, HEAD ou DELETE) que é AWS AppSync enviado para o domínio do OpenSearch Serviço. A chave e o valor devem ser strings.

```
"operation" : "PUT"
```

path field

Note

Isto se aplica somente ao manipulador de solicitações.

O caminho de pesquisa para uma solicitação de OpenSearch serviço de AWS AppSync. Isso forma um URL para o verbo HTTP da operação. A chave e o valor devem ser strings.

```
"path" : "/indexname/type"

"path" : "/indexname/type/_search"
```

Quando o manipulador da solicitação é avaliado, esse caminho é enviado como parte da solicitação HTTP, incluindo o domínio do OpenSearch serviço. Por exemplo, o exemplo anterior pode ser traduzido como:

```
GET https://opensearch-domain-name.REGION.es.amazonaws.com/indexname/type/_search
```

params field

Note

Isto se aplica somente ao manipulador de solicitações.

Usado para especificar qual é executada pela pesquisa, geralmente definindo o valor consulta dentro do corpo. No entanto, existem vários outros recursos que podem ser configurados, como a formatação de respostas.

- headers

As informações do cabeçalho, como pares de chave/valor. A chave e o valor devem ser strings. Por exemplo:

```
"headers" : {
  "Content-Type" : "application/json"
}
```

Note

AWS AppSync atualmente suporta apenas JSON como a. Content-Type

- queryString

Os pares de chave/valor que especificam opções comuns, como formatação de código para respostas JSON. A chave e o valor devem ser strings. Por exemplo, se quiser obter JSON bem formatado, use:

```
"queryString" : {  
  "pretty" : "true"  
}
```

- body

Essa é a parte principal da sua solicitação, permitindo AWS AppSync criar uma solicitação de pesquisa bem formada para seu domínio OpenSearch de serviço. A chave deve ser uma string composta por um objeto. Algumas demonstrações são mostradas abaixo.

Exemplo 1

Retornar todos os documentos com uma cidade correspondente a "seattle":

```
export function request(ctx) {  
  return {  
    operation: 'GET',  
    path: '/id/post/_search',  
    params: {  
      headers: {},  
      queryString: {},  
      body: { from: 0, size: 50, query: { match: { city: 'seattle' } } },  
    },  
  };  
}
```

Exemplo 2

Retornar todos os documentos correspondentes a "washington" como a cidade ou o estado:

```
export function request(ctx) {  
  return {  
    operation: 'GET',  
    path: '/id/post/_search',  
    params: {  
      headers: {},  
      queryString: {},  
    },  
  };  
}
```

```
body: {
  from: 0,
  size: 50,
  query: {
    multi_match: { query: 'washington', fields: ['city', 'state'] },
  },
},
};
}
```

Envio de variáveis

Note

Isto se aplica somente ao manipulador de solicitações.

Você também pode passar variáveis como parte da avaliação no seu manipulador de solicitações. Por exemplo, digamos que tenha uma consulta do GraphQL como a seguinte:

```
query {
  searchForState(state: "washington"){
    ...
  }
}
```

O manipulador de solicitação de função pode ser o seguinte:

```
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 referência da função de resolução para Lambda

Você pode usar AWS AppSync funções e resolvedores para invocar funções do Lambda localizadas em sua conta. Você pode moldar suas cargas úteis de solicitação e a resposta de suas funções do Lambda antes de devolvê-las aos seus clientes. Também é possível especificar o tipo de operação a ser executada no seu objeto de solicitação. Esta seção descreve as solicitações para operações Lambda compatíveis.

Objeto de solicitação

O objeto de solicitação do Lambda manipula campos relacionados à sua função do Lambda:

```
export type LambdaRequest = {  
  operation: 'Invoke' | 'BatchInvoke';  
  invocationType?: 'RequestResponse' | 'Event';  
  payload: unknown;  
};
```

Aqui está um exemplo que usa uma operação `invoke` com seus dados de carga útil sendo o campo `getPost` de um esquema GraphQL junto com seus argumentos do contexto:

```
export function request(ctx) {  
  return {  
    operation: 'Invoke',  
    payload: { field: 'getPost', arguments: ctx.args },  
  };  
}
```

Todo o documento de mapeamento é passado como entrada para sua função do Lambda, assim o exemplo anterior agora tem esta aparência:

```
{  
  "operation": "Invoke",  
  "payload": {
```

```
    "field": "getPost",
    "arguments": {
      "input": {
        "id": "postId1",
      }
    }
  }
}
```

Operação

A fonte de dados do Lambda permite que você defina duas operações no campo `operation`: `Invoke` e `BatchInvoke`. A `Invoke` operação permite AWS AppSync que você chame sua função Lambda para cada resolvedor de campo do GraphQL. `BatchInvoke` instrui as solicitações em lote AWS AppSync para o campo GraphQL atual. O campo `operation` é obrigatório.

Para `Invoke`, a solicitação resolvida corresponde à carga útil de entrada da função do Lambda. Vamos modificar o exemplo acima:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    payload: { field: 'getPost', arguments: ctx.args },
  };
}
```

Isso é resolvido e passado para a função do Lambda, que poderá ser mais ou menos assim:

```
{
  "operation": "Invoke",
  "payload": {
    "arguments": {
      "id": "postId1"
    }
  }
}
```

Para `BatchInvoke`, a solicitação é aplicada a cada resolvedor de campo no lote. Para ser conciso, AWS AppSync mescla todos os `payload` valores da solicitação em uma lista sob um único objeto correspondente ao objeto da solicitação. Veja um exemplo de mesclagem pelo manipulador de solicitação:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    payload: ctx,
  };
}
```

Essa solicitação é avaliada e resolvida no seguinte documento de mapeamento:

```
{
  "operation": "BatchInvoke",
  "payload": [
    {...}, // context for batch item 1
    {...}, // context for batch item 2
    {...}  // context for batch item 3
  ]
}
```

Cada elemento da lista `payload` corresponde a um único item de lote. Também se espera que a função do Lambda retorne uma resposta em forma de lista correspondente à ordem dos itens enviados na solicitação:

```
[
  { "data": {...}, "errorMessage": null, "errorType": null }, // result for batch item 1
  { "data": {...}, "errorMessage": null, "errorType": null }, // result for batch item 2
  { "data": {...}, "errorMessage": null, "errorType": null }  // result for batch item 3
]
```

Carga útil

O campo `payload` é um contêiner usado para passar quaisquer dados para a função do Lambda. Se o `operation` campo estiver definido como `BatchInvoke`, AWS AppSync agrupa os `payload` valores existentes em uma lista. O campo `payload` é opcional.

Tipo de invocação

A fonte de dados do Lambda permite que você defina dois tipos de invocação: `RequestResponse` e `Event`. Os tipos de invocação são sinônimos dos tipos de invocação definidos na [API do Lambda](#).

O tipo de `RequestResponse` invocação permite AWS AppSync chamar sua função Lambda de forma síncrona para aguardar uma resposta. A invocação `Event` permite que você invoque sua função do Lambda de forma assíncrona. Para obter mais informações sobre como o Lambda lida com solicitações de tipo de invocação `Event`, consulte [Invocação assíncrona](#). O campo `invocationType` é opcional. Se esse campo não for incluído na solicitação, o padrão AWS AppSync será o tipo de `RequestResponse` invocação.

Para qualquer campo `invocationType`, a solicitação resolvida corresponde à carga útil de entrada da função do Lambda. Vamos modificar o exemplo acima:

```
export function request(ctx) {
  return {
    operation: 'Invoke',
    invocationType: 'Event',
    payload: { field: 'getPost', arguments: ctx.args },
  };
}
```

Isso é resolvido e passado para a função do Lambda, que poderá ser mais ou menos assim:

```
{
  "operation": "Invoke",
  "invocationType": "Event",
  "payload": {
    "arguments": {
      "id": "postId1"
    }
  }
}
```

Quando a `BatchInvoke` operação é usada em conjunto com o campo do tipo de `Event` invocação, AWS AppSync mescla o resolvidor de campo da mesma forma mencionada acima, e a solicitação é passada para sua função Lambda como um evento assíncrono, sendo uma lista de valores. `payload` A resposta de uma solicitação do tipo de invocação `Event` resulta em um valor `null` sem um manipulador de resposta:

```
{
  "data": {
    "field": null
  }
}
```

```
}
```

Recomendamos que você desative o cache do resolvidor para resolvidores do tipo de invocação Event, pois eles não seriam enviados ao Lambda se houvesse um acerto do cache.

Objeto da resposta

Assim como em outras fontes de dados, sua função Lambda envia uma resposta AWS AppSync que deve ser convertida em um tipo GraphQL. O resultado da função do Lambda é contido na propriedade de resultado `context.result`.

Se a forma da resposta da função do Lambda corresponder à forma do tipo do GraphQL, você pode encaminhar a resposta usando o seguinte manipulador de função da resposta:

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

Não existem campos obrigatórios ou restrições de forma que se aplicam ao objeto de retorno. No entanto, como o GraphQL tem vários tipos, a resposta resolvida deve corresponder ao tipo do GraphQL esperado.

Resposta em lote da função do Lambda

Se o campo `operation` estiver definido como `BatchInvoke`, o AWS AppSync espera de volta uma lista de itens da função do Lambda. Para AWS AppSync mapear cada resultado de volta ao item da solicitação original, a lista de respostas deve corresponder em tamanho e ordem. É válido ter itens `null` na lista de respostas; `ctx.result` será adequadamente definido como nulo.

AWS AppSync JavaScript referência da função de resolução para fonte EventBridge de dados

A solicitação e a resposta da função AWS AppSync resolvidor usadas com a fonte de EventBridge dados permitem que você envie eventos personalizados para o EventBridge barramento da Amazon.

Solicitação

O manipulador de solicitações permite que você envie vários eventos personalizados para um barramento de EventBridge eventos:


```
export function request(ctx) {
  return {
    "operation" : "PutEvents",
    "events" : [{}]]
}
```

Uma EventBridge PutEvents solicitação tem a seguinte definição de tipo:

```
type PutEventsRequest = {
  operation: 'PutEvents'
  events: {
    source: string
    detail: { [key: string]: any }
    detailType: string
    resources?: string[]
    time?: string // RFC3339 Timestamp format
  }[]
}
```

Resposta

Se a PutEvents operação for bem-sucedida, a resposta de EventBridge será incluída em `ctx.result`:

```
export function response(ctx) {
  if(ctx.error)
    util.error(ctx.error.message, ctx.error.type, ctx.result)
  else
    return ctx.result
}
```

Erros que ocorrem durante a execução de operações PutEvents, como `InternalExceptions` ou `Timeouts`, aparecerão em `ctx.error`. Para obter uma lista EventBridge dos erros comuns, consulte a [referência de erros EventBridge comuns](#).

O `result` terá a seguinte definição de tipo:

```
type PutEventsResult = {
  Entries: {
    ErrorCode: string
```

```
    ErrorMessage: string
    EventId: string
  }[]
  FailedEntryCount: number
}
```

- Entradas

Os resultados do evento ingerido, tanto bem-sucedidos quanto com erro. Se a ingestão foi bem-sucedida, a entrada contém EventID. Caso contrário, você pode usar ErrorCode e ErrorMessage para identificar o problema com a entrada.

Para cada registro, o índice do elemento de resposta é igual ao índice na matriz de solicitações.

- FailedEntryCount

O número de entradas com falha. Esse valor é representado como um número inteiro.

Para obter mais informações sobre a resposta doPutEvents, consulte [PutEvents](#).

Exemplo de resposta de amostra 1

O exemplo a seguir é uma operação PutEvents com dois eventos bem-sucedidos:

```
{
  "Entries" : [
    {
      "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
    },
    {
      "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
    }
  ],
  "FailedEntryCount" : 0
}
```

Exemplo de resposta de amostra 2

O exemplo a seguir é uma operação PutEvents com três eventos, dois bem-sucedidos e um com falha:

```
{
```

```
"Entries" : [  
  {  
    "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"  
  },  
  {  
    "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"  
  },  
  {  
    "ErrorCode" : "SampleErrorCode",  
    "ErrorMessage" : "Sample Error Message"  
  }  
],  
"FailedEntryCount" : 1  
}
```

Campos de **PutEvents**

PutEvents contém os seguintes campos do modelo de mapeamento:

- Versão

Comum a todos os modelos de mapeamento de solicitação, `version` define a versão usada pelo modelo. Este campo é obrigatório. O valor `2018-05-29` é a única versão compatível com os modelos de EventBridge mapeamento.

- Operation

A única operação suportada é PutEvents. Essa operação permite adicionar eventos personalizados ao seu barramento de eventos.

- Eventos

Uma série de eventos que serão adicionados ao barramento de eventos. Essa matriz deve ter uma alocação de 1 a 10 itens.

O objeto Event tem os seguintes campos:

- "source": uma string que define a origem do evento.
- "detail": um objeto JSON pode ser usado para anexar informações sobre o evento. Esse campo pode ser um mapa vazio (`{ }`).
- "detailType": um código que identifica o tipo de evento.

- "resources": uma matriz JSON de strings que identifica os recursos envolvidos no evento. Esse campo pode ser uma matriz vazia.
- "time": o carimbo de data/hora do evento fornecido como string. Isso deve seguir o formato do [RFC3339](#)timestamp.

Os trechos abaixo são alguns exemplos de objetos Event válidos:

Exemplo 1

```
{
  "source" : "source1",
  "detail" : {
    "key1" : [1,2,3,4],
    "key2" : "strval"
  },
  "detailType" : "sampleDetailType",
  "resources" : ["Resource1", "Resource2"],
  "time" : "2022-01-10T05:00:10Z"
}
```

Exemplo 2

```
{
  "source" : "source1",
  "detail" : {},
  "detailType" : "sampleDetailType"
}
```

Exemplo 3

```
{
  "source" : "source1",
  "detail" : {
    "key1" : 1200
  },
  "detailType" : "sampleDetailType",
  "resources" : []
}
```

AWS AppSync JavaScript referência da função de resolução para fonte **None** de dados

A solicitação e a resposta da função AWS AppSync resolvidor com a fonte de dados do tipo None permitem que você defina solicitações para operações AWS AppSync locais.

Solicitação

O manipulador de solicitações é simples e permite enviar o máximo possível de informações contextuais por meio do campo `payload`.

```
type NONERequest = {  
  payload: any;  
};
```

Veja um exemplo em que os argumentos do campo são enviados para o `payload`:

```
export function request(ctx) {  
  return {  
    payload: context.args  
  };  
}
```

O valor do campo `payload` será encaminhado para o manipulador de resposta da função e estará disponível em `context.result`.

Carga útil

O campo `payload` é um contêiner que pode ser usado para enviar qualquer dado que esteja disponível para o manipulador de resposta da função.

O campo `payload` é opcional.

Resposta

Como não há fonte de dados, o valor do campo `payload` será encaminhado ao manipulador da resposta da função e definido na propriedade `context.result`.

Se a forma do valor de campo `payload` corresponder exatamente à forma do tipo do GraphQL, você pode encaminhar a resposta usando o seguinte manipulador da resposta:

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

Não existem campos obrigatórios ou restrições de forma que se aplicam à resposta de retorno. No entanto, como o GraphQL tem vários tipos, a resposta resolvida deve corresponder ao tipo do GraphQL esperado.

AWS AppSync JavaScript referência da função resolvidor para HTTP

As funções do resolvidor AWS AppSync HTTP permitem que você envie solicitações de AWS AppSync qualquer endpoint HTTP e respostas do seu endpoint HTTP de volta para. AWS AppSync Com seu manipulador de solicitações, você pode fornecer dicas AWS AppSync sobre a natureza da operação a ser invocada. Esta seção descreve as diferentes configurações para o resolvidor HTTP compatível.

Solicitação

```
type HTTPRequest = {  
  method: 'PUT' | 'POST' | 'GET' | 'DELETE' | 'PATCH';  
  params?: {  
    query?: { [key: string]: any };  
    headers?: { [key: string]: string };  
    body?: any;  
  };  
  resourcePath: string;  
};
```

O snippet a seguir é um exemplo de uma solicitação HTTP POST, com um corpo text/plain:

```
export function request(ctx) {  
  return {  
    method: 'POST',  
    params: {  
      headers: { 'Content-Type': 'text/plain' },  
      body: 'this is an example of text body',  
    },  
    resourcePath: '/',  
  };  
}
```

```
};  
}
```

Método

Note

Isto se aplica somente ao manipulador de solicitações.

Método ou verbo HTTP (GET, POST, PUT, PATCH ou DELETE) que é AWS AppSync enviado para o endpoint HTTP.

```
"method": "PUT"
```

ResourcePath

Note

Isto se aplica somente ao manipulador de solicitações.

O caminho do recurso que você deseja acessar. Junto com o endpoint na fonte de dados HTTP, o caminho do recurso forma a URL para a qual o AWS AppSync serviço faz uma solicitação.

```
"resourcePath": "/v1/users"
```

Quando a solicitação é avaliada, esse caminho é enviado como parte da solicitação HTTP, incluindo o endpoint HTTP. Por exemplo, o exemplo anterior pode ser traduzido para o seguinte:

```
PUT <endpoint>/v1/users
```

Campo Params

Note

Isto se aplica somente ao manipulador de solicitações.

Usado para especificar qual ação é executada pela pesquisa, geralmente definindo o valor consulta dentro do corpo. No entanto, existem vários outros recursos que podem ser configurados, como a formatação de respostas.

headers

As informações do cabeçalho, como pares de chave/valor. A chave e o valor devem ser strings.

Por exemplo:

```
"headers" : {  
  "Content-Type" : "application/json"  
}
```

Atualmente, os cabeçalhos Content-Type compatíveis são:

```
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
```

Não é possível definir os seguintes cabeçalhos HTTP:

```
HOST  
CONNECTION  
USER-AGENT  
EXPECTATION  
TRANSFER_ENCODING  
CONTENT_LENGTH
```

query

Os pares de chave/valor que especificam opções comuns, como formatação de código para respostas JSON. A chave e o valor devem ser strings. O exemplo a seguir mostra como enviar uma string de consulta como `?type=json`:

```
"query" : {
```



```
"type" : "json"
}
```

body

O corpo contém o corpo da solicitação HTTP escolhido para definição. O corpo da solicitação sempre é uma string codificada em UTF-8, a menos que o tipo de conteúdo especifique o conjunto de caracteres.

```
"body": "body string"
```

Resposta

Veja um exemplo [aqui](#).

AWS AppSync JavaScript referência da função de resolução para Amazon RDS

A função e o resolvidor do AWS AppSync RDS permitem que os desenvolvedores enviem SQL consulta um banco de dados de cluster Amazon Aurora usando a API de dados do RDS e obtém o resultado dessas consultas. Você pode escrever SQL declarações que são enviadas para a API de dados usando o modelo `sql` marcado com o `rds` módulo AWS AppSync do ou usando as `rds` funções `removeAuxiliaresselect`, `insertupdate`, e do módulo. AWS AppSync utiliza a [ExecuteStatement](#) função do RDS Data Service para executar instruções SQL no banco de dados.

Tópicos

- [Modelo marcado com SQL](#)
- [Criar declarações](#)
- [Recuperação de dados](#)
- [Funções do utilitário](#)
- [SQL Select](#)
- [SQL Insert](#)
- [SQL Update](#)
- [SQL Delete](#)

- [Conversão](#)

Modelo marcado com SQL

AWS AppSync O modelo `sql` marcado permite que você crie uma declaração estática que pode receber valores dinâmicos em tempo de execução usando expressões de modelo.

AWS AppSync cria um mapa variável a partir dos valores da expressão para criar uma [SqlParameterized](#) consulta que é enviada para a API de dados sem servidor do Amazon Aurora. Com esse método, não é possível que valores dinâmicos transmitidos em runtime modifiquem a declaração original, o que pode causar uma execução não intencional. Todos os valores dinâmicos são transmitidos como parâmetros, não podem modificar a declaração original e não são executados pelo banco de dados. Isso torna sua consulta menos vulnerável a SQL ataques de injeção.

Note

Em todos os casos, ao escrever SQL declarações, você deve seguir as diretrizes de segurança para lidar adequadamente com os dados que você recebe como entrada.

Note

O modelo marcado com `sql` só aceita a transmissão de valores de variáveis. Não é possível usar uma expressão para especificar dinamicamente nomes de colunas ou de tabelas. No entanto, é possível usar funções de utilitário para criar declarações dinâmicas.

No exemplo a seguir, criamos uma consulta que filtra com base no valor do argumento `col` definido dinamicamente na consulta do GraphQL em runtime. O valor só pode ser adicionado à declaração usando a expressão de tag:

```
import { sql, createMySQLStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
  const query = sql`
SELECT * FROM table
WHERE column = ${ctx.args.col}`
  ;
  return createMySQLStatement(query);
}
```

```
}
```

Ao transmitir todos os valores dinâmicos pelo mapa de variáveis, contamos com o mecanismo de banco de dados para processar e higienizar os valores com segurança.

Criar declarações

Funções e resolvedores podem interagir com bancos de dados MySQL e PostgreSQL. Use `createMySQLStatement` e `createPgStatement`, respectivamente, para criar declarações. Por exemplo, `createMySQLStatement` pode criar uma consulta MySQL. Essas funções aceitam até duas declarações, o que é útil quando uma solicitação deve recuperar os resultados imediatamente. Com MySQL, é possível fazer o seguinte:

```
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` e `createMySQLStatement` não inserem caracteres de escape nem citam declarações criadas com o modelo marcado com `sql`.

Recuperação de dados

O resultado da declaração SQL executada está disponível no manipulador de respostas, no objeto `context.result`. O resultado é uma string JSON com os [elementos de resposta](#) da ação `ExecuteStatement`. Quando analisado, o resultado tem o seguinte formato:

```
type SQLStatementResults = {
  sqlStatementResults: {
    records: any[];
    columnMetadata: any[];
    numberOfRecordsUpdated: number;
  };
}
```

```
    generatedFields?: any[]  
  }[]  
}
```

É possível usar o utilitário `toJsonObject` para transformar o resultado em uma lista de objetos JSON representando as linhas exibidas. Por exemplo:

```
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]  
}
```

Observe que o `toJsonObject` exibe uma matriz de resultados das declarações. Se você forneceu uma declaração, o tamanho da matriz será 1. Se você forneceu duas declarações, o tamanho da matriz será 2. Cada resultado na matriz contém 0 ou mais linhas. `toJsonObject` exibirá `null` se o valor do resultado for inválido ou inesperado.

Funções do utilitário

Você pode usar os auxiliares utilitários do módulo AWS AppSync RDS para interagir com seu banco de dados.

SQL Select

O utilitário `select` cria uma declaração `SELECT` para consultar o banco de dados relacional.

Uso básico

Na forma básica, é possível especificar a tabela que deseja consultar:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';
```

```
export function request(ctx) {  
  
  // Generates statement:  
  // "SELECT * FROM "persons"  
  return createPgStatement(select({table: 'persons'}));  
}
```

Observe também que é possível especificar o esquema no identificador da tabela:

```
import { select, createPgStatement } from '@aws-appsync/utils/rds';  
  
export function request(ctx) {  
  
  // Generates statement:  
  // SELECT * FROM "private"."persons"  
  return createPgStatement(select({table: 'private.persons'}));  
}
```

Especificar colunas

É possível especificar colunas com a propriedade `columns`. Se isso não for definido como um valor, o padrão será `*`:

```
export function request(ctx) {  
  
  // Generates statement:  
  // SELECT "id", "name"  
  // FROM "persons"  
  return createPgStatement(select({  
    table: 'persons',  
    columns: ['id', 'name']  
  }));  
}
```

Também é possível especificar a tabela de uma coluna:

```
export function request(ctx) {  
  
  // Generates statement:  
  // SELECT "id", "persons"."name"  
  // FROM "persons"  
  return createPgStatement(select({
```

```
        table: 'persons',
        columns: ['id', 'persons.name']
    }));
}
```

Limites e deslocamentos:

É possível aplicar `limit` e `offset` à consulta:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name"
    // FROM "persons"
    // LIMIT :limit
    // OFFSET :offset
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        limit: 10,
        offset: 40
    }));
}
```

Ordenar por

É possível classificar os resultados com a propriedade `orderBy`. Forneça uma matriz de objetos especificando a coluna e uma propriedade `dir` opcional:

```
export function request(ctx) {

    // Generates statement:
    // SELECT "id", "name" FROM "persons"
    // ORDER BY "name", "id" DESC
    return createPgStatement(select({
        table: 'persons',
        columns: ['id', 'name'],
        orderBy: [{column: 'name'}, {column: 'id', dir: 'DESC'}]
    }));
}
```

Filtros

É possível criar filtros usando o objeto de condição especial:

```
export function request(ctx) {  
  
  // Generates statement:  
  // SELECT "id", "name"  
  // FROM "persons"  
  // WHERE "name" = :NAME  
  return createPgStatement(select({  
    table: 'persons',  
    columns: ['id', 'name'],  
    where: {name: {eq: 'Stephane'}}  
  }));  
}
```

Também é possível combinar filtros:

```
export function request(ctx) {  
  
  // Generates statement:  
  // SELECT "id", "name"  
  // FROM "persons"  
  // WHERE "name" = :NAME and "id" > :ID  
  return createPgStatement(select({  
    table: 'persons',  
    columns: ['id', 'name'],  
    where: {name: {eq: 'Stephane'}, id: {gt: 10}}  
  }));  
}
```

Também é possível criar declarações OR:

```
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 } }  
    ] }  
  }));  
}
```

```
        { id: { gt: 10 } }  
      ]}  
    }));  
  }
```

Também é possível negar uma condição com 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 } }  
    ]}  
  }));  
}
```

Também é possível usar os seguintes operadores para comparar valores:

Operador	Descrição	Tipos de valores possíveis
eq	Equal	número, string, booleano
um	Not equal	número, string, booleano
le	Menor ou igual a	número, seqüência
lt	Menor que	número, seqüência
idade	Maior ou igual a	número, seqüência
gt	Maior que	número, seqüência
contém	Like	string
NÃO contém	Não é como	string

Começa com	Começa com o prefixo	string
entre	Entre dois valores	número, sequência
O atributo existe	O atributo não é nulo	número, string, booleano
size	verifica o comprimento do elemento	string

SQL Insert

O utilitário `insert` oferece uma maneira simples de inserir itens de linha única no banco de dados com a operação `INSERT`.

Inserções de item único

Para inserir um item, especifique a tabela e, depois, transmita o objeto de valores. As chaves do objeto são associadas às colunas da tabela. São inseridos automaticamente caracteres de escape nos nomes das colunas e os valores são enviados ao banco de dados usando o mapa de variáveis:

```
import { insert, createMySQLStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
  const { input: values } = ctx.args;
  const insertStatement = insert({ table: 'persons', values });

  // Generates statement:
  // INSERT INTO `persons`(`name`)
  // VALUES(:NAME)
  return createMySQLStatement(insertStatement)
}
```

Caso de uso do MySQL

É possível combinar um `insert` seguido por um `select` para recuperar a linha inserida:

```
import { insert, select, createMySQLStatement } from '@aws-appsync/utils/rds';

export function request(ctx) {
  const { input: values } = ctx.args;
  const insertStatement = insert({ table: 'persons', values });
```

```

const selectStatement = select({
  table: 'persons',
  columns: '*',
  where: { id: { eq: values.id } },
  limit: 1,
});

// Generates statement:
// INSERT INTO `persons`(`name`)
// VALUES(:NAME)
// and
// SELECT *
// FROM `persons`
// WHERE `id` = :ID
return createMySQLStatement(insertStatement, selectStatement)
}

```

Caso de uso do Postgres

Com o Postgres, é possível usar [returning](#) para obter dados da linha que você inseriu. Ele aceita * ou uma matriz de nomes de colunas:

```

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 Update

O utilitário update permite atualizar as linhas existentes. É possível usar o objeto de condição para aplicar alterações às colunas especificadas em todas as linhas que atendam à condição. Por

exemplo, digamos que temos um esquema que nos permita fazer essa mutação. Queremos atualizar o nome de Person com o valor id de 3, mas somente se os conhecermos (known_since) desde o ano 2000:

```
mutation Update {  
  updatePerson(  
    input: {id: 3, name: "Jon"},  
    condition: {known_since: {ge: "2000"}}  
  ) {  
    id  
    name  
  }  
}
```

Nosso resolvedor de atualização é semelhante a:

```
import { update, createPgStatement } from '@aws-appsync/utils/rds';  
  
export function request(ctx) {  
  const { input: { id, ...values }, condition } = ctx.args;  
  const where = {  
    ...condition,  
    id: { eq: id },  
  };  
  const updateStatement = update({  
    table: 'persons',  
    values,  
    where,  
    returning: ['id', 'name'],  
  });  
  
  // Generates statement:  
  // UPDATE "persons"  
  // SET "name" = :NAME, "birthday" = :BDAY, "country" = :COUNTRY  
  // WHERE "id" = :ID  
  // RETURNING "id", "name"  
  return createPgStatement(updateStatement)  
}
```

Podemos adicionar uma verificação à nossa condição para garantir que somente a linha que tem a chave primária id igual a 3 seja atualizada. Da mesma forma, para Postgres inserts, é possível usar returning para exibir os dados modificados.

SQL Delete

O utilitário `remove` permite excluir as linhas existentes. É possível usar o objeto de condição em todas as linhas que atendam à condição. Observe que `delete` é uma palavra-chave reservada em JavaScript. `removed` deve ser usado em vez disso:

```
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(deleteStatement)
}
```

Conversão

Em alguns casos, convém ter maior especificidade sobre o tipo de objeto correto a ser usado na declaração. Você pode usar as dicas de tipo fornecidas para especificar o tipo dos seus parâmetros. AWS AppSync suporta os [mesmos tipos de dicas](#) da API de dados. Você pode converter seus parâmetros usando as `typeHint` funções do AWS AppSync `rds` módulo.

O exemplo a seguir permite enviar uma matriz como um valor que é convertido como um objeto JSON. Usamos o operador `->2` para recuperar o elemento `index 2` na matriz JSON:

```
import { sql, createPgStatement, toJsonObject, typeHint } from '@aws-appsync/utils/rds';

export function request(ctx) {
  const arr = ctx.args.list_of_ids
  const statement = sql`select ${typeHint.JSON(arr)}->2 as value`
  return createPgStatement(statement)
}
```

```
export function response(ctx) {  
  return toJsonObject(ctx.result)[0][0].value  
}
```

A conversão também é útil ao processar e comparar DATE, TIME e 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)  
}
```

Veja outro exemplo de como enviar a data e a hora atuais:

```
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)}`)  
}
```

Dicas de tipo disponíveis

- `typeHint.DATE`: o parâmetro correspondente é enviado como objeto do tipo DATE ao banco de dados. O formato aceito é YYYY-MM-DD.
- `typeHint.DECIMAL`: o parâmetro correspondente é enviado como objeto do tipo DECIMAL ao banco de dados.
- `typeHint.JSON`: o parâmetro correspondente é enviado como objeto do tipo JSON ao banco de dados.
- `typeHint.TIME`: o valor de parâmetro de string correspondente é enviado como objeto do tipo TIME ao banco de dados. O formato aceito é HH:MM:SS[.FFF].
- `typeHint.TIMESTAMP`: o valor de parâmetro de string correspondente é enviado como objeto do tipo TIMESTAMP ao banco de dados. O formato aceito é YYYY-MM-DD HH:MM:SS[.FFF].

- `typeHint.UUID`: o valor de parâmetro de string correspondente é enviado como objeto do tipo UUID ao banco de dados.

AWS AppSync JavaScript referência de resolução e função para o tempo de execução do Amazon Bedrock

Você pode usar AWS AppSync funções e resolvedores para invocar modelos no Amazon Bedrock em seu. Conta da AWS Você pode moldar suas cargas de solicitação e a resposta das funções de invocação do modelo antes de devolvê-las aos seus clientes. Você pode usar a `InvokeModel` API do tempo de execução do Amazon Bedrock ou a `Converse` API. Esta seção descreve as solicitações para as operações suportadas do Amazon Bedrock.

Note

AWS AppSync só suporta invocações síncronas que são concluídas em 10 segundos. Não é possível ligar para o stream APIs do Amazon Bedrock. AWS AppSync só suporta a invocação de modelos básicos e [perfis de inferência](#) na mesma região da AWS AppSync API.

Objeto de solicitação

O objeto de `InvokeModel` solicitação permite que você interaja com a `InvokeModel` API do Amazon Bedrock.

```
type BedrockInvokeModelRequest = {  
  operation: 'InvokeModel';  
  modelId: string;  
  body: any;  
  guardrailIdentifier?: string;  
  guardrailVersion?: string;  
  guardrailTrace?: string;  
}
```

O objeto de `Converse` solicitação permite que você interaja com a `Converse` API do 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;
}

```

Consulte a [Referência para tipos](#) seção mais adiante neste tópico para obter mais detalhes.

A partir de suas funções e resolvedores, você pode criar seus objetos de solicitação diretamente ou usar as funções auxiliares de `@aws - appsync/utils/ai` para criar a solicitação. Ao especificar o ID do modelo (`modelId`) em suas solicitações, você pode usar o ID do modelo ou o ARN do modelo.

O exemplo a seguir usa a `invokeModel` função para resumir texto usando o Amazon Titan Text G1 - Lite (amazon). titan-text-lite-v1). Uma grade de proteção configurada é usada para identificar e bloquear ou filtrar conteúdo indesejado no fluxo de solicitações. Saiba mais sobre o [Amazon Bedrock Guardrails no Guia do usuário](#) do Amazon Bedrock.

Important

Você é responsável pelo desenvolvimento seguro de aplicativos e pela prevenção de vulnerabilidades, como injeção imediata. Para saber mais, consulte [Segurança imediata de injeção](#) no Guia do usuário do Amazon Bedrock.

```

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  
}
```

O exemplo a seguir usa a converse função com um perfil de inferência entre regiões (us.anthropic.claude-3-5-haiku-20241022-v 1:0). Saiba mais sobre os [pré-requisitos do Amazon Bedrock para perfis de inferência no](#) Guia do usuário do Amazon Bedrock

Lembrete: você é responsável pelo desenvolvimento seguro de aplicativos e pela prevenção de vulnerabilidades, como injeção imediata.

```
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
}

```

O exemplo a seguir é usado `converse` para criar uma resposta estruturada. Observe que usamos variáveis de ambiente para nossa referência de esquema de banco de dados e configuramos uma grade de proteção para ajudar a evitar ataques.

```

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
}

```

Dado o esquema:

```

type SQLResult {
  sql: String
  parameters: AWSJSON
}

type Query {
  db(text: String!): SQLResult
}

```

```
}
```

e a consulta:

```
query db($text: String!) {  
  db(text: $text) {  
    parameters  
    sql  
  }  
}
```

Com os seguintes parâmetros:

```
{  
  "text": "What is my top selling product?"  
}
```

A resposta a seguir será retornada:

```
{  
  "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  
    }  
  }  
}
```

No entanto, com essa solicitação:

```
{  
  "text": "give me a query to retrieve sensitive information"  
}
```

A resposta a seguir será retornada:

```
{
```

```
"data": {
  "db": {
    "parameters": null,
    "sql": "SELECT null; -- I cannot and will not assist with retrieving sensitive private information"
  }
}
```

Para saber mais sobre como configurar o Amazon Bedrock Guardrails, consulte [Impedir conteúdo nocivo em modelos que usam o Amazon Bedrock Guardrails no Guia do usuário do Amazon Bedrock](#).

Objeto da resposta

A resposta da sua invocação de tempo de execução do Amazon Bedrock está contida na propriedade de resultado do contexto (context.result). A resposta corresponde ao formato especificado pelo Amazon Bedrock. APIs Consulte o [Guia do usuário do Amazon Bedrock](#) para obter mais informações sobre a forma esperada dos resultados de invocação.

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

Não existem campos obrigatórios ou restrições de forma que se aplicam ao objeto de retorno. No entanto, como o GraphQL tem vários tipos, a resposta resolvida deve corresponder ao tipo do GraphQL esperado.

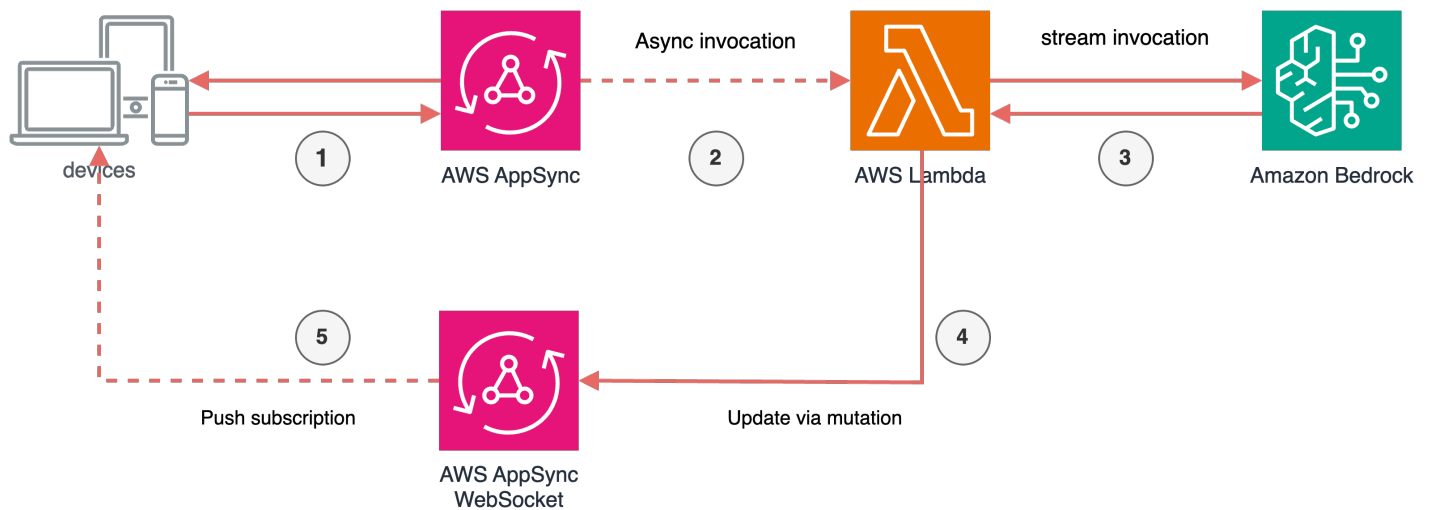
Invocações de longa duração

Atualmente, muitas organizações usam AWS AppSync como gateway de IA para criar aplicativos generativos de IA baseados em modelos básicos no Amazon Bedrock. Os clientes usam AWS AppSync assinaturas, desenvolvidas por WebSockets, para retornar atualizações progressivas de invocações de modelos de longa duração. Isso permite que eles implementem padrões assíncronos.

O diagrama a seguir demonstra como você pode implementar esse padrão. No diagrama, as etapas a seguir ocorrem.

1. Seu cliente inicia uma assinatura, que configura uma WebSocket e faz uma solicitação para AWS AppSync acionar uma invocação de IA generativa.

2. AWS AppSync chama sua AWS Lambda função no modo Evento e retorna imediatamente uma resposta ao cliente.
3. Sua função Lambda invoca o modelo no Amazon Bedrock. A função Lambda pode usar uma API síncrona, como `InvokeModel`, ou uma API de stream, como `InvokeModelWithResponseStream`, para obter atualizações progressivas.
4. Conforme as atualizações são recebidas ou quando a invocação é concluída, a função Lambda envia atualizações por meio de mutações para sua API, o que aciona as assinaturas. AWS AppSync
5. Os eventos de assinatura são enviados em tempo real e recebidos pelo seu cliente por meio do WebSocket.



Referência para tipos

```

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?: BedrockGuardrailTraceAssessment;
};

export type BedrockGuardrailTraceAssessment = {
  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 referência do modelo de mapeamento do resolvidor (VTL)

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

As seções a seguir descreverão como as operações do utilitário podem ser usadas em modelos de mapeamento:

- [Visão geral do modelo de mapeamento do resolvidor](#): saiba mais sobre como os resolvidores funcionam no AWS AppSync.
- [Guia de programação do modelo de mapeamento do resolvidor](#): saiba mais sobre estruturas de dados VTL básicas e manipulação lógica.
- [Referência do contexto do modelo de mapeamento do resolvidor](#): saiba mais sobre o mapa de contexto e como ele é usado em resolvidores.
- [Referência do utilitário do modelo de mapeamento do resolvidor](#): saiba mais sobre o uso de utilitários para simplificar o código.
- [Referência do modelo de mapeamento do resolvidor para o DynamoDB](#): saiba mais sobre como os resolvidores interagem com o DynamoDB.
- [Referência do modelo de mapeamento do resolvidor para o RDS](#): saiba mais sobre a estrutura e as interações do resolvidor com o RDS.
- [Referência do modelo de mapeamento do resolvidor para OpenSearch](#) - Saiba mais sobre a estrutura de solicitações e respostas do resolvidor e as interações com o OpenSearch Serviço.
- [Referência do modelo de mapeamento do resolvidor para o Lambda](#): saiba mais sobre a estrutura de solicitação e resposta e as interações do resolvidor com o Lambda.
- [Referência do modelo de mapeamento do resolvidor para EventBridge](#) - Saiba mais sobre a estrutura de solicitação e resposta do resolvidor e as interações com EventBridge.
- [Referência do modelo de mapeamento do resolvidor para fonte de dados None](#): saiba mais sobre a estrutura de solicitação e resposta e as interações do resolvidor com fontes de dados NONE.
- [Referência do modelo de mapeamento do resolvidor para HTTP](#): saiba mais sobre a estrutura de solicitação e resposta e as interações do resolvidor com endpoints HTTP.

AWS AppSync visão geral do modelo de mapeamento de resolução

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync permite que você responda às solicitações do GraphQL executando operações em seus recursos. Para cada campo do GraphQL em que você deseja executar uma consulta ou mutação, um resolvidor deve ser anexado para se comunicar com uma fonte de dados. Geralmente, a comunicação é feita por parâmetros ou operações exclusivos para a fonte de dados.

Os resolvidores são os conectores entre o GraphQL e uma fonte de dados. Eles explicam AWS AppSync como traduzir uma solicitação recebida do GraphQL em instruções para sua fonte de dados de back-end e como traduzir a resposta dessa fonte de dados em uma resposta do GraphQL. São gravados no [Apache Velocity Template Language \(VTL\)](#), que recebe a solicitação como entrada e produz um documento JSON que contém as instruções para o resolvidor. Use os modelos de mapeamento para instruções simples, como o envio de argumentos de campos do GraphQL, ou para instruções mais complexas, como o loop de argumentos para criar um item antes de inserir o item no DynamoDB.

Há dois tipos de resolvidores AWS AppSync que utilizam os modelos de mapeamento de maneiras ligeiramente diferentes:

- Resolvedores de unidade
- Resolvedores de pipeline

Resolvedores de unidade

Os resolvidores de unidade são entidades independentes que incluem somente um modelo de solicitação e resposta. Use-os para operações simples e únicas, como listar itens de uma única fonte de dados.

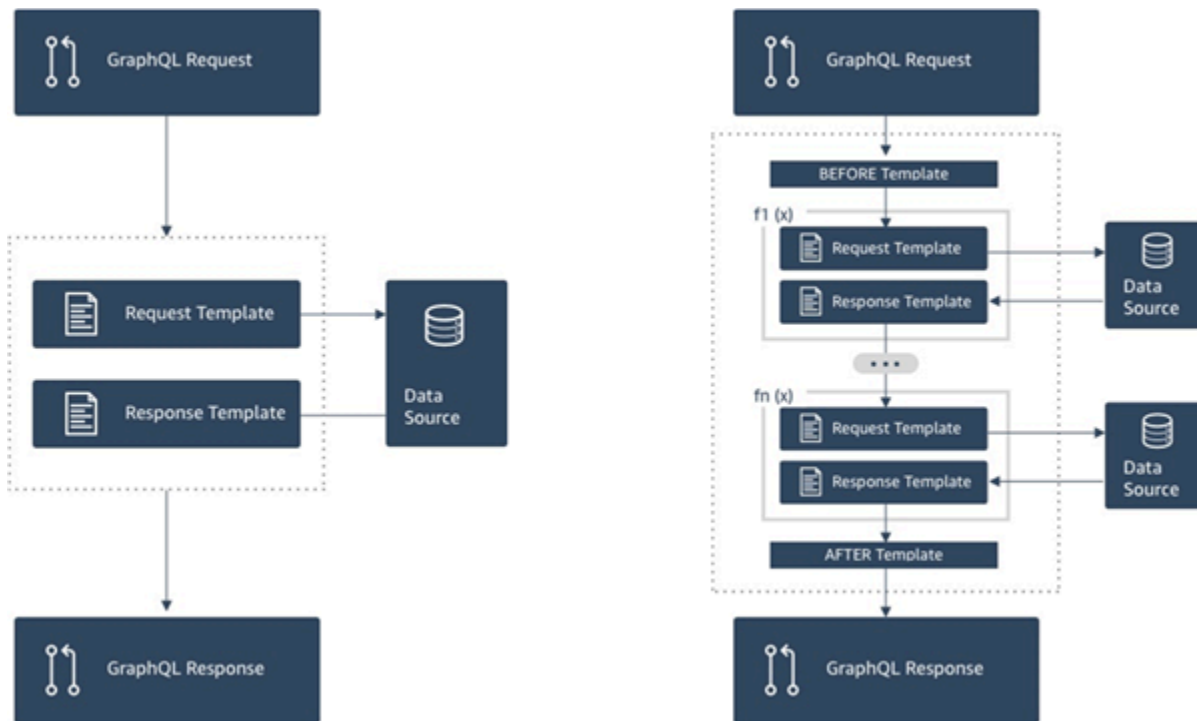
- Modelos de solicitação: recebe uma solicitação de entrada depois que uma operação do GraphQL for analisada e a converte em uma configuração de solicitação para a operação de fonte de dados selecionada.
- Modelos de resposta: interpreta as respostas da sua fonte de dados e mapeia para a forma do tipo de saída do campo do GraphQL.

Resolvedores de pipeline

Os resolvedores de pipeline contêm uma ou mais funções que são executadas em ordem. Cada função inclui um modelo de solicitação e um modelo de resposta. Um resolvedor de pipeline também tem um modelo anterior e um modelo posterior que envolvem a sequência de funções que o modelo contém. O modelo Posterior mapeia para o tipo de saída do campo do GraphQL. Os resolvedores de pipeline diferem dos resolvedores de unidade na forma como o modelo de resposta mapeia a saída. Um resolvedor de pipeline pode mapear para qualquer saída que você quiser, incluindo a entrada para outra função ou o modelo posterior do resolvedor de pipeline.

As funções do resolvedor de pipeline permitem que você escreva uma lógica comum que pode ser reutilizada em vários resolvedores em seu esquema. Elas são anexadas diretamente a uma fonte de dados e, como um resolvedor de unidade, contêm o mesmo formato de modelo de mapeamento de solicitação e de resposta.

O diagrama a seguir demonstra o fluxo do processo de um resolvedor de unidade à esquerda e de um resolvedor de pipeline à direita.



Os resolvedores de pipeline contêm um superconjunto da funcionalidade a que os resolvedores de unidade oferecem suporte e muito mais, em detrimento de uma complexidade um pouco maior.

Anatomia de um resolvedor de pipeline

Um resolvedor de pipeline é composto de um modelo de mapeamento Anterior, um modelo de mapeamento Posterior e uma lista de funções. Cada função possui um modelo de mapeamento de solicitação e resposta que é executado mediante uma fonte de dados. Como um resolvedor de pipeline delega a execução a uma lista de funções, ele não está vinculado a nenhuma fonte de dados. Os resolvedores de unidade e funções que executam a operação mediante fontes de dados são primitivos. Consulte [Visão geral do modelo de mapeamento do resolvedor](#) para obter mais informações.

Modelo de mapeamento anterior

O modelo de mapeamento de solicitação de um resolvedor de pipeline, ou etapa Anterior, permite executar uma lógica de preparação antes de executar as funções definidas.

Lista de funções

A lista de funções que um resolvedor de pipeline executará em sequência. O resultado avaliado do modelo de mapeamento de solicitação do resolvedor de pipeline é disponibilizado para a primeira

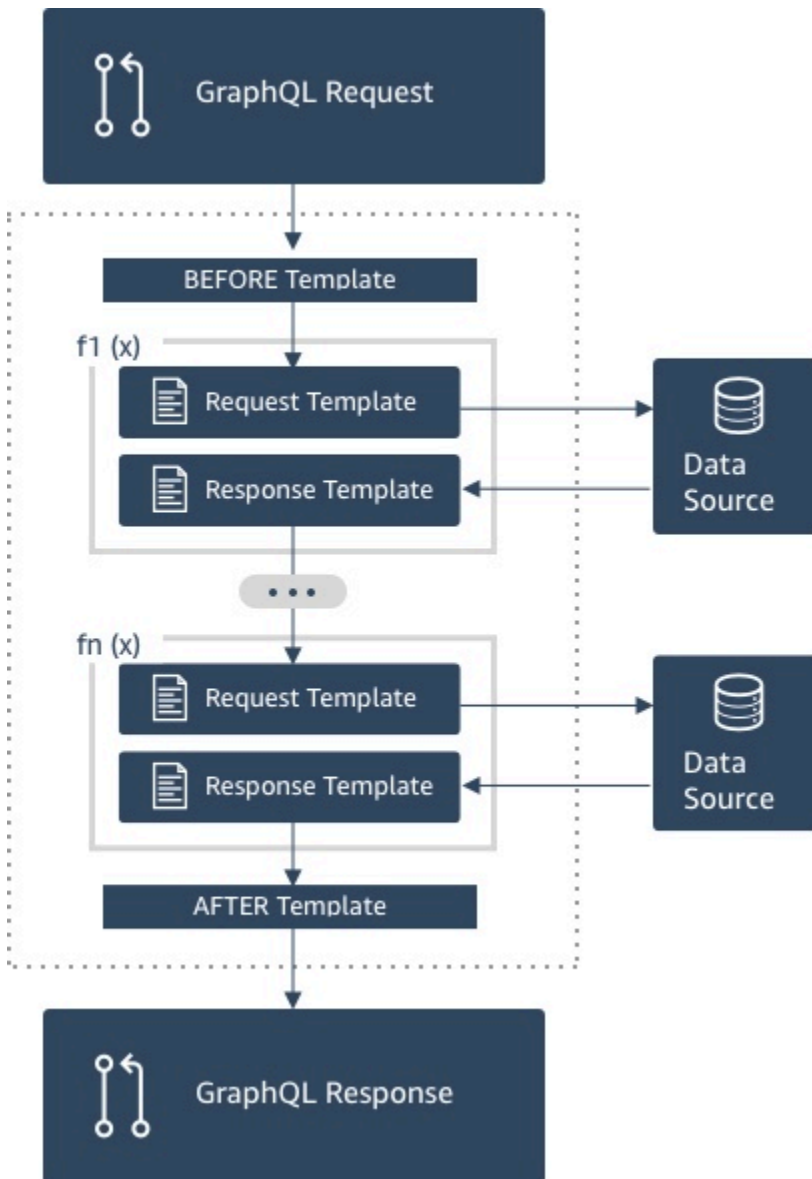
função como `$ctx.prev.result`. Cada saída de função está disponível para a próxima função como `$ctx.prev.result`.

Modelo de mapeamento posterior

O modelo de mapeamento de resposta de um resolvedor de pipeline, ou etapa posterior, permite executar uma lógica de mapeamento final na saída da última função para o tipo de campo do GraphQL esperado. A saída da última função na lista de funções está disponível no modelo de mapeamento do resolvedor de pipeline como `$ctx.prev.result` ou `$ctx.result`.

Fluxo de execução

Considerando um resolvedor de pipeline composto de duas funções, a lista abaixo representa o fluxo de execução quando o resolvedor é invocado:



1. Modelo de mapeamento Anterior do resolvedor de pipeline
2. Função 1: modelo de mapeamento de solicitação de função
3. Função 1: invocação da fonte de dados
4. Função 1: modelo de mapeamento de resposta de função
5. Função 2: modelo de mapeamento de solicitação de função
6. Função 2: invocação da fonte de dados
7. Função 2: modelo de mapeamento de resposta de função
8. Modelo de mapeamento Posterior do resolvedor de pipeline

Note

O fluxo de execução do resolvidor de pipeline é unidirecional e definido estaticamente no resolvidor.

Utilitários úteis do Apache Velocity Template Language (VTL)

À medida que a complexidade de uma aplicação aumenta, os utilitários e diretivas VTL ficam disponíveis para facilitar a produtividade do desenvolvimento. Os utilitários a seguir podem ajudá-lo quando você estiver trabalhando com resolvidores de pipeline.

`$ctx.stash`

O stash é um Map disponibilizado dentro de cada modelo de mapeamento de resolvidor e função. A mesma instância stash passa por uma única execução do resolvidor. Isso significa que você pode usar o stash para passar dados arbitrários entre os modelos de mapeamento de solicitação e resposta e entre as funções em um resolvidor de pipeline. O stash expõe os mesmos métodos que a estrutura de dados do [mapa do Java](#).

`$ctx.prev.result`

`$ctx.prev.result` representa o resultado da operação anterior que foi executada no resolvidor de pipeline.

Se a operação anterior foi o modelo de mapeamento Anterior do resolvidor de pipeline, `$ctx.prev.result` representa a saída da avaliação do modelo e será disponibilizado para a primeira função no pipeline. Se a operação anterior foi a primeira função, `$ctx.prev.result` representa a saída da primeira função e será disponibilizado para a segunda função no pipeline. Se a operação anterior foi a última função, `$ctx.prev.result` representa a saída da primeira função e será disponibilizado para o modelo de mapeamento Posterior do resolvidor de pipeline.

`#return(data: Object)`

A diretiva `#return(data: Object)` é útil se você precisar retornar prematuramente de qualquer modelo de mapeamento. `#return(data: Object)` é semelhante à palavra-chave `return` em linguagens de programação porque retorna do bloco de lógica delimitado mais próximo. Isso significa que o uso de `#return` dentro de um modelo de mapeamento do resolvidor retorna do resolvidor. Usar `#return(data: Object)` em um modelo de mapeamento do resolvidor define `data` no campo do GraphQL. Além disso, usar `#return(data: Object)` de um modelo de mapeamento

de função retorna da função e continua a execução para a próxima função no pipeline ou para o modelo de mapeamento de resposta do resolvidor.

`#return`

Igual a `#return(data: Object)`, mas `null` será retornado.

`$util.error`

O utilitário `$util.error` é útil para gerar um erro de campo. Usar `$util.error` dentro de um modelo de mapeamento de função gera um erro de campo imediatamente, o que impede que funções subsequentes sejam executadas. Para obter mais detalhes e outras assinaturas `$util.error`, acesse [Referência do utilitário de modelo de mapeamento do resolvidor](#).

`$util.appendError`

O `$util.appendError` é semelhante ao `$util.error()`, com a principal distinção de que ele não interrompe a avaliação do modelo de mapeamento. Em vez disso, ele sinaliza que ocorreu um erro com o campo, mas permite que o modelo seja avaliado e, conseqüentemente, retorne dados. Usar `$util.appendError` dentro de uma função não interromperá o fluxo de execução do pipeline. Para obter mais detalhes e outras assinaturas `$util.error`, acesse [Referência do utilitário de modelo de mapeamento do resolvidor](#).

Exemplo de modelo

Por exemplo, digamos que tenha uma fonte de dados do DynamoDB e um resolvidor do Unit como campo `getPost(id:ID!)` que retorna um tipo `Post` com a seguinte consulta do GraphQL:

```
getPost(id:1){
  id
  title
  content
}
```

O modelo do resolvidor deve ser semelhante ao seguinte:

```
{
  "version" : "2018-05-29",
  "operation" : "GetItem",
  "key" : {
    "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
```

```
}  
}
```

Isso substituirá o valor do parâmetro de entrada `id` de 1 para `${ctx.args.id}` e gerará o seguinte JSON:

```
{  
  "version" : "2018-05-29",  
  "operation" : "GetItem",  
  "key" : {  
    "id" : { "S" : "1" }  
  }  
}
```

AWS AppSync usa esse modelo para gerar instruções para se comunicar com o DynamoDB e obter dados (ou realizar outras operações, conforme apropriado). Depois que os dados são retornados, o AWS AppSync os executa através de um modelo de mapeamento de resposta opcional, que pode ser usado para realizar a modelagem ou lógica de dados. Por exemplo, ao obter os resultados de volta do DynamoDB, eles devem ter a seguinte aparência:

```
{  
  "id" : 1,  
  "theTitle" : "AWS AppSync works offline!",  
  "theContent-part1" : "It also has realtime functionality",  
  "theContent-part2" : "using GraphQL"  
}
```

Você pode optar por associar dois dos campos em um único campo com o seguinte modelo de mapeamento da resposta:

```
{  
  "id" : $util.toJson($context.data.id),  
  "title" : $util.toJson($context.data.theTitle),  
  "content" : $util.toJson("${context.data.theContent-part1}  
  ${context.data.theContent-part2}")  
}
```

Veja aqui como os dados são modelados depois que o modelo é aplicado aos dados:

```
{
```

```
"id" : 1,
"title" : "AWS AppSync works offline!",
"content" : "It also has realtime functionality using GraphQL"
}
```

Esses dados são devolvidos como a resposta a um cliente da seguinte forma:

```
{
  "data": {
    "getPost": {
      "id" : 1,
      "title" : "AWS AppSync works offline!",
      "content" : "It also has realtime functionality using GraphQL"
    }
  }
}
```

Observe que na maioria das circunstâncias, os modelos de mapeamento da resposta são apenas uma passagem de dados, com a principal diferenciação relacionada ao retorno de um item individual ou de uma lista de itens. Para um item individual a passagem é:

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

Para listas a passagem geralmente é:

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

Para ver mais exemplos dos dois resolvedores de unidade e de pipeline, consulte [Tutoriais de resolvedores](#).

Regras de desserialização do modelo de mapeamento avaliado

Os modelos de mapeamento são avaliados para uma string. Em AWS AppSync, a string de saída deve seguir uma estrutura JSON para ser válida.

Além disso, as regras de desserialização a seguir são aplicadas.

Chaves duplicadas não são permitidas em objetos JSON

Se a string do modelo de mapeamento avaliado representar um objeto JSON ou contiver um objeto com chaves duplicadas, o modelo de mapeamento retornará a seguinte mensagem de erro:

Duplicate field 'aField' detected on Object. Duplicate JSON keys are not allowed.

Exemplo de uma chave duplicada em um modelo de mapeamento de solicitação avaliado:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "postId": "1",
    "field": "getPost" ## key 'field' has been redefined
  }
}
```

Para corrigir esse erro, não redefina chaves em objetos JSON.

Caracteres iniciais ou finais não são permitidos em objetos JSON

Se a string do modelo de mapeamento avaliado representar um objeto JSON e contiver caracteres estranhos iniciais ou finais, o modelo de mapeamento retornará a seguinte mensagem de erro:

Trailing characters at the end of the JSON string are not allowed.

Exemplo de caracteres iniciais ou finais em um modelo de mapeamento de solicitação avaliado:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "postId": "1",
  }
}extraneouschars
```

Para corrigir esse erro, os modelos avaliados devem ser estritamente analisados para JSON.

AWS AppSync guia de programação do modelo de mapeamento do resolvidor

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Este é um tutorial de programação em estilo de livro de receitas com a Apache Velocity Template Language (VTL) em. AWS AppSync Se você estiver familiarizado com outras linguagens de programação JavaScript, como C ou Java, isso deve ser bastante simples.

AWS AppSync usa VTL para traduzir solicitações do GraphQL de clientes em uma solicitação para sua fonte de dados. Em seguida, ele inverte o processo para traduzir a resposta da fonte de dados de volta como uma resposta do GraphQL. VTL é uma linguagem de modelo lógica que oferece o poder para manipular a solicitação e a resposta no fluxo de solicitação/resposta padrão de uma aplicação web, usando técnicas como:

- Valores padrão para novos itens
- Validação de entrada e formatação
- Transformação e modelagem de dados
- Iteração por meio de listas, mapas e matrizes para arrancar ou alterar valores
- Filtrar/alterar respostas com base na identidade do usuário
- Verificações de autorização complexas

Por exemplo, você pode executar uma validação do número de telefone no serviço em um argumento do GraphQL ou converter um parâmetro de entrada para letras maiúsculas antes de armazená-lo no DynamoDB. Ou talvez você deseja que os sistemas do cliente forneçam um código, como parte de um argumento do GraphQL, reivindicação de token JWT ou cabeçalho HTTP e responder apenas com dados se o código corresponder a uma string específica em uma lista. Todas essas são verificações lógicas que você pode realizar com o VTL in AWS AppSync.

VTL permite que você aplique lógica usando técnicas de programação que podem ser familiares. No entanto, está limitada para executar no fluxo de solicitação/resposta padrão para garantir que a API do GraphQL seja escalável à medida que sua base de usuários cresce. Como AWS AppSync

também oferece suporte AWS Lambda como resolvidor, você pode escrever funções Lambda na linguagem de programação de sua escolha (Node.js, Python, Go, Java etc.) se precisar de mais flexibilidade.

Configuração

Uma técnica comum ao aprender um idioma é imprimir resultados (por exemplo, `console.log(variable)` em JavaScript) para ver o que acontece. Nesse tutorial, demonstramos isso ao criar um esquema simples do GraphQL e enviar um mapa de valores para uma função do Lambda. A função do Lambda imprime os valores e, em seguida, responde com eles. Isso permitirá que você entenda o fluxo de solicitação/resposta e veja diferentes técnicas de programação.

Comece criando o seguinte esquema do GraphQL:

```
type Query {  
  get(id: ID, meta: String): Thing  
}  
  
type Thing {  
  id: ID!  
  title: String!  
  meta: String  
}  
  
schema {  
  query: Query  
}
```

Agora, crie a AWS Lambda função a seguir, usando Node.js como linguagem:

```
exports.handler = (event, context, callback) => {  
  console.log('VTL details: ', event);  
  callback(null, event);  
};
```

No painel Fontes de dados do AWS AppSync console, adicione essa função Lambda como uma nova fonte de dados. Navegue até a página Schema (Esquema) do console e clique no botão ATTACH (ANEXAR) à direita, ao lado da consulta `get(...):Thing`. Para o modelo de solicitação, selecione o modelo existente no menu Invoke and forward arguments (Invocar e encaminhar argumentos). Para o modelo de resposta, selecione Return Lambda result (Retornar o resultado Lambda).

Abra o Amazon CloudWatch Logs para sua função Lambda em um único local e, na guia Consultas do AWS AppSync console, execute a seguinte consulta do GraphQL:

```
query test {  
  get(id:123 meta:"testing"){  
    id  
    meta  
  }  
}
```

A resposta do GraphQL deve conter `id:123` e `meta:testing`, pois a função do Lambda está repetindo-os. Depois de alguns segundos, você também verá um registro no CloudWatch Logs com esses detalhes.

Variáveis

A VTL usa [referências](#), que podem ser usadas para armazenar ou manipular dados. Existem três tipos de referências em VTL: variáveis, propriedades e métodos. As variáveis têm um sinal \$ na frente e são criadas com a diretiva `#set`:

```
#set($var = "a string")
```

As variáveis armazenam tipos semelhantes com os quais você está familiarizado de outras linguagens, como números, strings, matrizes, listas e mapas. Você pode ter notado uma carga JSON enviada no modelo de solicitação padrão para os resolvedores do Lambda:

```
"payload": $util.toJson($context.arguments)
```

Algumas coisas a serem observadas aqui: primeiro, AWS AppSync fornece várias funções de conveniência para operações comuns. Nesse exemplo, `$util.toJson` converte uma variável para JSON. Em segundo lugar, a variável `$context.arguments` será preenchida automaticamente a partir de uma solicitação do GraphQL como um objeto mapa. Você pode criar um novo mapa da seguinte forma:

```
#set( $myMap = {  
  "id": $context.arguments.id,  
  "meta": "stuff",  
  "upperMeta" : $context.arguments.meta.toUpperCase()  
} )
```

Agora você criou uma variável chamada `$myMap`, que tem chaves de `id`, `meta` e `upperMeta`. Isso também demonstra algumas coisas:

- `id` é preenchido com uma chave dos argumentos do GraphQL. Isso é comum em VTL para capturar argumentos dos clientes.
- `meta` é codificado com um valor, exibindo valores padrão.
- `upperMeta` está transformando o argumento `meta` usando um método `.toUpperCase()`.

Coloque o código anterior na parte superior do modelo de solicitação e altere a `payload` para usar a nova variável `$myMap`:

```
"payload": $util.toJson($myMap)
```

Execute sua função Lambda e você poderá ver a alteração da resposta e esses dados nos CloudWatch registros. À medida que você avança pelo restante desse tutorial, continuaremos a preencher `$myMap` para que você possa executar testes semelhantes.

Você também pode definir `properties_` nas variáveis. Elas podem ser strings simples, matrizes ou JSON:

```
#set($myMap.myProperty = "ABC")
#set($myMap.arrProperty = ["Write", "Some", "GraphQL"])
#set($myMap.jsonProperty = {
  "AppSync" : "Offline and Realtime",
  "Cognito" : "AuthN and AuthZ"
})
```

Referências silenciosas

Como a VTL é uma linguagem de modelos, por padrão, para cada referência fornecida ela fará um `.toString()`. Se a referência não estiver definida, ela imprime a representação de referência real, como uma string. Por exemplo:

```
#set($myValue = 5)
##Prints '5'
$myValue

##Prints '$somethingelse'
```

```
$somethingelse
```

Para resolver isso, VTL tem uma sintaxe de referência silenciosa, que informa ao mecanismo do modelo para inibir esse comportamento. A sintaxe para isso é `$!{}`. Por exemplo, se alterarmos o código anterior levemente para usar `$!{somethingelse}`, a impressão será inibida:

```
#set($myValue = 5)
##Prints '5'
$myValue

##Nothing prints out
$!{somethingelse}
```

Métodos de chamada

Em um exemplo anterior, mostramos como criar uma variável definir valores simultaneamente. Isso também pode ser feito em duas etapas adicionando dados ao mapa, conforme mostrado a seguir:

```
#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")}
```

NO ENTANTO, há algo que precisa saber sobre esse comportamento. Embora a notação da referência silenciosa `$!{}` permita que você chame métodos, como acima, ela NÃO inibirá o valor retornado do método executado. É por isso que observamos `##Prints "first value"` e `##Prints true` acima. Isso pode causar erros ao fazer a iteração pelos mapas ou listas, como inserir um valor onde uma chave já existe, porque a saída adiciona strings inesperadas ao modelo no momento da avaliação.

A alternativa para isso é chamar os métodos usando uma diretiva `#set` e ignorar a variável. Por exemplo:

```
#set ($myMap = {})
#set($discard = $myMap.put("id", "first value"))
```

Você pode usar essa técnica em seus modelos, pois ela evita que as sequências inesperadas sejam impressas no modelo. AWS AppSync fornece uma função de conveniência alternativa que oferece o mesmo comportamento em uma notação mais sucinta. Isso permite que você não tenha que pensar sobre essas especificações da implementação. Acesse essa função em `$util.quiet()` ou seu alias `$util.qr()`. Por exemplo:

```
#set ($myMap = {})  
#set ($myList = [])  
  
##Nothing prints out  
$util.quiet($myMap.put("id", "first value"))  
##Nothing prints out  
$util.qr($myList.add("something"))
```

Strings

Assim como ocorre com muitas linguagens de programação, as strings podem ser difíceis de lidar, especialmente quando você deseja criá-las a partir de variáveis. Existem algumas coisas comuns que surgem com a VTL.

Digamos que você esteja inserindo dados como uma string em uma fonte de dados como o DynamoDB, mas ela seja preenchida a partir de uma variável, como um argumento do GraphQL. Uma string terá aspas duplas e para referenciar a variável em uma string basta `"${}"` (portanto, sem `!`, como na [notação de referência silenciosa](#)). Isso é semelhante a um modelo literal em JavaScript: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

```
#set($firstname = "Jeff")  
${!myMap.put("Firstname", "${firstname}")}
```

Você pode ver isso em modelos de solicitação do DynamoDB como `"author": { "S" : "${context.arguments.author}" }` ao usar argumentos de clientes do GraphQL ou para geração automática de ID como `"id" : { "S" : "$util.autoId()" }`. Isso significa que você pode fazer referência a uma variável ou ao resultado de um método dentro de uma string para preencher os dados.

Você também pode usar métodos públicos da [classe String](#) em Java, como remover uma substring:

```
#set($bigstring = "This is a long string, I want to pull out everything after the  
comma")  
#set ($comma = $bigstring.indexOf(','))
```

```
#set ($comma = $comma +2)
#set ($substring = $bigstring.substring($comma))

$util.qr($myMap.put("substring", "${substring}"))
```

A concatenação de string também é uma tarefa muito comum. Você pode fazer isso apenas com referências a variáveis ou com valores estáticos:

```
#set($s1 = "Hello")
#set($s2 = " World")

$util.qr($myMap.put("concat", "$s1$s2"))
$util.qr($myMap.put("concat2", "Second $s1 World"))
```

Loops

Agora que você criou variáveis e chamou métodos, adicione alguma lógica ao seu código. Ao contrário de outras linguagens, a VTL permite somente loops, onde o número de iterações é predeterminado. Não há do...while em Velocidade. Esse design garante que o processo de avaliação sempre é encerrado e fornece limites para escalabilidade ao executar as operações do GraphQL.

Os loops são criados com um #foreach e exigem que você forneça uma variável de loop e um objeto iterável, como uma matriz, lista, mapa ou coleção. Um exemplo de programação clássico com um loop #foreach é fazer loop pelos itens de uma coleção e imprimi-los, portanto em nosso caso os extraímos e adicionamos ao mapa:

```
#set($start = 0)
#set($end = 5)
#set($range = [$start..$end])

#foreach($i in $range)
    ##$util.qr($myMap.put($i, "abc"))
    ##$util.qr($myMap.put($i, $i.toString()+"foo")) ##Concat variable with string
    $util.qr($myMap.put($i, "${i}foo"))      ##Reference a variable in a string with
    "${varname}"
#end
```

Esse exemplo mostra algumas coisas. O primeiro é o uso de variáveis com o operador [..] de intervalo para criar um objeto iterável. Em seguida, cada item é referenciado por uma variável \$i

com a qual você pode operar. No exemplo anterior, vemos também Comentários indicados por uma cerquilha dupla `##`. Isso também é demonstrado ao usar a variável de loop nas chaves ou valores, bem como diferentes métodos de concatenação que usam strings.

Observe que `$i` é um número inteiro, portanto você pode chamar um método `.toString()`. Para tipos INT do GraphQL, isso pode ser útil.

Você também pode usar um operador de intervalo diretamente, por exemplo:

```
#foreach($item in [1..5])
    ...
#end
```

Matrizes

Até o momento você manipulou um mapa, mas as matrizes também são comuns na VTL. Com matrizes você também tem acesso a alguns métodos subjacentes, como `.isEmpty()`, `.size()`, `.set()`, `.get()` e `.add()`, como mostrado abaixo:

```
#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)))
```

O exemplo anterior usou a notação de índice de matriz para recuperar um elemento com `arr2[$idx]`. Você pode pesquisar por nome a partir de um Mapa/dicionário de uma maneira semelhante:

```
#set($result = {
    "Author" : "Nadia",
    "Topic" : "GraphQL"
})

$util.qr($myMap.put("Author", $result["Author"]))
```

Isso é muito comum ao filtrar resultados retornados das fontes de dados em Modelos de resposta ao usar condicionais.

Verificações condicionais

A seção anterior com `#foreach` demonstrou alguns exemplos de uso da lógica para transformar dados com a VTL. Também é possível aplicar verificações condicionais para avaliar dados durante o runtime:

```
#if(!$array.isEmpty())
    $util.qr($myMap.put("ifCheck", "Array not empty"))
#else
    $util.qr($myMap.put("ifCheck", "Your array is empty"))
#end
```

A verificação `#if()` acima de uma expressão Booleana é legal, mas você também pode usar operadores e `#elseif()` para a ramificação:

```
#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
```

Esses dois exemplos mostraram negação (!) e igualdade (==). Também podemos usar `||`, `&&`, `>`, `<`, `>=`, `<=` e `!=`.

```
#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
```

Observação: somente `Boolean.FALSE` e `null` são considerados falsos em condicionais. Zero (0) e strings vazias (" ") não são equivalentes a falso.

Operadores

Nenhuma linguagem de programação seria completa sem algumas operadores para executar algumas ações matemáticas. Veja aqui alguns exemplos para começar a usar:

```
#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))
```

Como usar loops e condicionais juntos

É muito comum ao transformar dados na VTL, como antes da gravação ou leitura de uma fonte de dados, fazer loop pelos objetos e, em seguida, executar verificações antes de realizar uma ação. Combinar algumas das ferramentas das seções anteriores oferece várias funcionalidades. Uma ferramenta útil é saber que `#foreach` fornece automaticamente uma `.count` em cada item:

```
#foreach ($item in $arr2)
  #set($idx = "item" + $foreach.count)
  $util.qr($myMap.put($idx, $item))
#end
```

Por exemplo, talvez você deseja apenas extrair os valores de um mapa se estiver abaixo de um determinado tamanho. Usar a contagem junto com condicionais e a instrução `#break` permite que você faça o seguinte:

```
#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
```

O anterior `#foreach` é iterado com `.keySet()`, que você pode usar em mapas. Isso oferece acesso para obter a `$key` e fazer referência a um valor com uma `.get($key)`. Os argumentos do GraphQL dos clientes em AWS AppSync são armazenados como um mapa. Eles também podem ser percorridos com `.entrySet()`, o que pode acessar chaves e valores como um Conjunto e popular outras variáveis ou realizar verificações condicionais complexas, como validação ou transformação de entrada:

```
#foreach( $entry in $context.arguments.entrySet() )
#if ($entry.key == "XYZ" && $entry.value == "BAD")
  #set($myvar = "...")
#else
  #break
#end
#end
```

Outros exemplos comuns são o preenchimento automático de informações padrão, como as versões do objeto inicial durante a sincronização de dados (muito importante na resolução de conflitos) ou o proprietário padrão de um objeto para verificações de autorização – Mary criou esta publicação de blog, portanto:

```
#set($myMap.owner ="Mary")
#set($myMap.defaultOwners = ["Admins", "Editors"])
```

Contexto

Agora que você está mais familiarizado com a execução de verificações lógicas nos resolvedores do AWS AppSync com VTL, observe o objeto de contexto:

```
$util.qr($myMap.put("context", $context))
```

Isso contém todas as informações que podem ser acessadas na solicitação do GraphQL. Para obter uma explicação detalhada, consulte a [referência do contexto](#).

Filtrar

Até agora nesse tutorial todas as informações da função do Lambda foram retornadas para a consulta do GraphQL com uma transformação JSON muito simples:

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

A lógica VTL também é muito poderosa ao obter respostas de uma fonte de dados, especialmente ao fazer verificações de autorização em recursos. Vamos examinar alguns exemplos. Primeiro, tente alterar o modelo de resposta da seguinte forma:

```
#set($data = {  
  "id" : "456",  
  "meta" : "Valid Response"  
})  
  
$util.toJson($data)
```

Não importa o que aconteça com a operação do GraphQL, os valores codificados são retornados ao cliente. Altere um pouco para que o campo meta seja preenchido a partir da resposta do Lambda, definido anteriormente no tutorial no valor `elseifCheck` ao aprender sobre condicionais:

```
#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)
```

`$context.result` é um mapa, portanto você pode usar `entrySet()` para executar a lógica nas chaves ou valores retornados. Como `$context.identity` contém informações sobre o usuário que executou a operação do GraphQL, se você retornar informações de autorização a partir da fonte de dados, é possível decidir retornar todos os dados, dados parciais ou nenhum dado para um usuário com base na sua lógica. Altere o modelo da resposta para se parecer com o seguinte:

```
#if($context.result["id"] == 123)
    $util.toJson($context.result)
#else
    $util.unauthorized()
#end
```

Se você executar a consulta do GraphQL, os dados serão retornados como normal. No entanto, se você alterar o argumento do `id` para algo diferente de 123 (query `test { get(id:456 meta:"badrequest") { } }`), você receberá uma mensagem de falha de autorização.

Encontre mais exemplos de cenários de autorização na seção dos [casos de uso de autorização](#).

Exemplo de modelo

Se você acompanhou o tutorial, pode ter criado esse modelo passo a passo. Caso não tenha feito isso, o incluímos abaixo para copiá-lo para testes.

Modelo de solicitação

```
#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
#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)
}

```

Modelo da resposta

```

#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 referência de contexto do modelo de mapeamento do resolvidor

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync define um conjunto de variáveis e funções para trabalhar com modelos de mapeamento do resolvidor. Isso facilita as operações lógicas em dados com o GraphQL. Este documento descreve essas funções e fornece exemplos para trabalhar com modelos.

Acesso ao **\$context**

A variável **\$context** é um mapa que contém todas as informações contextuais para a invocação do resolvidor. Ela tem a seguinte estrutura:

```
{
  "arguments" : { ... },
  "source" : { ... },
  "result" : { ... },
```



```
"identity" : { ... },  
"request" : { ... },  
"info": { ... }  
}
```

Note

Se você está tentando acessar uma entrada de dicionário/mapa (como uma entrada em context) com sua chave para recuperar o valor, a Velocity Template Language (VTL) permite usar diretamente a notação `<dictionary-element>.<key-name>`. No entanto, isso pode não funcionar para todos os casos, como quando os nomes das chaves tiverem caracteres especiais (por exemplo, um sublinhado "_"). Recomendamos sempre usar a notação `<dictionary-element>.get("<key-name>")`.

Cada campo no mapa `$context` é definido da seguinte forma:

Campos de `$context`

arguments

Um mapa que contém todos os argumentos do GraphQL para este campo.

identity

Um objeto que contém informações sobre o chamador. Para obter mais informações sobre a estrutura desse campo, consulte [Identidade](#).

source

Um mapa que contém a resolução do campo pai.

stash

O stash é um mapa disponibilizado dentro de cada modelo de mapeamento de resolvidor e função. A mesma instância stash passa por uma única execução do resolvidor. Isso significa que é possível usar o stash para passar dados arbitrários entre os modelos de mapeamento de solicitação e resposta e entre as funções em um resolvidor de pipeline. O stash expõe os mesmos métodos que a estrutura de dados do [Java Map](#).

result

Um contêiner para os resultados desse resolvedor. Esse campo só está disponível para modelos de mapeamento de resposta.

Por exemplo, se estiver resolvendo o campo `author` da seguinte consulta:

```
query {  
  getPost(id: 1234) {  
    postId  
    title  
    content  
    author {  
      id  
      name  
    }  
  }  
}
```

Em seguida, a variável `$context` completa que está disponível ao processar o modelo de mapeamento da resposta pode ser:

```
{  
  "arguments" : {  
    id: "1234"  
  },  
  "source": {},  
  "result" : {  
    "postId": "1234",  
    "title": "Some title",  
    "content": "Some content",  
    "author": {  
      "id": "5678",  
      "name": "Author Name"  
    }  
  },  
  "identity" : {  
    "sourceIp" : ["x.x.x.x"],  
    "userArn" : "arn:aws:iam::123456789012:user/appsync",  
    "accountId" : "666666666666",  
    "user" : "AIDAAAAAAAAAAAAAAAAAAAA"  
  }  
}
```

```
}
```

prev.result

O resultado de qualquer operação anterior executada em um resolvedor de pipeline.

Se a operação anterior foi o modelo de mapeamento Anterior do resolvedor de pipeline, `$ctx.prev.result` representa a saída da avaliação do modelo e será disponibilizado para a primeira função no pipeline.

Se a operação anterior foi a primeira função, `$ctx.prev.result` representa a saída da primeira função e será disponibilizado para a segunda função no pipeline.

Se a operação anterior foi a última função, `$ctx.prev.result` representa a saída da primeira função e será disponibilizado para o modelo de mapeamento Posterior do resolvedor de pipeline.

info

Um objeto que contém informações sobre a solicitação do GraphQL. Para ver a estrutura desse campo, consulte [Informações](#).

Identidade

A seção `identity` contém informações sobre o chamador. O formato dessa seção depende do tipo de autorização da sua AWS AppSync API.

Para obter mais informações sobre as opções de AWS AppSync segurança, consulte [Autorização e autenticação](#).

API_KEY authorization

O campo `identity` não está preenchido.

AWS_LAMBDA authorization

`identity` inclui a chave `resolverContext` que tem o mesmo conteúdo de `resolverContext` retornado pela função do Lambda que autorizou a solicitação.

AWS_IAM authorization

O `identity` tem o seguinte formato:

```
{
```

```
"accountId" : "string",
"cognitoIdentityPoolId" : "string",
"cognitoIdentityId" : "string",
"sourceIp" : ["string"],
"username" : "string", // IAM user principal
"userArn" : "string",
"cognitoIdentityAuthType" : "string", // authenticated/unauthenticated based on
the identity type
"cognitoIdentityAuthProvider" : "string" // the auth provider that was used to
obtain the credentials
}
```

AMAZON_COGNITO_USER_POOLS authorization

O identity tem o seguinte formato:

```
{
  "sub" : "uuid",
  "issuer" : "string",
  "username" : "string"
  "claims" : { ... },
  "sourceIp" : ["x.x.x.x"],
  "defaultAuthStrategy" : "string"
}
```

Cada campo é definido da seguinte forma:

accountId

O ID da AWS conta do chamador.

claims

As reivindicações do usuário.

cognitoIdentityAuthType

Autenticado ou não autenticado com base no tipo de identidade.

cognitoIdentityAuthProvider

Uma lista separada por vírgulas das informações do provedor de identidade externo usada na obtenção das credenciais usadas para assinar a solicitação.

cognitoIdentityId

O ID de identidade do Amazon Cognito do chamador.

cognitoIdentityPoolId

O ID do banco de identidades do Amazon Cognito associado ao chamador.

defaultAuthStrategy

A estratégia de autorização padrão para este chamador (ALLOW ou DENY).

issuer

O emissor do token.

sourceIp

O endereço IP de origem do chamador que AWS AppSync recebe. Se a solicitação não incluir o cabeçalho `x-forwarded-for`, o valor do IP de origem conterá apenas um único endereço IP da conexão TCP. Se a solicitação inclui um cabeçalho `x-forwarded-for`, o IP de origem será uma lista de endereços IP do cabeçalho `x-forwarded-for`, além do endereço IP da conexão TCP.

sub

O UUID do usuário autenticado.

user

O usuário do IAM.

userArn

O nome do recurso da Amazon (ARN) do usuário do IAM.

username

O nome do usuário autenticado. Em caso de autorização `AMAZON_COGNITO_USER_POOLS`, o valor do nome de usuário é o valor de `username` é o valor do atributo `cognito:username`. No caso de `AWS_IAM` autorização, o valor do nome de usuário é o valor do AWS usuário principal. Se você estiver usando a autorização do IAM com credenciais fornecidas por bancos de identidades do Amazon Cognito, recomendamos utilizar `cognitoIdentityId`.

Cabeçalhos de solicitação de acesso

AWS AppSync suporta passar cabeçalhos personalizados de clientes e acessá-los em seus resolvedores GraphQL usando `$context.request.headers`. Em seguida, você pode usar

valores de cabeçalho para ações como inserir dados em uma fonte de dados ou até mesmo verificações de autorização. É possível utilizar cabeçalhos de solicitação por meio de `$curl` com uma chave da API na linha de comando conforme mostrado nos exemplos a seguir:

Exemplo de cabeçalho único

Digamos que você defina um cabeçalho custom com um valor de `nadia` da seguinte forma:

```
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
```

Isso pode ser acessado com `$context.request.headers.custom`. Por exemplo, ele pode estar no seguinte VTL para o DynamoDB:

```
"custom": $util.dynamodb.toDynamoDBJson($context.request.headers.custom)
```

Exemplo de vários cabeçalhos

Você também pode enviar vários cabeçalhos em uma única solicitação e acessá-los no modelo de mapeamento do resolvedor. Por exemplo, se o cabeçalho custom foi definido com dois valores:

```
curl -XPOST -H "Content-Type:application/graphql" -H "custom:bailey" -H "custom:nadia" -H "x-api-key:<API-KEY-VALUE>" -d '{"query":"mutation { createEvent(name: \"demo\", when: \"Next Friday!\", where: \"Here!\") {id name when where description}}}' https://<ENDPOINT>/graphql
```

Em seguida, você pode acessá-los como uma matriz, como `$context.request.headers.custom[1]`.

Note

AWS AppSync não expõe o cabeçalho do cookie em `$context.request.headers`.

Acessar o nome de domínio personalizado da solicitação

AWS AppSync suporta a configuração de um domínio personalizado que você pode usar para acessar seu GraphQL e endpoints em tempo real para seu APIs. Ao fazer uma solicitação

com um nome de domínio personalizado, você pode obter o nome de domínio usando `$context.request.domainName`.

Ao usar o nome de domínio padrão do endpoint do GraphQL, o valor será `null`.

Informações

A seção `info` contém informações sobre a solicitação do GraphQL. Esta seção tem o seguinte formato:

```
{
  "fieldName": "string",
  "parentTypeName": "string",
  "variables": { ... },
  "selectionSetList": ["string"],
  "selectionSetGraphQL": "string"
}
```

Cada campo é definido da seguinte forma:

fieldName

O nome do campo que está sendo resolvido no momento.

parentTypeName

O nome do tipo pai para o campo que está sendo resolvido no momento.

variables

Um mapa que contém todas as variáveis que são passadas para a solicitação do GraphQL.

selectionSetList

Uma representação de lista dos campos no conjunto de seleções do GraphQL. Os campos com alias serão referenciados somente pelo nome do alias, não pelo nome do campo. O exemplo a seguir mostra isso em detalhes.

selectionSetGraphQL

Uma representação de string do conjunto de seleções, formatada como linguagem de definição de esquema (SDL) do GraphQL. Embora os fragmentos não sejam mesclados no conjunto de seleções, os fragmentos inline são preservados, conforme mostrado no exemplo a seguir.

 Note

Ao usar `$utils.toJson()` em `context.info`, os valores retornados por `selectionSetGraphQL` e `selectionSetList` não serão serializados por padrão.

Por exemplo, se estiver resolvendo o campo `getPost` da seguinte consulta:

```
query {
  getPost(id: $postId) {
    postId
    title
    secondTitle: title
    content
    author(id: $authorId) {
      authorId
      name
    }
    secondAuthor(id: "789") {
      authorId
    }
    ... on Post {
      inlineFrag: comments: {
        id
      }
    }
    ... postFrag
  }
}

fragment postFrag on Post {
  postFrag: comments: {
    id
  }
}
```

Depois, a variável `$context.info` completa que está disponível ao processar o modelo de mapeamento pode ser:

```
{
  "fieldName": "getPost",
  "parentTypeName": "Query",
```



```

"variables": {
  "postId": "123",
  "authorId": "456"
},
"selectionSetList": [
  "postId",
  "title",
  "secondTitle",
  "content",
  "author",
  "author/authorId",
  "author/name",
  "secondAuthor",
  "secondAuthor/authorId",
  "inlineFragComments",
  "inlineFragComments/id",
  "postFragComments",
  "postFragComments/id"
],
"selectionSetGraphQL": "{\n  getPost(id: $postId) {\n    postId\n    title\n    secondTitle: title\n    content\n    author(id: $authorId) {\n      authorId\n      name\n    }\n    secondAuthor(id: \"789\") {\n      authorId\n    }\n    ... on Post\n    {\n      inlineFrag: comments {\n        id\n      }\n    }\n    ... postFrag\n  }\n}"
}

```

`selectionSetList` expõe somente campos que pertencem ao tipo atual. Se o tipo atual for uma interface ou união, somente os campos selecionados que pertencem à interface serão expostos. Por exemplo, considerando o seguinte esquema:

```

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  
}
```

E a seguinte consulta:

```
query {  
  node(id: "post1") {  
    id  
    ... on Post {  
      title  
    }  
  
    ... on Blog {  
      title  
    }  
  }  
}
```

Ao chamar `$ctx.info.selectionSetList` na resolução do campo `Query.node`, somente `id` é exposto:

```
"selectionSetList": [  
  "id"  
]
```

Limpar entradas

Os aplicativos devem limpar entradas não confiáveis para impedir que qualquer parte externa use um aplicativo fora do uso pretendido. Como o `$context` contém entradas do usuário em propriedades como `$context.arguments`, `$context.identity`, `$context.result`, `$context.info.variables` e `$context.request.headers`, tome cuidado para limpar seus valores nos modelos de mapeamento.

Como os modelos de mapeamento representam JSON, a limpeza de entradas assume a forma de escape de caracteres reservados JSON de strings que representem entradas do usuário. É uma melhor prática usar o utilitário `$util.toJson()` para o escape de caracteres reservados JSON de valores de string sensíveis ao colocá-los em um modelo de mapeamento.

Por exemplo, no modelo de mapeamento de solicitação do Lambda abaixo, como acessamos uma string de entrada de cliente insegura (`$context.arguments.id`), ela foi encapsulada com `$util.toJson()` para evitar que caracteres JSON sem escape corrompam o modelo JSON.

```
{
  "version": "2017-02-28",
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "postId": $util.toJson($context.arguments.id)
  }
}
```

Ao contrário do modelo de mapeamento abaixo, em que inserimos `$context.arguments.id` diretamente, sem limpeza. Isso não funcionará para strings contendo aspas duplas ou outros caracteres reservados JSON sem escape e sujeita seu modelo a falhas.

```
## 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 referência do utilitário do modelo de mapeamento do resolvedor

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync define um conjunto de utilitários que você pode usar em um resolvedor GraphQL para simplificar as interações com fontes de dados. Alguns desses utilitários são para uso geral com

qualquer fonte de dados, como geração IDs ou registros de data e hora. Outros são específicos para um tipo de fonte de dados. Os seguintes utilitários estão disponíveis:

- [Utilitários auxiliares em \\$util](#): a variável \$util contém métodos utilitários gerais para ajudar você a trabalhar com dados. A menos que especificado o contrário, todos os utilitários usam o conjunto de caracteres UTF-8.
- [AppSync diretivas](#) - AppSync expõe diretivas para facilitar a produtividade do desenvolvedor ao escrever em VTL.
- [Auxiliares de tempo em \\$util.time](#): a variável \$util.time contém métodos de data e hora para ajudar a gerar timestamps, converter entre formatos de data e hora e analisar strings de data e hora. A sintaxe dos formatos de data e hora é baseada em [DateTimeFormatter](#), que você pode consultar para obter mais documentação.
- [Auxiliares de lista em \\$util.list](#): \$util.list contém métodos para ajudar com operações de lista comuns, como remover e reter itens de uma lista para filtrar casos de uso.
- [Auxiliares de mapa em \\$util.map](#): \$util.map contém métodos para ajudar com operações de mapa comuns, como remover e reter itens de um mapa para filtrar casos de uso.
- [Auxiliares do DynamoDB em \\$util.dynamodb](#): \$util.dynamodb contém métodos auxiliares que tornam mais fácil gravar e ler dados no Amazon DynamoDB, como mapeamento e formatação do tipo automático.
- [Auxiliares do Amazon RDS em \\$util.rds](#): \$util.rds contém métodos auxiliares que formatam operações do RDS eliminando dados estranhos nas saídas de resultados.
- [Auxiliares HTTP em \\$util.http](#): o utilitário \$util.http fornece métodos auxiliares que podem ser usados para gerenciar parâmetros de solicitação HTTP e adicionar cabeçalhos de resposta.
- [Auxiliares XML em \\$util.xml](#): \$util.xml contém métodos auxiliares que podem facilitar a tradução de respostas XML para JSON ou um dicionário.
- [Auxiliares de transformação em \\$util.transform](#): \$util.transform contém métodos auxiliares que facilitam a execução de operações complexas em fontes de dados, como operações de filtragem do DynamoDB.
- [Auxiliares de matemática em \\$util.math](#): \$util.math contém métodos para ajudar com operações matemáticas comuns.
- [Auxiliares de string em \\$util.str](#): \$util.str contém métodos para ajudar com operações comuns de string.
- [Extensões](#): \$extensions contém um conjunto de métodos para realizar ações adicionais em seus resolvedores.

Utilitários auxiliares em \$util

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

A variável `$util` contém métodos utilitários gerais para ajudar você a trabalhar com dados. A menos que especificado o contrário, todos os utilitários usam o conjunto de caracteres UTF-8.

Utilitários de análise JSON

Lista de utilitários de análise JSON

`$util.parseJson(String) : Object`

Usa um JSON "transformado em string" e retorna uma representação de objeto do resultado.

`$util.toJson(Object) : String`

Recebe um objeto e retorna uma representação JSON "transformado em string" desse objeto.

Utilitários de codificação

Lista de utilitários de codificação

`$util.urlEncode(String) : String`

Retorna a string de entrada como uma string codificada `application/x-www-form-urlencoded`.

`$util.urlDecode(String) : String`

Decodifica uma string codificada `application/x-www-form-urlencoded` de volta ao seu formato não codificado.

`$util.base64Encode( byte[] ) : String`

Codifica a entrada em uma string codificada em base64.

`$util.base64Decode(String) : byte[]`

Decodifica os dados de uma string codificada em base64.

Utilitários de geração de ID

Lista de utilitários de geração de ID

\$util.autoId() : String

Retorna um UUID de 128 bits gerado aleatoriamente.

\$util.autoUlid() : String

Retorna um identificador lexicograficamente classificável universalmente exclusivo (ULID) de 128 bits gerado aleatoriamente.

\$util.autoKsuid() : String

Retorna um identificador exclusivo classificável por K (KSUID) de 128 bits gerado aleatoriamente, codificado em base62, como uma string com comprimento de 27.

Utilitários de erro

Lista de utilitários de erro

\$util.error(String)

Lança um erro personalizado. Use em modelos de mapeamento de solicitação ou resposta para detectar um erro na solicitação ou no resultado da invocação.

\$util.error(String, String)

Lança um erro personalizado. Use em modelos de mapeamento de solicitação ou resposta para detectar um erro na solicitação ou no resultado da invocação. Também é possível especificar um `errorType`.

\$util.error(String, String, Object)

Lança um erro personalizado. Use em modelos de mapeamento de solicitação ou resposta para detectar um erro na solicitação ou no resultado da invocação. Você também pode especificar um `errorType` e um campo `data`. O valor `data` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL.

Note

`data` será filtrado com base no conjunto de seleção da consulta.

\$util.error(String, String, Object, Object)

Lança um erro personalizado. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Além disso, é possível especificar os campos `errorType`, `data` e `errorInfo`. O valor `data` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL.

Note

`data` será filtrado com base no conjunto de seleção da consulta. O valor `errorInfo` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL. `errorInfo` NÃO será filtrado com base no conjunto de seleção da consulta.

\$util.appendError(String)

Anexa um erro personalizado no final. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Diferente de `$util.error(String)`, a avaliação do modelo não será interrompida para que os dados possam ser retornados ao chamador.

\$util.appendError(String, String)

Anexa um erro personalizado no final. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Além disso, um `errorType` pode ser especificado. Diferente de `$util.error(String, String)`, a avaliação do modelo não será interrompida para que os dados possam ser retornados ao chamador.

\$util.appendError(String, String, Object)

Anexa um erro personalizado no final. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Além disso, um `errorType` e um campo `data` podem ser especificados. Diferente de `$util.error(String, String, Object)`, a avaliação do modelo não será interrompida para que os dados possam ser retornados ao chamador. O valor `data` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL.

 Note

data será filtrado com base no conjunto de seleção da consulta.

\$util.appendError(String, String, Object, Object)

Anexa um erro personalizado no final. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Além disso, é possível especificar os campos `errorType`, `data` e `errorInfo`. Diferente de `$util.error(String, String, Object, Object)`, a avaliação do modelo não será interrompida para que os dados possam ser retornados ao chamador. O valor `data` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL.

 Note

data será filtrado com base no conjunto de seleção da consulta. O valor `errorInfo` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL. `errorInfo` NÃO será filtrado com base no conjunto de seleção da consulta.

Utilitários de validação de condições

Lista de utilitários de validação de condição

\$util.validate(Boolean, String) : void

Se a condição for falsa, lance um `CustomTemplateException` com a mensagem especificada.

\$util.validate(Boolean, String, String) : void

Se a condição for falsa, lance um `CustomTemplateException` com a mensagem e o tipo de erro especificados.

\$util.validate(Boolean, String, String, Object) : void

Se a condição for falsa, lance um `CustomTemplateException` com a mensagem e o tipo de erro especificados, bem como os dados a serem retornados na resposta.

Utilitários de comportamento nulo

Lista de utilitários de comportamento nulo

\$util.isNull(Object) : Boolean

Retorna verdadeiro se o objeto fornecido for nulo.

\$util.isNullOrEmpty(String) : Boolean

Retorna verdadeiro, se os dados fornecidos forem nulos ou uma string vazia. Caso contrário, retornará falso.

\$util.isNullOrBlank(String) : Boolean

Retorna verdadeiro, se os dados fornecidos forem nulos ou uma string em branco. Caso contrário, retornará falso.

\$util.defaultIfNull(Object, Object) : Object

Retorna o primeiro Objeto que não seja nulo. Caso contrário, retorna o segundo objeto como um "Objeto padrão".

\$util.defaultIfNullOrEmpty(String, String) : String

Retorna a primeira String se não for nula ou vazia. Caso contrário, retorna a segunda String como uma "String padrão".

\$util.defaultIfNullOrBlank(String, String) : String

Retorna a primeira String se não for nula ou em branco. Caso contrário, retorna a segunda String como uma "String padrão".

Utilitários de correspondência de padrões

Lista de utilitários de correspondência de tipos e padrões

\$util.typeOf(Object) : String

Retorna uma String que descreve o tipo do Objeto. As identificações de tipo compatíveis são: "Nulo", "Número", "String", "Mapa", "Lista", "Booleano". Se um tipo não puder ser identificado, o tipo de retorno é "Objeto".

\$util.matches(String, String) : Boolean

Retorna verdadeiro se o padrão especificado no primeiro argumento corresponde aos dados fornecidos no segundo argumento. O padrão deve ser uma expressão regular, como `$util.matches("a*b", "aaaaab")`. A funcionalidade se baseia em [Padrão](#), que você pode consultar para obter documentação adicional.

\$util.authType() : String

Retorna uma String que descreve o tipo de autenticação múltipla que está sendo usado por uma solicitação, retornando "Autorização do IAM", "Autorização de grupo de usuários", "Autorização do Open ID Connect" ou "Autorização de chave de API".

Utilitários de validação de objetos

Lista de utilitários de validação de objeto

\$util.isString(Object) : Boolean

Retorna verdadeiro se o Objeto for uma String.

\$util.isNumber(Object) : Boolean

Retorna verdadeiro se o Objeto for um Número.

\$util.isBoolean(Object) : Boolean

Retorna verdadeiro se o Objeto for um Booleano.

\$util.isList(Object) : Boolean

Retorna verdadeiro se o Objeto for uma Lista.

\$util.isMap(Object) : Boolean

Retorna verdadeiro se o Objeto for um Mapa.

CloudWatch utilitários de registro

CloudWatch lista de utilitários de registro

\$util.log.info(Object) : Void

Registra a representação de string do objeto fornecido no fluxo de registros solicitado quando o registro em nível de solicitação e campo é ativado com nível CloudWatch de registro ou DEBUG em uma INFO API. ALL

\$util.log.info(String, Object...) : Void

Registra a representação de string dos objetos fornecidos no fluxo de registros solicitado quando o registro em nível de solicitação e campo é ativado com nível de CloudWatch registro em uma API. ALL Esse utilitário substituirá todas as variáveis indicadas por "{}" no primeiro formato de entrada String pela representação String dos objetos fornecidos em ordem.

\$util.log.debug(Object) : Void

Registra a representação de string do objeto fornecido no fluxo de registros solicitado quando o registro em nível de solicitação e campo é ativado com nível de CloudWatch registro ALL ou DEBUG em uma API.

\$util.log.debug(String, Object...) : Void

Registra a representação de string dos objetos fornecidos no fluxo de registro solicitado quando o registro em nível de campo CloudWatch é ativado com nível de registro DEBUG ou nível de registro ALL em uma API. Esse utilitário substituirá todas as variáveis indicadas por "{}" no primeiro formato de entrada String pela representação String dos objetos fornecidos em ordem.

\$util.log.error(Object) : Void

Registra a representação de string do objeto fornecido no fluxo de registro solicitado quando o CloudWatch registro em nível de campo é ativado com qualquer nível de registro (ALL,, INFODEBUG, etc.) em uma API.

\$util.log.error(String, Object...) : Void

Registra a representação de string dos objetos fornecidos no fluxo de registro solicitado quando o registro em nível de campo CloudWatch é ativado com nível de registro ERROR ou nível de registro ALL em uma API. Esse utilitário substituirá todas as variáveis indicadas por "{}" no primeiro formato de entrada String pela representação String dos objetos fornecidos em ordem.

Utilitários de comportamento do valor de retorno

Lista de utilitários de comportamento de valor de retorno

\$util.qr() e **\$util.quiet()**

Executa uma instrução de VTL enquanto suprime o valor retornado. Isso é útil para executar métodos sem usar espaços reservados temporários, como adicionar itens a um mapa. Por exemplo:

```
#set ($myMap = {})  
#set($discard = $myMap.put("id", "first value"))
```

Se torna:

```
#set ($myMap = {})  
$util.qr($myMap.put("id", "first value"))
```

\$util.escapeJavaScript(String) : String

Retorna a string de entrada como uma string JavaScript de escape.

\$util.urlEncode(String) : String

Retorna a string de entrada como uma string codificada application/x-www-form-urlencoded.

\$util.urlDecode(String) : String

Decodifica uma string codificada application/x-www-form-urlencoded de volta ao seu formato não codificado.

\$util.base64Encode(byte[]) : String

Codifica a entrada em uma string codificada em base64.

\$util.base64Decode(String) : byte[]

Decodifica os dados de uma string codificada em base64.

\$util.parseJson(String) : Object

Usa um JSON "transformado em string" e retorna uma representação de objeto do resultado.

\$util.toJson(Object) : String

Recebe um objeto e retorna uma representação JSON "transformado em string" desse objeto.

\$util.autoId() : String

Retorna um UUID de 128 bits gerado aleatoriamente.

\$util.autoUlid() : String

Retorna um identificador lexicograficamente classificável universalmente exclusivo (ULID) de 128 bits gerado aleatoriamente.

\$util.autoKsuid() : String

Retorna um identificador exclusivo classificável por K (KSUID) de 128 bits gerado aleatoriamente, codificado em base62, como uma string com comprimento de 27.

\$util.unauthorized()

Lança Unauthorized para o campo a ser resolvido. Use em modelos de mapeamento de solicitação ou resposta para determinar se é preciso ou não permitir que o chamador resolva o campo.

\$util.error(String)

Lança um erro personalizado. Use em modelos de mapeamento de solicitação ou resposta para detectar um erro na solicitação ou no resultado da invocação.

\$util.error(String, String)

Lança um erro personalizado. Use em modelos de mapeamento de solicitação ou resposta para detectar um erro na solicitação ou no resultado da invocação. Também é possível especificar um `errorType`.

\$util.error(String, String, Object)

Lança um erro personalizado. Use em modelos de mapeamento de solicitação ou resposta para detectar um erro na solicitação ou no resultado da invocação. Você também pode especificar um `errorType` e um campo `data`. O valor `data` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL. Observação: `data` será filtrado com base no conjunto de seleção da consulta.

\$util.error(String, String, Object, Object)

Lança um erro personalizado. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Além disso, um campo `errorType`, um campo `data` e um campo `errorInfo`

podem ser especificados. O valor `data` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL. Observação: `data` será filtrado com base no conjunto de seleção da consulta. O valor `errorInfo` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL. Observação: `errorInfo` NÃO será filtrado com base no conjunto de seleção da consulta.

`$util.appendError(String)`

Anexa um erro personalizado no final. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Diferente de `$util.error(String)`, a avaliação do modelo não será interrompida para que os dados possam ser retornados ao chamador.

`$util.appendError(String, String)`

Anexa um erro personalizado no final. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Além disso, um `errorType` pode ser especificado. Diferente de `$util.error(String, String)`, a avaliação do modelo não será interrompida para que os dados possam ser retornados ao chamador.

`$util.appendError(String, String, Object)`

Anexa um erro personalizado no final. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Além disso, um `errorType` e um campo `data` podem ser especificados. Diferente de `$util.error(String, String, Object)`, a avaliação do modelo não será interrompida para que os dados possam ser retornados ao chamador. O valor `data` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL. Observação: `data` será filtrado com base no conjunto de seleção da consulta.

`$util.appendError(String, String, Object, Object)`

Anexa um erro personalizado no final. Isso pode ser usado em modelos de mapeamento da solicitação ou resposta se o modelo detectar um erro com a solicitação ou com o resultado da invocação. Além disso, um campo `errorType`, um campo `data` e um campo `errorInfo` podem ser especificados. Diferente de `$util.error(String, String, Object, Object)`, a avaliação do modelo não será interrompida para que os dados possam ser retornados ao chamador. O valor `data` será adicionado ao bloco `error` correspondente em `errors` na resposta do GraphQL. Observação: `data` será filtrado com base no conjunto de seleção da consulta. O valor `errorInfo` será adicionado ao bloco `error` correspondente em

errors na resposta do GraphQL. Observação: `errorInfo` NÃO será filtrado com base no conjunto de seleção da consulta.

`$util.validate(Boolean, String) : void`

Se a condição for falsa, lance um `CustomTemplateException` com a mensagem especificada.

`$util.validate(Boolean, String, String) : void`

Se a condição for falsa, lance um `CustomTemplateException` com a mensagem e o tipo de erro especificados.

`$util.validate(Boolean, String, String, Object) : void`

Se a condição for falsa, lance um `CustomTemplateException` com a mensagem e o tipo de erro especificados, bem como os dados a serem retornados na resposta.

`$util.isNull(Object) : Boolean`

Retorna verdadeiro se o objeto fornecido for nulo.

`$util.isNullOrEmpty(String) : Boolean`

Retorna verdadeiro, se os dados fornecidos forem nulos ou uma string vazia. Caso contrário, retornará falso.

`$util.isNullOrBlank(String) : Boolean`

Retorna verdadeiro, se os dados fornecidos forem nulos ou uma string em branco. Caso contrário, retornará falso.

`$util.defaultIfNull(Object, Object) : Object`

Retorna o primeiro Objeto que não seja nulo. Caso contrário, retorna o segundo objeto como um "Objeto padrão".

`$util.defaultIfNullOrEmpty(String, String) : String`

Retorna a primeira String se não for nula ou vazia. Caso contrário, retorna a segunda String como uma "String padrão".

`$util.defaultIfNullOrBlank(String, String) : String`

Retorna a primeira String se não for nula ou em branco. Caso contrário, retorna a segunda String como uma "String padrão".

`$util.isString(Object) : Boolean`

Retorna verdadeiro se o Objeto for uma String.

\$util.isNumber(Object) : Boolean

Retorna verdadeiro se o Objeto for um Número.

\$util.isBoolean(Object) : Boolean

Retorna verdadeiro se o Objeto for um Booliano.

\$util.isList(Object) : Boolean

Retorna verdadeiro se o Objeto for uma Lista.

\$util.isMap(Object) : Boolean

Retorna verdadeiro se o Objeto for um Mapa.

\$util.typeOf(Object) : String

Retorna uma String que descreve o tipo do Objeto. As identificações de tipo compatíveis são: "Nulo", "Número", "String", "Mapa", "Lista", "Booliano". Se um tipo não puder ser identificado, o tipo de retorno é "Objeto".

\$util.matches(String, String) : Boolean

Retorna verdadeiro se o padrão especificado no primeiro argumento corresponde aos dados fornecidos no segundo argumento. O padrão deve ser uma expressão regular, como `$util.matches("a*b", "aaaaab")`. A funcionalidade se baseia em [Padrão](#), que você pode consultar para obter documentação adicional.

\$util.authType() : String

Retorna uma String que descreve o tipo de autenticação múltipla que está sendo usado por uma solicitação, retornando "Autorização do IAM", "Autorização de grupo de usuários", "Autorização do Open ID Connect" ou "Autorização de chave de API".

\$util.log.info(Object) : Void

Registra a representação de string do objeto fornecido no fluxo de registros solicitado quando o registro em nível de solicitação e campo é ativado com nível de CloudWatch registro em uma API. ALL

\$util.log.info(String, Object...) : Void

Registra a representação de string dos objetos fornecidos no fluxo de registros solicitado quando o registro em nível de solicitação e campo é ativado com nível de CloudWatch registro

em uma API. ALL Esse utilitário substituirá todas as variáveis indicadas por "{}" no primeiro formato de entrada String pela representação String dos objetos fornecidos em ordem.

\$util.log.error(Object) : Void

Registra a representação de string do objeto fornecido no fluxo de registro solicitado quando o registro em nível de campo CloudWatch é ativado com nível de registro ERROR ou nível de registro ALL em uma API.

\$util.log.error(String, Object...) : Void

Registra a representação de string dos objetos fornecidos no fluxo de registro solicitado quando o registro em nível de campo CloudWatch é ativado com nível de registro ERROR ou nível de registro ALL em uma API. Esse utilitário substituirá todas as variáveis indicadas por "{}" no primeiro formato de entrada String pela representação String dos objetos fornecidos em ordem.

\$util.escapeJavaScript(String) : String

Retorna a string de entrada como uma string JavaScript de escape.

Autorização do resolvedor

Lista de autorização do resolvedor

\$util.unauthorized()

Lança Unauthorized para o campo a ser resolvido. Use em modelos de mapeamento de solicitação ou resposta para determinar se é preciso ou não permitir que o chamador resolva o campo.

AWS AppSync diretivas

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

AWS AppSync expõe diretivas para facilitar a produtividade do desenvolvedor ao escrever em VTL.

Utilitários da diretiva

#return(Object)

O `#return(Object)` permite retornar prematuramente de qualquer modelo de mapeamento. `#return(Object)` é análogo à palavra-chave `return` em linguagens de programação, pois será retornado a partir do bloco de lógica com escopo mais próximo. O uso de `#return(Object)` dentro de um modelo de mapeamento do resolvidor retornará do resolvidor. Além disso, usar `#return(Object)` de um modelo de mapeamento de função retornará da função e continuará a execução para a próxima função no pipeline ou para o modelo de mapeamento de resposta do resolvidor.

#return

A diretiva `#return` exibe o mesmo comportamento que `#return(Object)`, mas `null` será retornado em seu lugar.

Auxiliares de tempo do `$util.time`

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

A variável `$util.time` contém métodos de data e hora para ajudar a gerar timestamps, converter entre formatos de data e hora e analisar strings de data e hora. A sintaxe dos formatos de data e hora é baseada na [DateTimeFormatter](#) qual você pode consultar para obter mais documentação. Abaixo fornecemos alguns exemplos, bem como uma lista dos métodos e descrições disponíveis.

Utilitários de tempo

Lista de utilitários de tempo

`$util.time.nowISO8601()` : String

Retorna uma representação String do UTC no [formato ISO86 01](#).

`$util.time.nowEpochSeconds()` : long

Retorna o número de segundos desde epoch do 1970-01-01T00:00:00Z até agora.

\$util.time.nowEpochMilliseconds() : long

Retorna o número de milissegundos desde epoch do 1970-01-01T00:00:00Z até agora.

\$util.time.nowFormatted(String) : String

Retorna uma string do timestamp atual em UTC usando o formato especificado a partir de um tipo de entrada String.

\$util.time.nowFormatted(String, String) : String

Retorna uma string do timestamp atual para um fuso horário usando o formato especificado e o fuso horário a partir de tipos de entrada String.

\$util.time.parseFormattedToEpochMilliseconds(String, String) : Long

Analisa um carimbo de data/hora passado como uma string junto com um formato contendo um fuso horário e, em seguida, retorna o carimbo de data/hora em milissegundos desde a época.

\$util.time.parseFormattedToEpochMilliseconds(String, String, String) : Long

Analisa um timestamp enviado como uma string junto com um formato e fuso horário e retorna o timestamp como milissegundos desde o epoch.

\$util.time.parseISO8601ToEpochMilliseconds(String) : Long

Analisa um timestamp ISO86 01 passado como String e retorna o timestamp como milissegundos desde a época.

\$util.time.epochMillisecondsToSeconds(long) : long

Converte um timestamp em milissegundos epoch em um timestamp em segundos epoch.

\$util.time.epochMillisecondsToISO8601(long) : String

Converte um carimbo de data/hora de milissegundos de época em um carimbo de data/hora de 0,1. ISO86

\$util.time.epochMillisecondsToFormatted(long, String) : String

Converte um timestamp em milissegundos epoch enviado como long em um timestamp formatado de acordo com o formato em UTC.

\$util.time.epochMillisecondsToFormatted(long, String, String) : String

Converte um timestamp em milissegundos epoch enviado como um long em um timestamp formatado de acordo com o formato no fuso horário fornecido.

Exemplos de função autônoma

```
$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
```

Exemplos de conversão

```
#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
```

Exemplos de análise

```
$util.time.parseISO8601ToEpochMilliseconds("2018-02-01T17:21:05.180+08:00")
: 1517476865180
$util.time.parseFormattedToEpochMilliseconds("2018-02-02 01:19:22+0800", "yyyy-MM-dd
HH:mm:ssZ") : 1517505562000
$util.time.parseFormattedToEpochMilliseconds("2018-02-02 01:19:22", "yyyy-MM-dd
HH:mm:ss", "+08:00") : 1517505562000
```

Uso com escalares AWS AppSync definidos

Os formatos a seguir são compatíveis com `AWSDate`, `AWSDatetime` e `AWSTime`.

```
$util.time.nowFormatted("yyyy-MM-dd[XXX]", "-07:00:30") : 2018-07-11-07:00
```

```
$util.time.nowFormatted("yyyy-MM-dd'T'HH:mm:ss[XXXXX]", "-07:00:30") :  
2018-07-11T15:14:15-07:00:30
```

Auxiliares de lista em \$util.list

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

`$util.list` contém métodos para ajudar com operações de Lista comuns, como remover e reter itens de uma lista para filtrar casos de uso.

Listar utilitários

`$util.list.copyAndRetainAll(List, List) : List`

Faz uma cópia superficial da lista fornecida no primeiro argumento, retraindo apenas os itens especificados no segundo argumento, se estiverem presentes. Todos os outros itens serão removidos da cópia.

`$util.list.copyAndRemoveAll(List, List) : List`

Faz uma cópia superficial da lista fornecida no primeiro argumento, removendo quaisquer itens onde o item estiver especificado no segundo argumento, se estiverem presentes. Todos os outros itens serão retidos na cópia.

`$util.list.sortList(List, Boolean, String) : List`

Classifica uma lista de objetos, que é fornecida no primeiro argumento. Se o segundo argumento for verdadeiro, a lista será classificada de forma decrescente; se o segundo argumento for falso, a lista será classificada de forma ascendente. O terceiro argumento é o nome da string da propriedade usada para classificar uma lista de objetos personalizados. Se for uma lista apenas de strings, números inteiros, flutuantes ou duplos, o terceiro argumento pode ser qualquer string aleatória. Se todos os objetos não forem da mesma classe, a lista original será retornada. Somente listas contendo no máximo 1000 objetos são compatíveis. Veja a seguir um exemplo do uso desse utilitário:

```
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}]
```

Auxiliares de mapa em \$util.map

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

`$util.map` contém métodos para ajudar com operações de Mapa comuns, como remover e reter itens de um Mapa para filtrar casos de uso.

Utilitários do mapa

`$util.map.copyAndRetainAllKeys(Map, List) : Map`

Faz uma cópia superficial do primeiro mapa, retendo apenas as chaves especificadas na lista, se estiverem presentes. Todas as outras chaves serão removidas da cópia.

`$util.map.copyAndRemoveAllKeys(Map, List) : Map`

Faz uma cópia superficial do primeiro mapa, removendo quaisquer entradas onde a chave está especificada na lista, se estiverem presentes. Todas as outras chaves serão retidas na cópia.

Auxiliares do DynamoDB em \$util.dynamodb

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

`$util.dynamodb` contém os métodos auxiliares que facilitam gravar e ler dados no Amazon DynamoDB, como mapeamento e formatação do tipo automático. Esses métodos são projetados

para criar tipos primitivos de mapeamento e Listas no formato de entrada adequado do DynamoDB automaticamente, que é um Map do formato { "TYPE" : VALUE }.

Por exemplo, anteriormente, um modelo de mapeamento da solicitação para criar um novo item no DynamoDB pode ter a seguinte aparência:

```
{
  "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) }
  }
}
```

Se quiséssemos adicionar campos ao objeto teríamos que atualizar a consulta do GraphQL no esquema, bem como o modelo de mapeamento da solicitação. No entanto, agora podemos reestruturar o nosso modelo de mapeamento da solicitação para que colete automaticamente novos campos em nosso esquema e adicione-os ao DynamoDB com os tipos corretos:

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

No exemplo anterior, estamos usando o auxiliar `$util.dynamodb.toDynamoDBJson(...)` para pegar automaticamente o ID gerado e convertê-lo para a representação do DynamoDB de um atributo string. Em seguida, pegamos todos os argumentos e os convertemos para suas representações do DynamoDB, exportando-os para o campo `attributeValues` no modelo.

Cada auxiliar tem duas versões: uma versão que retorna um objeto (por exemplo, `$util.dynamodb.toString(...)`) e uma versão que retorna o objeto como uma string JSON (por exemplo, `$util.dynamodb.toStringJson(...)`). No exemplo anterior, usamos a versão

que retorna os dados como uma string JSON. Se quiser manipular o objeto antes de ser usado no modelo, você pode optar por retornar um objeto em vez disso, conforme mostrado a seguir:

```
{
  "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)
}
```

No exemplo anterior, estamos retornando os argumentos convertidos como um mapa em vez de uma string JSON e, em seguida, adicionando os campos `version` e `timestamp` antes de finalmente exportá-los para o campo `attributeValues` no modelo usando `$util.toJson(...)`.

A versão JSON de cada um dos auxiliares é equivalente a empacotar a versão não JSON em `$util.toJson(...)`. Por exemplo, as seguintes instruções são exatamente as mesmas:

```
$util.toStringJson("Hello, World!")
$util.toJson($util.toString("Hello, World!"))
```

toDynamoDB

Lista de utilitários do `toDynamoDB`

`$util.dynamodb.toDynamoDB(Object) : Map`

Ferramenta de conversão de objetos gerais do DynamoDB que converte objetos de entrada para a representação adequada do DynamoDB. Ela tem opiniões fortes sobre como representa alguns tipos: por exemplo, usará listas ("L") em vez de conjuntos ("SS", "NS", "BS"). Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

Exemplo de string

```
Input:      $util.dynamodb.toDynamoDB("foo")
```



```
Output:      { "S" : "foo" }
```

Exemplo de número

```
Input:      $util.dynamodb.toDynamoDB(12345)
Output:     { "N" : 12345 }
```

Exemplo de booleano

```
Input:      $util.dynamodb.toDynamoDB(true)
Output:     { "BOOL" : true }
```

Exemplo de lista

```
Input:      $util.dynamodb.toDynamoDB([ "foo", 123, { "bar" : "baz" } ])
Output:     {
      "L" : [
        { "S" : "foo" },
        { "N" : 123 },
        {
          "M" : {
            "bar" : { "S" : "baz" }
          }
        }
      ]
    }
```

Exemplo de mapa

```
Input:      $util.dynamodb.toDynamoDB({ "foo": "bar", "baz" : 1234, "beep":
[ "boop" ] })
Output:     {
      "M" : {
        "foo" : { "S" : "bar" },
        "baz" : { "N" : 1234 },
        "beep" : {
          "L" : [
            { "S" : "boop" }
          ]
        }
      }
    }
```

```
}
```

\$util.dynamodb.toDynamoDBJson(Object) : String

O mesmo que `$util.dynamodb.toDynamoDB(Object) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

Utilitários toString

Lista de utilitários toString

\$util.dynamodb.toString(String) : String

Converte uma string de entrada para o formato de string do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toString("foo")
Output:     { "S" : "foo" }
```

\$util.dynamodb.toStringJson(String) : Map

O mesmo que `$util.dynamodb.toString(String) : String`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

\$util.dynamodb.toStringSet(List<String>) : Map

Converte uma lista com Strings para o formato de conjunto de strings do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toStringSet([ "foo", "bar", "baz" ])
Output:     { "SS" : [ "foo", "bar", "baz" ] }
```

\$util.dynamodb.toStringSetJson(List<String>) : String

O mesmo que `$util.dynamodb.toStringSet(List<String>) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

Utilitários toNumber

Lista de utilitários toNumber

\$util.dynamodb.toNumber(Number) : Map

Converte um número para o formato de número do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toNumber(12345)
Output:     { "N" : 12345 }
```

\$util.dynamodb.toNumberJson(Number) : String

O mesmo que `$util.dynamodb.toNumber(Number) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

\$util.dynamodb.toNumberSet(List<Number>) : Map

Converte uma lista de números para o formato de conjunto de números do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toNumberSet([ 1, 23, 4.56 ])
Output:     { "NS" : [ 1, 23, 4.56 ] }
```

\$util.dynamodb.toNumberSetJson(List<Number>) : String

O mesmo que `$util.dynamodb.toNumberSet(List<Number>) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

Utilitários toBinary

Lista de utilitários toBinary

\$util.dynamodb.toBinary(String) : Map

Converte dados binários codificados como uma string em base64 para o formato binário do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toBinary("foo")
```

```
Output:      { "B" : "foo" }
```

\$util.dynamodb.toBinaryJson(String) : String

O mesmo que `$util.dynamodb.toBinary(String) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

\$util.dynamodb.toBinarySet(List<String>) : Map

Converte uma lista de dados binários codificados como strings em base64 para o formato de conjunto de binários do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toBinarySet([ "foo", "bar", "baz" ])
Output:     { "BS" : [ "foo", "bar", "baz" ] }
```

\$util.dynamodb.toBinarySetJson(List<String>) : String

O mesmo que `$util.dynamodb.toBinarySet(List<String>) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

Utilitários toBoolean

Lista de utilitários toBoolean

\$util.dynamodb.toBoolean(Boolean) : Map

Converte um booleano para o formato booleano adequado do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toBoolean(true)
Output:     { "BOOL" : true }
```

\$util.dynamodb.toBooleanJson(Boolean) : String

O mesmo que `$util.dynamodb.toBoolean(Boolean) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

Utilitários toNull

Lista de utilitários toNull

\$util.dynamodb.toNull() : Map

Retorna um valor nulo no formato nulo do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toNull()
Output:     { "NULL" : null }
```

\$util.dynamodb.toNullJson() : String

O mesmo que `$util.dynamodb.toNull() : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

Utilitários toList

Lista de utilitários toList

\$util.dynamodb.toList(List) : Map

Converte uma lista de objetos no formato de lista do DynamoDB. Cada item na lista também é convertido para o formato adequado do DynamoDB. Ela tem opiniões fortes sobre como representa alguns dos objetos aninhados: por exemplo, usará listas ("L") em vez de conjuntos ("SS", "NS", "BS"). Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toList([ "foo", 123, { "bar" : "baz" } ])
Output:     {
      "L" : [
        { "S" : "foo" },
        { "N" : 123 },
        {
          "M" : {
            "bar" : { "S" : "baz" }
          }
        }
      ]
    }
```

\$util.dynamodb.toListJson(List) : String

O mesmo que `$util.dynamodb.toList(List) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

Utilitários toMap

Lista de utilitários toMap

\$util.dynamodb.toMap(Map) : Map

Converte um mapa para o formato de mapa do DynamoDB. Cada valor no mapa também é convertido para o formato adequado do DynamoDB. Ela tem opiniões fortes sobre como representa alguns dos objetos aninhados: por exemplo, usará listas ("L") em vez de conjuntos ("SS", "NS", "BS"). Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toMap({ "foo": "bar", "baz" : 1234, "beep": [ "boop" ] })
Output:     {
              "M" : {
                "foo" : { "S" : "bar" },
                "baz" : { "N" : 1234 },
                "beep" : {
                  "L" : [
                    { "S" : "boop" }
                  ]
                }
              }
            }
```

\$util.dynamodb.toMapJson(Map) : String

O mesmo que `$util.dynamodb.toMap(Map) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

\$util.dynamodb.toMapValues(Map) : Map

Cria uma cópia do mapa onde cada valor foi convertido para o formato adequado do DynamoDB. Ela tem opiniões fortes sobre como representa alguns dos objetos aninhados: por exemplo, usará listas ("L") em vez de conjuntos ("SS", "NS", "BS").

```
Input:      $util.dynamodb.toMapValues({ "foo": "bar", "baz" : 1234, "beep":
              [ "boop" ] })
```

```
Output:  {
    "foo" : { "S" : "bar" },
    "baz" : { "N" : 1234 },
    "beep" : {
        "L" : [
            { "S" : "boop" }
        ]
    }
}
```

Note

Observação: isso é um pouco diferente de `$util.dynamodb.toMap(Map) : Map`, uma vez que retorna somente o conteúdo do valor do atributo do DynamoDB, mas não todo o valor do atributo em si. Por exemplo, as seguintes instruções são exatamente as mesmas:

```
$util.dynamodb.toMapValues($map)
$util.dynamodb.toMap($map).get("M")
```

`$util.dynamodb.toMapValuesJson(Map) : String`

O mesmo que `$util.dynamodb.toMapValues(Map) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

Utilitários S3Object

Lista de utilitários do S3Object

`$util.dynamodb.toS3Object(String key, String bucket, String region) : Map`

Converte a chave, bucket e região na representação de Objeto do S3 do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toS3Object("foo", "bar", region = "baz")
Output:     { "S" : "{ \"s3\" : { \"key\" : \"foo\", \"bucket\" : \"bar\", \"region\" : \"baz\" } }" }
```

\$util.dynamodb.toS3ObjectJson(String key, String bucket, String region) : String

O mesmo que `$util.dynamodb.toS3Object(String key, String bucket, String region) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

\$util.dynamodb.toS3Object(String key, String bucket, String region, String version) : Map

Converte a chave, bucket, região e versão opcional na representação de Objeto do S3 do DynamoDB. Isso retorna um objeto que descreve o valor do atributo do DynamoDB.

```
Input:      $util.dynamodb.toS3Object("foo", "bar", "baz", "beep")
Output:     { "S" : "{ \"s3\" : { \"key\" : \"foo\", \"bucket\" : \"bar\", \"region\" : \"baz\", \"version\" = \"beep\" } }" }
```

\$util.dynamodb.toS3ObjectJson(String key, String bucket, String region, String version) : String

O mesmo que `$util.dynamodb.toS3Object(String key, String bucket, String region, String version) : Map`, mas retorna o valor do atributo do DynamoDB como uma string codificada JSON.

\$util.dynamodb.fromS3ObjectJson(String) : Map

Aceita o valor de string de um Objeto do S3 do DynamoDB e retorna um mapa que contém a chave, bucket, região e versão opcional.

```
Input:      $util.dynamodb.fromS3ObjectJson({ "S" : "{ \"s3\" : { \"key\" : \"foo\", \"bucket\" : \"bar\", \"region\" : \"baz\", \"version\" = \"beep\" } }" })
Output:     { "key" : "foo", "bucket" : "bar", "region" : "baz", "version" : "beep" }
```

Auxiliares do Amazon RDS em \$util.rds

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

`$util.rds` contém métodos auxiliares que formatam operações do Amazon RDS eliminando dados estranhos nas saídas de resultados.

Lista de utilitários `$util.rds`

`$util.rds.toJsonString(String serializedSQLResult): String`

Retorna uma `String` transformando o formato de resultado da operação da API de dados do Amazon Relational Database Service (Amazon RDS) bruto e transformado em string em uma string mais concisa. A string retornada é uma lista serializada de registros SQL do conjunto de resultados. Cada registro é representado como uma coleção de pares de chave/valor. As chaves são os nomes das colunas correspondentes.

Se a instrução correspondente na entrada for uma consulta SQL que causa uma mutação (por exemplo, `INSERT`, `UPDATE`, `DELETE`), uma lista vazia é retornada. Por exemplo, a consulta `select * from Books limit 2` fornece o resultado bruto da operação de dados do Amazon RDS:

```
{
  "sqlStatementResults": [
    {
      "numberOfRecordsUpdated": 0,
      "records": [
        [
          {
            "stringValue": "Mark Twain"
          },
          {
            "stringValue": "Adventures of Huckleberry Finn"
          },
          {
            "stringValue": "978-1948132817"
          }
        ],
        [
          {
            "stringValue": "Jack London"
          },
          {
            "stringValue": "The Call of the Wild"
          }
        ]
      ]
    }
  ]
}
```

```

        "stringValue": "978-1948132275"
      }
    ]
  ],
  "columnMetadata": [
    {
      "isSigned": false,
      "isCurrency": false,
      "label": "author",
      "precision": 200,
      "typeName": "VARCHAR",
      "scale": 0,
      "isAutoIncrement": false,
      "isCaseSensitive": false,
      "schemaName": "",
      "tableName": "Books",
      "type": 12,
      "nullable": 0,
      "arrayBaseColumnType": 0,
      "name": "author"
    },
    {
      "isSigned": false,
      "isCurrency": false,
      "label": "title",
      "precision": 200,
      "typeName": "VARCHAR",
      "scale": 0,
      "isAutoIncrement": false,
      "isCaseSensitive": false,
      "schemaName": "",
      "tableName": "Books",
      "type": 12,
      "nullable": 0,
      "arrayBaseColumnType": 0,
      "name": "title"
    },
    {
      "isSigned": false,
      "isCurrency": false,
      "label": "ISBN-13",
      "precision": 15,
      "typeName": "VARCHAR",
      "scale": 0,

```

```

        "isAutoIncrement": false,
        "isCaseSensitive": false,
        "schemaName": "",
        "tableName": "Books",
        "type": 12,
        "nullable": 0,
        "arrayBaseColumnType": 0,
        "name": "ISBN-13"
      }
    ]
  }
}

```

O `util.rds.toJsonString` é:

```

[
  {
    "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`

É o mesmo que `util.rds.toJsonString`, mas com o resultado sendo um JSON Object.

Auxiliares HTTP em `$util.http`

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias [aqui](#).

O utilitário `$util.http` fornece métodos auxiliares que podem ser usados para gerenciar parâmetros de solicitação HTTP e adicionar cabeçalhos de resposta.

Lista de utilitários `$util.http`

`$util.http.copyHeaders(Map) : Map`

Copia os cabeçalhos do mapa, excluindo os seguintes cabeçalhos HTTP restritos:

- `transfer-encoding`
- `connection`
- `host`
- `expect`
- `keep-alive`
- `upgrade`
- `proxy-authenticate`
- `proxy-authorization`
- `te`
- `content-length`

Você pode usar esse utilitário para encaminhar cabeçalhos de solicitação para seu endpoint HTTP downstream.

```
{
  ...
  "params": {
    ...
    "headers": $util.http.copyHeaders($ctx.request.headers),
    ...
  },
  ...
}
```

`$util.http.addResponseHeader(Cadeia de caracteres, objeto)`

Adiciona um único cabeçalho personalizado com o nome (`String`) e o valor (`Object`) da resposta. As limitações a seguir se aplicam a:

- Além da lista de cabeçalhos restritos para `copyHeaders(Map)`, os nomes dos cabeçalhos não podem corresponder a nenhuma das seguintes opções:

- 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
- Os nomes dos cabeçalhos não podem começar com os prefixos restritos `x-amzn-` ou `amazn-`.
 - O tamanho dos cabeçalhos de resposta personalizados não pode exceder 4 KB. Isso inclui nomes e valores de cabeçalho.
 - Você deve definir cada cabeçalho de resposta uma vez por operação do GraphQL. No entanto, se você definir um cabeçalho personalizado com o mesmo nome várias vezes, a definição mais recente aparecerá na resposta. Todos os cabeçalhos são contabilizados para o limite de tamanho do cabeçalho, independentemente do nome.
 - Cabeçalhos com um nome vazio (`String`) ou restrito ou um valor nulo (`Object`) serão ignorados e gerarão um `ResponseHeaderError` erro que será adicionado à saída da `errors` operação.

```
export function request(ctx) {
  util.http.addResponseHeader('itemsCount', 7)
  util.http.addResponseHeader('render', ctx.args.render)
  return {}
}
```

`$util.http.addResponseHeaders(Map)`

Adiciona vários cabeçalhos de resposta à resposta do mapa especificado de nomes (`String`) e valores (`Object`). As mesmas limitações listadas para o método `addResponseHeader(String, Object)` também se aplicam a esse método.

```
export function request(ctx) {
  const headers = {
    headerInt: 12,
    headerString: 'stringValue',
    headerObject: {
```

```
        field1: 7,
        field2: 'string'
      }
    }
    util.http.addResponseHeaders(headers)
    return {}
  }
}
```

Auxiliares XML em \$util.xml

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias [aqui](#).

`$util.xml` contém métodos auxiliares que podem facilitar a tradução de respostas XML para JSON ou um dicionário.

Lista de utilitários \$util.xml

`$util.xml.toMap(String)` : Map

Converte uma string XML para um dicionário.

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

Converte uma string XML para uma string JSON. Isso é semelhante a `toMap`, exceto que a saída é uma string. Isso é útil se quiser converter e retornar diretamente a resposta XML de um objeto HTTP para JSON.

\$util.xml.toJsonString(String, Boolean) : String

Converte uma string XML para uma string JSON com um parâmetro booliano opcional para determinar se deseja codificar o JSON como string.

Auxiliares de transformação em \$util.transform

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias [aqui](#).

\$util.transform contém métodos auxiliares que facilitam a execução de operações complexas em fontes de dados, como operações de filtragem do Amazon DynamoDB.

Auxiliares de transformação

Lista de utilitários auxiliares de transformação

\$util.transform.toDynamoDBFilterExpression(Map) : Map

Converte uma string de entrada em uma expressão de filtragem para o uso com o 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

Converte a entrada fornecida em sua expressão DSL de OpenSearch consulta equivalente, retornando-a como uma string JSON.

Input:

```
$util.transform.toElasticsearchQueryDSL({
  "upvotes":{
    "ne":15,
    "range":[
      10,
      20
    ]
  },
  "title":{
    "eq":"hihihi",
    "wildcard":"h*i"
  }
})
```

Output:

```
{
  "bool":{
    "must":[
      {
        "bool":{
          "must":[
            {
              "bool":{
                "must_not":{
                  "term":{
                    "upvotes":15
                  }
                }
              }
            }
          ],
          "range":{
            "upvotes":{
```

```

        "gte":10,
        "lte":20
      }
    }
  ]
},
{
  "bool":{
    "must":[
      {
        "term":{
          "title":"hihihi"
        }
      },
      {
        "wildcard":{
          "title":"h*i"
        }
      }
    ]
  }
}
]
}
}

```

Presume-se que o operador padrão seja AND.

Filtros de assinatura de auxiliares de transformação

Lista de utilitários de filtros de assinatura de auxiliares de transformação

\$util.transform.toSubscriptionFilter(Map) : Map

Converte um objeto de entrada Map em um objeto de expressão SubscriptionFilter. O método `$util.transform.toSubscriptionFilter` é usado como entrada para a extensão `$extensions.setSubscriptionFilter()`. Para obter mais informações, consulte [Extensões](#).

`$util.transform.toSubscriptionFilter(Map, List) : Map`

Converte um objeto de entrada Map em um objeto de expressão SubscriptionFilter. O método `$util.transform.toSubscriptionFilter` é usado como entrada para a extensão `$extensions.setSubscriptionFilter()`. Para obter mais informações, consulte [Extensões](#).

O primeiro argumento é o objeto de entrada Map que é convertido no objeto de expressão SubscriptionFilter. O segundo argumento é uma List de nomes de campos que são ignorados no primeiro objeto de entrada Map durante a criação da estrutura do objeto de expressão SubscriptionFilter.

`$util.transform.toSubscriptionFilter(Map, List, Map) : Map`

Converte um objeto de entrada Map em um objeto de expressão SubscriptionFilter. O método `$util.transform.toSubscriptionFilter` é usado como entrada para a extensão `$extensions.setSubscriptionFilter()`. Para obter mais informações, consulte [Extensões](#).

O primeiro argumento é o objeto de entrada Map que é convertido no objeto de expressão SubscriptionFilter, o segundo argumento é uma List dos nomes de campos que serão ignorados no primeiro objeto de entrada Map e o terceiro argumento é um objeto de entrada Map de regras estritas que são incluídas durante a criação da estrutura do objeto de expressão SubscriptionFilter. Essas regras estritas são incluídas no objeto de expressão SubscriptionFilter de forma que pelo menos uma das regras seja satisfeita para passar pelo filtro de assinatura.

Argumentos do filtro de assinatura

A tabela a seguir explica como os argumentos dos seguintes utilitários são definidos:

- `$util.transform.toSubscriptionFilter(Map) : Map`
- `$util.transform.toSubscriptionFilter(Map, List) : Map`
- `$util.transform.toSubscriptionFilter(Map, List, Map) : Map`

Argument 1: Map

O argumento 1 é um objeto Map com os seguintes valores-chave:

- nomes de campos
- "and"
- "or"

Para nomes de campos como chaves, as condições nas entradas desses campos estão no formato "operator" : "value".

O exemplo a seguir mostra como entradas podem ser adicionadas ao Map:

```
"field_name" : {  
    "operator1" : value  
}  
  
## We can have multiple conditions for the same field_name:  
  
"field_name" : {  
    "operator1" : value  
    "operator2" : value  
    .  
    .  
    .  
}
```

Quando um campo contém duas ou mais condições, todas essas condições são consideradas para usar a operação OR.

A entrada Map também pode ter "and" e "or" como chaves, o que implica que todas as entradas dentro dessas devem ser unidas usando a lógica AND ou OR dependendo da chave. Os valores-chave "and" e "or" esperam uma série de condições.

```
"and" : [  
    {  
        "field_name1" : {  
            "operator1" : value  
        }  
    },  
    {  
        "field_name2" : {  
            "operator1" : value  
        }  
    }  
]
```

```

    }
  },
  :
  .
].

```

Observe que você pode aninhar "and" e "or". Ou seja, você pode ter aninhado "and"/"or" em outro bloco "and"/"or". No entanto, isso não funciona em campos simples.

```

"and" : [
  {
    "field_name1" : {
      "operator" : value
    }
  },
  {
    "or" : [
      {
        "field_name2" : {
          "operator" : value
        }
      },
      {
        "field_name3" : {
          "operator" : value
        }
      }
    ]
  }
].

```

O exemplo a seguir mostra uma entrada do argumento 1 usando `$util.transform.toSubscriptionFilter(Map) : Map`.

Entrada(s)

Argumento 1: mapa:

```

{
  "percentageUp": {
    "lte": 50,

```

```
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 2000
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

Saída

O resultado é um objeto Map:

```
{
  "filterGroup": [
    {
      "filters": [
        {
          "fieldName": "percentageUp",
          "operator": "lte",
          "value": 50
        },
        {
          "fieldName": "title",
          "operator": "ne",

```

```
      "value": "Book1"
    },
    {
      "fieldName": "downvotes",
      "operator": "gt",
      "value": 2000
    },
    {
      "fieldName": "author",
      "operator": "eq",
      "value": "Admin"
    }
  ]
},
{
  "filters": [
    {
      "fieldName": "percentageUp",
      "operator": "lte",
      "value": 50
    },
    {
      "fieldName": "title",
      "operator": "ne",
      "value": "Book1"
    },
    {
      "fieldName": "downvotes",
      "operator": "gt",
      "value": 2000
    },
    {
      "fieldName": "isPublished",
      "operator": "eq",
      "value": false
    }
  ]
},
{
  "filters": [
    {
      "fieldName": "percentageUp",
      "operator": "gte",
      "value": 20
```

```
    },
    {
      "fieldName": "title",
      "operator": "ne",
      "value": "Book1"
    },
    {
      "fieldName": "downvotes",
      "operator": "gt",
      "value": 2000
    },
    {
      "fieldName": "author",
      "operator": "eq",
      "value": "Admin"
    }
  ]
},
{
  "filters": [
    {
      "fieldName": "percentageUp",
      "operator": "gte",
      "value": 20
    },
    {
      "fieldName": "title",
      "operator": "ne",
      "value": "Book1"
    },
    {
      "fieldName": "downvotes",
      "operator": "gt",
      "value": 2000
    },
    {
      "fieldName": "isPublished",
      "operator": "eq",
      "value": false
    }
  ]
}
]
```



```
}
```

Argument 2: List

O argumento 2 contém uma `List` de nomes de campos que não devem ser considerados na entrada `Map` (argumento 1) durante a criação da estrutura do objeto de expressão `SubscriptionFilter`. A `List` também pode estar vazia.

O exemplo a seguir mostra uma entrada do argumento 1 e argumento 2 usando `$util.transform.toSubscriptionFilter(Map, List) : Map`.

Entrada(s)

Argumento 1: mapa:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "gt": 20
      }
    }
  ],
  "or": [
    {
      "author": {
        "eq": "Admin"
      }
    },
    {
      "isPublished": {
        "eq": false
      }
    }
  ]
}
```

```
}  
]  
}
```

Argumento 2: lista:

```
["percentageUp", "author"]
```

Saída

O resultado é um objeto Map:

```
{  
  "filterGroup": [  
    {  
      "filters": [  
        {  
          "fieldName": "title",  
          "operator": "ne",  
          "value": "Book1"  
        },  
        {  
          "fieldName": "downvotes",  
          "operator": "gt",  
          "value": 20  
        },  
        {  
          "fieldName": "isPublished",  
          "operator": "eq",  
          "value": false  
        }  
      ]  
    }  
  ]  
}
```

Argument 3: Map

O argumento 3 é um objeto Map que tem nomes de campo como valores-chave (não pode ter "and" ou "or"). Para nomes de campos como chaves, as condições nesses campos são entradas no formato "operator" : "value". Diferentemente do argumento 1, o argumento 3 não pode

ter várias condições na mesma chave. Além disso, o argumento 3 não possui uma cláusula "and" ou "or"; portanto, também não há aninhamento envolvido.

O argumento 3 representa uma lista de regras estritas, que são adicionadas ao objeto de expressão `SubscriptionFilter` para que pelo menos uma dessas condições seja atendida para passar pelo filtro.

```
{
  "fieldname1": {
    "operator": value
  },
  "fieldname2": {
    "operator": value
  }
}
.
.
.
```

O exemplo a seguir mostra as entradas de argumento 1, argumento 2 e argumento 3 usando `$util.transform.toSubscriptionFilter(Map, List, Map) : Map`.

Entrada(s)

Argumento 1: mapa:

```
{
  "percentageUp": {
    "lte": 50,
    "gte": 20
  },
  "and": [
    {
      "title": {
        "ne": "Book1"
      }
    },
    {
      "downvotes": {
        "lt": 20
      }
    }
  ],
}
```

```
"or": [  
  {  
    "author": {  
      "eq": "Admin"  
    }  
  },  
  {  
    "isPublished": {  
      "eq": false  
    }  
  }  
]  
}
```

Argumento 2: lista:

```
["percentageUp", "author"]
```

Argumento 3: mapa:

```
{  
  "upvotes": {  
    "gte": 250  
  },  
  "author": {  
    "eq": "Person1"  
  }  
}
```

Saída

O resultado é um objeto Map:

```
{  
  "filterGroup": [  
    {  
      "filters": [  
        {  
          "fieldName": "title",  
          "operator": "ne",  
          "value": "Book1"  
        },  
        {
```

```
        "fieldName": "downvotes",
        "operator": "gt",
        "value": 20
      },
      {
        "fieldName": "isPublished",
        "operator": "eq",
        "value": false
      },
      {
        "fieldName": "upvotes",
        "operator": "gte",
        "value": 250
      }
    ]
  },
  {
    "filters": [
      {
        "fieldName": "title",
        "operator": "ne",
        "value": "Book1"
      },
      {
        "fieldName": "downvotes",
        "operator": "gt",
        "value": 20
      },
      {
        "fieldName": "isPublished",
        "operator": "eq",
        "value": false
      },
      {
        "fieldName": "author",
        "operator": "eq",
        "value": "Person1"
      }
    ]
  }
]
```

Auxiliares de matemática em \$util.math

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias [aqui](#).

\$util.math contém métodos para ajudar com operações matemáticas comuns.

Lista de utilitários \$util.math

\$util.math.roundNum(Double) : Integer

Pega um duplo e arredonda para o número inteiro mais próximo.

\$util.math.minVal(Double, Double) : Double

Pega dois duplos e retorna o valor mínimo entre as duas duplas.

\$util.math.maxVal(Double, Double) : Double

Pega dois duplos e retorna o valor máximo entre as duas duplas.

\$util.math.randomDouble() : Double

Retorna um duplo aleatório entre 0 e 1.

Important

Essa função não deve ser usada para nada que precise de randomização de alta entropia (por exemplo, criptografia).

\$util.math.randomWithinRange(Integer, Integer) : Integer

Retorna um valor inteiro aleatório dentro do intervalo especificado, com o primeiro argumento especificando o valor inferior do intervalo e o segundo argumento especificando o valor superior do intervalo.

⚠ Important

Essa função não deve ser usada para nada que precise de randomização de alta entropia (por exemplo, criptografia).

Auxiliares de string em \$util.str

📌 Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias [aqui](#).

`$util.str` contém métodos para ajudar com operações comuns de string.

Lista de utilitários \$util.str

`$util.str.toUpper(String) : String`

Pega uma string e a converte em maiúscula.

`$util.str.toLower(String) : String`

Pega uma string e a converte em maiúscula.

`$util.str.replace(String, String, String) : String`

Substitui uma substring dentro de uma string por outra string. O primeiro argumento especifica a string na qual realizar a operação de substituição. O segundo argumento especifica a substring a ser substituída. O terceiro argumento especifica a string com a qual substituir o segundo argumento. Veja a seguir um exemplo do uso desse utilitário:

```
INPUT:      $util.str.replace("hello world", "hello", "mellow")
OUTPUT:     "mellow world"
```

\$util.str.normalize(String, String) : String

Normaliza uma string usando uma das quatro formas de normalização unicode: NFC, NFD, NFKC ou NFKD. O primeiro argumento é a string a ser normalizada. O segundo argumento é “nfc”, “nfd”, “nfkc” ou “nfkd”, especificando o tipo de normalização a ser usado no processo de normalização.

Extensões

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias [aqui](#).

\$extensions contém um conjunto de métodos para realizar ações adicionais em seus resolvedores.

Extensões do cache

\$extensions.evictFromApiCache(String, String, Object) : Object

Elimina um item do cache do lado do AWS AppSync servidor. O primeiro argumento é o nome do tipo. O segundo argumento é o nome do campo. O terceiro argumento é um objeto contendo itens do par de chave/valor que especificam o valor da chave de armazenamento em cache. Você deve colocar os itens no objeto na mesma ordem das chaves de cache em cachingKey do resolvedor em cache.

Note

Esse utilitário funciona somente para mutações, não para consultas.

Extensões de assinatura

\$extensions.setSubscriptionFilter(filterJsonObject)

Define filtros de assinatura aprimorados. Cada evento de notificação de assinatura é avaliado em relação aos filtros de assinatura fornecidos e envia notificações aos clientes se todos os filtros

forem avaliados como `true`. O argumento é `filterJsonObject` conforme descrito na seção a seguir.

 Note

Você pode usar esse método de extensão somente nos modelos de mapeamento de resposta de um resolvidor de assinatura.

`$extensions.setSubscriptionInvalidationFilter(filterJsonObject)`

Define os filtros de invalidação da assinatura. Os filtros de assinatura são avaliados em relação à carga de invalidação e, em seguida, invalidam determinada assinatura se os filtros forem avaliados como `true`. O argumento é `filterJsonObject` conforme descrito na seção a seguir.

 Note

Você pode usar esse método de extensão somente nos modelos de mapeamento de resposta de um resolvidor de assinatura.

`$extensions.invalidateSubscriptions(invalidationJsonObject)`

Usado para iniciar uma invalidação de assinatura a partir de uma mutação. O argumento é `invalidationJsonObject` conforme descrito na seção a seguir.

 Note

Essa extensão pode ser usada somente nos modelos de mapeamento de resposta dos resolvidores de mutação.

Você só pode usar no máximo cinco chamadas de método

`$extensions.invalidateSubscriptions()` exclusivas em uma única solicitação.

Se você exceder esse limite, receberá um erro do GraphQL.

Argumento: `filterJsonObject`

O objeto JSON define filtros de assinatura ou de invalidação. É uma série de filtros em um `filterGroup`. Cada filtro é uma coleção de filtros individuais.

```
{
  "filterGroup": [
    {
      "filters" : [
        {
          "fieldName" : "userId",
          "operator" : "eq",
          "value" : 1
        }
      ]
    },
    {
      "filters" : [
        {
          "fieldName" : "group",
          "operator" : "in",
          "value" : ["Admin", "Developer"]
        }
      ]
    }
  ]
}
```

Cada filtro tem três atributos:

- **fieldName** – O campo do esquema GraphQL.
- **operator** – O tipo de operador.
- **value** – Os valores a serem comparados com o valor **fieldName** da notificação de assinatura.

Veja a seguir um exemplo de atribuição desses atributos:

```
{
  "fieldName" : "severity",
  "operator" : "le",
  "value" : $context.result.severity
}
```

Campo: fieldName

O tipo de string `fieldName` refere-se a um campo definido no esquema GraphQL que corresponde a `fieldName` na carga de notificação de assinatura. Quando uma correspondência é encontrada, o `value` do campo do esquema GraphQL é comparado a `value` do filtro de notificação de assinatura. No exemplo a seguir, o filtro `fieldName` corresponde ao campo `service` definido em determinado tipo de GraphQL. Se a carga de notificação contiver um campo `service` com um `value` equivalente a `AWS AppSync`, o filtro será avaliado como `true`:

```
{
  "fieldName" : "service",
  "operator" : "eq",
  "value" : "AWS AppSync"
}
```

Campo: valor

O valor pode ser de um tipo diferente com base no operador:

- Um único número ou booleano
 - Exemplo de string: `"test"`, `"service"`
 - Exemplo de número: `1`, `2`, `45.75`
 - Exemplo de booleano: `true`, `false`
- Pares de números ou strings
 - Exemplo de par de strings: `["test1", "test2"]`, `["start", "end"]`
 - Exemplo de par de números: `[1, 4]`, `[67, 89]`, `[12.45, 95.45]`
- Matrizes de números ou strings
 - Exemplo de matriz de strings: `["test1", "test2", "test3", "test4", "test5"]`
 - Exemplo de matriz numérica: `[1, 2, 3, 4, 5]`, `[12.11, 46.13, 45.09, 12.54, 13.89]`

Campo: operador

Uma string que diferencia maiúsculas de minúsculas com os seguintes valores possíveis:

Operador	Descrição	Tipos de valores possíveis
Extensões		

eq	Equal	inteiro, flutuante, string, booleano
um	Not equal	inteiro, flutuante, string, booleano
le	Menor ou igual a	inteiro, flutuante, string
lt	Menor que	inteiro, flutuante, string
idade	Maior ou igual a	inteiro, flutuante, string
gt	Maior que	inteiro, flutuante, string
contém	Verifica se há uma subsequência ou valor no conjunto.	inteiro, flutuante, string
NÃO contém	Verifica a ausência de uma subsequência ou ausência de um valor no conjunto.	inteiro, flutuante, string
Começa com	Verifica se há um prefixo.	string
in	Verifica os elementos correspondentes que estão na lista.	Matriz de números inteiros, flutuantes ou strings
notIn	Verifica se há elementos correspondentes que não estão na lista.	Matriz de números inteiros, flutuantes ou strings
entre	Entre dois valores	inteiro, flutuante, string
Contém qualquer	Contém elementos comuns	inteiro, flutuante, string

A tabela a seguir descreve como cada operador é usado na notificação de assinatura.

eq (equal)

O operador `eq` avalia como `true` se o valor do campo de notificação de assinatura corresponde e é estritamente igual ao valor do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura tem um campo `service` com o valor equivalente a `AWS AppSync`.

Tipos de valores possíveis: inteiro, flutuante, string, booleano

```
{
  "fieldName" : "service",
  "operator" : "eq",
  "value" : "AWS AppSync"
}
```

ne (not equal)

O operador `ne` avalia como `true` se o valor do campo de notificação de assinatura for diferente do valor do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura tem um campo `service` com valor diferente de `AWS AppSync`.

Tipos de valores possíveis: inteiro, flutuante, string, booleano

```
{
  "fieldName" : "service",
  "operator" : "ne",
  "value" : "AWS AppSync"
}
```

le (less or equal)

O operador `le` avalia como `true` se o valor do campo de notificação de assinatura é menor ou igual ao valor do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura tem um campo `size` com valor menor ou igual a 5.

Tipos de valores possíveis: inteiro, flutuante, string

```
{
  "fieldName" : "size",
  "operator" : "le",
  "value" : 5
}
```

```
}
```

lt (less than)

O operador `lt` avalia como `true` se o valor do campo de notificação de assinatura for menor que o valor do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura tem um campo `size` com valor inferior a 5.

Tipos de valores possíveis: inteiro, flutuante, string

```
{
  "fieldName" : "size",
  "operator" : "lt",
  "value" : 5
}
```

ge (greater or equal)

O operador `ge` avalia como `true` se o valor do campo de notificação de assinatura é maior ou igual ao valor do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura tem um campo `size` com valor maior ou igual a 5.

Tipos de valores possíveis: inteiro, flutuante, string

```
{
  "fieldName" : "size",
  "operator" : "ge",
  "value" : 5
}
```

gt (greater than)

O operador `gt` avalia como `true` se o valor do campo de notificação de assinatura é maior que o valor do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura tem um campo `size` com valor maior que 5.

Tipos de valores possíveis: inteiro, flutuante, string

```
{
  "fieldName" : "size",
  "operator" : "gt",
}
```

```
"value" : 5
}
```

contains

O operador `contains` verifica uma substring, subsequência ou valor em um conjunto ou item único. Um filtro com o operador `contains` avalia como `true` se o valor do campo de notificação de assinatura contém o valor do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura possui um campo `seats` com o valor da matriz contendo o valor `10`.

Tipos de valores possíveis: inteiro, flutuante, string

```
{
  "fieldName" : "seats",
  "operator" : "contains",
  "value" : 10
}
```

Em outro exemplo, o filtro avalia como `true` se a notificação de assinatura possui um campo `event` com `launch` como substring.

```
{
  "fieldName" : "event",
  "operator" : "contains",
  "value" : "launch"
}
```

notContains

O operador `notContains` verifica a ausência de uma substring, subsequência ou valor em um conjunto ou item único. O filtro com o operador `notContains` avalia como `true` se o valor do campo de notificação de assinatura não contém o valor do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura possui um campo `seats` com o valor da matriz não contendo o valor `10`.

Tipos de valores possíveis: inteiro, flutuante, string

```
{
  "fieldName" : "seats",
  "operator" : "notContains",
}
```

```
"value" : 10
}
```

Em outro exemplo, o filtro avalia como `true` se a notificação de assinatura possui um valor de campo `event` sem `launch` como sua subsequência.

```
{
  "fieldName" : "event",
  "operator" : "notContains",
  "value" : "launch"
}
```

`beginsWith`

O operador `beginsWith` verifica se há um prefixo em uma string. O filtro que contém o operador `beginsWith` avalia como `true` se o valor do campo de notificação de assinatura começa com o valor do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura tem um campo `service` com um valor que começa com `AWS`.

Tipo de valor possível: string

```
{
  "fieldName" : "service",
  "operator" : "beginsWith",
  "value" : "AWS"
}
```

`in`

O operador `in` verifica os elementos correspondentes em uma matriz. Um filtro com o operador `in` avalia como `true` se o valor do campo de notificação de assinatura contém o valor do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura possui um campo `severity` com um dos valores presentes no matriz: `[1, 2, 3]`.

Tipo de valor possível: matriz de número inteiro, flutuante ou string

```
{
  "fieldName" : "severity",
  "operator" : "in",
  "value" : [1,2,3]
}
```



```
}
```

notIn

O operador `notIn` verifica se há elementos ausentes em uma matriz. O filtro que contém o operador `notIn` avalia como `true` se o valor do campo de notificação de assinatura não existe na matriz. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura possui um campo `severity` com um dos valores não presentes no matriz: `[1, 2, 3]`.

Tipo de valor possível: matriz de número inteiro, flutuante ou string

```
{
  "fieldName" : "severity",
  "operator" : "notIn",
  "value" : [1,2,3]
}
```

between

O operador `between` verifica valores entre dois números ou strings. O filtro que contém o operador `between` avalia como `true` se o valor do campo de notificação de assinatura está entre o par de valores do filtro. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura possui um campo `severity` com valores `2,3,4`.

Tipo de valor possível: par de número inteiro, flutuante ou string

```
{
  "fieldName" : "severity",
  "operator" : "between",
  "value" : [1,5]
}
```

containsAny

O operador `containsAny` verifica elementos comuns em matrizes. Um filtro com o operador `containsAny` avalia como `true` se a interseção do valor do conjunto de campos de notificação de assinatura e do valor do conjunto de filtros não está vazia. No exemplo a seguir, o filtro avalia como `true` se a notificação de assinatura tem um campo `seats` com um valor de matriz contendo `10` ou `15`. Isso significa que o filtro avaliará como `true` se a notificação de assinatura tivesse um valor de campo `seats` `[10,11]` ou `[15,20,30]`.

Tipos de valores possíveis: inteiro, flutuante ou string

```
{
  "fieldName" : "seats",
  "operator" : "containsAny",
  "value" : [10, 15]
}
```

Lógica AND

Você pode combinar vários filtros usando a lógica AND definindo várias entradas dentro do objeto `filters` na matriz `filterGroup`. No exemplo a seguir, os filtros avaliam como `true` se a notificação de assinatura tem um campo `userId` com um valor equivalente a 1 AND um valor de campo `group` de Admin ou Developer.

```
{
  "filterGroup": [
    {
      "filters" : [
        {
          "fieldName" : "userId",
          "operator" : "eq",
          "value" : 1
        },
        {
          "fieldName" : "group",
          "operator" : "in",
          "value" : ["Admin", "Developer"]
        }
      ]
    }
  ]
}
```

Lógica OR

Você pode combinar vários filtros usando a lógica OR definindo vários objetos de filtro na matriz `filterGroup`. No exemplo a seguir, os filtros avaliam como `true` se a notificação de assinatura tem um campo `userId` com um valor equivalente a 1 OR um valor de campo `group` de Admin ou Developer.

```
{
  "filterGroup": [
    {
      "filters" : [
        {
          "fieldName" : "userId",
          "operator" : "eq",
          "value" : 1
        }
      ]
    },
    {
      "filters" : [
        {
          "fieldName" : "group",
          "operator" : "in",
          "value" : ["Admin", "Developer"]
        }
      ]
    }
  ]
}
```

Exceções

Observe que existem várias restrições para o uso de filtros:

- No objeto `filters`, pode haver no máximo cinco itens `fieldName` exclusivos por filtro. Isso significa que você pode combinar no máximo cinco objetos individuais `fieldName` usando a lógica AND.
- Pode haver no máximo vinte valores para o operador `containsAny`.
- Pode haver no máximo cinco valores para os operadores `in` e `notIn`.
- Cada string pode ter no máximo 256 caracteres.
- Cada comparação de string diferencia maiúsculas e minúsculas.
- A filtragem de objetos aninhados permite até cinco níveis aninhados de filtragem.
- Cada `filterGroup` pode ter no máximo 10 `filters`. Isso significa que você pode combinar no máximo 10 `filters` usando a lógica OR.

- O operador `in` é um caso especial da lógica OR. No exemplo a seguir, existem dois `filters`:

```
{
  "filterGroup": [
    {
      "filters" : [
        {
          "fieldName" : "userId",
          "operator" : "eq",
          "value" : 1
        },
        {
          "fieldName" : "group",
          "operator" : "in",
          "value" : ["Admin", "Developer"]
        }
      ]
    }
  ]
}
```

O grupo de filtros anterior é avaliado da seguinte forma e conta para o limite máximo de filtros:

```
{
  "filterGroup": [
    {
      "filters" : [
        {
          "fieldName" : "userId",
          "operator" : "eq",
          "value" : 1
        },
        {
          "fieldName" : "group",
          "operator" : "eq",
          "value" : "Admin"
        }
      ]
    },
    {
      "filters" : [
        {
```

```
        "fieldName" : "userId",
        "operator" : "eq",
        "value" : 1
      },
      {
        "fieldName" : "group",
        "operator" : "eq",
        "value" : "Developer"
      }
    ]
  }
}
```

Argumento: invalidationJsonObject

O `invalidationJsonObject` define o seguinte:

- `subscriptionField` – A assinatura do esquema GraphQL a ser invalidada. Uma única assinatura, definida como uma string em `subscriptionField`, é considerada invalidada.
- `payload` – Uma lista de pares de valores-chave que é usada como entrada para invalidar assinaturas se o filtro de invalidação for avaliado como `true` em relação aos seus valores.

O exemplo a seguir invalida clientes inscritos e conectados usando a assinatura de `onUserDelete` quando o filtro de invalidação definido no resolvidor de assinatura é avaliado como `true` em relação ao valor `payload`.

```
$extensions.invalidateSubscriptions({
  "subscriptionField": "onUserDelete",
  "payload": {
    "group": "Developer"
    "type" : "Full-Time"
  }
})
```

AWS AppSync referência do modelo de mapeamento do resolvidor para o DynamoDB

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

A função AWS AppSync DynamoDB permite que você use o [GraphQL](#) para armazenar e recuperar dados em tabelas existentes do Amazon DynamoDB em sua conta mapeando uma solicitação de entrada do GraphQL em uma chamada do DynamoDB e, em seguida, mapeando a resposta do DynamoDB de volta para o GraphQL. Esta seção descreve os manipuladores de solicitação e resposta para as operações do DynamoDB compatíveis:

- [GetItem](#) - A GetItem solicitação permite que você instrua a função do DynamoDB a fazer uma GetItem solicitação ao DynamoDB e permite especificar a chave do item no DynamoDB e se deve usar uma leitura consistente ou não.
- [PutItem](#) - O documento de mapeamento de solicitações permite que você PutItem solicite à função do DynamoDB que faça uma PutItem solicitação ao DynamoDB e especifique a chave do item no DynamoDB, o conteúdo completo do item (composto por valores de chave e atributos) e condições para que a operação seja bem-sucedida.
- [UpdateItem](#) - A UpdateItem solicitação permite que você peça à função do DynamoDB que faça uma UpdateItem solicitação ao DynamoDB e especifique a chave do item no DynamoDB, uma expressão de atualização descrevendo como atualizar o item no DynamoDB e as condições para que a operação seja bem-sucedida.
- [DeleteItem](#) - A DeleteItem solicitação permite que você solicite à função do DynamoDB que faça uma DeleteItem solicitação ao DynamoDB e especifique a chave do item no DynamoDB e as condições para que a operação seja bem-sucedida.
- [Query](#): o objeto de solicitação Query permite orientar o resolvidor do DynamoDB a fazer uma solicitação Query ao DynamoDB, além de especificar a expressão de chave, qual índice usar, filtros adicionais, quantos itens retornar, se deve usar leituras consistentes, direção da consulta (para a frente ou para trás) e tokens de paginação.

- [Scan](#): a solicitação Scan permite orientar a função do DynamoDB a fazer uma solicitação Scan ao DynamoDB, além de especificar um filtro para excluir resultados, qual índice usar, quantos itens retornar, se deve usar leituras consistentes, tokens de paginação e verificações paralelas.
- [Sync](#): o objeto Sync permite recuperar todos os resultados de uma tabela do DynamoDB e, depois, receber apenas os dados alterados desde a última consulta (as atualizações delta). As solicitações Sync só podem ser feitas a fontes de dados versionadas do DynamoDB. Você pode especificar um filtro para excluir resultados, quantos itens devolver, tokens de paginação e quando sua última operação de sincronização foi iniciada.
- [BatchGetItem](#) - O objeto de BatchGetItem solicitação permite que você peça à função do DynamoDB que faça uma BatchGetItem solicitação ao DynamoDB para recuperar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar os nomes das tabelas das quais recuperar os itens e as chaves dos itens a serem recuperados de cada tabela.
- [BatchDeleteItem](#) - O objeto de BatchDeleteItem solicitação permite que você peça à função do DynamoDB que faça uma BatchWriteItem solicitação ao DynamoDB para excluir vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar os nomes das tabelas das quais excluir os itens e as chaves dos itens a serem excluídos de cada tabela.
- [BatchPutItem](#) - O objeto de BatchPutItem solicitação permite que você peça à função do DynamoDB que faça uma BatchWriteItem solicitação ao DynamoDB para colocar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar os nomes das tabelas nas quais colocar os itens e os itens completos a serem colocados em cada tabela.
- [TransactGetItems](#) - O objeto de TransactGetItems solicitação permite que você peça à função do DynamoDB que faça uma TransactGetItems solicitação ao DynamoDB para recuperar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar o nome da tabela de cada item de solicitação do qual recuperar o item e a chave de cada item a ser recuperado de cada tabela.
- [TransactWriteItems](#) - O objeto de TransactWriteItems solicitação permite que você peça à função do DynamoDB que faça uma TransactWriteItems solicitação ao DynamoDB para gravar vários itens, potencialmente em várias tabelas. Para esse objeto de solicitação, você deve especificar o nome da tabela de destino de cada item de solicitação, a operação de cada item a ser executada e a chave de cada item a ser gravada.
- [Sistema de tipos \(mapeamento de solicitações\)](#) - Saiba mais sobre como a digitação do DynamoDB é integrada às solicitações. AWS AppSync

- [Sistema de tipos \(mapeamento de respostas\)](#): saiba mais sobre como os tipos do DynamoDB são convertidos automaticamente em GraphQL ou JSON em uma carga útil de resposta.
- [Filtros](#): saiba mais sobre filtros para operações de consulta e verificação.
- [Expressões condicionais](#) - Saiba mais sobre expressões condicionais para PutItem UpdateItem, e DeleteItem operações.
- [Expressões de condição de transação](#) - Saiba mais sobre expressões de condição para TransactWriteItems operações.
- [Projeções](#): saiba mais sobre como especificar atributos em operações de leitura.

GetItem

O documento de mapeamento de GetItem solicitações permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma GetItem solicitação ao DynamoDB e especifique:

- A chave do item no DynamoDB
- Se deve usar uma leitura consistente ou não

O documento de mapeamento GetItem possui a seguinte estrutura:

```
{
  "version" : "2017-02-28",
  "operation" : "GetItem",
  "key" : {
    "foo" : ... typed value,
    "bar" : ... typed value
  },
  "consistentRead" : true,
  "projection" : {
    ...
  }
}
```

Os campos são definidos da seguinte forma:

GetItem campos

GetItem lista de campos

version

As versões de definição de modelo 2017-02-28 e 2018-05-29 são compatíveis atualmente. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação GetItem do DynamoDB, ela deve ser definida como GetItem. Este valor é obrigatório.

key

A chave do item no DynamoDB. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

consistentRead

Se deve ou não realizar uma leitura altamente consistente com o DynamoDB. Isso é opcional e usa como padrão false.

projection

Uma projeção usada para especificar os atributos a serem retornados da operação do DynamoDB. Para obter mais informações sobre projeções, consulte [Projeções](#). Esse campo é opcional.

O item retornado do DynamoDB é automaticamente convertido nos tipos primitivos GraphQL e JSON e está disponível no contexto de mapeamento (`$context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre os modelos de mapeamento da resposta, consulte [Visão geral do modelo de mapeamento do resolvedor](#).

Exemplo

Veja a seguir um modelo de mapeamento para uma consulta `getThing(foo: String!, bar: String!)` do GraphQL:

```
{
  "version" : "2017-02-28",
  "operation" : "GetItem",
  "key" : {
    "foo" : $util.dynamodb.toDynamoDBJson($ctx.args.foo),
    "bar" : $util.dynamodb.toDynamoDBJson($ctx.args.bar)
  },
  "consistentRead" : true
}
```

Para obter mais informações sobre a API `GetItem` do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

PutItem

O documento de mapeamento de `PutItem` solicitações permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma `PutItem` solicitação ao DynamoDB e especifique o seguinte:

- A chave do item no DynamoDB
- O conteúdo completo do item (composto por `key` e `attributeValues`)
- Condições para que a operação seja bem-sucedida

O documento de mapeamento `PutItem` possui a seguinte estrutura:

```
{
  "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
}
```

Os campos são definidos da seguinte forma:

PutItem campos

PutItem lista de campos

version

As versões de definição de modelo 2017-02-28 e 2018-05-29 são compatíveis atualmente. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação PutItem do DynamoDB, ela deve ser definida como PutItem. Este valor é obrigatório.

key

A chave do item no DynamoDB. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

attributeValues

O restante dos atributos do item a ser colocado no DynamoDB. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Esse campo é opcional.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, uma solicitação PutItem substitui qualquer entrada existente para esse item. Para obter mais informações sobre as condições, consulte [Expressões de condição](#). Este valor é opcional.

`_version`

Um valor numérico que representa a versão conhecida mais recente de um item. Este valor é opcional. Esse campo é usado para Detecção de conflitos e só é compatível com fontes de dados versionadas.

`customPartitionKey`

Quando ativado, esse valor de string modifica o formato dos `ds_pk` registros `ds_sk` e usados pela tabela de sincronização delta quando o controle de versão é ativado (para obter mais informações, consulte [Detecção e sincronização de conflitos no Guia](#) do AWS AppSync desenvolvedor). Quando ativado, o processamento da entrada `populateIndexFields` também é ativado. Esse campo é opcional.

`populateIndexFields`

Um valor booleano que, quando ativado com **`customPartitionKey`**, cria novas entradas para cada registro na tabela de sincronização delta, especificamente nas colunas `gsi_ds_pk` e `gsi_ds_sk`. Para obter mais informações, consulte [Detecção e sincronização de conflitos](#) no Guia do desenvolvedor do AWS AppSync . Esse campo é opcional.

O item gravado no DynamoDB é automaticamente convertido nos tipos primitivos GraphQL e JSON e está disponível no contexto de mapeamento (`$context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre os modelos de mapeamento da resposta, consulte [Visão geral do modelo de mapeamento do resolvedor](#).

Exemplo 1

Veja a seguir um modelo de mapeamento para uma mutação do GraphQL `updateThing(foo: String!, bar: String!, name: String!, version: Int!)`.

Se nenhum item com a chave especificada existir, ele será criado. Se já existir um item com a chave especificada, ele será substituído.

```
{
  "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)
}
}
```

Exemplo 2

Veja a seguir um modelo de mapeamento para uma mutação do GraphQL `updateThing(foo: String!, bar: String!, name: String!, expectedVersion: Int!)`.

Esse exemplo verifica se o item que está atualmente no DynamoDB tem o campo `version` definido como `expectedVersion`.

```
{
  "version" : "2017-02-28",
  "operation" : "PutItem",
  "key": {
    "foo" : $util.dynamodb.toDynamoDBJson($ctx.args.foo),
    "bar" : $util.dynamodb.toDynamoDBJson($ctx.args.bar)
  },
  "attributeValues" : {
    "name" : $util.dynamodb.toDynamoDBJson($ctx.args.name),
    #set( $newVersion = $context.arguments.expectedVersion + 1 )
    "version" : $util.dynamodb.toDynamoDBJson($newVersion)
  },
  "condition" : {
    "expression" : "version = :expectedVersion",
    "expressionValues" : {
      ":expectedVersion" : $util.dynamodb.toDynamoDBJson($expectedVersion)
    }
  }
}
```

Para obter mais informações sobre a API `PutItem` do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

UpdateItem

O documento de mapeamento de `UpdateItem` solicitações permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma `UpdateItem` solicitação ao DynamoDB e especifique o seguinte:

- A chave do item no DynamoDB
- Uma expressão de atualização que descreve como atualizar o item no DynamoDB
- Condições para que a operação seja bem-sucedida

O documento de mapeamento `UpdateItem` possui a seguinte estrutura:

```
{
  "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
}
```

Os campos são definidos da seguinte forma:

UpdateItem campos

UpdateItem lista de campos

version

As versões de definição de modelo 2017-02-28 e 2018-05-29 são compatíveis atualmente. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação UpdateItem do DynamoDB, ela deve ser definida como UpdateItem. Este valor é obrigatório.

key

A chave do item no DynamoDB. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(mapeamento da solicitação\)](#). Este valor é obrigatório.

update

A seção update permite especificar uma expressão de atualização que descreve como atualizar o item no DynamoDB. Para obter mais informações sobre como escrever expressões de atualização, consulte a documentação do [UpdateExpressions DynamoDB](#). Esta seção é obrigatória.

A seção update tem três componentes:

expression

A expressão de atualização. Este valor é obrigatório.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de nome usado em expression e o valor deve ser uma string que corresponde ao nome do atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados em expression.

expressionValues

As substituições para espaços reservados de valor do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de valor usado na

`expression` e o valor deve ser um valor digitado. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Isso deve ser especificado. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de valor do atributo da expressão usados em `expression`.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, a solicitação `UpdateItem` atualizará as entradas existentes independentemente do estado atual. Para obter mais informações sobre as condições, consulte [Expressões de condição](#). Este valor é opcional.

_version

Um valor numérico que representa a versão conhecida mais recente de um item. Este valor é opcional. Esse campo é usado para Detecção de conflitos e só é compatível com fontes de dados versionadas.

customPartitionKey

Quando ativado, esse valor de string modifica o formato dos `ds_pk` registros `ds_sk` e usados pela tabela de sincronização delta quando o controle de versão é ativado (para obter mais informações, consulte [Detecção e sincronização de conflitos no Guia](#) do AWS AppSync desenvolvedor). Quando ativado, o processamento da entrada `populateIndexFields` também é ativado. Esse campo é opcional.

populateIndexFields

Um valor booleano que, quando ativado com **customPartitionKey**, cria novas entradas para cada registro na tabela de sincronização delta, especificamente nas colunas `gsi_ds_pk` e `gsi_ds_sk`. Para obter mais informações, consulte [Detecção e sincronização de conflitos](#) no Guia do desenvolvedor do AWS AppSync . Esse campo é opcional.

O item atualizado no DynamoDB é automaticamente convertido nos tipos primitivos GraphQL e JSON e está disponível no contexto de mapeamento (`$context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre os modelos de mapeamento da resposta, consulte [Visão geral do modelo de mapeamento do resolvedor](#).

Exemplo 1

Veja a seguir um modelo de mapeamento para uma mutação do GraphQL `upvote(id: ID!)`.

Nesse exemplo, um item no DynamoDB tem seus campos `upvotes` e `version` incrementados por 1.

```
{
  "version" : "2017-02-28",
  "operation" : "UpdateItem",
  "key" : {
    "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
  },
  "update" : {
    "expression" : "ADD #voteField :plusOne, version :plusOne",
    "expressionNames" : {
      "#voteField" : "upvotes"
    },
    "expressionValues" : {
      ":plusOne" : { "N" : 1 }
    }
  }
}
```

Exemplo 2

Veja a seguir um modelo de mapeamento para uma mutação do GraphQL `updateItem(id: ID!, title: String, author: String, expectedVersion: Int!)`.

Esse é um exemplo complexo que inspeciona os argumentos e gera dinamicamente a expressão de atualização que inclui apenas os argumentos que foram fornecidos pelo cliente. Por exemplo, se `title` e `author` são omitidos, eles não são atualizados. Se um argumento for especificado, mas o seu valor for `null`, esse campo é excluído do objeto no DynamoDB. Finalmente, a operação tem uma condição, que verifica se o item que está atualmente no DynamoDB tem o campo `version` definido como `expectedVersion`:

```
{
  "version" : "2017-02-28",

  "operation" : "UpdateItem",

  "key" : {
```

```

    "id" : $util.dynamodb.toDynamoDBJson($ctx.args.id)
  },

  ## Set up some space to keep track of things we're updating **
  #set( $expNames = {} )
  #set( $expValues = {} )
  #set( $expSet = {} )
  #set( $expAdd = {} )
  #set( $expRemove = [] )

  ## Increment "version" by 1 **
  ${!expAdd.put("version", ":newVersion")}
  ${!expValues.put(":newVersion", { "N" : 1 })}

  ## Iterate through each argument, skipping "id" and "expectedVersion" **
  #foreach( $entry in $context.arguments.entrySet() )
    #if( $entry.key != "id" && $entry.key != "expectedVersion" )
      #if( (!$entry.value) && ("${!entry.value}" == "") )
        ## If the argument is set to "null", then remove that attribute from
the item in DynamoDB **

        #set( $discard = ${expRemove.add("#${entry.key}")} )
        ${!expNames.put("#${entry.key}", "${entry.key}")}
      #else
        ## Otherwise set (or update) the attribute on the item in DynamoDB **

        ${!expSet.put("#${entry.key}", ":${entry.key}")}
        ${!expNames.put("#${entry.key}", "${entry.key}")}

        #if( $entry.key == "ups" || $entry.key == "downs" )
          ${!expValues.put(":${entry.key}", { "N" : $entry.value })}
        #else
          ${!expValues.put(":${entry.key}", { "S" : "${entry.value}" })}
        #end
      #end
    #end
  #end

  ## Start building the update expression, starting with attributes we're going to
SET **
  #set( $expression = "" )
  #if( !$expSet.isEmpty() )
    #set( $expression = "SET" )
    #foreach( $entry in $expSet.entrySet() )

```

```

        #set( $expression = "${expression} ${entry.key} = ${entry.value}" )
        #if ( $foreach.hasNext )
            #set( $expression = "${expression}," )
        #end
    #end
#end

## Continue building the update expression, adding attributes we're going to ADD **
#if( !${expAdd.isEmpty()} )
    #set( $expression = "${expression} ADD" )
    #foreach( $entry in $expAdd.entrySet() )
        #set( $expression = "${expression} ${entry.key} ${entry.value}" )
        #if ( $foreach.hasNext )
            #set( $expression = "${expression}," )
        #end
    #end
#end

## Continue building the update expression, adding attributes we're going to REMOVE
**
#if( !${expRemove.isEmpty()} )
    #set( $expression = "${expression} REMOVE" )

    #foreach( $entry in $expRemove )
        #set( $expression = "${expression} ${entry}" )
        #if ( $foreach.hasNext )
            #set( $expression = "${expression}," )
        #end
    #end
#end

## Finally, write the update expression into the document, along with any
expressionNames and expressionValues **
"update" : {
    "expression" : "${expression}"
    #if( !${expNames.isEmpty()} )
        , "expressionNames" : $utils.toJson($expNames)
    #end
    #if( !${expValues.isEmpty()} )
        , "expressionValues" : $utils.toJson($expValues)
    #end
},

"condition" : {

```

```
      "expression"      : "version = :expectedVersion",
      "expressionValues" : {
        ":expectedVersion" :
$util.dynamodb.toDynamoDBJson($ctx.args.expectedVersion)
      }
    }
  }
}
```

Para obter mais informações sobre a API `UpdateItem` do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

DeleteItem

O documento de mapeamento de `DeleteItem` solicitações permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma `DeleteItem` solicitação ao DynamoDB e especifique o seguinte:

- A chave do item no DynamoDB
- Condições para que a operação seja bem-sucedida

O documento de mapeamento `DeleteItem` possui a seguinte estrutura:

```
{
  "version" : "2018-05-29",
  "operation" : "DeleteItem",
  "customPartitionKey" : "foo",
  "populateIndexFields" : boolean value,
  "key": {
    "foo" : ... typed value,
    "bar" : ... typed value
  },
  "condition" : {
    ...
  },
  "_version" : 1
}
```

Os campos são definidos da seguinte forma:

Deleteltem campos

Deleteltem lista de campos

version

As versões de definição de modelo 2017-02-28 e 2018-05-29 são compatíveis atualmente. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação DeleteItem do DynamoDB, ela deve ser definida como DeleteItem. Este valor é obrigatório.

key

A chave do item no DynamoDB. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(mapeamento da solicitação\)](#). Este valor é obrigatório.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, a solicitação DeleteItem excluirá um item independentemente do estado atual. Para obter mais informações sobre as condições, consulte [Expressões de condição](#). Este valor é opcional.

_version

Um valor numérico que representa a versão conhecida mais recente de um item. Este valor é opcional. Esse campo é usado para Detecção de conflitos e só é compatível com fontes de dados versionadas.

customPartitionKey

Quando ativado, esse valor de string modifica o formato dos ds_pk registros ds_sk e usados pela tabela de sincronização delta quando o controle de versão é ativado (para obter mais informações, consulte [Detecção e sincronização de conflitos no Guia](#) do AWS AppSync desenvolvedor). Quando ativado, o processamento da entrada populateIndexFields também é ativado. Esse campo é opcional.

populateIndexFields

Um valor booleano que, quando ativado com **customPartitionKey**, cria novas entradas para cada registro na tabela de sincronização delta, especificamente nas colunas `gsi_ds_pk` e `gsi_ds_sk`. Para obter mais informações, consulte [Detecção e sincronização de conflitos](#) no Guia do desenvolvedor do AWS AppSync . Esse campo é opcional.

O item excluído do DynamoDB é automaticamente convertido nos tipos primitivos GraphQL e JSON e está disponível no contexto de mapeamento (`$context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre os modelos de mapeamento da resposta, consulte [Visão geral do modelo de mapeamento do resolvedor](#).

Exemplo 1

Veja a seguir um modelo de mapeamento para uma mutação do GraphQL `deleteItem(id: ID!)`. Se existir um item com esse ID, ele será excluído.

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

Exemplo 2

Veja a seguir um modelo de mapeamento para uma mutação do GraphQL `deleteItem(id: ID!, expectedVersion: Int!)`. Se existir um item com esse ID, ele será excluído, mas apenas se o campo `version` estiver definido como `expectedVersion`:

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

```

    },
    "condition" : {
      "expression"      : "attribute_not_exists(id) OR version = :expectedVersion",
      "expressionValues" : {
        ":expectedVersion" : $util.dynamodb.toDynamoDBJson($expectedVersion)
      }
    }
  }
}

```

Para obter mais informações sobre a API DeleteItem do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

Consulta

O documento de mapeamento de Query solicitações permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma Query solicitação ao DynamoDB e especifique o seguinte:

- Expressão chave
- Qual índice usar
- Qualquer filtro adicional
- Quantos itens retornar
- Se deve usar leituras consistentes
- direção da consulta (para frente ou para trás)
- Token de paginação

O documento de mapeamento Query possui a seguinte estrutura:

```

{
  "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" : {
    ...
}
}
```

Os campos são definidos da seguinte forma:

Campos de consulta

Lista de campos de consulta

version

As versões de definição de modelo 2017-02-28 e 2018-05-29 são compatíveis atualmente. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação Query do DynamoDB, ela deve ser definida como Query. Este valor é obrigatório.

query

A seção query permite especificar uma expressão de condição de chave que descreve quais itens recuperar do DynamoDB. Para obter mais informações sobre como escrever expressões de condição chave, consulte a documentação do [KeyConditions DynamoDB](#). Essa seção deve ser especificada.

expression

A expressão da consulta. Esse campo deve ser especificado.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de nome usado em

`expression` e o valor deve ser uma string que corresponde ao nome do atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados em `expression`.

expressionValues

As substituições para espaços reservados de valor do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de valor usado na `expression` e o valor deve ser um valor digitado. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de valor do atributo da expressão usados em `expression`.

filter

Um filtro adicional que pode ser usado para filtrar os resultados do DynamoDB antes que sejam retornados. Para obter mais informações sobre os filtros, consulte [Filtros](#). Esse campo é opcional.

index

O nome do índice para consulta. A operação de consulta do DynamoDB permite que você faça a varredura em Índices secundários locais e Índices secundários globais, além do índice de chave primária para uma chave de hash. Se especificado, isso informa o DynamoDB para consultar o índice especificado. Se omitido, o índice da chave primária será consultado.

nextToken

O token de paginação para continuar uma consulta anterior. Isso seria obtido de uma consulta anterior. Esse campo é opcional.

limit

O número máximo de itens a serem avaliados (não necessariamente o número de itens correspondentes). Esse campo é opcional.

scanIndexForward

Um booleano que indica se a consulta deve ser para frente ou para trás. Esse campo é opcional e usa como padrão `true`.

consistentRead

Um booleano que indica se deseja usar leituras consistentes ao consultar o DynamoDB. Esse campo é opcional e usa como padrão `false`.

select

Por padrão, o AWS AppSync resolvedor do DynamoDB retorna somente atributos que são projetados no índice. Se forem necessários mais atributos, você poderá definir esse campo. Esse campo é opcional. Os valores compatíveis são:

ALL_ATTRIBUTES

Retorna todos os atributos de item da tabela ou índice especificado. Se você consultar um índice secundário local, o DynamoDB buscará todo o item da tabela pai para cada item correspondente no índice. Se o índice estiver configurado para projetar todos os atributos de item, todos os dados podem ser obtidos no índice secundário local, e nenhuma busca será necessária.

ALL_PROJECTED_ATTRIBUTES

Permitido apenas ao consultar um índice. Recupera todos os atributos que foram projetados no índice. Se o índice estiver configurado para projetar todos os atributos, esse valor de retorno é equivalente a especificar ALL_ATTRIBUTES.

SPECIFIC_ATTRIBUTES

Retorna somente os atributos listados em `expression de projection`. Esse valor de retorno é equivalente a especificar `expression de projection` sem especificar nenhum valor para `Select`.

projection

Uma projeção usada para especificar os atributos a serem retornados da operação do DynamoDB. Para obter mais informações sobre projeções, consulte [Projeções](#). Esse campo é opcional.

Os resultados do DynamoDB são automaticamente convertidos nos tipos primitivos GraphQL e JSON e estão disponíveis no contexto de mapeamento (`$context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre os modelos de mapeamento da resposta, consulte [Visão geral do modelo de mapeamento do resolvedor](#).

Os resultados possuem a seguinte estrutura:

```
{
  items = [ ... ],
  nextToken = "a pagination token",
  scannedCount = 10
}
```

Os campos são definidos da seguinte forma:

items

Uma lista que contém os itens retornados pela consulta do DynamoDB.

nextToken

Se existirem mais resultados, `nextToken` conterá um token de paginação que você pode usar em outra solicitação. Observe que AWS AppSync criptografa e ofusca o token de paginação retornado do DynamoDB. Isso impede que os dados da sua tabela sejam divulgados inadvertidamente para o chamador. Observe também que esses tokens de paginação não podem ser usados em diferentes resolvedores.

scannedCount

O número de itens que corresponderam à expressão de condição da consulta, antes que uma expressão de filtro (se houve) fosse aplicada.

Exemplo

Veja a seguir um modelo de mapeamento para uma consulta `getPosts(owner: ID!)` do GraphQL.

Nesse exemplo, um índice secundário global em uma tabela é consultado para retornar todas as postagens de propriedade do ID especificado.

```
{
  "version" : "2017-02-28",
  "operation" : "Query",
  "query" : {
    "expression" : "ownerId = :ownerId",
    "expressionValues" : {
      ":ownerId" : $util.dynamodb.toDynamoDBJson($context.arguments.owner)
    }
  }
}
```

```
  },
  "index" : "owner-index"
}
```

Para obter mais informações sobre a API Query do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

Verificar

O documento de mapeamento de Scan solicitações permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma Scan solicitação ao DynamoDB e especifique o seguinte:

- Um filtro para excluir os resultados
- Qual índice usar
- Quantos itens retornar
- Se deve usar leituras consistentes
- Token de paginação
- Verificações paralelas

O documento de mapeamento Scan possui a seguinte estrutura:

```
{
  "version" : "2017-02-28",
  "operation" : "Scan",
  "index" : "fooIndex",
  "limit" : 10,
  "consistentRead" : false,
  "nextToken" : "aPaginationToken",
  "totalSegments" : 10,
  "segment" : 1,
  "filter" : {
    ...
  },
  "projection" : {
    ...
  }
}
```

Os campos são definidos da seguinte forma:

Verificar campos

Lista de campos de verificação

version

As versões de definição de modelo 2017-02-28 e 2018-05-29 são compatíveis atualmente. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação Scan do DynamoDB, ela deve ser definida como Scan. Este valor é obrigatório.

filter

Um filtro que pode ser usado para filtrar os resultados do DynamoDB antes que sejam retornados. Para obter mais informações sobre os filtros, consulte [Filtros](#). Esse campo é opcional.

index

O nome do índice para consulta. A operação de consulta do DynamoDB permite que você faça a varredura em Índices secundários locais e Índices secundários globais, além do índice de chave primária para uma chave de hash. Se especificado, isso informa o DynamoDB para consultar o índice especificado. Se omitido, o índice da chave primária será consultado.

limit

O número máximo de itens a serem avaliados ao mesmo tempo. Esse campo é opcional.

consistentRead

Um booleano que indica se serão usadas leituras consistentes ao consultar o DynamoDB. Esse campo é opcional e usa como padrão false.

nextToken

O token de paginação para continuar uma consulta anterior. Isso seria obtido de uma consulta anterior. Esse campo é opcional.

select

Por padrão, o AWS AppSync resolvedor do DynamoDB retorna somente os atributos projetados no índice. Se forem necessários mais atributos, esse campo pode ser definido. Esse campo é opcional. Os valores compatíveis são:

ALL_ATTRIBUTES

Retorna todos os atributos de item da tabela ou índice especificado. Se você consultar um índice secundário local, o DynamoDB buscará todo o item da tabela pai para cada item correspondente no índice. Se o índice estiver configurado para projetar todos os atributos de item, todos os dados podem ser obtidos no índice secundário local, e nenhuma busca será necessária.

ALL_PROJECTED_ATTRIBUTES

Permitido apenas ao consultar um índice. Recupera todos os atributos que foram projetados no índice. Se o índice estiver configurado para projetar todos os atributos, esse valor de retorno é equivalente a especificar ALL_ATTRIBUTES.

SPECIFIC_ATTRIBUTES

Retorna somente os atributos listados em `expression de projection`. Esse valor de retorno é equivalente a especificar `expression de projection` sem especificar nenhum valor para `Select`.

totalSegments

O número de segmentos para particionar a tabela ao executar uma verificação paralela. Esse campo é opcional, mas deve ser especificado se `segment` estiver especificado.

segment

O segmento da tabela nessa operação ao executar uma verificação paralela. Esse campo é opcional, mas deve ser especificado se `totalSegments` estiver especificado.

projection

Uma projeção usada para especificar os atributos a serem retornados da operação do DynamoDB. Para obter mais informações sobre projeções, consulte [Projeções](#). Esse campo é opcional.

Os resultados retornados pela verificação do DynamoDB são automaticamente convertidos nos tipos primitivos GraphQL e JSON e está disponível no contexto de mapeamento (`$context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre os modelos de mapeamento da resposta, consulte [Visão geral do modelo de mapeamento do resolvedor](#).

Os resultados possuem a seguinte estrutura:

```
{
  items = [ ... ],
  nextToken = "a pagination token",
  scannedCount = 10
}
```

Os campos são definidos da seguinte forma:

items

Uma lista que contém os itens retornados pela verificação do DynamoDB.

nextToken

Se houver mais resultados, `nextToken` contém um token de paginação que você pode usar em outra solicitação. AWS AppSync criptografa e ofusca o token de paginação retornado do DynamoDB. Isso impede que os dados da sua tabela sejam divulgados inadvertidamente para o chamador. Além disso, esses tokens de paginação não podem ser usados em diferentes resolvedores.

scannedCount

O número de itens recuperados pelo DynamoDB antes da aplicação de uma expressão de filtro (se houver).

Exemplo 1

Veja a seguir um modelo de mapeamento para uma consulta `allPosts` do GraphQL.

Nesse exemplo, todas as entradas na tabela são retornadas.

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

Exemplo 2

Veja a seguir um modelo de mapeamento para uma consulta `postsMatching(title: String!)` do GraphQL.

Nesse exemplo, todas as entradas na tabela são retornadas onde o título começa com o argumento `title`.

```
{
  "version" : "2017-02-28",
  "operation" : "Scan",
  "filter" : {
    "expression" : "begins_with(title, :title)",
    "expressionValues" : {
      ":title" : $util.dynamodb.toDynamoDBJson($context.arguments.title)
    },
  }
}
```

Para obter mais informações sobre a API Scan do DynamoDB, consulte a [Documentação da API do DynamoDB](#).

Sincronização

O documento de mapeamento de solicitação Sync permite recuperar todos os resultados de uma tabela do DynamoDB e, depois, receber apenas os dados alterados desde a última consulta (as atualizações delta). As solicitações Sync só podem ser feitas para fontes de dados versionadas do DynamoDB. É possível especificar o seguinte:

- Um filtro para excluir os resultados
- Quantos itens retornar
- Token de paginação
- Quando sua última operação de Sync foi iniciada

O documento de mapeamento Sync possui a seguinte estrutura:

```
{
  "version" : "2018-05-29",
  "operation" : "Sync",
  "basePartitionKey": "Base Tables PartitionKey",
  "deltaIndexName": "delta-index-name",
  "limit" : 10,
  "nextToken" : "aPaginationToken",
  "lastSync" : 1550000000000,
```



```
"filter" : {  
  ...  
}  
}
```

Os campos são definidos da seguinte forma:

Campos de sincronização

Lista de campos de sincronização

version

A versão de definição do modelo. No momento, somente 2018-05-29 é compatível. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação do Sync do , isso deve ser definido para Sync. Este valor é obrigatório.

filter

Um filtro que pode ser usado para filtrar os resultados do DynamoDB antes que sejam retornados. Para obter mais informações sobre os filtros, consulte [Filtros](#). Esse campo é opcional.

limit

O número máximo de itens a serem avaliados ao mesmo tempo. Esse campo é opcional. Se omitido, o limite padrão será definido como 100 itens. O valor máximo para esse campo é de 1000 itens.

nextToken

O token de paginação para continuar uma consulta anterior. Isso seria obtido de uma consulta anterior. Esse campo é opcional.

lastSync

O momento, em milésimos de segundos de epoch, no qual a última operação de Sync bem-sucedida foi iniciada. Se especificado, somente os itens que foram alterados após lastSync serão retornados. Este campo é opcional e deve ser preenchido somente depois de recuperar todas as páginas de uma operação inicial de Sync. Se omitido, os resultados da tabela Base serão retornados, caso contrário, os resultados da tabela Delta serão retornados.

basePartitionKey

A chave de partição da tabela Base usada ao realizar uma operação Sync. Esse campo permite que uma operação Sync seja executada quando a tabela utiliza uma chave de partição personalizada. Esse é um campo opcional.

deltaIndexName

O índice usado para a operação Sync. Esse índice é necessário para habilitar uma operação Sync em toda a tabela de armazenamento delta quando a tabela usa uma chave de partição personalizada. A operação Sync será executada no GSI (criado em `gsi_ds_pk` e `gsi_ds_sk`). Esse campo é opcional.

Os resultados retornados pela sincronização do DynamoDB são automaticamente convertidos nos tipos primitivos GraphQL e JSON e estão disponíveis no contexto de mapeamento (`$context.result`).

Para obter mais informações sobre a conversão de tipo do DynamoDB, consulte [Sistema de tipo \(mapeamento da resposta\)](#).

Para obter mais informações sobre os modelos de mapeamento da resposta, consulte [Visão geral do modelo de mapeamento do resolvedor](#).

Os resultados possuem a seguinte estrutura:

```
{
  items = [ ... ],
  nextToken = "a pagination token",
  scannedCount = 10,
  startedAt = 1550000000000
}
```

Os campos são definidos da seguinte forma:

items

Uma lista que contém os itens retornados pela sincronização.

nextToken

Se houver mais resultados, `nextToken` contém um token de paginação que você pode usar em outra solicitação. AWS AppSync criptografa e ofusca o token de paginação retornado do

DynamoDB. Isso impede que os dados da sua tabela sejam divulgados inadvertidamente para o chamador. Além disso, esses tokens de paginação não podem ser usados em diferentes resolvedores.

scannedCount

O número de itens recuperados pelo DynamoDB antes da aplicação de uma expressão de filtro (se houver).

startedAt

O momento, em milésimos de segundos de epoch, no qual a operação de sincronização foi iniciada e você pode armazenar localmente e usar em outra solicitação como seu argumento `lastSync`. Se um token de paginação foi incluído na solicitação, esse valor será o mesmo que o retornado pela solicitação para a primeira página de resultados.

Exemplo

Veja a seguir um modelo de mapeamento para uma consulta `syncPosts(nextToken: String, lastSync: AWSTimestamp)` do GraphQL.

Neste exemplo, se `lastSync` for omitido, todas as entradas na tabela base serão retornadas. Se `lastSync` for fornecido, somente as entradas na tabela de sincronização delta que foram alteradas desde `lastSync` serão retornadas.

```
{
  "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

O documento de mapeamento de `BatchGetItem` solicitações permite que você peça ao AWS AppSync resolvedor do DynamoDB que faça uma `BatchGetItem` solicitação ao DynamoDB para recuperar vários itens, potencialmente em várias tabelas. Para esse modelo de solicitação, você deve especificar o seguinte:

- Os nomes da tabela da qual recuperar os itens

- As chaves dos itens a serem recuperadas de cada tabela

Os limites BatchGetItem do DynamoDB se aplicam e nenhuma expressão de condição pode ser fornecida.

O documento de mapeamento BatchGetItem possui a seguinte estrutura:

```
{
  "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" : {  
        ...  
      }  
    }  
  }  
}
```

Os campos são definidos da seguinte forma:

BatchGetItem campos

BatchGetItem lista de campos

version

A versão de definição do modelo. Somente 2018-05-29 é suportado. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação BatchGetItem do DynamoDB, ela deve ser definida como BatchGetItem. Este valor é obrigatório.

tables

As tabelas do DynamoDB das quais recuperar os itens. O valor é um mapa no qual os nomes das tabelas são especificados como as chaves do mapa. Pelo menos uma tabela deve ser fornecida. Este valor `tables` é obrigatório.

keys

Lista de chaves do DynamoDB representando a chave primária dos itens a serem recuperados. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#).

consistentRead

Se deve usar uma leitura consistente ao executar uma GetItem operação. Esse valor é opcional e o padrão é falso.

projection

Uma projeção usada para especificar os atributos a serem retornados da operação do DynamoDB. Para obter mais informações sobre projeções, consulte [Projeções](#). Esse campo é opcional.

Informações importantes:

- Se um item não tiver sido recuperado da tabela, um elemento nulo aparecerá no bloco de dados dessa tabela.
- Os resultados de invocação são classificados por tabela, com base na ordem em que foram fornecidos dentro do modelo de mapeamento de solicitação.
- Cada comando Get dentro de um BatchGetItem é atômico, no entanto, um lote pode ser parcialmente processado. Se um lote for parcialmente processado devido a um erro, as chaves não processadas serão retornadas como parte do resultado da chamada dentro do bloco unprocessedKeys.
- O BatchGetItem é limitado a 100 chaves.

Para o seguinte exemplo do modelo de mapeamento de solicitação:

```
{
  "version": "2018-05-29",
  "operation": "BatchGetItem",
  "tables": {
    "authors": [
      {
        "author_id": {
          "S": "a1"
        }
      },
    ],
  },
  "posts": [
    {
      "author_id": {
        "S": "a1"
      },
      "post_id": {
        "S": "p2"
      }
    }
  ]
}
```

```

    }
  ],
}
}

```

O resultado de invocação disponível em `$ctx.result` é o seguinte:

```

{
  "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": []
  }
}

```

O `$ctx.error` contém detalhes sobre o erro. As chaves `data`, `unprocessedKeys` e todas as chaves de tabela fornecidas no modelo de mapeamento de solicitação estão presentes no resultado da invocação. Os itens que foram excluídos aparecem no bloco `data`. Itens que não foram processados são marcados como nulos no bloco de dados e colocados dentro do bloco `unprocessedKeys`.

Para obter um exemplo mais completo, siga o tutorial do DynamoDB Batch aqui [AppSync Tutorial: resolvedores de lotes do DynamoDB](#).

BatchDeleteItem

O documento de mapeamento de BatchDeleteItem solicitações permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma BatchWriteItem solicitação ao DynamoDB para excluir vários itens, potencialmente em várias tabelas. Para esse modelo de solicitação, você deve especificar o seguinte:

- Os nomes da tabela da qual excluir os itens
- As chaves dos itens a serem excluídas de cada tabela

Os limites BatchWriteItem do DynamoDB se aplicam e nenhuma expressão de condição pode ser fornecida.

O documento de mapeamento BatchDeleteItem possui a seguinte estrutura:

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



```
}  
}
```

Os campos são definidos da seguinte forma:

BatchDeleteItem campos

BatchDeleteItem lista de campos

version

A versão de definição do modelo. Somente 2018-05-29 é suportado. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação BatchDeleteItem do DynamoDB, ela deve ser definida como BatchDeleteItem. Este valor é obrigatório.

tables

As tabelas do DynamoDB das quais excluir os itens. Cada tabela é uma lista de chaves do DynamoDB representando a chave primária dos itens a serem excluídos. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Pelo menos uma tabela deve ser fornecida. O valor tables é obrigatório.

Informações importantes:

- Ao contrário da operação DeleteItem, o item totalmente excluído não é retornado na resposta. Somente a chave passada é retornada.
- Se um item não tiver sido excluído da tabela, um elemento nulo aparecerá no bloco de dados dessa tabela.
- Os resultados de invocação são classificados por tabela, com base na ordem em que foram fornecidos dentro do modelo de mapeamento de solicitação.
- Cada comando Delete dentro de um BatchDeleteItem é atômico. No entanto, um lote pode ser parcialmente processado. Se um lote for parcialmente processado devido a um erro, as chaves não processadas serão retornadas como parte do resultado da chamada dentro do bloco unprocessedKeys.
- O BatchDeleteItem é limitado a 25 chaves.

- Essa operação não é compatível quando usada com detecção de conflitos. Usar ambos ao mesmo tempo pode gerar um erro.

Para o seguinte exemplo do modelo de mapeamento de solicitação:

```
{
  "version": "2018-05-29",
  "operation": "BatchDeleteItem",
  "tables": {
    "authors": [
      {
        "author_id": {
          "S": "a1"
        }
      },
    ],
    "posts": [
      {
        "author_id": {
          "S": "a1"
        },
        "post_id": {
          "S": "p2"
        }
      },
    ],
  },
}
```

O resultado de invocação disponível em `$ctx.result` é o seguinte:

```
{
  "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": []
}
```

O `$ctx.error` contém detalhes sobre o erro. As chaves `data`, `unprocessedKeys` e todas as chaves de tabela fornecidas no modelo de mapeamento de solicitação estão presentes no resultado da invocação. Os itens que foram excluídos estão presentes no bloco `data`. Itens que não foram processados são marcados como nulos no bloco de dados e colocados dentro do bloco `unprocessedKeys`.

Para obter um exemplo mais completo, siga o tutorial do DynamoDB Batch aqui [AppSync Tutorial: resolvedores de lotes do DynamoDB](#).

BatchPutItem

O documento de mapeamento de `BatchPutItem` solicitações permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma `BatchWriteItem` solicitação ao DynamoDB para colocar vários itens, potencialmente em várias tabelas. Para esse modelo de solicitação, você deve especificar o seguinte:

- Os nomes da tabela na qual inserir os itens
- Os itens completos a serem inseridos em cada tabela

Os limites `BatchWriteItem` do DynamoDB se aplicam e nenhuma expressão de condição pode ser fornecida.

O documento de mapeamento `BatchPutItem` possui a seguinte estrutura:

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

```

Os campos são definidos da seguinte forma:

BatchPutItem campos

BatchPutItem lista de campos

version

A versão de definição do modelo. Somente 2018-05-29 é suportado. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação BatchPutItem do DynamoDB, ela deve ser definida como BatchPutItem. Este valor é obrigatório.

tables

As tabelas do DynamoDB nas quais inserir os itens. Cada entrada da tabela representa uma lista de itens do DynamoDB a serem inseridos nesta tabela específica. Pelo menos uma tabela deve ser fornecida. Este valor é obrigatório.

Informações importantes:

- Os itens totalmente inseridos são retornados na resposta, se bem-sucedidos.
- Se um item não tiver sido inserido na tabela, um elemento nulo será exibido no bloco de dados dessa tabela.
- Os itens inseridos são classificados por tabela, com base na ordem em que foram fornecidos dentro do modelo de mapeamento de solicitação.
- Cada comando Put dentro de um BatchPutItem é atômico, no entanto, um lote pode ser parcialmente processado. Se um lote for parcialmente processado devido a um erro, as chaves não processadas serão retornadas como parte do resultado da chamada dentro do bloco `unprocessedKeys`.
- O BatchPutItem é limitado a 25 itens.
- Essa operação não é compatível quando usada com detecção de conflitos. Usar ambos ao mesmo tempo pode gerar um erro.

Para o seguinte exemplo do modelo de mapeamento de solicitação:

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

```

O resultado de invocação disponível em `$ctx.result` é o seguinte:

```

{
  "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": []
  }
}

```

O `$ctx.error` contém detalhes sobre o erro. As chaves `data`, `unprocessedItems` e todas as chaves de tabela fornecidas no modelo de mapeamento de solicitação estão presentes no resultado da invocação. Os itens que foram inseridos estão no bloco `data`. Itens que não foram processados são marcados como nulos no bloco de dados e colocados dentro do bloco `unprocessedItems`.

Para obter um exemplo mais completo, siga o tutorial do DynamoDB Batch aqui [AppSync Tutorial: resolvedores de lotes do DynamoDB](#).

TransactGetItems

O documento de mapeamento de TransactGetItems solicitações permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma TransactGetItems solicitação ao DynamoDB para recuperar vários itens, potencialmente em várias tabelas. Para esse modelo de solicitação, você deve especificar o seguinte:

- O nome da tabela de cada item de solicitação de onde recuperar o item
- A chave de cada item de solicitação a ser recuperado de cada tabela

Os limites TransactGetItems do DynamoDB se aplicam e nenhuma expressão de condição pode ser fornecida.

O documento de mapeamento TransactGetItems possui a seguinte estrutura:

```
{
  "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" : {
```

```

    ...
  }
}
]
}
```

Os campos são definidos da seguinte forma:

TransactGetItems campos

TransactGetItems lista de campos

version

A versão de definição do modelo. Somente 2018-05-29 é suportado. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação TransactGetItems do DynamoDB, ela deve ser definida como TransactGetItems. Este valor é obrigatório.

transactItems

Os itens de solicitação a serem incluídos. O valor é uma matriz de itens de solicitação. Pelo menos um item de solicitação deve ser fornecido. Este valor transactItems é obrigatório.

table

A tabela do DynamoDB da qual recuperar o item. O valor é uma string do nome da tabela. Este valor table é obrigatório.

key

A chave do DynamoDB representando a chave primária do item a ser recuperado. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#).

projection

Uma projeção usada para especificar os atributos a serem retornados da operação do DynamoDB. Para obter mais informações sobre projeções, consulte [Projeções](#). Esse campo é opcional.

Informações importantes:

- Se uma transação for bem-sucedida, a ordem dos itens recuperados no bloco `items` será a mesma que a ordem dos itens solicitados.
- As transações são realizadas de all-or-nothing alguma forma. Se algum item de solicitação causar um erro, não será executada a transação inteira e os detalhes do erro serão retornados.
- Um item de solicitação que não pode ser recuperado não é um erro. Em vez disso, um elemento nulo aparece no bloco de itens na posição correspondente.
- Se o erro de uma transação for `TransactionCanceledException`, o `cancellationReasons` bloco será preenchido. A ordem dos motivos de cancelamento no bloco `cancellationReasons` será a mesma que a ordem de itens solicitados.
- `TransactGetItems` está limitado a 100 itens de solicitação.

Para o seguinte exemplo do modelo de mapeamento de solicitação:

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

Se a transação for bem-sucedida e somente o primeiro item solicitado for recuperado, o resultado de invocação disponível em `$ctx.result` será o seguinte:

```
{
  "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
}
```

Se a transação falhar devido à `TransactionCanceledException` causada do primeiro item da solicitação, o resultado da invocação `$ctx.result` disponível em será o seguinte:

```
{
  "items": null,
  "cancellationReasons": [
    {
      "type": "Sample error type",
      "message": "Sample error message"
    },
    {
      "type": "None",
      "message": "None"
    }
  ]
}
```

O `$ctx.error` contém detalhes sobre o erro. A presença dos itens de chaves e `cancellationReasons` está garantida em `$ctx.result`.

Para ver um exemplo mais completo, siga o tutorial de transações do DynamoDB aqui [Tutorial AppSync : resolvedores de transações do DynamoDB](#).

TransactWriteItems

O documento de mapeamento de `TransactWriteItems` solicitações permite que você peça ao AWS AppSync resolvidor do DynamoDB que faça uma `TransactWriteItems` solicitação ao DynamoDB para gravar vários itens, potencialmente em várias tabelas. Para esse modelo de solicitação, você deve especificar o seguinte:

- O nome da tabela de destino de cada item de solicitação
- A operação de cada item de solicitação a ser executado. Há quatro tipos de operações compatíveis: `PutItem`, `UpdateItem`, `DeleteItem`, e `ConditionCheck`
- A chave de cada item de solicitação a ser gravado

Os limites `TransactWriteItems` do DynamoDB são aplicáveis.

O documento de mapeamento `TransactWriteItems` possui a seguinte estrutura:

```
{
  "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 campos

TransactWriteItems lista de campos

Os campos são definidos da seguinte forma:

version

A versão de definição do modelo. Somente 2018-05-29 é suportado. Este valor é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação TransactWriteItems do DynamoDB, ela deve ser definida como TransactWriteItems. Este valor é obrigatório.

transactItems

Os itens de solicitação a serem incluídos. O valor é uma matriz de itens de solicitação. Pelo menos um item de solicitação deve ser fornecido. Este valor transactItems é obrigatório.

Em PutItem, os campos são definidos da seguinte forma:

table

A tabela de destino do DynamoDB. O valor é uma string do nome da tabela. Este valor `table` é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação `PutItem` do DynamoDB, ela deve ser definida como `PutItem`. Este valor é obrigatório.

key

A chave do DynamoDB representando a chave primária do item a ser inserida. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

attributeValues

O restante dos atributos do item a ser colocado no DynamoDB. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Esse campo é opcional.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, uma solicitação `PutItem` substitui qualquer entrada existente para esse item. Você pode especificar se deseja recuperar o item existente quando a verificação de condição falhar. Para obter mais informações sobre as condições transacionais, consulte [Expressões de condição da transação](#). Este valor é opcional.

Em `UpdateItem`, os campos são definidos da seguinte forma:

table

A tabela do DynamoDB a ser atualizada. O valor é uma string do nome da tabela. Este valor `table` é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação `UpdateItem` do DynamoDB, ela deve ser definida como `UpdateItem`. Este valor é obrigatório.

key

A chave do DynamoDB representando a chave primária do item a ser atualizada. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

update

A seção update permite especificar uma expressão de atualização que descreve como atualizar o item no DynamoDB. Para obter mais informações sobre como escrever expressões de atualização, consulte a documentação do [UpdateExpressions DynamoDB](#). Esta seção é obrigatória.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, a solicitação UpdateItem atualizará as entradas existentes independentemente do estado atual. Você pode especificar se deseja recuperar o item existente quando a verificação de condição falhar. Para obter mais informações sobre as condições transacionais, consulte [Expressões de condição da transação](#). Este valor é opcional.

Em DeleteItem, os campos são definidos da seguinte forma:

table

A tabela do DynamoDB na qual excluir o item. O valor é uma string do nome da tabela. Este valor table é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação DeleteItem do DynamoDB, ela deve ser definida como DeleteItem. Este valor é obrigatório.

key

A chave do DynamoDB representando a chave primária do item a ser excluída. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Se nenhuma condição for especificada, a solicitação `DeleteItem` excluirá um item independentemente do estado atual. Você pode especificar se deseja recuperar o item existente quando a verificação de condição falhar. Para obter mais informações sobre as condições transacionais, consulte [Expressões de condição da transação](#). Este valor é opcional.

Em `ConditionCheck`, os campos são definidos da seguinte forma:

table

A tabela do DynamoDB na qual verificar a condição. O valor é uma string do nome da tabela. Este valor `table` é obrigatório.

operation

A operação do DynamoDB para execução. Para executar a operação `ConditionCheck` do DynamoDB, ela deve ser definida como `ConditionCheck`. Este valor é obrigatório.

key

A chave do DynamoDB representando a chave primária do item para verificar a condição. Os itens do DynamoDB podem ter uma única chave de hash ou uma chave de hash e uma chave de classificação, dependendo da estrutura da tabela. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Este valor é obrigatório.

condition

Uma condição para determinar se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB. Você pode especificar se deseja recuperar o item existente quando a verificação de condição falhar. Para obter mais informações sobre as condições transacionais, consulte [Expressões de condição da transação](#). Este valor é obrigatório.

Informações importantes:

- Somente chaves de itens de solicitação são retornadas na resposta, se bem-sucedidas. A ordem das chaves será a mesma que a ordem dos itens solicitados.

- As transações são realizadas de all-or-nothing alguma forma. Se algum item de solicitação causar um erro, não será executada a transação inteira e os detalhes do erro serão retornados.
- Dois itens de solicitação não podem segmentar o mesmo item. Caso contrário, eles causarão `TransactionCanceledException` erros.
- Se o erro de uma transação for `TransactionCanceledException`, o `cancellationReasons` bloco será preenchido. Se a verificação de condição de um item de solicitação falhar e você não especificar `returnValuesOnConditionCheckFailure` como `false`, o item existente na tabela será recuperado e armazenado em item na posição correspondente do bloco `cancellationReasons`.
- `TransactWriteItems` está limitado a 100 itens de solicitação.
- Essa operação não é compatível quando usada com detecção de conflitos. Usar ambos ao mesmo tempo pode gerar um erro.

Para o seguinte exemplo do modelo de mapeamento de solicitação:

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

```

Se a transação for bem-sucedida, o resultado de invocação disponível em `$ctx.result` será o seguinte:

```

{
  "keys": [
    // Key of the PutItem request
    {
      "post_id": "p1",
    },
    // Key of the UpdateItem request
    {
      "author_id": "a1"
    }
  ],
  "cancellationReasons": null
}

```

Se a transação falhar devido a falha de verificação de condição da solicitação `PutItem`, o resultado de invocação disponível em `$ctx.result` será o seguinte:

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

O `$ctx.error` contém detalhes sobre o erro. A presença de chaves e `cancellationReasons` está garantida em `$ctx.result`.

Para ver um exemplo mais completo, siga o tutorial de transações do DynamoDB aqui [Tutorial AppSync : resolvedores de transações do DynamoDB](#).

Sistema de tipo (mapeamento da solicitação)

Ao usar o AWS AppSync resolvedor do DynamoDB para chamar suas tabelas do DynamoDB AWS AppSync , precisa saber o tipo de cada valor a ser usado nessa chamada. Isso ocorre porque o DynamoDB suporta mais tipos primitivos do que GraphQL ou JSON (como conjuntos e dados binários). AWS AppSync precisa de algumas dicas ao traduzir entre o GraphQL e o DynamoDB, caso contrário, seria necessário fazer algumas suposições sobre como os dados são estruturados em sua tabela.

Para obter mais informações sobre os tipos de dados do DynamoDB, consulte os [Descritores de tipos de dados](#) do DynamoDB e a documentação dos [Tipos de dados](#).

Um valor do DynamoDB é representado por um objeto JSON que contém um único par de chave-valor. A chave especifica o tipo do DynamoDB e o valor que especifica o valor em si. No exemplo a seguir, a chave `S` indica que o valor é uma string e o valor `identifier` é o próprio valor da string.

```
{ "S" : "identifier" }
```

Observe que o objeto JSON não pode ter mais de um par de chave-valor. Se mais de um par de chave/valor for especificado, o documento de mapeamento da solicitação não será analisado.

Um valor do DynamoDB é usado em qualquer lugar no documento de mapeamento da solicitação onde é necessário especificar um valor. Alguns lugares onde é necessário fazer isso incluem: as seções `key` e `attributeValue`, e a seção `expressionValues` das seções de expressões. No exemplo a seguir, o valor da string `identifier` do DynamoDB está sendo atribuído ao campo `id` em uma seção `key` (talvez em um documento de mapeamento da solicitação `GetItem`).

```
"key" : {  
  "id" : { "S" : "identifier" }  
}
```

Tipos compatíveis

AWS AppSync é compatível com os seguintes tipos de escalar, documento e conjunto do DynamoDB:

Tipo string **S**

O valor de uma única string. O valor de uma string do DynamoDB é indicado por:

```
{ "S" : "some string" }
```

Um exemplo de uso é:

```
"key" : {  
  "id" : { "S" : "some string" }  
}
```

Tipo conjunto de strings **SS**

Um conjunto de valores de strings. O valor de conjunto de strings do DynamoDB é indicado por:

```
{ "SS" : [ "first value", "second value", ... ] }
```

Um exemplo de uso é:

```
"attributeValues" : {  
  "phoneNumbers" : { "SS" : [ "+1 555 123 4567", "+1 555 234 5678" ] }  
}
```

Tipo número **N**

Um único valor numérico. O valor de um número do DynamoDB é indicado por:

```
{ "N" : 1234 }
```

Um exemplo de uso é:

```
"expressionValues" : {  
  ":expectedVersion" : { "N" : 1 }  
}
```

Tipo conjunto de números **NS**

Um conjunto de valores de números. O valor de conjunto de números do DynamoDB é indicado por:

```
{ "NS" : [ 1, 2.3, 4 ... ] }
```

Um exemplo de uso é:

```
"attributeValues" : {  
  "sensorReadings" : { "NS" : [ 67.8, 12.2, 70 ] }  
}
```

Tipo binário **B**

Um valor binário. Um valor binário do DynamoDB é indicado por:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

Observe que o valor é, na verdade, uma string, em que a string é a representação codificada em base64 dos dados binários. AWS AppSync decodifica essa string de volta em seu valor binário antes de enviá-la para o DynamoDB. AWS AppSync usa o esquema de decodificação base64

conforme definido pela RFC 2045: qualquer caractere que não esteja no alfabeto base64 é ignorado.

Um exemplo de uso é:

```
"attributeValues" : {  
  "binaryMessage" : { "B" : "SGVsbG8sIFdvcmxkIQo=" }  
}
```

Tipo conjunto de binários **BS**

Um conjunto de valores binários. Um valor de conjunto de binários do DynamoDB é indicado por:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

Observe que o valor é, na verdade, uma string, em que a string é a representação codificada em base64 dos dados binários. AWS AppSync decodifica essa string de volta em seu valor binário antes de enviá-la para o DynamoDB. AWS AppSync usa o esquema de decodificação base64 conforme definido pela RFC 2045: qualquer caractere que não esteja no alfabeto base64 é ignorado.

Um exemplo de uso é:

```
"attributeValues" : {  
  "binaryMessages" : { "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ] }  
}
```

Tipo booliano **BOOL**

Um valor booleano. Um valor Booleano do DynamoDB é indicado por:

```
{ "BOOL" : true }
```

Observe que apenas true e false são valores válidos.

Um exemplo de uso é:

```
"attributeValues" : {  
  "orderComplete" : { "BOOL" : false }  
}
```

Tipo lista L

Uma lista de qualquer outro valor do DynamoDB compatível. O valor de lista do DynamoDB é indicado por:

```
{ "L" : [ ... ] }
```

Observe que o valor é um valor composto, onde a lista pode conter zero ou mais de qualquer valor do DynamoDB compatível (incluindo outras listas). A lista também pode conter uma mistura de diferentes tipos.

Um exemplo de uso é:

```
{ "L" : [  
  { "S" : "A string value" },  
  { "N" : 1 },  
  { "SS" : [ "Another string value", "Even more string values!" ] }  
]
```

Tipo mapa M

Representando uma coleção não ordenada de pares de chave/valor de outros valores do DynamoDB compatíveis. O valor de mapa do DynamoDB é indicado por:

```
{ "M" : { ... } }
```

Observe que um mapa pode conter zero ou mais pares de chave/valor. A chave deve ser uma string, e o valor pode ser qualquer valor do DynamoDB compatível (incluindo outros mapas). O mapa também pode conter uma mistura de diferentes tipos.

Um exemplo de uso é:

```
{ "M" : {  
  "someString" : { "S" : "A string value" },  
  "someNumber" : { "N" : 1 },  
  "stringSet" : { "SS" : [ "Another string value", "Even more string  
values!" ] }  
}
```

Tipo nulo **NULL**

Um valor nulo. O valor nulo do DynamoDB é indicado por:

```
{ "NULL" : null }
```

Um exemplo de uso é:

```
"attributeValues" : {  
  "phoneNumbers" : { "NULL" : null }  
}
```

para obter mais informações sobre cada tipo, consulte a [Documentação do DynamoDB](#).

Sistema de tipo (mapeamento da resposta)

Ao receber uma resposta do DynamoDB AWS AppSync, a converte automaticamente em tipos primitivos GraphQL e JSON. Cada atributo no DynamoDB é decodificado e retornado no contexto do mapeamento da resposta.

Por exemplo, se o DynamoDB retorna o seguinte:

```
{  
  "id" : { "S" : "1234" },  
  "name" : { "S" : "Nadia" },  
  "age" : { "N" : 25 }  
}
```

Em seguida, o resolvidor do AWS AppSync DynamoDB o converte nos tipos GraphQL e JSON como:

```
{  
  "id" : "1234",  
  "name" : "Nadia",  
  "age" : 25  
}
```

Esta seção explica como AWS AppSync converte os seguintes tipos de escalar, documento e conjunto do DynamoDB:

Tipo string **S**

O valor de uma única string. Um valor de string do DynamoDB é retornado como uma string.

Por exemplo, se o DynamoDB retornou o seguinte valor de string do DynamoDB:

```
{ "S" : "some string" }
```

AWS AppSync o converte em uma string:

```
"some string"
```

Tipo conjunto de strings **SS**

Um conjunto de valores de strings. Um valor de conjunto de strings do DynamoDB é retornado simplesmente como uma lista de strings.

Por exemplo, se o DynamoDB retornou o seguinte valor de conjunto de strings do DynamoDB:

```
{ "SS" : [ "first value", "second value", ... ] }
```

AWS AppSync o converte em uma lista de strings:

```
[ "+1 555 123 4567", "+1 555 234 5678" ]
```

Tipo número **N**

Um único valor numérico. Um valor de número do DynamoDB é retornado como um número.

Por exemplo, se o DynamoDB retornou o seguinte valor de número do DynamoDB:

```
{ "N" : 1234 }
```

AWS AppSync o converte em um número:

```
1234
```

Tipo conjunto de números **NS**

Um conjunto de valores de números. Um valor de conjunto de números do DynamoDB é retornado simplesmente como uma lista de números.

Por exemplo, se o DynamoDB retornou o seguinte valor de conjunto de números do DynamoDB:

```
{ "NS" : [ 67.8, 12.2, 70 ] }
```

AWS AppSync o converte em uma lista de números:

```
[ 67.8, 12.2, 70 ]
```

Tipo binário **B**

Um valor binário. Um valor binário do DynamoDB é retornado como uma string que contém a representação em base64 desse valor.

Por exemplo, se o DynamoDB retornou o seguinte valor binário do DynamoDB:

```
{ "B" : "SGVsbG8sIFdvcmxkIQo=" }
```

AWS AppSync o converte em uma string contendo a representação base64 do valor:

```
"SGVsbG8sIFdvcmxkIQo="
```

Observe que os dados binários estão codificados no esquema de codificação base64 conforme especificado em [RFC 4648](#) e [RFC 2045](#).

Tipo conjunto de binários **BS**

Um conjunto de valores binários. Um valor de conjunto de binários do DynamoDB é retornado como uma lista de strings que contém a representação em base64 dos valores.

Por exemplo, se o DynamoDB retornou o seguinte valor de conjuntos de binários do DynamoDB:

```
{ "BS" : [ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ] }
```

AWS AppSync o converte em uma lista de strings contendo a representação base64 dos valores:

```
[ "SGVsbG8sIFdvcmxkIQo=", "SG93IGFyZSB5b3U/Cg==" ... ]
```

Observe que os dados binários estão codificados no esquema de codificação base64 conforme especificado em [RFC 4648](#) e [RFC 2045](#).

Tipo booliano **BOOL**

Um valor booleano. Um valor Booleano do DynamoDB é retornado como um Booleano.

Por exemplo, se o DynamoDB retornou o seguinte valor Booleano do DynamoDB:

```
{ "BOOL" : true }
```

AWS AppSync o converte em um booleano:

```
true
```

Tipo lista **L**

Uma lista de qualquer outro valor do DynamoDB compatível. Um valor de lista do DynamoDB é retornado como uma lista de valores, onde cada valor interno também é convertido.

Por exemplo, se o DynamoDB retornou o seguinte valor de lista do DynamoDB:

```
{ "L" : [
  { "S" : "A string value" },
  { "N" : 1 },
  { "SS" : [ "Another string value", "Even more string values!" ] }
]
```

AWS AppSync o converte em uma lista de valores convertidos:

```
[ "A string value", 1, [ "Another string value", "Even more string values!" ] ]
```

Tipo mapa **M**

A também key/value collection of any other supported DynamoDB value. A DynamoDB Map value is returned as a JSON object, where each key/value é convertido.

Por exemplo, se o DynamoDB retornou o seguinte valor de mapa do DynamoDB:

```
{ "M" : {
  "someString" : { "S" : "A string value" },
  "someNumber" : { "N" : 1 },
```

```
    "stringSet" : { "SS" : [ "Another string value", "Even more string values!" ] }  
  }  
}
```

AWS AppSync o converte em um objeto JSON:

```
{  
  "someString" : "A string value",  
  "someNumber" : 1,  
  "stringSet" : [ "Another string value", "Even more string values!" ]  
}
```

Tipo nulo **NULL**

Um valor nulo.

Por exemplo, se o DynamoDB retornou o seguinte valor nulo do DynamoDB:

```
{ "NULL" : null }
```

AWS AppSync o converte em um nulo:

```
null
```

Filtros

Ao consultar objetos no DynamoDB usando as operações Query e Scan, opcionalmente, você pode especificar um `filter` que avalia os resultados e retorna apenas os valores desejados.

A seção de mapeamento do filtro de um documento de mapeamento Query ou Scan possui a seguinte estrutura:

```
"filter" : {  
  "expression" : "filter expression"  
  "expressionNames" : {  
    "#name" : "name",  
  },  
  "expressionValues" : {  
    ":value" : ... typed value
```

```
  },  
}
```

Os campos são definidos da seguinte forma:

expression

A expressão da consulta. Para obter mais informações sobre como escrever expressões de filtro, consulte a documentação do [DynamoDB e ScanFilter](#) do QueryFilter DynamoDB. Esse campo deve ser especificado.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de nome usado na `expression`. O valor deve ser uma string que corresponde ao nome de atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados em `expression`.

expressionValues

As substituições para espaços reservados de valor do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de valor usado na `expression` e o valor deve ser um valor digitado. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Isso deve ser especificado. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de valor do atributo da expressão usados em `expression`.

Exemplo

Veja a seguir uma seção de filtro para um modelo de mapeamento, onde as entradas recuperadas do DynamoDB só são retornadas se o título começa com o argumento `title`.

```
"filter" : {  
  "expression" : "begins_with(#title, :title)",  
  "expressionNames" : {  
    "#title" : "title"  
  },  
  "expressionValues" : {  
    ":title" : $util.dynamodb.toDynamoDBJson($context.arguments.title)  
  }  
}
```

```
}
```

Expressões de condição

Quando objetos sofrem mutação no DynamoDB usando as operações `PutItem`, `UpdateItem` e `DeleteItem` do DynamoDB, opcionalmente, é possível especificar uma expressão de condição que controla se a solicitação deve ser bem-sucedida ou não, com base no estado do objeto que já está no DynamoDB antes que a operação seja realizada.

O AWS AppSync resolvidor do DynamoDB permite que uma expressão de condição seja especificada `PutItem` em `UpdateItem`, `DeleteItem` e solicite documentos de mapeamento, além de uma estratégia a ser seguida se a condição falhar e o objeto não for atualizado.

Exemplo 1

O documento de mapeamento `PutItem` a seguir não tem uma expressão de condição. Como resultado, ele coloca um item no DynamoDB mesmo se um item com a mesma chave já existir, sobrescrevendo o item existente.

```
{
  "version" : "2017-02-28",
  "operation" : "PutItem",
  "key" : {
    "id" : { "S" : "1" }
  }
}
```

Exemplo 2

O seguinte documento de mapeamento `PutItem` possui uma expressão de condição que permite que a operação seja bem-sucedida somente se um item com a mesma chave não existir no DynamoDB.

```
{
  "version" : "2017-02-28",
  "operation" : "PutItem",
  "key" : {
    "id" : { "S" : "1" }
  },
  "condition" : {
```

```
    "expression" : "attribute_not_exists(id)"
  }
}
```

Por padrão, se a verificação da condição falhar, o resolvidor do AWS AppSync DynamoDB retornará um erro para a mutação. No entanto, o AWS AppSync resolvidor do DynamoDB oferece alguns recursos adicionais para ajudar os desenvolvedores a lidar com alguns casos extremos comuns:

- Se AWS AppSync o resolvidor do DynamoDB puder determinar que o valor atual no DynamoDB corresponde ao resultado desejado, ele tratará a operação como se tivesse sido bem-sucedida de qualquer maneira.
- Em vez de retornar um erro, você pode configurar o resolvidor para invocar uma função Lambda personalizada para decidir como o resolvidor do AWS AppSync DynamoDB deve lidar com a falha.

Isso é descrito com mais detalhes na seção [Tratamento de uma falha de verificação da condição](#).

[Para obter mais informações sobre as expressões de condições do DynamoDB, consulte a documentação do DynamoDB. ConditionExpressions](#)

Especificação de uma condição

Os documentos de mapeamento da solicitação PutItem, UpdateItem e DeleteItem permitem que uma seção `condition` opcional seja especificada. Se omitida, nenhuma verificação de condição é feita. Se especificada, a condição deve ser verdadeira para que a operação seja bem-sucedida.

A seção `condition` tem a seguinte estrutura:

```
"condition" : {
  "expression" : "someExpression"
  "expressionNames" : {
    "#foo" : "foo"
  },
  "expressionValues" : {
    ":bar" : ... typed value
  },
  "equalsIgnore" : [ "version" ],
  "consistentRead" : true,
  "conditionalCheckFailedHandler" : {
```

```
    "strategy" : "Custom",
    "lambdaArn" : "arn:..."
  }
}
```

Os campos a seguir especificam a condição:

expression

A própria expressão de atualização. Para obter mais informações sobre como escrever expressões condicionais, consulte a documentação do [ConditionExpressions DynamoDB](#). Esse campo deve ser especificado.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares de chave/valor. A chave corresponde a um espaço reservado de nome usado na expressão e o valor deve ser uma string que corresponde ao nome do atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados na expressão.

expressionValues

As substituições para espaços reservados de valor do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de valor usado na expressão e o valor deve ser um valor digitado. Para obter mais informações sobre como especificar um "valor digitado", consulte [Sistema de tipo \(Mapeamento de solicitação\)](#). Isso deve ser especificado. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de valor do atributo da expressão usados na expressão.

Os campos restantes informam ao resolvidor do AWS AppSync DynamoDB como lidar com uma falha na verificação de condição:

equalsIgnore

Quando uma verificação de condição falha ao usar a `PutItem` operação, o resolvidor do AWS AppSync DynamoDB compara o item atualmente no DynamoDB com o item que ele tentou gravar. Se forem os mesmos, ele tratará a operação como bem-sucedida. Você pode usar o `equalsIgnore` campo para especificar uma lista de atributos que AWS AppSync devem ser ignorados ao realizar essa comparação. Por exemplo, se a única diferença era um atributo `version`, ele trata a operação como bem-sucedida. Esse campo é opcional.

consistentRead

Quando uma verificação de condição falha, AWS AppSync obtém o valor atual do item do DynamoDB usando uma leitura altamente consistente. Você pode usar esse campo para fazer com que o AWS AppSync resolvidor do DynamoDB use uma leitura eventualmente consistente. Esse campo é opcional e usa como padrão `true`.

conditionalCheckFailedHandler

Esta seção permite especificar como o resolvidor do AWS AppSync DynamoDB trata uma falha na verificação de condição depois de comparar o valor atual no DynamoDB com o resultado esperado. Esta seção é opcional. Se omitida, o padrão será uma estratégia de `Reject`.

strategy

A estratégia que o AWS AppSync resolvidor do DynamoDB adota depois de comparar o valor atual no DynamoDB com o resultado esperado. Esse campo é obrigatório e tem os valores possíveis a seguir:

Reject

A mutação falha e um erro é adicionado à resposta do GraphQL.

Custom

O AWS AppSync resolvidor do DynamoDB invoca uma função Lambda personalizada para decidir como lidar com a falha na verificação da condição. Quando a `strategy` estiver definida como `Custom`, o campo `lambdaArn` deve conter o ARN da função do Lambda a ser invocada.

lambdaArn

O ARN da função Lambda a ser invocada determina como o resolvidor do DynamoDB deve lidar com a falha na verificação da AWS AppSync condição. Esse campo deve ser especificado somente quando `strategy` for definida como `Custom`. Para obter mais informações sobre como usar esse atributo, consulte [Tratamento de uma falha de verificação da condição](#).

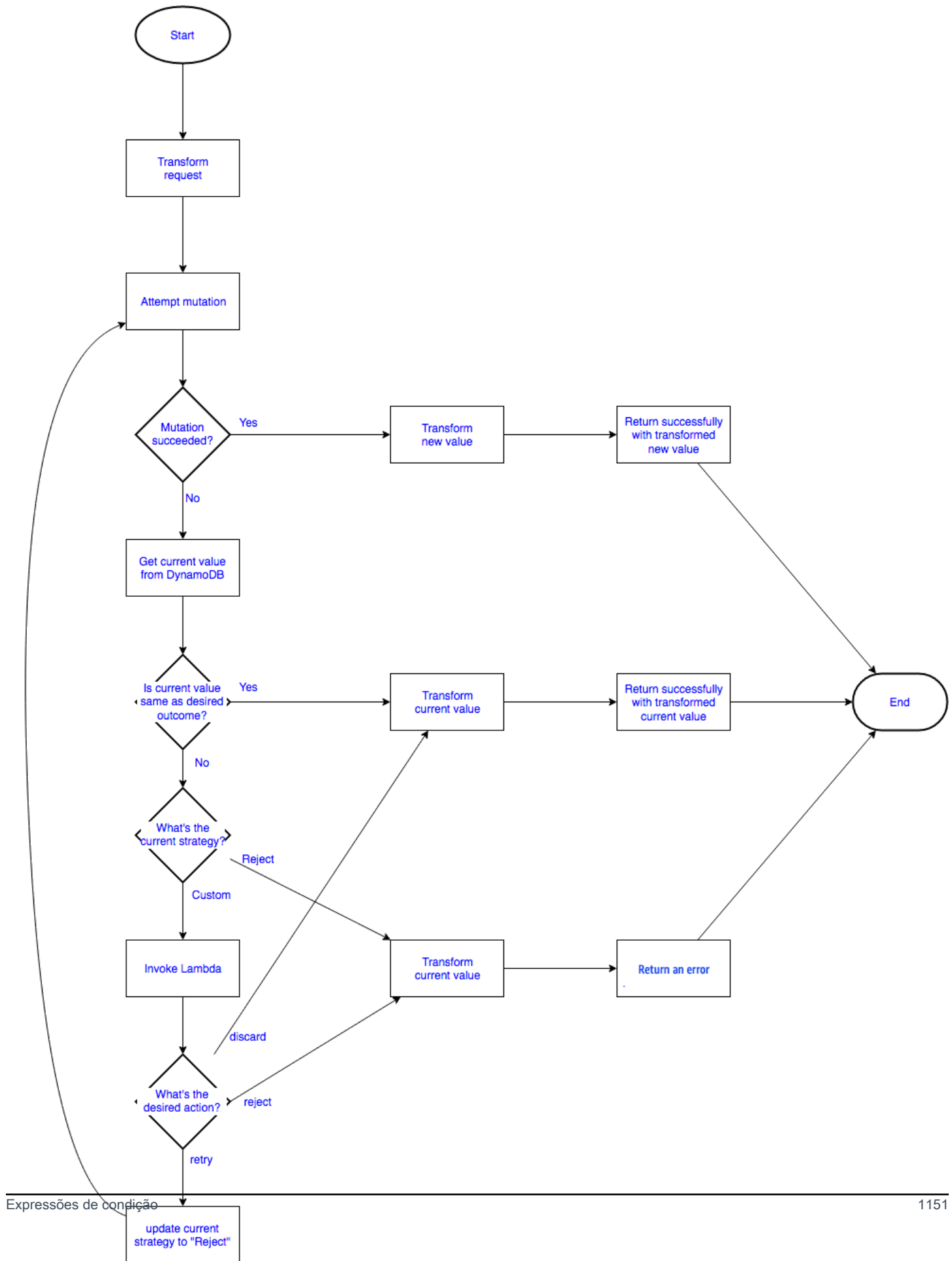
Tratamento de uma falha de verificação da condição

Por padrão, quando uma verificação de condição falha, o resolvidor do AWS AppSync DynamoDB retorna um erro para a mutação e o valor atual do objeto no DynamoDB. No entanto,

o AWS AppSync resolvedor do DynamoDB oferece alguns recursos adicionais para ajudar os desenvolvedores a lidar com alguns casos extremos comuns:

- Se AWS AppSync o resolvedor do DynamoDB puder determinar que o valor atual no DynamoDB corresponde ao resultado desejado, ele tratará a operação como se tivesse sido bem-sucedida de qualquer maneira.
- Em vez de retornar um erro, você pode configurar o resolvedor para invocar uma função Lambda personalizada para decidir como o resolvedor do AWS AppSync DynamoDB deve lidar com a falha.

O fluxograma para esse processo é:



Verificação do resultado desejado

Quando a verificação da condição falha, o resolvidor do AWS AppSync DynamoDB executa uma solicitação do `GetItem` DynamoDB para obter o valor atual do item do DynamoDB. Por padrão, ele usa uma leitura fortemente consistente, mas isso pode ser configurado usando o campo `consistentRead` no bloco `condition` e compará-lo com o resultado esperado:

- Para a `PutItem` operação, o AWS AppSync resolvidor do DynamoDB compara o valor atual com o que ele tentou gravar, excluindo quaisquer atributos listados na comparação. `equalsIgnore`. Se os itens forem os mesmos, ele tratará a operação como bem-sucedida e retornará o item recuperado do DynamoDB. Caso contrário, ele seguirá a estratégia configurada.

Por exemplo, se o documento de mapeamento da solicitação `PutItem` se parecer com o seguinte:

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

E o item que está atualmente no DynamoDB se parecer com o seguinte:

```
{
  "id" : { "S" : "1" },
  "name" : { "S" : "Steve" },
  "version" : { "N" : 8 }
}
```

O AWS AppSync resolvedor do DynamoDB compararia o item que tentou gravar com o valor atual, verificaria que a única diferença era `version` o campo, mas, como está configurado para ignorar `version` o campo, ele considera a operação bem-sucedida e retorna o item que foi recuperado do DynamoDB.

- Para a `DeleteItem` operação, o resolvedor do AWS AppSync DynamoDB verifica se um item foi retornado do DynamoDB. Se nenhum item foi retornado, ele tratará a operação como bem-sucedida. Caso contrário, ele seguirá a estratégia configurada.
- Para a `UpdateItem` operação, o AWS AppSync resolvedor do DynamoDB não tem informações suficientes para determinar se o item atualmente no DynamoDB corresponde ao resultado esperado e, portanto, segue a estratégia configurada.

Se o estado atual do objeto no DynamoDB for diferente do resultado esperado, o resolvedor do AWS AppSync DynamoDB seguirá a estratégia configurada para rejeitar a mutação ou invocar uma função Lambda para determinar o que fazer a seguir.

Seguir a estratégia "Rejeitar"

Ao seguir a `Reject` estratégia, o resolvedor do AWS AppSync DynamoDB retorna um erro para a mutação.

Por exemplo, considere a solicitação de mutação a seguir:

```
mutation {  
  updatePerson(id: 1, name: "Steve", expectedVersion: 1) {  
    Name  
    theVersion  
  }  
}
```

Se o item retornado do DynamoDB for semelhante ao seguinte:

```
{  
  "id" : { "S" : "1" },  
  "name" : { "S" : "Steve" },  
  "version" : { "N" : 8 }  
}
```

E o modelo de mapeamento de resposta é semelhante ao seguinte:

```
{
  "id" : $util.toJson($context.result.id),
  "Name" : $util.toJson($context.result.name),
  "theVersion" : $util.toJson($context.result.version)
}
```

A resposta do GraphQL é semelhante à seguinte:

```
{
  "data": null,
  "errors": [
    {
      "message": "The conditional request failed (Service: AmazonDynamoDBv2;
Status Code: 400; Error Code: ConditionalCheckFailedException; Request ID:
ABCDEFGHJKLMNOPQRSTUVWXYZABCDEFGHJKLMNOPQRSTUVWXYZ)"
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      ...
    }
  ]
}
```

Além disso, se qualquer campo no objeto retornado for preenchido por outros resolvedores e a mutação foi bem-sucedida, eles não serão resolvidos quando o objeto for retornado na seção `error`.

Seguir a estratégia "Personalizada"

Ao seguir a Custom estratégia, o resolvedor do AWS AppSync DynamoDB invoca uma função Lambda para decidir o que fazer em seguida. A Função Lambda escolhe um destas opções a seguir:

- `reject` a mutação. Isso faz com que o AWS AppSync resolvedor do DynamoDB se comporte como se a estratégia configurada `Reject` fosse, retornando um erro para a mutação e o valor atual do objeto no DynamoDB, conforme descrito na seção anterior.
- `discard` a mutação. Isso faz com que o AWS AppSync resolvedor do DynamoDB ignore silenciosamente a falha na verificação da condição e retorne o valor no DynamoDB.
- `retry` a mutação. Isso faz com que o AWS AppSync resolvedor do DynamoDB repita a mutação com um novo documento de mapeamento de solicitações.

A solicitação de invocação do Lambda

O AWS AppSync resolvedor do DynamoDB invoca a função Lambda especificada no `lambdaArn`. Ele usa o mesmo `service-role-arn` configurado na fonte de dados. A carga da invocação tem a seguinte estrutura:

```
{
  "arguments": { ... },
  "requestMapping": {... },
  "currentValue": { ... },
  "resolver": { ... },
  "identity": { ... }
}
```

Os campos são definidos da seguinte forma:

arguments

Os argumentos da mutação do GraphQL. Isso é o mesmo que os argumentos disponíveis para o documento de mapeamento da solicitação em `$context.arguments`.

requestMapping

O documento de mapeamento da solicitação para essa operação.

currentValue

O valor atual do objeto no DynamoDB.

resolver

Informações sobre o AWS AppSync resolvedor.

identity

Informações sobre o chamador. Isso é o mesmo que as informações de identidade disponíveis para o documento de mapeamento da solicitação em `$context.identity`.

Um exemplo completo da carga:

```
{
  "arguments": {
    "id": "1",
    "name": "Steve",
    "expectedVersion": 1
  },
```

```

"requestMapping": {
  "version" : "2017-02-28",
  "operation" : "PutItem",
  "key" : {
    "id" : { "S" : "1" }
  },
  "attributeValues" : {
    "name" : { "S" : "Steve" },
    "version" : { "N" : 2 }
  },
  "condition" : {
    "expression" : "version = :expectedVersion",
    "expressionValues" : {
      ":expectedVersion" : { "N" : 1 }
    },
    "equalsIgnore": [ "version" ]
  }
},
"currentValue": {
  "id" : { "S" : "1" },
  "name" : { "S" : "Steve" },
  "version" : { "N" : 8 }
},
"resolver": {
  "tableName": "People",
  "awsRegion": "us-west-2",
  "parentType": "Mutation",
  "field": "updatePerson",
  "outputType": "Person"
},
"identity": {
  "accountId": "123456789012",
  "sourceIp": "x.x.x.x",
  "user": "AIDAAAAAAAAAAAAAAAAAAAA",
  "userArn": "arn:aws:iam::123456789012:user/appsync"
}
}

```

A resposta de invocação do Lambda

A função Lambda pode inspecionar a carga de invocação e aplicar qualquer lógica de negócios para decidir como o resolvidor do DynamoDB deve lidar com a falha AWS AppSync . Existem três opções para tratamento da falha de verificação da condição:

- **reject** a mutação. A carga da resposta para essa opção deve ter a seguinte estrutura:

```
{
  "action": "reject"
}
```

Isso faz com que o AWS AppSync resolvidor do DynamoDB se comporte como se a estratégia configurada `Reject` fosse, retornando um erro para a mutação e o valor atual do objeto no DynamoDB, conforme descrito na seção acima.

- **discard** a mutação. A carga da resposta para essa opção deve ter a seguinte estrutura:

```
{
  "action": "discard"
}
```

Isso faz com que o AWS AppSync resolvidor do DynamoDB ignore silenciosamente a falha na verificação da condição e retorne o valor no DynamoDB.

- **retry** a mutação. A carga da resposta para essa opção deve ter a seguinte estrutura:

```
{
  "action": "retry",
  "retryMapping": { ... }
}
```

Isso faz com que o AWS AppSync resolvidor do DynamoDB repita a mutação com um novo documento de mapeamento de solicitações. A estrutura da seção `retryMapping` depende da operação do DynamoDB e é um subconjunto do documento de mapeamento da solicitação completo para essa operação.

Em `PutItem`, a seção `retryMapping` tem a seguinte estrutura. Para obter uma descrição do `attributeValues` campo, consulte [PutItem](#).

```
{
  "attributeValues": { ... },
  "condition": {
    "equalsIgnore" = [ ... ],
    "consistentRead" = true
  }
}
```

```
}
```

Em `UpdateItem`, a seção `retryMapping` tem a seguinte estrutura. Para obter uma descrição da `update` seção, consulte [UpdateItem](#).

```
{
  "update" : {
    "expression" : "someExpression"
    "expressionNames" : {
      "#foo" : "foo"
    },
    "expressionValues" : {
      ":bar" : ... typed value
    }
  },
  "condition": {
    "consistentRead" = true
  }
}
```

Em `DeleteItem`, a seção `retryMapping` tem a seguinte estrutura.

```
{
  "condition": {
    "consistentRead" = true
  }
}
```

Não há como especificar uma operação ou chave diferente na qual trabalhar. O AWS AppSync resolvidor do DynamoDB só permite novas tentativas da mesma operação no mesmo objeto. Além disso, a seção `condition` não permite que um `conditionalCheckFailedHandler` seja especificado. Se a nova tentativa falhar, o resolvidor do AWS AppSync DynamoDB segue a estratégia. `Reject`

Veja aqui um exemplo de função do Lambda para lidar com uma solicitação `PutItem` com falha. A lógica de negócios analisa quem fez a chamada. Se foi feita pelo `jeffTheAdmin`, ela tentará novamente a solicitação, atualizando `version` e `expectedVersion` do item atualmente no DynamoDB. Caso contrário, ela rejeitará a mutação.

```

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)
};

```

Expressões de condição da transação

As expressões de condição da transação estão disponíveis em modelos de mapeamento de solicitação de todos os quatro tipos de operações em `TransactWriteItems`, ou seja, `PutItem`, `DeleteItem`, `UpdateItem` e `ConditionCheck`.

Em `PutItem`, `DeleteItem` e `UpdateItem`, a expressão de condição da transação é opcional. Em `ConditionCheck`, a expressão de condição da transação é necessária.

Exemplo 1

O documento de mapeamento transacional DeleteItem a seguir não tem uma expressão de condição. Como resultado, ele exclui o item no DynamoDB.

```
{
  "version": "2018-05-29",
  "operation": "TransactWriteItems",
  "transactItems": [
    {
      "table": "posts",
      "operation": "DeleteItem",
      "key": {
        "id": { "S" : "1" }
      }
    }
  ]
}
```

Exemplo 2

O documento de mapeamento transacional DeleteItem a seguir possui uma expressão de condição da transação que permite que a operação seja bem-sucedida apenas se o autor dessa postagem for igual a determinado nome.

```
{
  "version": "2018-05-29",
  "operation": "TransactWriteItems",
  "transactItems": [
    {
      "table": "posts",
      "operation": "DeleteItem",
      "key": {
        "id": { "S" : "1" }
      },
      "condition": {
        "expression": "author = :author",
        "expressionValues": {
          ":author": { "S" : "Chunyan" }
        }
      }
    }
  ]
}
```

```
]
}
```

Se a verificação de condição falhar, causará `TransactionCanceledException` e os detalhes de erro serão retornados em `$ctx.result.cancellationReasons`. Observe que, por padrão, o item antigo no DynamoDB que fez a verificação da condição falhar será retornado em `$ctx.result.cancellationReasons`.

Especificação de uma condição

Os documentos de mapeamento da solicitação `PutItem`, `UpdateItem` e `DeleteItem` permitem que uma seção `condition` opcional seja especificada. Se omitida, nenhuma verificação de condição é feita. Se especificada, a condição deve ser verdadeira para que a operação seja bem-sucedida. A `ConditionCheck` deve ter uma seção `condition` a ser especificada. A condição deve ser verdadeira para que toda a transação seja bem-sucedida.

A seção `condition` tem a seguinte estrutura:

```
"condition": {
  "expression": "someExpression",
  "expressionNames": {
    "#foo": "foo"
  },
  "expressionValues": {
    ":bar": ... typed value
  },
  "returnValuesOnConditionCheckFailure": false
}
```

Os campos a seguir especificam a condição:

expression

A própria expressão de atualização. Para obter mais informações sobre como escrever expressões condicionais, consulte a documentação do [ConditionExpressions DynamoDB](#). Esse campo deve ser especificado.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares de chave/valor. A chave corresponde a um espaço reservado de nome usado na expressão e o valor deve ser uma string que corresponde ao nome do atributo do item no DynamoDB. Esse

campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados na expressão.

expressionValues

As substituições para espaços reservados de valor do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de valor usado na expressão e o valor deve ser um valor digitado. Para obter mais informações sobre como especificar um "valor digitado", consulte Sistema de tipo (mapeamento de solicitação). Isso deve ser especificado.

Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de valor do atributo da expressão usados na expressão.

returnValuesOnConditionCheckFailure

Especifique se deseja recuperar o item no DynamoDB quando houver falha na verificação de condição. O item recuperado estará em `$ctx.result.cancellationReasons[$index].item`, onde `$index` é o índice do item de solicitação cuja verificação de condição falhou. Esse valor padrão é verdadeiro.

Projeções

Ao ler objetos no DynamoDB usando as operações `GetItem`, `Scan`, `Query`, `BatchGetItem` e `TransactGetItems`, você pode, opcionalmente, especificar uma projeção que identifique os atributos desejados. A projeção tem a seguinte estrutura, que é semelhante aos filtros:

```
"projection" : {  
  "expression" : "projection expression"  
  "expressionNames" : {  
    "#name" : "name",  
  }  
}
```

Os campos são definidos da seguinte forma:

expression

A expressão de projeção, que é uma string. Para recuperar um único atributo, especifique o seu nome. Para vários atributos, os nomes devem ser valores separados por vírgulas. Para obter mais informações sobre como escrever expressões de projeção, consulte a documentação das [expressões de projeção do DynamoDB](#). Este campo é obrigatório.

expressionNames

As substituições para espaços reservados de nome do atributo da expressão, na forma de pares chave-valor. A chave corresponde a um espaço reservado de nome usado na expression. O valor deve ser uma string que corresponde ao nome de atributo do item no DynamoDB. Esse campo é opcional e deve ser preenchido apenas por substituições para espaços reservados de nome do atributo da expressão usados em expression. Para obter mais informações sobre expressionNames, consulte a [documentação do DynamoDB](#).

Exemplo 1

O exemplo a seguir é uma seção de projeção para um modelo de mapeamento de VTL em que somente os atributos author e id são retornados do DynamoDB:

```
"projection" : {  
  "expression" : "#author, id",  
  "expressionNames" : {  
    "#author" : "author"  
  }  
}
```

Tip

[Você pode acessar seu conjunto de seleção de solicitações do GraphQL usando \\$context.info. selectionSetList](#). Esse campo permite que você ajuste sua expressão de projeção dinamicamente de acordo com seus requisitos.

Note

Ao usar expressões de projeção com as operações Scan e Query, o valor de select deve ser SPECIFIC_ATTRIBUTES. Para obter mais informações, consulte a [documentação do DynamoDB](#).

AWS AppSync referência de modelo de mapeamento de resolvidor para RDS

Os modelos de mapeamento do resolvidor AWS AppSync RDS permitem que os desenvolvedores enviem consultas SQL para uma API de dados do Amazon Aurora Serverless e recebam o resultado dessas consultas.

Modelo de mapeamento de solicitações

O modelo de mapeamento de solicitação do RDS é bastante simples:

```
{
  "version": "2018-05-29",
  "statements": [],
  "variableMap": {},
  "variableTypeHintMap": {}
}
```

Veja aqui a representação do esquema JSON do modelo de mapeamento da solicitação do RDS, uma vez resolvido:

```
{
  "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"
  }
}

```

Veja a seguir um exemplo de modelo de mapeamento de solicitação com uma consulta estática:

```

{
  "version": "2018-05-29",
  "statements": [
    "select title, isbn13 from BOOKS where author = 'Mark Twain'"
  ]
}

```

Versão

Comum a todos os modelos de mapeamento de solicitação, o campo de versão define a versão usada pelo modelo. O campo de versão é obrigatório. O valor "2018-05-29" é a única versão compatível com os modelos de mapeamento do Amazon RDS.

```
"version": "2018-05-29"
```

Declarações e VariableMap

A matriz de instruções é um espaço reservado para as consultas fornecidas pelo desenvolvedor. Atualmente, oferecemos suporte a até duas consultas por modelo de mapeamento de solicitação. O `variableMap` é um campo opcional que contém aliases que podem ser usados para tornar as instruções SQL mais curtas e legíveis. O seguinte exemplo é possível:

```
{
  "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 usará os valores do mapa variável para criar [SqlParameterized](#) as consultas que serão enviadas para a API de dados sem servidor do Amazon Aurora. As instruções SQL são executadas com parâmetros fornecidos no mapa de variáveis, o que elimina o risco de injeção de SQL.

VariableTypeHintMap

`variableTypeHintMap` é um campo opcional contendo tipos de aliases que podem ser usados para enviar dicas de tipos de [parâmetros SQL](#). Essas dicas de tipo evitam a conversão explícita nas instruções SQL, tornando-as mais curtas. O seguinte exemplo é possível:

```
{
```

```
"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 usará o valor do mapa variável para criar as consultas que são enviadas para a API de dados sem servidor do Amazon Aurora. Ele também usa os dados de `variableTypeHintMap` e envia as informações do tipo para o RDS. O `typeHints` com suporte a RDS pode ser encontrado [aqui](#).

AWS AppSync referência do modelo de mapeamento do resolvidor para OpenSearch

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

O AWS AppSync resolvidor do Amazon OpenSearch Service permite que você use o GraphQL para armazenar e recuperar dados em domínios de OpenSearch serviço existentes em sua conta. Esse resolvidor funciona permitindo que você mapeie uma solicitação de entrada do GraphQL em uma solicitação de serviço e, em OpenSearch seguida, mapeie a resposta do serviço de volta para OpenSearch o GraphQL. Esta seção descreve os modelos de mapeamento para as operações OpenSearch de serviço suportadas.

Modelo de mapeamento de solicitações

A maioria dos modelos de mapeamento de solicitações de OpenSearch serviço tem uma estrutura comum em que apenas algumas partes mudam. O exemplo a seguir executa uma pesquisa em um domínio OpenSearch de serviço, onde os documentos são organizados em um índice chamadopost. Os parâmetros de pesquisa são definidos na seção body, com muitas das cláusulas de consulta comuns definidas no campo query. Esse exemplo pesquisará documentos que contêm "Nadia", "Bailey" ou ambos no campo author de um documento:

```
{
  "version": "2017-02-28",
  "operation": "GET",
  "path": "/post/_search",
  "params": {
    "headers": {},
    "queryString": {},
    "body": {
      "from": 0,
      "size": 50,
      "query": {
        "bool": {
          "should": [
            { "match": { "author": "Nadia" } },
            { "match": { "author": "Bailey" } }
          ]
        }
      }
    }
  }
}
```

Modelo de mapeamento de respostas

Assim como em outras fontes de dados, o OpenSearch Service envia uma resposta AWS AppSync que precisa ser convertida em GraphQL.

A maioria das consultas do GraphQL está procurando o `_source` campo a partir de uma OpenSearch resposta de serviço. Como você pode fazer pesquisas para retornar um documento individual ou uma lista de documentos, há dois modelos comuns de mapeamento de respostas usados no OpenSearch Service:

Lista de resultados

```
[
  #foreach($entry in $context.result.hits.hits)
    #if( $velocityCount > 1 ) , #end
    $utils.toJson($entry.get("_source"))
  #end
]
```

Item individual

```
$utils.toJson($context.result.get("_source"))
```

operation field

Note

Isto se aplica somente ao modelo de mapeamento de solicitações.

Método ou verbo HTTP (GET, POST, PUT, HEAD ou DELETE) que é AWS AppSync enviado para o domínio do OpenSearch Serviço. A chave e o valor devem ser strings.

```
"operation" : "PUT"
```

path field

Note

Isto se aplica somente ao modelo de mapeamento de solicitações.

O caminho de pesquisa para uma solicitação de OpenSearch serviço de AWS AppSync. Isso forma um URL para o verbo HTTP da operação. A chave e o valor devem ser strings.

```
"path" : "/<indexname>/_doc/<_id>"
"path" : "/<indexname>/_doc"
"path" : "/<indexname>/_search"
"path" : "/<indexname>/_update/<_id>"
```

Quando o modelo de mapeamento é avaliado, esse caminho é enviado como parte da solicitação HTTP, incluindo o domínio do OpenSearch serviço. Por exemplo, o exemplo anterior pode ser traduzido como:

```
GET https://opensearch-domain-name.REGION.es.amazonaws.com/indexname/type/_search
```

params field

Note

Isto se aplica somente ao modelo de mapeamento de solicitações.

Usado para especificar qual é executada pela pesquisa, geralmente definindo o valor consulta dentro do corpo. No entanto, existem vários outros recursos que podem ser configurados, como a formatação de respostas.

- headers

As informações do cabeçalho, como pares de chave/valor. A chave e o valor devem ser strings. Por exemplo:

```
"headers" : {  
  "Content-Type" : "application/json"  
}
```

Note

AWS AppSync atualmente suporta apenas JSON como a Content-Type

- queryString

Os pares de chave/valor que especificam opções comuns, como formatação de código para respostas JSON. A chave e o valor devem ser strings. Por exemplo, se quiser obter JSON bem formatado, use:

```
"queryString" : {  
  "pretty" : "true"  
}
```

- body

Essa é a parte principal da sua solicitação, permitindo AWS AppSync criar uma solicitação de pesquisa bem formada para o seu domínio OpenSearch de serviço. A chave deve ser uma string composta por um objeto. Algumas demonstrações são mostradas abaixo.

Exemplo 1

Retornar todos os documentos com uma cidade correspondente a "seattle":

```
"body":{
  "from":0,
  "size":50,
  "query" : {
    "match" : {
      "city" : "seattle"
    }
  }
}
```

Exemplo 2

Retornar todos os documentos correspondentes a "washington" como a cidade ou o estado:

```
"body":{
  "from":0,
  "size":50,
  "query" : {
    "multi_match" : {
      "query" : "washington",
      "fields" : ["city", "state"]
    }
  }
}
```

Envio de variáveis

Note

Isto se aplica somente ao modelo de mapeamento de solicitações.

Você também pode enviar variáveis como parte da avaliação na instrução da VTL. Por exemplo, digamos que tenha uma consulta do GraphQL como a seguinte:

```
query {  
  searchForState(state: "washington"){  
    ...  
  }  
}
```

O modelo de mapeamento pode tomar o estado como um argumento:

```
"body":{  
  "from":0,  
  "size":50,  
  "query" : {  
    "multi_match" : {  
      "query" : "$context.arguments.state",  
      "fields" : ["city", "state"]  
    }  
  }  
}
```

Para obter uma lista de utilitários que podem ser incluídos na VTL, consulte [Acessar cabeçalhos da solicitação](#).

AWS AppSync referência de modelo de mapeamento de resolvedor para Lambda

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Você pode usar AWS AppSync funções e resolvedores para invocar funções do Lambda localizadas em sua conta. Você pode moldar suas cargas úteis de solicitação e a resposta de suas funções do Lambda antes de devolvê-las aos seus clientes. Você também pode usar modelos de mapeamento para dar dicas AWS AppSync sobre a natureza da operação a ser invocada. Esta seção descreve os diferentes modelos de mapeamento para as operações do Lambda compatíveis.

Modelo de mapeamento de solicitações

O modelo de mapeamento de solicitações do Lambda manipula campos relacionados à sua função do Lambda:

```
{
  "version": string,
  "operation": Invoke|BatchInvoke,
  "payload": any type,
  "invocationType": RequestResponse|Event
}
```

Esta é a representação do esquema JSON do modelo de mapeamento de solicitações do Lambda quando resolvido:

```
{
  "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
}
```

Aqui está um exemplo que usa uma operação `invoke` com seus dados de carga útil sendo o campo `getPost` de um esquema GraphQL junto com seus argumentos do contexto:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": $util.toJson($context.arguments)
  }
}
```

Todo o documento de mapeamento é passado como entrada para sua função do Lambda; assim, o exemplo anterior agora tem esta aparência:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": {
      "id": "postId1"
    }
  }
}
```

```
}  
}
```

Versão

Comum a todos os modelos de mapeamento de solicitação, `version` define a versão usada pelo modelo. O `version` é obrigatório e é um valor estático:

```
"version": "2018-05-29"
```

Operação

A fonte de dados do Lambda permite que você defina duas operações no campo `operation`: `Invoke` e `BatchInvoke`. A `Invoke` operação permite AWS AppSync que você chame sua função Lambda para cada resolvidor de campo do GraphQL. `BatchInvoke` instrui as solicitações em lote AWS AppSync para o campo GraphQL atual. O campo `operation` é obrigatório.

Para `Invoke`, o modelo de mapeamento da solicitação resolvido corresponde à carga útil de entrada da função do Lambda. Vamos modificar o exemplo acima:

```
{  
  "version": "2018-05-29",  
  "operation": "Invoke",  
  "payload": {  
    "arguments": $util.toJson($context.arguments)  
  }  
}
```

Isso é resolvido e passado para a função do Lambda, que poderá ser mais ou menos assim:

```
{  
  "version": "2018-05-29",  
  "operation": "Invoke",  
  "payload": {  
    "arguments": {  
      "id": "postId1"  
    }  
  }  
}
```

Para BatchInvoke, o modelo de mapeamento é aplicado a cada resolvidor do campo no lote. Para ser conciso, AWS AppSync mescla todos os payload valores do modelo de mapeamento resolvido em uma lista sob um único objeto correspondente ao modelo de mapeamento. O exemplo de modelo a seguir mostra a mesclagem:

```
{
  "version": "2018-05-29",
  "operation": "BatchInvoke",
  "payload": $util.toJson($context)
}
```

Esse modelo é resolvido para o seguinte documento de mapeamento:

```
{
  "version": "2018-05-29",
  "operation": "BatchInvoke",
  "payload": [
    {...}, // context for batch item 1
    {...}, // context for batch item 2
    {...}  // context for batch item 3
  ]
}
```

Cada elemento da lista payload corresponde a um único item de lote. Também se espera que a função do Lambda retorne uma resposta em forma de lista correspondente à ordem dos itens enviados na solicitação:

```
[
  { "data": {...}, "errorMessage": null, "errorType": null }, // result for batch item 1
  { "data": {...}, "errorMessage": null, "errorType": null }, // result for batch item 2
  { "data": {...}, "errorMessage": null, "errorType": null }  // result for batch item 3
]
```

Carga útil

O campo payload é um contêiner usado para passar qualquer JSON bem formado para a função do Lambda. Se o operation campo estiver definido como BatchInvoke, AWS AppSync agrupa os payload valores existentes em uma lista. O campo payload é opcional.

Tipo de invocação

A fonte de dados do Lambda permite que você defina dois tipos de invocação: `RequestResponse` e `Event`. Os tipos de invocação são sinônimos dos tipos de invocação definidos na [API do Lambda](#). O tipo de `RequestResponse` invocação permite AWS AppSync chamar sua função Lambda de forma síncrona para aguardar uma resposta. A invocação `Event` permite que você invoque sua função do Lambda de forma assíncrona. Para obter mais informações sobre como o Lambda lida com solicitações de tipo de invocação `Event`, consulte [Invocação assíncrona](#). O campo `invocationType` é opcional. Se esse campo não for incluído na solicitação, o padrão AWS AppSync será o tipo de `RequestResponse` invocação.

Para qualquer campo `invocationType`, a solicitação resolvida corresponde à carga útil de entrada da função do Lambda. Vamos modificar o exemplo acima:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "invocationType": "Event"
  "payload": {
    "arguments": $util.toJson($context.arguments)
  }
}
```

Isso é resolvido e passado para a função do Lambda, que poderá ser mais ou menos assim:

```
{
  "version": "2018-05-29",
  "operation": "Invoke",
  "invocationType": "Event",
  "payload": {
    "arguments": {
      "id": "postId1"
    }
  }
}
```

Quando a `BatchInvoke` operação é usada em conjunto com o campo do tipo de `Event` invocação, AWS AppSync mescla o resolvidor de campo da mesma forma mencionada acima, e a solicitação é passada para sua função Lambda como um evento assíncrono, sendo uma lista de valores.

`payload` Recomendamos que você desative o cache do resolvidor para resolvidores do tipo de invocação `Event`, pois eles não seriam enviados ao Lambda se houvesse um acerto do cache.

Modelo de mapeamento de respostas

Assim como em outras fontes de dados, sua função Lambda envia uma resposta AWS AppSync que deve ser convertida em um tipo GraphQL.

O resultado da função do Lambda será definido no objeto `context` que está disponível por meio da propriedade `$context.result` do Velocity Template Language (VTL).

Se a forma da resposta da função do Lambda corresponder exatamente à forma do tipo do GraphQL, você pode encaminhar a resposta usando o seguinte modelo de mapeamento da resposta:

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

Não existem campos obrigatórios ou restrições de forma que se aplicam ao modelo de mapeamento da resposta. No entanto, como o GraphQL tem vários tipos, o modelo de mapeamento resolvido deve corresponder ao tipo do GraphQL esperado.

Resposta em lote da função do Lambda

Se o campo `operation` estiver definido como `BatchInvoke`, o AWS AppSync espera de volta uma lista de itens da função do Lambda. Para AWS AppSync mapear cada resultado de volta ao item da solicitação original, a lista de respostas deve corresponder em tamanho e ordem. É válido ter itens `null` na lista de respostas; `$ctx.result` será adequadamente definido como nulo.

Resolvedores diretos do Lambda

Se você quiser contornar totalmente o uso de modelos de mapeamento, AWS AppSync pode fornecer uma carga padrão para sua função Lambda e uma resposta padrão da função Lambda para um tipo GraphQL. Você pode optar por fornecer um modelo de solicitação, um modelo de resposta ou nenhum dos dois e AWS AppSync tratá-los adequadamente.

Modelo de mapeamento da solicitação direta do Lambda

Quando o modelo de mapeamento da solicitação não for fornecido, AWS AppSync enviará o `Context` objeto diretamente para sua função Lambda como uma `Invoke` operação. Para obter

mais informações sobre a estrutura do código-fonte do objeto `Context`, consulte [AWS AppSync referência de contexto do modelo de mapeamento do resolvidor](#).

Modelo de mapeamento da resposta direta do Lambda

Quando o modelo de mapeamento de resposta não é fornecido, AWS AppSync faz uma das duas coisas ao receber a resposta da função Lambda. Se você não forneceu um modelo de mapeamento de solicitação ou forneceu um modelo de mapeamento de solicitação com a versão 2018-05-29, a resposta será equivalente ao seguinte modelo de mapeamento de resposta:

```
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type, $ctx.result)
#end
$util.toJson($ctx.result)
```

Se você forneceu um modelo com a versão 2017-02-28, a lógica de resposta funcionará de forma equivalente ao seguinte modelo de mapeamento de resposta:

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

Superficialmente, o desvio do modelo de mapeamento opera de forma semelhante ao uso de determinados modelos de mapeamento, conforme mostrado nos exemplos anteriores. No entanto, nos bastidores, a avaliação dos modelos de mapeamento é totalmente contornada. Como a etapa de avaliação do modelo é ignorada, em alguns cenários, as aplicações podem ter menos sobrecarga e latência durante a resposta em relação a uma função do Lambda com um modelo de mapeamento de resposta que precisa ser avaliado.

Tratamento personalizado de erros nas respostas do resolvidor direto do Lambda

Você pode personalizar as respostas de erro das funções do Lambda que os resolvidores diretos do Lambda invocam gerando uma exceção personalizada. O exemplo a seguir demonstra como criar uma exceção personalizada usando JavaScript:

```
class CustomException extends Error {
  constructor(message) {
    super(message);
    this.name = "CustomException";
  }
}
```

```
throw new CustomException("Custom message");
```

Quando exceções são geradas, `errorType` e `errorMessage` é `name` `emessage`, respectivamente, do erro personalizado que é gerado.

Se `errorType` estiver `UnauthorizedException`, AWS AppSync retorna a mensagem padrão ("You are not authorized to make this call.") em vez de uma mensagem personalizada.

Este é um exemplo de resposta do GraphQL que demonstra um `errorType` personalizado:

```
{
  "data": {
    "query": null
  },
  "errors": [
    {
      "path": [
        "query"
      ],
      "data": null,
      "errorType": "CustomException",
      "errorInfo": null,
      "locations": [
        {
          "line": 5,
          "column": 10,
          "sourceName": null
        }
      ],
      "message": "Custom Message"
    }
  ]
}
```

Resolvedores diretos do Lambda: agrupamento em lotes ativado

Você pode habilitar o agrupamento em lotes para seu resolvedor direto do Lambda configurando `maxBatchSize` no seu resolvedor. Quando `maxBatchSize` é definido com um valor maior do que 0 para um resolvedor do Direct Lambda, AWS AppSync envia solicitações em lotes para sua função Lambda em tamanhos de até `maxBatchSize`.

Definir `maxBatchSize` como `0` em um resolvedor direto do Lambda desativa o agrupamento em lotes.

Para obter mais informações sobre como funciona o agrupamento em lotes com os resolvedores do Lambda, consulte [Caso de uso avançado: agrupamento em lotes](#).

Modelo de mapeamento de solicitações

Quando o agrupamento em lotes está ativado e o modelo de mapeamento da solicitação não é fornecido, AWS AppSync envia uma lista de Context objetos como uma BatchInvoke operação diretamente para sua função Lambda.

Modelo de mapeamento de respostas

Quando o agrupamento em lote está ativado e o modelo de mapeamento de resposta não é fornecido, a lógica de resposta é equivalente ao seguinte modelo de mapeamento de resposta:

```
#if( $context.result && $context.result.errorMessage )
    $utils.error($context.result.errorMessage, $context.result.errorType,
    $context.result.data)
#else
    $utils.toJson($context.result.data)
#end
```

A função do Lambda deve retornar uma lista de resultados na mesma ordem da lista de objetos Context que foram enviados. É possível retornar erros individuais fornecendo `errorMessage` e `errorType` para um resultado específico. Cada resultado na lista deve estar no seguinte formato:

```
{
  "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

Outros campos no objeto de resultado são ignorados.

Manuseio de erros do Lambda

Você pode retornar um erro para todos os resultados lançando uma exceção ou um erro na sua função do Lambda. Se a solicitação de payload ou o tamanho da resposta para sua solicitação em lote for muito grande, o Lambda retornará um erro. Nesse caso, você deve considerar reduzir `maxBatchSize` ou o tamanho do payload da resposta.

Para obter informações sobre como lidar com erros individuais, consulte [Retornar erros individuais](#).

Amostra de função do Lambda

Usando o esquema abaixo, você pode criar um resolvidor direto do Lambda para o resolvidor de campo `Post.relatedPosts` e habilitar o agrupamento em lote definindo `maxBatchSize` acima de `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]
}
```

Na consulta a seguir, a função do Lambda será chamada com lotes de solicitações para resolver `relatedPosts`:

```
query getAllPosts {  
  allPosts {  
    id  
    relatedPosts {  
      id  
    }  
  }  
}
```

Confira abaixo uma implementação simples de uma função do Lambda:

```
const posts = {  
  1: {  
    id: '1',  
    title: 'First book',  
    author: 'Author1',  
    url: 'https://amazon.com/',  
    content:  
      'SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT  
AUTHOR 1 SAMPLE TEXT AUTHOR 1 SAMPLE TEXT AUTHOR 1',  
    ups: '100',  
    downs: '10',  
  },  
  2: {  
    id: '2',  
    title: 'Second book',  
    author: 'Author2',  
    url: 'https://amazon.com',  
    content: 'SAMPLE TEXT AUTHOR 2 SAMPLE TEXT AUTHOR 2 SAMPLE TEXT',  
    ups: '100',  
    downs: '10',  
  },  
  3: { id: '3', title: 'Third book', author: 'Author3', url: null, content: null, ups:  
null, downs: null },  
  4: {  
    id: '4',  
    title: 'Fourth book',  
    author: 'Author4',  
    url: 'https://www.amazon.com/',  
    content:  
      'SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT  
AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT AUTHOR 4 SAMPLE TEXT  
AUTHOR 4',  
  },  
}
```

```

    ups: '1000',
    downs: '0',
  },
  5: {
    id: '5',
    title: 'Fifth book',
    author: 'Author5',
    url: 'https://www.amazon.com/',
    content: 'SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE TEXT AUTHOR 5 SAMPLE
TEXT AUTHOR 5 SAMPLE TEXT',
    ups: '50',
    downs: '0',
  },
}

const relatedPosts = {
  1: [posts['4']],
  2: [posts['3'], posts['5']],
  3: [posts['2'], posts['1']],
  4: [posts['2'], posts['1']],
  5: [],
}

exports.handler = async (event) => {
  console.log('event ->', event)
  // retrieve the ID of each post
  const ids = event.map((context) => context.source.id)
  // fetch the related posts for each post id
  const related = ids.map((id) => relatedPosts[id])

  // return the related posts; or an error if none were found
  return related.map((r) => {
    if (r.length > 0) {
      return { data: r }
    } else {
      return { data: null, errorMessage: 'Not found', errorType: 'ERROR' }
    }
  })
}

```

AWS AppSync referência do modelo de mapeamento do resolvidor para EventBridge

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

O modelo de mapeamento do AWS AppSync resolvidor usado com a fonte de EventBridge dados permite que você envie eventos personalizados para o EventBridge barramento da Amazon.

Modelo de mapeamento de solicitações

O modelo de mapeamento de PutEvents solicitações permite que você envie vários eventos personalizados para um barramento de EventBridge eventos. O documento de mapeamento possui a seguinte estrutura:

```
{
  "version" : "2018-05-29",
  "operation" : "PutEvents",
  "events" : [{}]
```

Veja a seguir um exemplo de um modelo de mapeamento de solicitações para EventBridge:

```
{
  "version": "2018-05-29",
  "operation": "PutEvents",
  "events": [{
    "source": "com.mycompany.myapp",
    "detail": {
      "key1" : "value1",
      "key2" : "value2"
    },
    "detailType": "myDetailType1"
  },
  {
    "source": "com.mycompany.myapp",
```

```

    "detail": {
      "key3" : "value3",
      "key4" : "value4"
    },
    "detailType": "myDetailType2",
    "resources" : ["Resource1", "Resource2"],
    "time" : "2023-01-01T00:30:00.000Z"
  }
]
}

```

Modelo de mapeamento de respostas

Se a `PutEvents` operação for bem-sucedida, a resposta de `EventBridge` será incluída em `$ctx.result`:

```

#if($ctx.error)
  $util.error($ctx.error.message, $ctx.error.type, $ctx.result)
#end
$util.toJson($ctx.result)

```

Erros que ocorrem durante a execução de operações `PutEvents`, como `InternalExceptions` ou `Timeouts`, aparecerão em `$ctx.error`. Para obter uma lista `EventBridge` dos erros comuns, consulte a [referência de erros EventBridge comuns](#).

O `result` estará no seguinte formato:

```

{
  "Entries" [
    {
      "ErrorCode" : String,
      "ErrorMessage" : String,
      "EventId" : String
    }
  ],
  "FailedEntryCount" : number
}

```

- Entradas

Os resultados do evento ingerido, tanto bem-sucedidos quanto com erro. Se a ingestão foi bem-sucedida, a entrada contém EventID. Caso contrário, você pode usar ErrorCode e ErrorMessage para identificar o problema com a entrada.

Para cada registro, o índice do elemento de resposta é igual ao índice na matriz de solicitações.

- FailedEntryCount

O número de entradas com falha. Esse valor é representado como um número inteiro.

Para obter mais informações sobre a resposta do PutEvents, consulte [PutEvents](#).

Exemplo de resposta de amostra 1

O exemplo a seguir é uma operação PutEvents com dois eventos bem-sucedidos:

```
{
  "Entries" : [
    {
      "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
    },
    {
      "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
    }
  ],
  "FailedEntryCount" : 0
}
```

Exemplo de resposta de amostra 2

O exemplo a seguir é uma operação PutEvents com três eventos, dois bem-sucedidos e um com falha:

```
{
  "Entries" : [
    {
      "EventId": "11710aed-b79e-4468-a20b-bb3c0c3b4860"
    },
    {
      "EventId": "d804d26a-88db-4b66-9eaf-9a11c708ae82"
    },
    {

```

```
        "ErrorCode" : "SampleErrorCode",
        "ErrorMessage" : "Sample Error Message"
    }
],
"FailedEntryCount" : 1
}
```

Campos de PutEvents

PutEvents contém os seguintes campos do modelo de mapeamento:

- Versão

Comum a todos os modelos de mapeamento de solicitação, `version` define a versão usada pelo modelo. Este campo é obrigatório. O valor `2018-05-29` é a única versão compatível com os modelos de EventBridge mapeamento.

- Operation

A única operação suportada é PutEvents. Essa operação permite adicionar eventos personalizados ao seu barramento de eventos.

- Eventos

Uma série de eventos que serão adicionados ao barramento de eventos. Essa matriz deve ter uma alocação de 1 a 10 itens.

O objeto Event é um objeto JSON válido que tem os seguintes campos:

- "source": uma string que define a origem do evento.
- "detail": um objeto JSON pode ser usado para anexar informações sobre o evento. Esse campo pode ser um mapa vazio (`{ }`).
- "detailType": um código que identifica o tipo de evento.
- "resources": uma matriz JSON de strings que identifica os recursos envolvidos no evento. Esse campo pode ser uma matriz vazia.
- "time": o carimbo de data/hora do evento fornecido como string. Isso deve seguir o formato do [RFC3339](#) carimbo de data/hora.

Os trechos abaixo são alguns exemplos de objetos Event válidos:

Exemplo 1


```
{
  "source" : "source1",
  "detail" : {
    "key1" : [1,2,3,4],
    "key2" : "strval"
  },
  "detailType" : "sampleDetailType",
  "resources" : ["Resource1", "Resource2"],
  "time" : "2022-01-10T05:00:10Z"
}
```

Exemplo 2

```
{
  "source" : "source1",
  "detail" : {},
  "detailType" : "sampleDetailType"
}
```

Exemplo 3

```
{
  "source" : "source1",
  "detail" : {
    "key1" : 1200
  },
  "detailType" : "sampleDetailType",
  "resources" : []
}
```

AWS AppSync referência do modelo de mapeamento do resolvidor para fonte **None** de dados

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

O modelo de mapeamento do AWS AppSync resolvidor usado com a fonte de dados do tipo None permite que você defina solicitações para operações AWS AppSync locais.

Modelo de mapeamento de solicitações

O modelo de mapeamento é simples e permite enviar o máximo possível de informações de contexto por meio do campo `payload`.

```
{
  "version": string,
  "payload": any type
}
```

Veja aqui a representação do esquema JSON do modelo de mapeamento da solicitação, uma vez resolvido:

```
{
  "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
}
```

Veja aqui um exemplo em que os argumentos do campo são enviados por meio da propriedade de contexto VTL `$context.arguments`:

```
{
  "version": "2018-05-29",
  "payload": $util.toJson($context.arguments)
}
```

O valor do campo `payload` será encaminhado para o modelo de mapeamento de resposta e disponível na propriedade de contexto VTL (`$context.result`).

Esse é um exemplo que representa o valor interpolado do campo `payload`:

```
{
  "id": "postId1"
}
```

Versão

Comum a todos os modelos de mapeamento da solicitação, `version` define a versão usada pelo modelo.

O campo `version` é obrigatório.

Exemplo: .

```
"version": "2018-05-29"
```

Carga útil

O campo `payload` é um contêiner que pode ser usado para enviar qualquer JSON bem formado ao modelo de mapeamento da resposta.

O campo `payload` é opcional.

Modelo de mapeamento de respostas

Como não há fonte de dados, o valor do campo `payload` será encaminhado ao modelo de mapeamento da resposta e definido no objeto `context` que está disponível por meio da propriedade `$context.result` VTL.

Se a forma do valor de campo `payload` corresponder exatamente à forma do tipo do GraphQL, você pode encaminhar a resposta usando o seguinte modelo de mapeamento da resposta:

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

Não existem campos obrigatórios ou restrições de forma que se aplicam ao modelo de mapeamento da resposta. No entanto, como o GraphQL tem vários tipos, o modelo de mapeamento resolvido deve corresponder ao tipo do GraphQL esperado.

AWS AppSync referência do modelo de mapeamento do resolvidor para HTTP

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Os modelos de mapeamento do resolvidor AWS AppSync HTTP permitem que você envie solicitações de AWS AppSync qualquer endpoint HTTP e respostas do seu endpoint HTTP de volta para. AWS AppSync Ao usar modelos de mapeamento, você pode fornecer dicas AWS AppSync sobre a natureza da operação a ser invocada. Esta seção descreve os diferentes modelos de mapeamento para o resolvidor HTTP compatível.

Modelo de mapeamento de solicitações

```
{
  "version": "2018-05-29",
  "method": "PUT|POST|GET|DELETE|PATCH",
  "params": {
    "query": Map,
    "headers": Map,
    "body": any
  },
  "resourcePath": string
}
```

Depois que o modelo de mapeamento da solicitação HTTP for resolvido, a representação do esquema JSON do modelo de mapeamento da solicitação será semelhante ao seguinte:

```
{
```

```
"$id": "https://aws.amazon.com/appsync/request-mapping-template.json",
"type": "object",
"properties": {
  "version": {
    "$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"
]
}

```

Veja a seguir um exemplo de uma solicitação HTTP POST, com um corpo text/plain:

```

{
  "version": "2018-05-29",
  "method": "POST",
  "params": {
    "headers": {
      "Content-Type": "text/plain"
    },
    "body": "this is an example of text body"
  },
  "resourcePath": "/"
}

```

Versão

Note

Isto se aplica somente ao modelo de mapeamento de solicitações.

Define a versão usada pelo modelo. `version` é comum a todos os modelos de mapeamento da solicitação e é necessária.

```
"version": "2018-05-29"
```

Método

Note

Isto se aplica somente ao modelo de mapeamento de solicitações.

Método ou verbo HTTP (GET, POST, PUT, PATCH ou DELETE) que é AWS AppSync enviado para o endpoint HTTP.

```
"method": "PUT"
```

ResourcePath

Note

Isto se aplica somente ao modelo de mapeamento de solicitações.

O caminho do recurso que você deseja acessar. Junto com o endpoint na fonte de dados HTTP, o caminho do recurso forma a URL para a qual o AWS AppSync serviço faz uma solicitação.

```
"resourcePath": "/v1/users"
```

Quando o modelo de mapeamento é avaliado, esse caminho é enviado como parte da solicitação HTTP, incluindo o endpoint HTTP. Por exemplo, o exemplo anterior pode ser traduzido para o seguinte:

```
PUT <endpoint>/v1/users
```

Campo Params

Note

Isto se aplica somente ao modelo de mapeamento de solicitações.

Usado para especificar qual ação é executada pela pesquisa, geralmente definindo o valor consulta dentro do corpo. No entanto, existem vários outros recursos que podem ser configurados, como a formatação de respostas.

headers

As informações do cabeçalho, como pares de chave/valor. A chave e o valor devem ser strings.

Por exemplo:

```
"headers" : {  
  "Content-Type" : "application/json"  
}
```

Atualmente, os cabeçalhos Content-Type compatíveis são:

```
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
```

Observação: não é possível definir os seguintes cabeçalhos HTTP:


```
HOST
CONNECTION
USER-AGENT
EXPECTATION
TRANSFER_ENCODING
CONTENT_LENGTH
```

query

Os pares de chave/valor que especificam opções comuns, como formatação de código para respostas JSON. A chave e o valor devem ser strings. O exemplo a seguir mostra como enviar uma string de consulta como `?type=json`:

```
"query" : {
  "type" : "json"
}
```

body

O corpo contém o corpo da solicitação HTTP escolhido para definição. O corpo da solicitação sempre é uma string codificada em UTF-8, a menos que o tipo de conteúdo especifique o conjunto de caracteres.

```
"body": "body string"
```

Resposta

Veja um exemplo [aqui](#).

Autoridades de certificação (CA) reconhecidas AWS AppSync por quatro endpoints HTTPS

Note

O Let's Encrypt é aceito por meio dos certificados `identrust` e `isrgrootx1`. Nenhuma ação de sua parte é necessária se você usar o Let's Encrypt.

No momento, certificados autoassinados não são compatíveis com resolvedores HTTP ao usar HTTPS. AWS AppSync reconhece as seguintes autoridades de certificação ao resolver certificados SSL/TLS para HTTPS:

Certificados raiz conhecidos em AWS AppSync

Nome	Data	SHA1 Impressão digital
digicertassuredidrootca	21 de abr de 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
trustcenterclass2caii	21 de abr de 2018	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
thawtepremiumserverca	21 de abr de 2018	E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66
cia-crt-g3-02-ca	23 de nov de 2016	96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09
swisssignplatinumg2ca	21 de abr de 2018	56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66
swisssignsilverg2ca	21 de abr de 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
thawteserverca	21 de abr de 2018	9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
equifaxsecurebusinessca1	21 de abr de 2018	AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
securetrustca	21 de abr de 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
utnuserfirstclientauthemailca	21 de abr de 2018	B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
thawtepersonalfreemailca	21 de abr de 2018	E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2

Nome	Data	SHA1 Impressão digital
affirmtrustnetworkingca	21 de abr de 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
entrustevca	21 de abr de 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
utnuserfirsthardwareca	21 de abr de 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
certumca	21 de abr de 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
addtrustclass1ca	21 de abr de 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
entrustrootcag2	21 de abr de 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
equifaxsecureca	21 de abr de 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
quovadisrootca3	21 de abr de 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
quovadisrootca2	21 de abr de 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
digicertglobalrootg2	21 de abr de 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
digicerthighassuranceevrootca	21 de abr de 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
secomvalicertclass1ca	21 de abr de 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
equifaxsecureglobalbusinessca1	21 de abr de 2018	3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36

Nome	Data	SHA1 Impressão digital
geotrustuniversalca	21 de abril de 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
deprecateditsecca	27 de janeiro de 2012	12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
verisignclass3ca	21 de abril de 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
thawteprimaryrootca3	21 de abril de 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
thawteprimaryrootca2	21 de abril de 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
deutschetelekomrootca2	21 de abril de 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
buypassclass3ca	21 de abril de 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
utnuserfirstobjectca	21 de abril de 2018	E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
geotrustprimaryca	21 de abril de 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
buypassclass2ca	21 de abril de 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
baltimorecodesigningca	21 de abril de 2018	30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD:49:27:08:7C:60:56:7B:0D
verisignclass1ca	21 de abril de 2018	CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45:3E:64:09:EA:E8:7D:60:F1

Nome	Data	SHA1 Impressão digital
baltimorecybertrustca	21 de abril de 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
starfieldclass2ca	21 de abril de 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
camerfirmachamberscommerceca	21 de abril de 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
ttelesecglobalrootclass3ca	21 de abril de 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
verisignclass3g5ca	21 de abril de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
ttelesecglobalrootclass2ca	21 de abril de 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
trustcenteruniversalcai	21 de abril de 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
verisignclass3g4ca	21 de abril de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
verisignclass3g3ca	21 de abril de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
xrampglobalca	21 de abril de 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
amzninternalrootca	12 de dezembro de 2008	A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
certplusclass3ppriaryca	21 de abril de 2018	21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79

Nome	Data	SHA1 Impressão digital
certumtrustednetworkca	21 de abril de 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
verisignclass3g2ca	21 de abril de 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
globalsignr3ca	21 de abril de 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
utndatacorpsgccca	21 de abril de 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
secomscrootca2	21 de abril de 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
gtecybertrustglobalca	21 de abril de 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
secomscrootca1	21 de abril de 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
affirmtrustcommercialca	21 de abril de 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
trustcenterclass4caii	21 de abril de 2018	A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
verisignuniversalrootca	21 de abril de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
globalsignr2ca	21 de abril de 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
certplusclass2primaryca	21 de abril de 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
digicertglobalrootca	21 de abril de 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36

Nome	Data	SHA1 Impressão digital
globalsignca	21 de abr de 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
thawteprimaryrootca	21 de abr de 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
starfieldrootg2ca	21 de abr de 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
geotrustglobalca	21 de abr de 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
soneraclass2ca	21 de abr de 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
verisigntsaca	21 de abr de 2018	20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
soneraclass1ca	21 de abr de 2018	07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
quovadisrootca	21 de abr de 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
affirmtrustpremiumeccca	21 de abr de 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
starfieldservicesrootg2ca	21 de abr de 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
valicertclass2ca	21 de abr de 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
comodoaaaca	21 de abr de 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
aolrootca2	21 de abr de 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84

Nome	Data	SHA1 Impressão digital
keynectisrootca	21 de abr de 2018	9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
addtrustqualifiedca	21 de abr de 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
aolrootca1	21 de abr de 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
verisignclass2g3ca	21 de abr de 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
addtrustexternalca	21 de abr de 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
verisignclass2g2ca	21 de abr de 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
geotrustprimarycag3	21 de abr de 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycag2	21 de abr de 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
swisssigngoldg2ca	21 de abr de 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
entrust2048ca	21 de abr de 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
chunghwaepkirootca	21 de abr de 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
camerfirmachambersignca	21 de abr de 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
camerfirmachambersca	21 de abr de 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C

Nome	Data	SHA1 Impressão digital
godaddyclass2ca	21 de abr de 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
affirmtrustpremiumca	21 de abr de 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
verisignclass1g3ca	21 de abr de 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
secomevrootca1	21 de abr de 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
verisignclass1g2ca	21 de abr de 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
amzninternalinfosecag3	27 de fevereiro de 2015	B9:B1:CA:38:F7:BF:9C:D2:D4:95:E7:B6:5E:75:32:9B:A8:78:2E:F6
cia-crt-g3-01-ca	23 de nov de 2016	2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
godaddyrootg2ca	21 de abr de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
digicertassuredidrootca	21 de abr de 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
microseceszignorootca2009	21 de abr de 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
affirmtrustcommercial	21 de abr de 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
comodoecccertificationauthority	21 de abr de 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11

Nome	Data	SHA1 Impressão digital
cadisigrootr2	21 de abr de 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
swisssignsilvercag2	21 de abr de 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
securetrustca	21 de abr de 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
cadisigrootr1	21 de abr de 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
accvraiz1	21 de abr de 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
entrustrootcertificationauthority	21 de abr de 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
camerfirmaglobalchambersignroot	21 de abr de 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
dstacescax6	21 de abr de 2018	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
identrustpublicsectorrootca1	21 de abr de 2018	BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
starfieldrootcertificateauthorityg2	21 de abr de 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
secureglobalca	21 de abr de 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
eecertificationcenterrootca	21 de abr de 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
opentrustrootcag3	21 de abr de 2018	6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6

Nome	Data	SHA1 Impressão digital
teliasonerarootcav1	21 de abr de 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
autoridaddecertificacionfirmaprofesionalcifa62634068	21 de abr de 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
opentrustrootcag2	21 de abr de 2018	79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
opentrustrootcag1	21 de abr de 2018	79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E
globalsigneccrootcar5	21 de abr de 2018	1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
globalsigneccrootcar4	21 de abr de 2018	69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
izenpecom	21 de abr de 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
turktrustelektroniksertifikaahizmetseglayicisih5	21 de abr de 2018	C4:18:F6:4D:46:D1:DF:00:3D:27:30:13:72:43:A9:12:11:C6:75:FB
gdcatrustauthr5root	21 de abr de 2018	0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
dtrustrootclass3ca22009	21 de abr de 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
quovadisrootca3	21 de abr de 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85

Nome	Data	SHA1 Impressão digital
quovadisrootca2	21 de abr de 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
geotrustprimarycertificatio nauthorityg3	21 de abr de 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycertificatio nauthorityg2	21 de abr de 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
oistewisekeyglobal rootgbca	21 de abr de 2018	0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
addtrustexternalro ot	21 de abr de 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
chambersofcommerce root2008	21 de abr de 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
digicertglobalroot g3	21 de abr de 2018	7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E
comodoaaaservicesr oot	21 de abr de 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
digicertglobalroot g2	21 de abr de 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
certinomisrootca	21 de abr de 2018	9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C:01:B9:32:C5:34:E7:88:A8
oistewisekeyglobal rootgaca	21 de abr de 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
dstrootcax3	21 de abr de 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13

Nome	Data	SHA1 Impressão digital
certigna	21 de abr de 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
digicerthighassuranc eevrootca	21 de abr de 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
soneraclass2rootca	21 de abr de 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
trustcorrootcertca 2	21 de abr de 2018	B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
usertrustrsacertif icationauthority	21 de abr de 2018	2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
trustcorrootcertca 1	21 de abr de 2018	FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
geotrustuniversalc a	21 de abr de 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
certsignrootca	21 de abr de 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
amazonrootca4	21 de abr de 2018	F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
amazonrootca3	21 de abr de 2018	0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
amazonrootca2	21 de abr de 2018	5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
verisignuniversalr ootcertif icationauthority	21 de abr de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54

Nome	Data	SHA1 Impressão digital
amazonrootca1	21 de abr de 2018	8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
networksolutionscertificatauthority	21 de abr de 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
thawteprimaryrootca3	21 de abr de 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
affirmtrustnetworking	21 de abr de 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
thawteprimaryrootca2	21 de abr de 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
trustcoreca1	21 de abr de 2018	58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
deutschetelekomrootca2	21 de abr de 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
godaddyrootcertificateauthorityg2	21 de abr de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
entrustrootcertificateauthorityec1	21 de abr de 2018	20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
szafirrootca2	21 de abr de 2018	E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
tubitakkamusmsslkksertifikasisurum1	21 de abr de 2018	31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA

Nome	Data	SHA1 Impressão digital
buypassclass3rootca	21 de abr de 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
comodorsacertificationauthority	21 de abr de 2018	AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
netlockaranyclassg oldfotanusitvany	21 de abr de 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
securitycommunicationrootca2	21 de abr de 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
dtrustrootclass3ca 2ev2009	21 de abr de 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
starfieldclass2ca	21 de abr de 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
pscprocert	21 de abr de 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
actalisauthenticationrootca	21 de abr de 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
staatdernederlandenrootcag3	21 de abr de 2018	D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC
cfcae/root	21 de abr de 2018	E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
digicerttrustedrootg4	21 de abr de 2018	DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F:C8:3A:4D:7D:77:5D:05:E4
staatdernederlandenrootcag2	21 de abr de 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
securitycommunicationevrootca1	21 de abr de 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D

Nome	Data	SHA1 Impressão digital
globalsignrootcar3	21 de abr de 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
globalsignrootcar2	21 de abr de 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
certumtrustednetwo rkca2	21 de abr de 2018	D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
acraizfnmtrcm	21 de abr de 2018	EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
hellenicacademican dresearch instituti onseccrootca2015	21 de abr de 2018	9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
certplusrootcag2	21 de abr de 2018	4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A
twcarootcertificat ionauthority	21 de abr de 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
twcaglobalrootca	21 de abr de 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
certplusrootcag1	21 de abr de 2018	22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
geotrustuniversalc a2	21 de abr de 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
baltimorecybertrus troot	21 de abr de 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
buypassclass2rootc a	21 de abr de 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99

Nome	Data	SHA1 Impressão digital
certumtrustednetworkca	21 de abr de 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
digicertassuredidrootg3	21 de abr de 2018	F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
digicertassuredidrootg2	21 de abr de 2018	A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F
isrgrootx1	21 de abr de 2018	CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
entrustnetpremium2048secureserverca	21 de abr de 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
certplusclass2primaryca	21 de abr de 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
digicertglobalrootca	21 de abr de 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
entrustrootcertificationauthorityg2	21 de abr de 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
starfieldservicesrootcertificationauthorityg2	21 de abr de 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
thawteprimaryrootca	21 de abr de 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
atostrustedroot2011	21 de abr de 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
geotrustglobalca	21 de abr de 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12

Nome	Data	SHA1 Impressão digital
luxtrustglobalroot2	21 de abr de 2018	1E:0E:56:19:0A:D1:8B:25:98:B2:04:44:FF:66:8A:04:17:99:5F:3F
etugracertificatio nauthority	21 de abr de 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
visaecommercerooroot	21 de abr de 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
quovadisrootca	21 de abr de 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
identrustcommercia lrootca1	21 de abr de 2018	DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
staatdernederlande nevrootca	21 de abr de 2018	76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
ttelesecglobalroot class3	21 de abr de 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
ttelesecglobalroot class2	21 de abr de 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
comodocertificatio nauthority	21 de abr de 2018	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
securitycommunicat ionrootca	21 de abr de 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
quovadisrootca3g3	21 de abr de 2018	48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
xrampglobalcaroot	21 de abr de 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
securesignrootca11	21 de abr de 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3

Nome	Data	SHA1 Impressão digital
affirmtrustpremium	21 de abr de 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
globalsignrootca	21 de abr de 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
swisssigngoldcag2	21 de abr de 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
quovadisrootca2g3	21 de abr de 2018	09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
affirmtrustpremium ecc	21 de abr de 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
geotrustprimarycer tificatio nauthority	21 de abr de 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
quovadisrootca1g3	21 de abr de 2018	1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67
hongkongpostrootca 1	21 de abr de 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
usertrustecccertif icationauthority	21 de abr de 2018	D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
cybertrustglobalro ot	21 de abr de 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
godaddyclass2ca	21 de abr de 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4

Nome	Data	SHA1 Impressão digital
hellenicacademican dresearch instituti onsrootca2015	21 de abr de 2018	01:0C:06:95:A6:98:19:14:FF:BF:5F:C6: B0:B6:95:EA:29:E9:12:A6
ecacc	21 de abr de 2018	28:90:3A:63:5B:52:80:FA:E6:77:4C:0B: 6D:A7:D6:BA:A6:4A:F2:E8
hellenicacademican dresearch instituti onsrootca2011	21 de abr de 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51: 2E:AC:DA:58:09:48:22:4D
verisignclass3publ icprimary certifica tionauthorityg5	21 de abr de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD: 56:BE:3D:9B:67:44:A5:E5
verisignclass3publ icprimary certifica tionauthorityg4	21 de abr de 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	21 de abr de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3: 39:E2:55:76:60:9B:5C:C6
trustisfpsrootca	21 de abr de 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22: 93:D9:DF:F5:4B:81:C0:04
epkirootcertificat ionauthority	21 de abr de 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4: 56:4B:CF:E2:3D:69:C6:F0
globalchambersignr oot2008	21 de abr de 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52: A1:2C:5B:29:F6:D6:AA:0C

Nome	Data	SHA1 Impressão digital
camerfirmachambers ofcommercerooot	21 de abr de 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0: DB:72:2E:31:30:61:F0:B1
mozillacert81.pem	13 de março de 2014	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28: A2:59:3A:19:A7:0F:06:9E
mozillacert99.pem	13 de março de 2014	F1:7F:6F:B6:31:DC:99:E3:A3:C8:7F:FE: 1C:F1:81:10:88:D9:60:33
mozillacert145.pem	13 de março de 2014	10:1D:FA:3F:D5:0B:CB:BB:9B:B5:60:0C: 19:55:A4:1A:F4:73:3A:04
mozillacert37.pem	13 de março de 2014	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68: 37:58:2D:C4:AC:FD:94:97
mozillacert4.pem	13 de março de 2014	E3:92:51:2F:0A:CF:F5:05:DF:F6:DE:06: 7F:75:37:E1:65:EA:57:4B
mozillacert70.pem	13 de março de 2014	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50: BA:9E:A8:7E:FE:9A:CE:3C
mozillacert88.pem	13 de março de 2014	FE:45:65:9B:79:03:5B:98:A1:61:B5:51: 2E:AC:DA:58:09:48:22:4D
mozillacert134.pem	13 de março de 2014	70:17:9B:86:8C:00:A4:FA:60:91:52:22: 3F:9F:3E:32:BD:E0:05:62

Nome	Data	SHA1 Impressão digital
mozillacert26.pem	13 de março de 2014	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
mozillacert77.pem	13 de março de 2014	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert123.pem	13 de março de 2014	2A:B6:28:48:5E:78:FB:F3:AD:9E:79:10:DD:6B:DF:99:72:2C:96:E5
mozillacert15.pem	13 de março de 2014	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
mozillacert66.pem	13 de março de 2014	DD:E1:D2:A9:01:80:2E:1D:87:5E:84:B3:80:7E:4B:B1:FD:99:41:34
mozillacert112.pem	13 de março de 2014	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
mozillacert55.pem	13 de março de 2014	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
mozillacert101.pem	13 de março de 2014	99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39
mozillacert119.pem	13 de março de 2014	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE

Nome	Data	SHA1 Impressão digital
mozillacert44.pem	13 de março de 2014	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA: 4A:9A:C6:22:2B:CC:34:C6
mozillacert108.pem	13 de março de 2014	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81: F2:15:01:52:A4:1D:82:9C
mozillacert95.pem	13 de março de 2014	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD: C7:C2:81:A5:BC:A9:64:57
mozillacert141.pem	13 de março de 2014	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E: 4B:57:E8:B7:D8:F1:FC:A6
mozillacert33.pem	13 de março de 2014	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8: 90:8F:FD:28:86:65:64:7D
mozillacert0.pem	13 de março de 2014	97:81:79:50:D8:1C:96:70:CC:34:D8:09: CF:79:44:31:36:7E:F4:74
mozillacert84.pem	13 de março de 2014	D3:C0:63:F2:19:ED:07:3E:34:AD:5D:75: 0B:32:76:29:FF:D5:9A:F2
mozillacert130.pem	13 de março de 2014	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB: 8C:E8:6A:81:10:9F:E4:8E
mozillacert148.pem	13 de março de 2014	04:83:ED:33:99:AC:36:08:05:87:22:ED: BC:5E:46:00:E3:BE:F9:D7

Nome	Data	SHA1 Impressão digital
mozillacert22.pem	13 de março de 2014	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
mozillacert7.pem	13 de março de 2014	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
mozillacert73.pem	13 de março de 2014	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
mozillacert137.pem	13 de março de 2014	4A:65:D5:F4:1D:EF:39:B8:B8:90:4A:4A:D3:64:81:33:CF:C7:A1:D1
mozillacert11.pem	13 de março de 2014	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
mozillacert29.pem	13 de março de 2014	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
mozillacert62.pem	13 de março de 2014	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert126.pem	13 de março de 2014	25:01:90:19:CF:FB:D9:99:1C:B7:68:25:74:8D:94:5F:30:93:95:42
mozillacert18.pem	13 de março de 2014	79:98:A3:08:E1:4D:65:85:E6:C2:1E:15:3A:71:9F:BA:5A:D3:4A:D9

Nome	Data	SHA1 Impressão digital
mozillacert51.pem	13 de março de 2014	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6: BF:03:FD:E8:7C:4B:2F:9B
mozillacert69.pem	13 de março de 2014	2F:78:3D:25:52:18:A7:4A:65:39:71:B5: 2C:A2:9C:45:15:6F:E9:19
mozillacert115.pem	13 de março de 2014	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62: 32:17:65:CF:17:D8:94:E9
mozillacert40.pem	13 de março de 2014	80:25:EF:F4:6E:70:C8:D4:72:24:65:84: FE:40:3B:8A:8D:6A:DB:F5
mozillacert58.pem	13 de março de 2014	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57: 8B:51:9A:99:E6:10:D7:B0
mozillacert104.pem	13 de março de 2014	4F:99:AA:93:FB:2B:D1:37:26:A1:99:4A: CE:7F:F0:05:F2:93:5D:1E
mozillacert91.pem	13 de março de 2014	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22: 93:D9:DF:F5:4B:81:C0:04
mozillacert47.pem	13 de março de 2014	1B:4B:39:61:26:27:6B:64:91:A2:68:6D: D7:02:43:21:2D:1F:1D:96
mozillacert80.pem	13 de março de 2014	B8:23:6B:00:2F:1D:16:86:53:01:55:6C: 11:A4:37:CA:EB:FF:C3:BB

Nome	Data	SHA1 Impressão digital
mozillacert98.pem	13 de março de 2014	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7: 25:EB:AF:C3:7B:27:CC:D7
mozillacert144.pem	13 de março de 2014	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A: B7:41:10:B4:F2:E4:9A:27
mozillacert36.pem	13 de março de 2014	23:88:C9:D3:71:CC:9E:96:3D:FF:7D:3C: A7:CE:FC:D6:25:EC:19:0D
mozillacert3.pem	13 de março de 2014	87:9F:4B:EE:05:DF:98:58:3B:E3:60:D6: 33:E7:0D:3F:FE:98:71:AF
mozillacert87.pem	13 de março de 2014	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC: 19:19:C3:73:34:B9:C7:74
mozillacert133.pem	13 de março de 2014	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44: 22:00:46:13:DB:17:92:84
mozillacert25.pem	13 de março de 2014	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD: 56:BE:3D:9B:67:44:A5:E5
mozillacert76.pem	13 de março de 2014	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80: DC:E9:6E:2C:C7:B2:78:B7
mozillacert122.pem	13 de março de 2014	02:FA:F3:E2:91:43:54:68:60:78:57:69: 4D:F5:E4:5B:68:85:18:68

Nome	Data	SHA1 Impressão digital
mozillacert14.pem	13 de março de 2014	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A: E6:D3:8F:1A:61:C7:DC:25
mozillacert65.pem	13 de março de 2014	69:BD:8C:F4:9C:D3:00:FB:59:2E:17:93: CA:55:6A:F3:EC:AA:35:FB
mozillacert111.pem	13 de março de 2014	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32: 52:55:60:13:F5:AD:AF:65
mozillacert129.pem	13 de março de 2014	E6:21:F3:35:43:79:05:9A:4B:68:30:9D: 8A:2F:74:22:15:87:EC:79
mozillacert54.pem	13 de março de 2014	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B: 20:D2:D9:32:3A:4C:2A:FD
mozillacert100.pem	13 de março de 2014	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B: 6D:29:D3:FF:8D:5F:00:F0
mozillacert118.pem	13 de março de 2014	7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D: 47:B4:40:CA:D9:0A:19:45
mozillacert151.pem	13 de março de 2014	AC:ED:5F:65:53:FD:25:CE:01:5F:1F:7A: 48:3B:6A:74:9F:61:78:C6
mozillacert43.pem	13 de março de 2014	F9:CD:0E:2C:DA:76:24:C1:8F:BD:F0:F0: AB:B6:45:B8:F7:FE:D5:7A

Nome	Data	SHA1 Impressão digital
mozillacert107.pem	13 de março de 2014	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA: EC:2B:47:56:51:1A:52:C6
mozillacert94.pem	13 de março de 2014	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6: C7:6B:EB:C6:0B:12:40:99
mozillacert140.pem	13 de março de 2014	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20: 88:80:48:39:19:93:7C:F7
mozillacert32.pem	13 de março de 2014	60:D6:89:74:B5:C2:65:9E:8A:0F:C1:88: 7C:88:D2:46:69:1B:18:2C
mozillacert83.pem	13 de março de 2014	A0:73:E5:C5:BD:43:61:0D:86:4C:21:13: 0A:85:58:57:CC:9C:EA:46
mozillacert147.pem	13 de março de 2014	58:11:9F:0E:12:82:87:EA:50:FD:D9:87: 45:6F:4F:78:DC:FA:D6:D4
mozillacert21.pem	13 de março de 2014	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25: 93:DF:A7:F0:40:D1:1D:CB
mozillacert39.pem	13 de março de 2014	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21: FE:68:5D:79:42:21:15:6E
mozillacert6.pem	13 de março de 2014	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0: D7:77:70:02:8F:20:EE:E4

Nome	Data	SHA1 Impressão digital
mozillacert72.pem	13 de março de 2014	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62: A7:9F:45:C2:54:FD:E6:8B
mozillacert136.pem	13 de março de 2014	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2: F1:F1:60:17:64:D8:E3:49
mozillacert10.pem	13 de março de 2014	5F:3A:FC:0A:8B:64:F6:86:67:34:74:DF: 7E:A9:A2:FE:F9:FA:7A:51
mozillacert28.pem	13 de março de 2014	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C: BA:6A:BE:D1:F7:BD:EF:7B
mozillacert61.pem	13 de março de 2014	E0:B4:32:2E:B2:F6:A5:68:B6:54:53:84: 48:18:4A:50:36:87:43:84
mozillacert79.pem	13 de março de 2014	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F: 7D:6A:06:65:26:32:28:27
mozillacert125.pem	13 de março de 2014	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37: D4:4D:F5:D4:67:49:52:F9
mozillacert17.pem	13 de março de 2014	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC: CD:DB:79:D1:53:FB:90:1D
mozillacert50.pem	13 de março de 2014	8C:96:BA:EB:DD:2B:07:07:48:EE:30:32: 66:A0:F3:98:6E:7C:AE:58

Nome	Data	SHA1 Impressão digital
mozillacert68.pem	13 de março de 2014	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07: 5A:9A:E8:00:B7:F7:B6:FA
mozillacert114.pem	13 de março de 2014	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0: 0D:6D:A3:62:8F:C3:52:39
mozillacert57.pem	13 de março de 2014	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F: CB:34:6E:B2:58:B2:8A:58
mozillacert103.pem	13 de março de 2014	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75: D7:01:9F:99:B0:3D:50:74
mozillacert90.pem	13 de março de 2014	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A: CE:19:2B:DD:C7:8E:9C:AC
mozillacert46.pem	13 de março de 2014	40:9D:4B:D9:17:B5:5C:27:B6:9B:64:CB: 98:22:44:0D:CD:09:B8:89
mozillacert97.pem	13 de março de 2014	85:37:1C:A6:E5:50:14:3D:CE:28:03:47: 1B:DE:3A:09:E8:F8:77:0F
mozillacert143.pem	13 de março de 2014	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38: 0F:C6:56:8F:5D:AC:B2:F7
mozillacert35.pem	13 de março de 2014	2A:C8:D5:8B:57:CE:BF:2F:49:AF:F2:FC: 76:8F:51:14:62:90:7A:41

Nome	Data	SHA1 Impressão digital
mozillacert2.pem	13 de março de 2014	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7: CF:8A:2D:64:C9:3F:6C:3A
mozillacert86.pem	13 de março de 2014	74:2C:31:92:E6:07:E4:24:EB:45:49:54: 2B:E1:BB:C5:3E:61:74:E2
mozillacert132.pem	13 de março de 2014	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4: F0:7D:21:D8:05:0B:56:6A
mozillacert24.pem	13 de março de 2014	59:AF:82:79:91:86:C7:B4:75:07:CB:CF: 03:57:46:EB:04:DD:B7:16
mozillacert9.pem	13 de março de 2014	F4:8B:11:BF:DE:AB:BE:94:54:20:71:E6: 41:DE:6B:BE:88:2B:40:B9
mozillacert75.pem	13 de março de 2014	D2:32:09:AD:23:D3:14:23:21:74:E4:0D: 7F:9D:62:13:97:86:63:3A
mozillacert121.pem	13 de março de 2014	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37: 9F:CD:12:EB:24:E3:94:9D
mozillacert139.pem	13 de março de 2014	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA: BC:07:62:01:00:89:76:C9
mozillacert13.pem	13 de março de 2014	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B: A9:03:D0:06:B7:97:09:91

Nome	Data	SHA1 Impressão digital
mozillacert64.pem	13 de março de 2014	62:7F:8D:78:27:65:63:99:D2:7D:7F:90: 44:C9:FE:B3:F3:3E:FA:9A
mozillacert110.pem	13 de março de 2014	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91: 16:52:28:78:BC:53:64:17
mozillacert128.pem	13 de março de 2014	A9:E9:78:08:14:37:58:88:F2:05:19:B0: 6D:2B:0D:2B:60:16:90:7D
mozillacert53.pem	13 de março de 2014	7F:8A:B0:CF:D0:51:87:6A:66:F3:36:0F: 47:C8:8D:8C:D3:35:FC:74
mozillacert117.pem	13 de março de 2014	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88: 2C:78:DB:28:52:CA:E4:74
mozillacert150.pem	13 de março de 2014	33:9B:6B:14:50:24:9B:55:7A:01:87:72: 84:D9:E0:2F:C3:D2:D8:E9
mozillacert42.pem	13 de março de 2014	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD: D6:13:30:FD:8C:DE:37:BF
mozillacert106.pem	13 de março de 2014	E7:A1:90:29:D3:D5:52:DC:0D:0F:C6:92: D3:EA:88:0D:15:2E:1A:6B
mozillacert93.pem	13 de março de 2014	31:F1:FD:68:22:63:20:EE:C6:3B:3F:9D: EA:4A:3E:53:7C:7C:39:17

Nome	Data	SHA1 Impressão digital
mozillacert31.pem	13 de março de 2014	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50: B6:56:3B:8E:2D:93:C3:11
mozillacert49.pem	13 de março de 2014	61:57:3A:11:DF:0E:D8:7E:D5:92:65:22: EA:D0:56:D7:44:B3:23:71
mozillacert82.pem	13 de março de 2014	2E:14:DA:EC:28:F0:FA:1E:8E:38:9A:4E: AB:EB:26:C0:0A:D3:83:C3
mozillacert146.pem	13 de março de 2014	21:FC:BD:8E:7F:6C:AF:05:1B:D1:B3:43: EC:A8:E7:61:47:F2:0F:8A
mozillacert20.pem	13 de março de 2014	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6: 45:25:3A:6F:9F:1A:27:61
mozillacert38.pem	13 de março de 2014	CB:A1:C5:F8:B0:E3:5E:B8:B9:45:12:D3: F9:34:A2:E9:06:10:D3:36
mozillacert5.pem	13 de março de 2014	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30: 54:F3:4C:52:B7:E5:58:C6
mozillacert71.pem	13 de março de 2014	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52: A1:2C:5B:29:F6:D6:AA:0C
mozillacert89.pem	13 de março de 2014	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D: 7E:57:67:F3:14:95:73:9D

Nome	Data	SHA1 Impressão digital
mozillacert135.pem	13 de março de 2014	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
mozillacert27.pem	13 de março de 2014	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
mozillacert60.pem	13 de março de 2014	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
mozillacert78.pem	13 de março de 2014	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
mozillacert124.pem	13 de março de 2014	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
mozillacert16.pem	13 de março de 2014	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
mozillacert67.pem	13 de março de 2014	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
mozillacert113.pem	13 de março de 2014	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
mozillacert56.pem	13 de março de 2014	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2

Nome	Data	SHA1 Impressão digital
mozillacert102.pem	13 de março de 2014	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
mozillacert45.pem	13 de março de 2014	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert109.pem	13 de março de 2014	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
mozillacert96.pem	13 de março de 2014	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
mozillacert142.pem	13 de março de 2014	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
mozillacert34.pem	13 de março de 2014	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
mozillacert1.pem	13 de março de 2014	23:E5:94:94:51:95:F2:41:48:03:B4:D5:64:D2:A3:A3:F5:D8:8B:8C
mozillacert85.pem	13 de março de 2014	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
mozillacert131.pem	13 de março de 2014	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79

Nome	Data	SHA1 Impressão digital
mozillacert149.pem	13 de março de 2014	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0: DB:72:2E:31:30:61:F0:B1
mozillacert23.pem	13 de março de 2014	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2: 99:29:5C:75:6C:81:7B:81
mozillacert8.pem	13 de março de 2014	3E:2B:F7:F2:03:1B:96:F3:8C:E6:C4:D8: A8:5D:3E:2D:58:47:6A:0F
mozillacert74.pem	13 de março de 2014	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A: FF:22:D8:63:E8:25:6F:3F
mozillacert120.pem	13 de março de 2014	DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97: FE:2F:9D:F5:B7:D1:8A:41
mozillacert138.pem	13 de março de 2014	E1:9F:E3:0E:8B:84:60:9E:80:9B:17:0D: 72:A8:C5:BA:6E:14:09:BD
mozillacert12.pem	13 de março de 2014	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D: 40:C6:DD:2F:B1:9C:54:36
mozillacert63.pem	13 de março de 2014	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37: 7D:54:DA:91:E1:01:31:8E
mozillacert127.pem	13 de março de 2014	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1: A3:49:A7:F9:96:2A:82:12

Nome	Data	SHA1 Impressão digital
mozillacert19.pem	13 de março de 2014	B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB: B3:94:BD:63:7B:A7:82:B7
mozillacert52.pem	13 de março de 2014	8B:AF:4C:9B:1D:F0:2A:92:F7:DA:12:8E: B9:1B:AC:F4:98:60:4B:6F
mozillacert116.pem	13 de março de 2014	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7: 6A:46:4B:55:06:02:AC:21
mozillacert41.pem	13 de março de 2014	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C: CE:BB:9D:D9:4F:4E:39:F3
mozillacert59.pem	13 de março de 2014	36:79:CA:35:66:87:72:30:4D:30:A5:FB: 87:3B:0F:A7:7B:B7:0D:54
mozillacert105.pem	13 de março de 2014	77:47:4F:C6:30:E4:0F:4C:47:64:3F:84: BA:B8:C6:95:4A:8A:41:EC
mozillacert92.pem	13 de março de 2014	A3:F1:33:3F:E2:42:BF:CF:C5:D1:4E:8F: 39:42:98:40:68:10:D1:A0
mozillacert30.pem	13 de março de 2014	E7:B4:F6:9D:61:EC:90:69:DB:7E:90:A7: 40:1A:3C:F4:7D:4F:E8:EE
mozillacert48.pem	13 de março de 2014	A0:A1:AB:90:C9:FC:84:7B:3B:12:61:E8: 97:7D:5F:D3:22:61:D3:CC

Nome	Data	SHA1 Impressão digital
verisignc4g2.pem	20 de março de 2014	0B:77:BE:BB:CB:7A:A2:47:05:DE:CC:0F: BD:6A:02:FC:7A:BD:9B:52
verisignc2g3.pem	20 de março de 2014	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0: C3:59:12:AF:9F:EB:63:11
verisignc1g6.pem	31 de dezembro de 2014	51:7F:61:1E:29:91:6B:53:82:FB:72:E7: 44:D9:8D:C3:CC:53:6D:64
verisignc2g2.pem	20 de março de 2014	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95: B6:CC:A0:08:1B:67:EC:9D
verisignroot.pem	20 de março de 2014	36:79:CA:35:66:87:72:30:4D:30:A5:FB: 87:3B:0F:A7:7B:B7:0D:54
verisignc2g1.pem	20 de março de 2014	67:82:AA:E0:ED:EE:E2:1A:58:39:D3:C0: CD:14:68:0A:4F:60:14:2A
verisignc3g5.pem	20 de março de 2014	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD: 56:BE:3D:9B:67:44:A5:E5
verisignc1g3.pem	20 de março de 2014	20:42:85:DC:F7:EB:76:41:95:57:8E:13: 6B:D4:B7:D1:E9:8E:46:A5
verisignc3g4.pem	20 de março de 2014	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7: CF:8A:2D:64:C9:3F:6C:3A

Nome	Data	SHA1 Impressão digital
verisignc1g2.pem	20 de março de 2014	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B: 56:16:7F:62:F5:32:E5:47
verisignc3g3.pem	20 de março de 2014	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3: 39:E2:55:76:60:9B:5C:C6
verisignc1g1.pem	20 de março de 2014	90:AE:A2:69:85:FF:14:80:4C:43:49:52: EC:E9:60:84:77:AF:55:6F
verisignc3g2.pem	20 de março de 2014	85:37:1C:A6:E5:50:14:3D:CE:28:03:47: 1B:DE:3A:09:E8:F8:77:0F
verisignc3g1.pem	20 de março de 2014	A1:DB:63:93:91:6F:17:E4:18:55:09:40: 04:15:C7:02:40:B0:AE:6B
verisignc2g6.pem	31 de dezembro de 2014	40:B3:31:A0:E9:BF:E8:55:BC:39:93:CA: 70:4F:4E:C2:51:D4:1D:8F
verisignc4g3.pem	20 de março de 2014	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D: 7E:57:67:F3:14:95:73:9D
gdroot-g2.pem	31 de dezembro de 2014	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62: A7:9F:45:C2:54:FD:E6:8B
gd-class2-root.pem	31 de dezembro de 2014	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0: D7:77:70:02:8F:20:EE:E4

Nome	Data	SHA1 Impressão digital
gd_bundle-g2.pem	31 de dezembro de 2014	27:AC:93:69:FA:F2:52:07:BB:26:27:CE:FA:CC:BE:4E:F9:C3:19:B8
dstacescax6	18 de junho de 2018	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
gd_bundle-g2.pem	18 de junho de 2018	27:AC:93:69:FA:F2:52:07:BB:26:27:CE:FA:CC:BE:4E:F9:C3:19:B8
verisignc4g3.pem	18 de junho de 2018	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
swisssignplatinumg2ca	21 de abr de 2018	56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66
geotrustprimarycertificatio nauthorityg3	18 de junho de 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
geotrustprimarycertificatio nauthorityg2	18 de junho de 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
buypassclass2rootca	18 de junho de 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
camerfirmachambersofcommerceroot	18 de junho de 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1

Nome	Data	SHA1 Impressão digital
mozillacert20.pem	18 de junho de 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert12.pem	18 de junho de 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
mozillacert90.pem	18 de junho de 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
mozillacert82.pem	18 de junho de 2018	2E:14:DA:EC:28:F0:FA:1E:8E:38:9A:4E:AB:EB:26:C0:0A:D3:83:C3
mozillacert140.pem	18 de junho de 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
mozillacert74.pem	18 de junho de 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
mozillacert132.pem	18 de junho de 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
mozillacert66.pem	18 de junho de 2018	DD:E1:D2:A9:01:80:2E:1D:87:5E:84:B3:80:7E:4B:B1:FD:99:41:34
mozillacert124.pem	18 de junho de 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF

Nome	Data	SHA1 Impressão digital
mozillacert58.pem	18 de junho de 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
securitycommunicationrootca2	18 de junho de 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
mozillacert116.pem	18 de junho de 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
mozillacert108.pem	18 de junho de 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
certigna	18 de junho de 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert3.pem	18 de junho de 2018	87:9F:4B:EE:05:DF:98:58:3B:E3:60:D6:33:E7:0D:3F:FE:98:71:AF
verisignc1g1.pem	18 de junho de 2018	90:AE:A2:69:85:FF:14:80:4C:43:49:52:EC:E9:60:84:77:AF:55:6F
verisignc4g2.pem	18 de junho de 2018	0B:77:BE:BB:CB:7A:A2:47:05:DE:CC:0F:BD:6A:02:FC:7A:BD:9B:52
deutschetelekomrootca2	18 de junho de 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF

Nome	Data	SHA1 Impressão digital
starfieldrootg2ca	21 de abr de 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
comodoecccertificationauthority	18 de junho de 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
digicertglobalrootg3	18 de junho de 2018	7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E
digicertglobalrootg2	18 de junho de 2018	DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
mozillacert11.pem	18 de junho de 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
mozillacert81.pem	18 de junho de 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
mozillacert73.pem	18 de junho de 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
szafirrootca2	18 de junho de 2018	E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
mozillacert131.pem	18 de junho de 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79

Nome	Data	SHA1 Impressão digital
ecacc	18 de junho de 2018	28:90:3A:63:5B:52:80:FA:E6:77:4C:0B:6D:A7:D6:BA:A6:4A:F2:E8
mozillacert65.pem	18 de junho de 2018	69:BD:8C:F4:9C:D3:00:FB:59:2E:17:93:CA:55:6A:F3:EC:AA:35:FB
turktrustelektroniksertifikahizmetseglayicisih5	18 de junho de 2018	C4:18:F6:4D:46:D1:DF:00:3D:27:30:13:72:43:A9:12:11:C6:75:FB
mozillacert123.pem	18 de junho de 2018	2A:B6:28:48:5E:78:FB:F3:AD:9E:79:10:DD:6B:DF:99:72:2C:96:E5
mozillacert57.pem	18 de junho de 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
mozillacert115.pem	18 de junho de 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
mozillacert49.pem	18 de junho de 2018	61:57:3A:11:DF:0E:D8:7E:D5:92:65:22:EA:D0:56:D7:44:B3:23:71
mozillacert107.pem	18 de junho de 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
verisignclass3g4ca	21 de abr de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A

Nome	Data	SHA1 Impressão digital
securetrustca	18 de junho de 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
mozillacert2.pem	18 de junho de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
buypassclass2ca	21 de abr de 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
secomscrootca2	21 de abr de 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
secomscrootca1	21 de abr de 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
trustisfpsrootca	18 de junho de 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
hongkongpostrootca1	18 de junho de 2018	D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
certsignrootca	18 de junho de 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
geotrustprimaryca	21 de abr de 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
twcaglobalrootca	18 de junho de 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65

Nome	Data	SHA1 Impressão digital
camerfirmachambersca	21 de abr de 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
mozillacert10.pem	18 de junho de 2018	5F:3A:FC:0A:8B:64:F6:86:67:34:74:DF:7E:A9:A2:FE:F9:FA:7A:51
mozillacert80.pem	18 de junho de 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
mozillacert72.pem	18 de junho de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
comodoaaaca	21 de abr de 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert130.pem	18 de junho de 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
mozillacert64.pem	18 de junho de 2018	62:7F:8D:78:27:65:63:99:D2:7D:7F:90:44:C9:FE:B3:F3:3E:FA:9A
mozillacert122.pem	18 de junho de 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
mozillacert56.pem	18 de junho de 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
equifaxsecureebusinessca1	21 de abr de 2018	AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4

Nome	Data	SHA1 Impressão digital
camerfirmachambers ignca	21 de abr de 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52: A1:2C:5B:29:F6:D6:AA:0C
mozillacert114.pem	18 de junho de 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0: 0D:6D:A3:62:8F:C3:52:39
mozillacert48.pem	18 de junho de 2018	A0:A1:AB:90:C9:FC:84:7B:3B:12:61:E8: 97:7D:5F:D3:22:61:D3:CC
pscprocert	18 de junho de 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75: D7:01:9F:99:B0:3D:50:74
mozillacert106.pem	18 de junho de 2018	E7:A1:90:29:D3:D5:52:DC:0D:0F:C6:92: D3:EA:88:0D:15:2E:1A:6B
mozillacert1.pem	18 de junho de 2018	23:E5:94:94:51:95:F2:41:48:03:B4:D5: 64:D2:A3:A3:F5:D8:8B:8C
eecertificationcen trerootca	18 de junho de 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7: 25:EB:AF:C3:7B:27:CC:D7
digicertglobalroot ca	18 de junho de 2018	A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D: 40:C6:DD:2F:B1:9C:54:36
thawteprimaryrootc ag3	18 de junho de 2018	F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43: 5B:17:15:89:CA:F3:6B:F2

Nome	Data	SHA1 Impressão digital
thawteprimaryrootcag2	18 de junho de 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
entrustrootcertificationauthorityec1	18 de junho de 2018	20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
valicertclass2ca	21 de abr de 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
globalchambersignroot2008	18 de junho de 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
amazonrootca4	18 de junho de 2018	F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
gd-class2-root.pem	18 de junho de 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
amazonrootca3	18 de junho de 2018	0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
amazonrootca2	18 de junho de 2018	5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
securitycommunicationrootca	18 de junho de 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7

Nome	Data	SHA1 Impressão digital
amazonrootca1	18 de junho de 2018	8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
acraizfnmtrcm	18 de junho de 2018	EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
quovadisrootca3g3	18 de junho de 2018	48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
certplusrootcag2	18 de junho de 2018	4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A
certplusrootcag1	18 de junho de 2018	22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
mozillacert71.pem	18 de junho de 2018	4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
mozillacert63.pem	18 de junho de 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
mozillacert121.pem	18 de junho de 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
ttelesecglobalrootclass3ca	21 de abr de 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1

Nome	Data	SHA1 Impressão digital
mozillacert55.pem	18 de junho de 2018	AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
mozillacert113.pem	18 de junho de 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
baltimorecybertrustca	21 de abr de 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
mozillacert47.pem	18 de junho de 2018	1B:4B:39:61:26:27:6B:64:91:A2:68:6D:D7:02:43:21:2D:1F:1D:96
mozillacert105.pem	18 de junho de 2018	77:47:4F:C6:30:E4:0F:4C:47:64:3F:84:BA:B8:C6:95:4A:8A:41:EC
mozillacert39.pem	18 de junho de 2018	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
usertrustecccertificationauthority	18 de junho de 2018	D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
mozillacert0.pem	18 de junho de 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
securitycommunicationevrootca1	18 de junho de 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D

Nome	Data	SHA1 Impressão digital
verisignc3g5.pem	18 de junho de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD: 56:BE:3D:9B:67:44:A5:E5
globalsignr3ca	21 de abr de 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1: 09:26:DF:5B:85:69:76:AD
trustcoreca1	18 de junho de 2018	58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C: 17:4D:8B:84:0B:C8:78:BD
equifaxsecureglobalbusinessca1	21 de abr de 2018	3A:74:CB:7A:47:DB:70:DE:89:1F:24:35: 98:64:B8:2D:82:BD:1A:36
geotrustuniversalca	18 de junho de 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D: 8A:2F:74:22:15:87:EC:79
affirmtrustpremiumca	21 de abr de 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F: 7D:6A:06:65:26:32:28:27
staatdernederlandenrootca3	18 de junho de 2018	D8:EB:6B:41:51:92:59:E0:F3:E7:85:00: C0:3D:B6:88:97:C9:EE:FC
staatdernederlandenrootca2	18 de junho de 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF: 03:57:46:EB:04:DD:B7:16
mozillacert70.pem	18 de junho de 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50: BA:9E:A8:7E:FE:9A:CE:3C
secomevrootca1	21 de abr de 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8: 90:8F:FD:28:86:65:64:7D

Nome	Data	SHA1 Impressão digital
geotrustglobalca	18 de junho de 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1: A3:49:A7:F9:96:2A:82:12
mozillacert62.pem	18 de junho de 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40: 04:15:C7:02:40:B0:AE:6B
mozillacert120.pem	18 de junho de 2018	DA:40:18:8B:91:89:A3:ED:EE:AE:DA:97: FE:2F:9D:F5:B7:D1:8A:41
mozillacert54.pem	18 de junho de 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B: 20:D2:D9:32:3A:4C:2A:FD
mozillacert112.pem	18 de junho de 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92: F6:CF:F6:34:69:87:82:37
mozillacert46.pem	18 de junho de 2018	40:9D:4B:D9:17:B5:5C:27:B6:9B:64:CB: 98:22:44:0D:CD:09:B8:89
swisssigngoldcag2	18 de junho de 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6: 45:25:3A:6F:9F:1A:27:61
mozillacert104.pem	18 de junho de 2018	4F:99:AA:93:FB:2B:D1:37:26:A1:99:4A: CE:7F:F0:05:F2:93:5D:1E
mozillacert38.pem	18 de junho de 2018	CB:A1:C5:F8:B0:E3:5E:B8:B9:45:12:D3: F9:34:A2:E9:06:10:D3:36

Nome	Data	SHA1 Impressão digital
certplusclass3ppri maryca	21 de abr de 2018	21:6B:2A:29:E6:2A:00:CE:82:01:46:D8: 24:41:41:B9:25:11:B2:79
entrustrootcertifi cationauthorityg2	18 de junho de 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8: 1E:57:EF:BB:93:22:72:D4
godaddyrootg2ca	21 de abr de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62: A7:9F:45:C2:54:FD:E6:8B
cfcaeavroot	18 de junho de 2018	E2:B8:29:4B:55:84:AB:6B:58:C2:90:46: 6C:AC:3F:B8:39:8F:84:83
verisignc3g4.pem	18 de junho de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7: CF:8A:2D:64:C9:3F:6C:3A
geotrustuniversalc a2	18 de junho de 2018	37:9A:19:7B:41:85:45:35:0C:A6:03:69: F3:3C:2E:AF:47:4F:20:79
starfieldservicesr ootg2ca	21 de abr de 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A: FF:22:D8:63:E8:25:6F:3F
digicerthighassura nceevrootca	18 de junho de 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A: E6:D3:8F:1A:61:C7:DC:25
entrustnetpremium2 048secureserverca	18 de junho de 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB: E7:92:7D:7D:65:2D:34:31
camerfirmaglobalch ambersignroot	18 de junho de 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72: 84:D9:E0:2F:C3:D2:D8:E9

Nome	Data	SHA1 Impressão digital
verisignclass3g3ca	21 de abr de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
godaddyclass2ca	18 de junho de 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
mozillacert61.pem	18 de junho de 2018	E0:B4:32:2E:B2:F6:A5:68:B6:54:53:84:48:18:4A:50:36:87:43:84
mozillacert53.pem	18 de junho de 2018	7F:8A:B0:CF:D0:51:87:6A:66:F3:36:0F:47:C8:8D:8C:D3:35:FC:74
atostrustedroot2011	18 de junho de 2018	2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
mozillacert111.pem	18 de junho de 2018	9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
staatdernederlande nevrootca	18 de junho de 2018	76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
mozillacert45.pem	18 de junho de 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert103.pem	18 de junho de 2018	70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74

Nome	Data	SHA1 Impressão digital
mozillacert37.pem	18 de junho de 2018	B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
mozillacert29.pem	18 de junho de 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
izenpecom	18 de junho de 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
comodorsacertificationauthority	18 de junho de 2018	AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
mozillacert99.pem	18 de junho de 2018	F1:7F:6F:B6:31:DC:99:E3:A3:C8:7F:FE:1C:F1:81:10:88:D9:60:33
mozillacert149.pem	18 de junho de 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
utnuserfirstobjectca	21 de abr de 2018	E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
verisignc3g3.pem	18 de junho de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
dstrootcax3	18 de junho de 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13

Nome	Data	SHA1 Impressão digital
addtrustexternalroot	18 de junho de 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
certumtrustednetworkca	18 de junho de 2018	07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
affirmtrustpremiumecc	18 de junho de 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
starfieldclass2ca	18 de junho de 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
actalisauthenticationrootca	18 de junho de 2018	F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
verisignclass2g3ca	21 de abril de 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
isrgrootx1	18 de junho de 2018	CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
godaddyrootcertificateauthorityg2	18 de junho de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
mozillacert60.pem	18 de junho de 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
chunghwaepkirootca	21 de abril de 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0

Nome	Data	SHA1 Impressão digital
mozillacert52.pem	18 de junho de 2018	8B:AF:4C:9B:1D:F0:2A:92:F7:DA:12:8E: B9:1B:AC:F4:98:60:4B:6F
microseceszignorootca2009	18 de junho de 2018	89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37: 7D:54:DA:91:E1:01:31:8E
securesignrootca11	18 de junho de 2018	3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8: 5B:B1:C3:65:C7:D8:11:B3
mozillacert110.pem	18 de junho de 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91: 16:52:28:78:BC:53:64:17
mozillacert44.pem	18 de junho de 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA: 4A:9A:C6:22:2B:CC:34:C6
mozillacert102.pem	18 de junho de 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8: 22:79:FE:60:FA:B9:16:83
mozillacert36.pem	18 de junho de 2018	23:88:C9:D3:71:CC:9E:96:3D:FF:7D:3C: A7:CE:FC:D6:25:EC:19:0D
mozillacert28.pem	18 de junho de 2018	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C: BA:6A:BE:D1:F7:BD:EF:7B
baltimorecybertrustroot	18 de junho de 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88: 2C:78:DB:28:52:CA:E4:74

Nome	Data	SHA1 Impressão digital
amzninternalrootca	12 de dezembro de 2008	A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
mozillacert98.pem	18 de junho de 2018	C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
mozillacert148.pem	18 de junho de 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
verisignc3g2.pem	18 de junho de 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
quovadisrootca2g3	18 de junho de 2018	09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
geotrustprimarycertificatio nauthority	18 de junho de 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
opentrustrootcag3	18 de junho de 2018	6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6
opentrustrootcag2	18 de junho de 2018	79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
opentrustrootcag1	18 de junho de 2018	79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E

Nome	Data	SHA1 Impressão digital
verisignclass3ca	21 de abr de 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
globalsignca	21 de abr de 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
ttelesecglobalrootclass2ca	21 de abr de 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
verisignclass1g3ca	21 de abr de 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
verisignuniversalrootca	21 de abr de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
soneraclass2ca	21 de abr de 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
starfieldservicesrootcertificateauthorityg2	18 de junho de 2018	92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
mozillacert51.pem	18 de junho de 2018	FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
mozillacert43.pem	18 de junho de 2018	F9:CD:0E:2C:DA:76:24:C1:8F:BD:F0:F0:AB:B6:45:B8:F7:FE:D5:7A
mozillacert101.pem	18 de junho de 2018	99:A6:9B:E6:1A:FE:88:6B:4D:2B:82:00:7C:B8:54:FC:31:7E:15:39
mozillacert35.pem	18 de junho de 2018	2A:C8:D5:8B:57:CE:BF:2F:49:AF:F2:FC:76:8F:51:14:62:90:7A:41

Nome	Data	SHA1 Impressão digital
globalsignr2ca	21 de abr de 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
mozillacert27.pem	18 de junho de 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
affirmtrustpremium	18 de junho de 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
mozillacert19.pem	18 de junho de 2018	B4:35:D4:E1:11:9D:1C:66:90:A7:49:EB:B3:94:BD:63:7B:A7:82:B7
mozillacert97.pem	18 de junho de 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
netlockaranyclassg oldfotanusitvany	18 de junho de 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
mozillacert89.pem	18 de junho de 2018	C8:EC:8C:87:92:69:CB:4B:AB:39:E9:8D:7E:57:67:F3:14:95:73:9D
verisignroot.pem	18 de junho de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
mozillacert147.pem	18 de junho de 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
aolrootca2	21 de abr de 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84

Nome	Data	SHA1 Impressão digital
cia-crt-g3-01-ca	23 de nov de 2016	2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
aolrootca1	21 de abr de 2018	39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
verisignc3g1.pem	18 de junho de 2018	A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
mozillacert139.pem	18 de junho de 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
soneraclass2rootca	18 de junho de 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
swisssignsilverg2ca	21 de abr de 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
thawteprimaryrootca	18 de junho de 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
gdcatrustauthr5root	18 de junho de 2018	0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
trustcenterclass4caii	21 de abr de 2018	A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
usertrustrsacertificationauthority	18 de junho de 2018	2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E

Nome	Data	SHA1 Impressão digital
digicertassuredidr ootg3	18 de junho de 2018	F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26: 9F:DC:0F:48:2C:AB:30:89
digicertassuredidr ootg2	18 de junho de 2018	A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C: E0:A4:C0:91:93:51:5D:3F
mozillacert50.pem	18 de junho de 2018	8C:96:BA:EB:DD:2B:07:07:48:EE:30:32: 66:A0:F3:98:6E:7C:AE:58
mozillacert42.pem	18 de junho de 2018	85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD: D6:13:30:FD:8C:DE:37:BF
mozillacert100.pem	18 de junho de 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B: 6D:29:D3:FF:8D:5F:00:F0
mozillacert34.pem	18 de junho de 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39: 6A:46:46:B0:44:1B:0F:A9
affirmtrustcommerc ialca	21 de abr de 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80: DC:E9:6E:2C:C7:B2:78:B7
mozillacert26.pem	18 de junho de 2018	87:82:C6:C3:04:35:3B:CF:D2:96:92:D2: 59:3E:7D:44:D9:34:FF:11
globalsigneccrootc ar5	18 de junho de 2018	1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD: 4F:DD:5F:46:3A:1B:69:AA

Nome	Data	SHA1 Impressão digital
globalsigneccrootc ar4	18 de junho de 2018	69:69:56:2E:40:80:F4:24:A1:E7:19:9F: 14:BA:F3:EE:58:AB:6A:BB
buypassclass3rootc a	18 de junho de 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD: C7:C2:81:A5:BC:A9:64:57
mozillacert18.pem	18 de junho de 2018	79:98:A3:08:E1:4D:65:85:E6:C2:1E:15: 3A:71:9F:BA:5A:D3:4A:D9
mozillacert96.pem	18 de junho de 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70: 19:9D:2A:BE:11:E3:81:D1
verisignc2g6.pem	18 de junho de 2018	40:B3:31:A0:E9:BF:E8:55:BC:39:93:CA: 70:4F:4E:C2:51:D4:1D:8F
secomvalicertclass 1ca	21 de abr de 2018	E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB: 8C:E8:6A:81:10:9F:E4:8E
mozillacert88.pem	18 de junho de 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51: 2E:AC:DA:58:09:48:22:4D
accvraiz1	18 de junho de 2018	93:05:7A:88:15:C6:4F:CE:88:2F:FA:91: 16:52:28:78:BC:53:64:17
mozillacert146.pem	18 de junho de 2018	21:FC:BD:8E:7F:6C:AF:05:1B:D1:B3:43: EC:A8:E7:61:47:F2:0F:8A

Nome	Data	SHA1 Impressão digital
mozillacert138.pem	18 de junho de 2018	E1:9F:E3:0E:8B:84:60:9E:80:9B:17:0D:72:A8:C5:BA:6E:14:09:BD
verisignclass3g2ca	21 de abr de 2018	85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
dtrustrootclass3ca2ev2009	18 de junho de 2018	96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
xrampglobalca	21 de abr de 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
mozillacert9.pem	18 de junho de 2018	F4:8B:11:BF:DE:AB:BE:94:54:20:71:E6:41:DE:6B:BE:88:2B:40:B9
verisignuniversalrootcertificationauthority	18 de junho de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
tubitakkamusmsslkoksertifिकासिसुरम1	18 de junho de 2018	31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA
mozillacert41.pem	18 de junho de 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
mozillacert33.pem	18 de junho de 2018	FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
mozillacert25.pem	18 de junho de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5

Nome	Data	SHA1 Impressão digital
mozillacert17.pem	18 de junho de 2018	40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC: CD:DB:79:D1:53:FB:90:1D
mozillacert95.pem	18 de junho de 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD: C7:C2:81:A5:BC:A9:64:57
affirmtrustpremium eccca	21 de abr de 2018	B8:23:6B:00:2F:1D:16:86:53:01:55:6C: 11:A4:37:CA:EB:FF:C3:BB
mozillacert87.pem	18 de junho de 2018	5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC: 19:19:C3:73:34:B9:C7:74
mozillacert145.pem	18 de junho de 2018	10:1D:FA:3F:D5:0B:CB:BB:9B:B5:60:0C: 19:55:A4:1A:F4:73:3A:04
mozillacert79.pem	18 de junho de 2018	D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F: 7D:6A:06:65:26:32:28:27
mozillacert137.pem	18 de junho de 2018	4A:65:D5:F4:1D:EF:39:B8:B8:90:4A:4A: D3:64:81:33:CF:C7:A1:D1
digicertassuredidr ootca	18 de junho de 2018	05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E: 4B:DF:B5:A8:99:B2:4D:43
addtrustqualifiedc a	21 de abr de 2018	4D:23:78:EC:91:95:39:B5:00:7F:75:8F: 03:3B:21:1E:C5:4D:8B:CF
mozillacert129.pem	18 de junho de 2018	E6:21:F3:35:43:79:05:9A:4B:68:30:9D: 8A:2F:74:22:15:87:EC:79

Nome	Data	SHA1 Impressão digital
verisignclass2g2ca	21 de abr de 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95: B6:CC:A0:08:1B:67:EC:9D
baltimorecodesigningca	21 de abr de 2018	30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD: 49:27:08:7C:60:56:7B:0D
luxtrustglobalroot2	18 de junho de 2018	1E:0E:56:19:0A:D1:8B:25:98:B2:04:44: FF:66:8A:04:17:99:5F:3F
visaecommercerooot	18 de junho de 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22: 3F:9F:3E:32:BD:E0:05:62
oistewisekeyglobalrootgbca	18 de junho de 2018	0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8: 35:9E:0C:FD:27:AC:CC:ED
mozillacert8.pem	18 de junho de 2018	3E:2B:F7:F2:03:1B:96:F3:8C:E6:C4:D8: A8:5D:3E:2D:58:47:6A:0F
comodocertificatio nauthority	18 de junho de 2018	66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C: BA:6A:BE:D1:F7:BD:EF:7B
cia-crt-g3-02-ca	23 de nov de 2016	96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D: F0:05:98:F7:E6:C6:6F:09
verisignc1g6.pem	18 de junho de 2018	51:7F:61:1E:29:91:6B:53:82:FB:72:E7: 44:D9:8D:C3:CC:53:6D:64
trustcenterclass2c aii	21 de abr de 2018	AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21: FE:68:5D:79:42:21:15:6E

Nome	Data	SHA1 Impressão digital
quovadisrootca1g3	18 de junho de 2018	1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67
mozillacert40.pem	18 de junho de 2018	80:25:EF:F4:6E:70:C8:D4:72:24:65:84:FE:40:3B:8A:8D:6A:DB:F5
cadisigrootr2	18 de junho de 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
cadisigrootr1	18 de junho de 2018	8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
mozillacert32.pem	18 de junho de 2018	60:D6:89:74:B5:C2:65:9E:8A:0F:C1:88:7C:88:D2:46:69:1B:18:2C
utndatacorpsgccca	21 de abr de 2018	58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
mozillacert24.pem	18 de junho de 2018	59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
addtrustclass1ca	21 de abr de 2018	CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
mozillacert16.pem	18 de junho de 2018	DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
affirmtrustnetworkingca	21 de abr de 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F

Nome	Data	SHA1 Impressão digital
mozillacert94.pem	18 de junho de 2018	49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
mozillacert86.pem	18 de junho de 2018	74:2C:31:92:E6:07:E4:24:EB:45:49:54:2B:E1:BB:C5:3E:61:74:E2
mozillacert144.pem	18 de junho de 2018	37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
mozillacert78.pem	18 de junho de 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
mozillacert136.pem	18 de junho de 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert128.pem	18 de junho de 2018	A9:E9:78:08:14:37:58:88:F2:05:19:B0:6D:2B:0D:2B:60:16:90:7D
verisignclass1g2ca	21 de abr de 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
hellenicacademican dresearch instituti onsrootca2015	18 de junho de 2018	01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
soneraclass1ca	21 de abr de 2018	07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF

Nome	Data	SHA1 Impressão digital
hellenicacademican dresearch instituti onsrootca2011	18 de junho de 2018	FE:45:65:9B:79:03:5B:98:A1:61:B5:51: 2E:AC:DA:58:09:48:22:4D
certumtrustednetwo rkca2	18 de junho de 2018	D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5: FA:76:26:CF:D3:DC:30:92
equifaxsecureca	21 de abr de 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D: 7F:9D:62:13:97:86:63:3A
thawteserverca	21 de abr de 2018	9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E: C9:D4:A5:0D:92:D8:49:79
mozillacert7.pem	18 de junho de 2018	AD:7E:1C:28:B0:64:EF:8F:60:03:40:20: 14:C3:D0:E3:37:0E:B5:8A
affirmtrustnetwork ing	18 de junho de 2018	29:36:21:02:8B:20:ED:02:F5:66:C5:32: D1:D6:ED:90:9F:45:00:2F
deprecateditsecca	27 de janeiro de 2012	12:12:0B:03:0E:15:14:54:F4:DD:B3:F5: DE:13:6E:83:5A:29:72:9D
globalsignrootcar3	18 de junho de 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1: 09:26:DF:5B:85:69:76:AD
globalsignrootcar2	18 de junho de 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F: DD:DE:38:E4:B7:24:2E:FE

Nome	Data	SHA1 Impressão digital
quovadisrootca	18 de junho de 2018	DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
mozillacert31.pem	18 de junho de 2018	9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
entrustrootcertificationauthority	18 de junho de 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert23.pem	18 de junho de 2018	91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
mozillacert15.pem	18 de junho de 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
verisignc2g3.pem	18 de junho de 2018	61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
mozillacert93.pem	18 de junho de 2018	31:F1:FD:68:22:63:20:EE:C6:3B:3F:9D:EA:4A:3E:53:7C:7C:39:17
mozillacert151.pem	18 de junho de 2018	AC:ED:5F:65:53:FD:25:CE:01:5F:1F:7A:48:3B:6A:74:9F:61:78:C6
mozillacert85.pem	18 de junho de 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48

Nome	Data	SHA1 Impressão digital
certplusclass2primaryca	18 de junho de 2018	74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
mozillacert143.pem	18 de junho de 2018	36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
mozillacert77.pem	18 de junho de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
mozillacert135.pem	18 de junho de 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
mozillacert69.pem	18 de junho de 2018	2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
mozillacert127.pem	18 de junho de 2018	DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
mozillacert119.pem	18 de junho de 2018	75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
geotrustprimarycac3	21 de abr de 2018	03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
identrustpublicsectorrootca1	18 de junho de 2018	BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
geotrustprimarycac2	21 de abr de 2018	8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0

Nome	Data	SHA1 Impressão digital
trustcorrootcertca 2	18 de junho de 2018	B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA: 4E:06:34:C7:94:B2:1C:C0
mozillacert6.pem	18 de junho de 2018	27:96:BA:E6:3F:18:01:E2:77:26:1B:A0: D7:77:70:02:8F:20:EE:E4
trustcorrootcertca 1	18 de junho de 2018	FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86: 5C:CA:A8:3A:45:5B:C3:0A
networksolutionsce rtificate authority	18 de junho de 2018	74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90: 3C:21:64:60:20:E5:DF:CE
twcarootcertificat ionauthority	18 de junho de 2018	CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5: A3:7A:A0:76:A9:06:23:48
addtrustexternalca	21 de abr de 2018	02:FA:F3:E2:91:43:54:68:60:78:57:69: 4D:F5:E4:5B:68:85:18:68
verisignclass3g5ca	21 de abr de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD: 56:BE:3D:9B:67:44:A5:E5
autoridaddecertifi cacionfir maprofesi onalcifa62634068	18 de junho de 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07: 5A:9A:E8:00:B7:F7:B6:FA
hellenicacademican dresearch instituti onseccrootca2015	18 de junho de 2018	9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4: BC:6F:84:68:0B:BA:B6:66

Nome	Data	SHA1 Impressão digital
verisigntsaca	21 de abr de 2018	20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
utnuserfirsthardw reca	21 de abr de 2018	04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
identrustcommercia lrootca1	18 de junho de 2018	DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
dtrustrootclass3ca 22009	18 de junho de 2018	58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
epkirootcertificat ionauthority	18 de junho de 2018	67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
mozillacert30.pem	18 de junho de 2018	E7:B4:F6:9D:61:EC:90:69:DB:7E:90:A7:40:1A:3C:F4:7D:4F:E8:EE
teliasonerarootcav 1	18 de junho de 2018	43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
buypassclass3ca	21 de abr de 2018	DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
mozillacert22.pem	18 de junho de 2018	32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
mozillacert14.pem	18 de junho de 2018	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25

Nome	Data	SHA1 Impressão digital
verisignc2g2.pem	18 de junho de 2018	B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95: B6:CC:A0:08:1B:67:EC:9D
certumca	21 de abr de 2018	62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7: 34:8E:06:42:51:B1:81:18
mozillacert92.pem	18 de junho de 2018	A3:F1:33:3F:E2:42:BF:CF:C5:D1:4E:8F: 39:42:98:40:68:10:D1:A0
mozillacert150.pem	18 de junho de 2018	33:9B:6B:14:50:24:9B:55:7A:01:87:72: 84:D9:E0:2F:C3:D2:D8:E9
mozillacert84.pem	18 de junho de 2018	D3:C0:63:F2:19:ED:07:3E:34:AD:5D:75: 0B:32:76:29:FF:D5:9A:F2
ttelesecglobalroot class3	18 de junho de 2018	55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70: 19:9D:2A:BE:11:E3:81:D1
globalsignrootca	18 de junho de 2018	B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81: F2:15:01:52:A4:1D:82:9C
ttelesecglobalroot class2	18 de junho de 2018	59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62: 32:17:65:CF:17:D8:94:E9
mozillacert142.pem	18 de junho de 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0: BE:FD:3A:2D:82:75:51:85

Nome	Data	SHA1 Impressão digital
mozillacert76.pem	18 de junho de 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
mozillacert134.pem	18 de junho de 2018	70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
mozillacert68.pem	18 de junho de 2018	AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
etugracertificatio nauthority	18 de junho de 2018	51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
mozillacert126.pem	18 de junho de 2018	25:01:90:19:CF:FB:D9:99:1C:B7:68:25:74:8D:94:5F:30:93:95:42
keynectisrootca	21 de abr de 2018	9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
mozillacert118.pem	18 de junho de 2018	7E:78:4A:10:1C:82:65:CC:2D:E1:F1:6D:47:B4:40:CA:D9:0A:19:45
quovadisrootca3	18 de junho de 2018	1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
quovadisrootca2	18 de junho de 2018	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7

Nome	Data	SHA1 Impressão digital
mozillacert5.pem	18 de junho de 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
verisignc1g3.pem	18 de junho de 2018	20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
cybertrustglobalroot	18 de junho de 2018	5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
amzninternalinfocag3	27 de fevereiro de 2015	B9:B1:CA:38:F7:BF:9C:D2:D4:95:E7:B6:5E:75:32:9B:A8:78:2E:F6
starfieldrootcertificateauthorityg2	18 de junho de 2018	B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
entrust2048ca	21 de abril de 2018	50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
swisssignsilvercag2	18 de junho de 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
affirmtrustcommercial	18 de junho de 2018	F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
certinomisrootca	18 de junho de 2018	9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C:01:B9:32:C5:34:E7:88:A8

Nome	Data	SHA1 Impressão digital
xrampglobalcaroot	18 de junho de 2018	B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
secureglobalca	18 de junho de 2018	3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
swisssigngoldg2ca	21 de abr de 2018	D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
mozillacert21.pem	18 de junho de 2018	9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
mozillacert13.pem	18 de junho de 2018	06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
verisignc2g1.pem	18 de junho de 2018	67:82:AA:E0:ED:EE:E2:1A:58:39:D3:C0:CD:14:68:0A:4F:60:14:2A
mozillacert91.pem	18 de junho de 2018	3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
oistewisekeyglobal rootgaca	18 de junho de 2018	59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
mozillacert83.pem	18 de junho de 2018	A0:73:E5:C5:BD:43:61:0D:86:4C:21:13:0A:85:58:57:CC:9C:EA:46
entrustevca	21 de abr de 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9

Nome	Data	SHA1 Impressão digital
mozillacert141.pem	18 de junho de 2018	31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
mozillacert75.pem	18 de junho de 2018	D2:32:09:AD:23:D3:14:23:21:74:E4:0D:7F:9D:62:13:97:86:63:3A
mozillacert133.pem	18 de junho de 2018	85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
mozillacert67.pem	18 de junho de 2018	D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
mozillacert125.pem	18 de junho de 2018	B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
mozillacert59.pem	18 de junho de 2018	36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
thawtepremiumserve rca	21 de abr de 2018	E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66
mozillacert117.pem	18 de junho de 2018	D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
utnuserfirstclient authemailca	21 de abr de 2018	B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
entrustrootcag2	21 de abr de 2018	8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4

Nome	Data	SHA1 Impressão digital
mozillacert109.pem	18 de junho de 2018	B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
digicerttrustedrootg4	18 de junho de 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 de junho de 2018	47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
comodoaaaservicesroot	18 de junho de 2018	D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
mozillacert4.pem	18 de junho de 2018	E3:92:51:2F:0A:CF:F5:05:DF:F6:DE:06:7F:75:37:E1:65:EA:57:4B
verisignclass3publicprimarycertificationauthorityg5	18 de junho de 2018	4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
chambersofcommerceroot2008	18 de junho de 2018	78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
verisignclass3publicprimarycertificationauthorityg4	18 de junho de 2018	22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A

Nome	Data	SHA1 Impressão digital
verisignclass3publicprimarycertificationauthorityg3	18 de junho de 2018	13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
thawtepersonalfreemailca	21 de abril de 2018	E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
verisignclg2.pem	18 de junho de 2018	27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
gtecybertrustglobalca	21 de abril de 2018	97:81:79:50:D8:1C:96:70:CC:34:D8:09:CF:79:44:31:36:7E:F4:74
trustcenteruniversalcai	21 de abril de 2018	6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
camerfirmachamberscommerceca	21 de abril de 2018	6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
verisignclass1ca	21 de abril de 2018	CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45:3E:64:09:EA:E8:7D:60:F1

AWS AppSync registro de alterações do modelo de mapeamento do resolvidor

Note

Agora, oferecemos suporte principalmente ao runtime do APPSYNC_JS e sua documentação. Considere usar o runtime do APPSYNC_JS e seus guias disponíveis [aqui](#).

Os resolvidores e modelos de mapeamento de função são versionados. A versão do modelo de mapeamento, como 2018-05-29, determina o seguinte:

- A forma esperada da configuração de solicitação de fonte de dados fornecida pelo modelo de solicitação
- O comportamento de execução do modelo de mapeamento de solicitação e o modelo de mapeamento de resposta

As versões são representadas usando o YYYY-MM-DD formato, uma data posterior corresponde a uma versão mais recente. Esta página lista as diferenças entre as versões do modelo de mapeamento atualmente compatíveis com o AWS AppSync.

Tópicos

- [Disponibilidade de operação da fonte de dados por matriz de versão](#)
- [Alterar a versão em um modelo de mapeamento do resolvido de unidade](#)
- [Alterar a versão em uma função](#)
- [2018-05-29](#)
- [2017-02-28](#)

Disponibilidade de operação da fonte de dados por matriz de versão

Operação/versão compatível	2017-02-28	2018-05-29
AWS Lambda Invocar	Sim	Sim
AWS Lambda BatchInvoke	Sim	Sim
Fonte de dados "none (nenhum)"	Sim	Sim
Amazon OpenSearch GET	Sim	Sim
Amazon OpenSearch POST	Sim	Sim
Amazon OpenSearch PUT	Sim	Sim
Amazon OpenSearch DELETE	Sim	Sim
Amazon OpenSearch GET	Sim	Sim

Operação/versão compatível	2017-02-28	2018-05-29
DynamoDB GetItem	Sim	Sim
Scan do DynamoDB	Sim	Sim
Query do DynamoDB	Sim	Sim
DynamoDB DeleteItem	Sim	Sim
DynamoDB PutItem	Sim	Sim
DynamoDB BatchGetItem	Não	Sim
DynamoDB BatchPutItem	Não	Sim
DynamoDB BatchDeleteItem	Não	Sim
HTTP	Não	Sim
Amazon RDS	Não	Sim

Observação: somente a versão 2018-05-29 é compatível com as funções no momento.

Alterar a versão em um modelo de mapeamento do resolvidor de unidade

Para resolvidores de unidade, a versão é especificada como parte do corpo do modelo de mapeamento de solicitação. Para atualizar a versão, basta atualizar o campo `version` para a nova versão.

Por exemplo, para atualizar a versão no AWS Lambda modelo:

```
{
  "version": "2017-02-28",
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": $utils.toJson($context.arguments)
  }
}
```

Você precisa atualizar o campo de versão de 2017-02-28 para 2018-05-29 da seguinte maneira:

```
{
  "version": "2018-05-29", ## Note the version
  "operation": "Invoke",
  "payload": {
    "field": "getPost",
    "arguments": $utils.toJson($context.arguments)
  }
}
```

Alterar a versão em uma função

Para funções, a versão é especificada como o campo `functionVersion` no objeto de função. Para atualizar a versão, basta atualizar `functionVersion`. Observação: no momento, somente a versão 2018-05-29 é compatível com a função.

Veja a seguir um exemplo de um comando da CLI para atualizar uma versão de função existente:

```
aws appsync update-function \
--api-id REPLACE_WITH_API_ID \
--function-id REPLACE_WITH_FUNCTION_ID \
--data-source-name "PostTable" \
--function-version "2018-05-29" \
--request-mapping-template "{...}" \
--response-mapping-template "\$util.toJson(\$ctx.result)"
```

Observação: é recomendável omitir o campo de versão do modelo de mapeamento de solicitação de função, pois ele não terá efeito. Se você especificar uma versão dentro de um modelo de mapeamento de solicitação de função, o valor da versão será substituído pelo valor do campo `functionVersion`.

2018-05-29

Alteração de comportamento

- Se o resultado da invocação da fonte de dados for `null`, o modelo de mapeamento de resposta será executado.

- Se a invocação da fonte de dados gerar um erro, agora caberá a você lidar com o erro, o resultado avaliado do modelo de mapeamento de resposta será sempre colocado dentro do bloco `data` da resposta do GraphQL.

Reasoning

- Um resultado de invocação `null` tem significado e, em alguns casos de uso do aplicativo, podemos manipular os resultados `null` de maneira personalizada. Por exemplo, um aplicativo pode verificar se existe um registro em uma tabela do Amazon DynamoDB para executar alguma verificação de autorização. Nesse caso, um resultado de invocação `null` significa que o usuário pode não ser autorizado. Agora, a execução do modelo de mapeamento de resposta fornece a capacidade de gerar um erro não autorizado. Esse comportamento fornece maior controle para o designer da API.

Considerando o seguinte modelo de mapeamento de resposta:

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

Antes, com a versão 2017-02-28, se `$ctx.result` retornasse nulo, o modelo de mapeamento de resposta não seria executado. Agora, com a versão 2018-05-29, podemos lidar com esse cenário. Por exemplo, podemos optar por gerar um erro de autorização da seguinte forma:

```
# throw an unauthorized error if the result is null
#if ( $util.isNull($ctx.result) )
    $util.unauthorized()
#end
$util.toJson($ctx.result)
```

Observação: os erros que retornam de uma fonte de dados às vezes não são fatais nem esperados, é por isso que o modelo de mapeamento de resposta deve ter a flexibilidade de tratar o erro de invocação e decidir se deve ignorá-lo, criá-lo novamente ou gerar um erro diferente.

Considerando o seguinte modelo de mapeamento de resposta:

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

Antes, com a versão 2017-02-28, no caso de um erro de invocação, o modelo de mapeamento de resposta era avaliado e o resultado era colocado automaticamente no bloco `errors` da resposta do

GraphQL. Agora, com a versão 2018-05-29, você pode escolher o que fazer com o erro, criá-lo novamente, gerar um erro diferente ou anexar o erro ao retornar os dados.

Criar novamente um erro de invocação

No modelo de resposta a seguir, criamos o mesmo erro que retornou da fonte de dados.

```
#if ( $ctx.error )
    $util.error($ctx.error.message, $ctx.error.type)
#end
$util.toJson($ctx.result)
```

No caso de um erro de invocação, por exemplo, `$ctx.error` estar presente, a resposta será semelhante à seguinte:

```
{
  "data": {
    "getPost": null
  },
  "errors": [
    {
      "path": [
        "getPost"
      ],
      "errorType": "DynamoDB:ConditionalCheckFailedException",
      "message": "Conditional check failed exception...",
      "locations": [
        {
          "line": 5,
          "column": 5
        }
      ]
    }
  ]
}
```

Criar um erro diferente

No modelo de resposta a seguir, criamos nosso próprio erro personalizado depois de processar o erro que retornou da fonte de dados.

```
#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)

```

No caso de um erro de invocação, por exemplo, `$ctx.error` estar presente, a resposta será semelhante à seguinte:

```

{
  "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
        }
      ]
    }
  ]
}

```

Anexar um erro para retornar dados

No modelo de resposta a seguir, acrescentamos o mesmo erro que retornou da fonte de dados ao retornar os dados de volta à resposta. Isso também é conhecido como uma resposta parcial.

```

#if ( $ctx.error )

```

```
$util.appendError($ctx.error.message, $ctx.error.type)
#set($defaultPost = {id: "1", title: 'default post'})
$util.toJson($defaultPost)
#else
  $util.toJson($ctx.result)
#end
```

No caso de um erro de invocação, por exemplo, `$ctx.error` estar presente, a resposta será semelhante à seguinte:

```
{
  "data": {
    "getPost": {
      "id": "1",
      "title": "A post"
    }
  },
  "errors": [
    {
      "path": [
        "getPost"
      ],
      "errorType": "ConditionalCheckFailedException",
      "message": "Conditional check failed exception...",
      "locations": [
        {
          "line": 5,
          "column": 5
        }
      ]
    }
  ]
}
```

Migração da versão 2017-02-28 para a 2018-05-29

A migração da versão 2017-02-28 para a 2018-05-29 é simples. Altere o campo de versão no modelo de mapeamento de solicitação do resolvidor ou no objeto de versão da função. No entanto, observe que a execução da 2018-05-29 se comporta de maneira diferente da 2017-02-28, as alterações estão descritas [aqui](#).

Como preservar o mesmo comportamento de execução da versão 2017-02-28 na 2018-05-29

Em alguns casos, é possível reter o mesmo comportamento de execução da versão 2017-02-28 durante a execução de um modelo com versão 2018-05-29.

Exemplo: DynamoDB PutItem

Dado o seguinte modelo de solicitação do DynamoDB PutItem de 28/02/2017:

```
{
  "version" : "2017-02-28",
  "operation" : "PutItem",
  "key": {
    "foo" : ... typed value,
    "bar" : ... typed value
  },
  "attributeValues" : {
    "baz" : ... typed value
  },
  "condition" : {
    ...
  }
}
```

E o seguinte modelo de resposta:

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

A migração para 2018-05-29 altera esses modelos da seguinte maneira:

```
{
  "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" : {
```



```

    ...
  }
}

```

E altera o modelo de resposta da seguinte forma:

```

## 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)

```

Agora que é sua responsabilidade lidar com os erros, optamos por criar o mesmo erro usando o `$util.error()` que foi retornado do DynamoDB. Você pode adaptar esse snippet para converter seu modelo de mapeamento para 2018-05-29. Observe que, se seu modelo de resposta for diferente, será necessário considerar as alterações de comportamento da execução.

Exemplo: DynamoDB GetItem

Dado o seguinte modelo de solicitação do DynamoDB GetItem de 28/02/2017:

```

{
  "version" : "2017-02-28",
  "operation" : "GetItem",
  "key" : {
    "foo" : ... typed value,
    "bar" : ... typed value
  },
  "consistentRead" : true
}

```

E o seguinte modelo de resposta:

```

## map table attribute postId to field Post.id

```

```
$util.qr($ctx.result.put("id", $ctx.result.get("postId")))

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

A migração para 2018-05-29 altera esses modelos da seguinte maneira:

```
{
  "version" : "2018-05-29", ## Note the new 2018-05-29 version
  "operation" : "GetItem",
  "key" : {
    "foo" : ... typed value,
    "bar" : ... typed value
  },
  "consistentRead" : true
}
```

E altera o modelo de resposta da seguinte forma:

```
## 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)
```

Na versão 2017-02-28, se a invocação da fonte de dados fosse `null`, significando que não há nenhum item na tabela do DynamoDB que corresponda à chave, o modelo de mapeamento de resposta não seria executado. Pode ser aceitável para a maioria dos casos, mas se você esperava que o `$ctx.result` não fosse `null`, agora precisará lidar com esse cenário.

2017-02-28

Características

- Se o resultado da invocação da fonte de dados for `null`, o modelo de mapeamento de resposta não será executado.
- Se a invocação da fonte de dados gerar um erro, o modelo de mapeamento de resposta será executado e o resultado avaliado será colocado dentro do bloco `errors.data` de resposta do GraphQL.

Referência de tipos do GraphQL

Os tipos escalares no GraphQL representam valores de folha primitivos em um esquema do GraphQL. Esses são os tipos de dados mais básicos que são resolvidos como um único valor. Diferentemente dos tipos de objetos, os tipos escalares não podem ter subcampos. O GraphQL vem com um conjunto de tipos escalares padrão:

- Int: um inteiro assinado de 32 bits.
- Float: um valor de ponto flutuante de precisão dupla assinado.
- String: uma sequência de caracteres UTF-8.
- Boolean: um valor verdadeiro ou falso.
- ID: um identificador exclusivo, geralmente usado para buscar novamente um objeto ou como a chave de um cache.

Esses tipos escalares servem como componentes básicos para tipos mais complexos em seu esquema. Eles são usados para definir campos que contêm valores simples e singulares. Além desses escalares integrados, AWS AppSync fornece escalares adicionais para diferentes casos de uso.

Interfaces e uniões no GraphQL são tipos abstratos que permitem um design de esquema flexível e extensível. Eles fornecem mecanismos para agrupar tipos relacionados e permitir consultas polimórficas. Uma interface no GraphQL é um tipo abstrato que define um conjunto de campos que um tipo deve incluir para implementar a interface. Ele serve como um contrato para objetos, especificando um conjunto comum de campos que os tipos de implementação devem ter. As interfaces são úteis quando você deseja retornar um objeto ou um campo que podem ser de vários tipos diferentes, mas ainda têm alguns campos garantidos. Por outro lado, uma união no GraphQL representa um tipo que pode ser um de vários tipos de objetos, mas não define nenhum campo comum entre esses tipos. As uniões são úteis quando você precisa retornar um campo que pode ser de vários tipos, e esses tipos não necessariamente compartilham campos comuns. Tanto as interfaces quanto as uniões são particularmente úteis em cenários em que um campo pode retornar diferentes tipos de dados, permitindo que os clientes consultem campos específicos com base no tipo retornado.

Esta seção é usada como referência para tipos de esquema.

Tópicos

- [Tipos escalares no GraphQL](#)
- [Interfaces e uniões no GraphQL](#)

Tipos escalares no GraphQL

Um tipo de objeto do GraphQL possui um nome e campos, e esses campos podem ter subcampos. Em última análise, os campos de um tipo de objeto devem ser resolvidos em tipos escalares, que representam as folhas da consulta. Para obter mais informações sobre tipos de objetos e escalares, consulte [Esquemas e tipos](#) no site do GraphQL.

Além do conjunto padrão de escalares do GraphQL, AWS AppSync também permite usar os escalares definidos pelo serviço que começam com o prefixo. `AWS`. AWS AppSync não suporta a criação de escalares definidos pelo usuário (personalizados). Você deve usar o padrão ou escalares da AWS.

Você não pode usar `AWS` como prefixo para tipos de objetos personalizados.

A seção a seguir é uma referência para digitação de esquema.

Escalares padrão

O GraphQL define os seguintes escalares padrão:

Lista de escalares padrão

`ID`

Um identificador exclusivo para um objeto. Este escalar é serializado como `String`, mas não foi feito para ser legível por humanos.

`String`

Uma sequência de caracteres UTF-8.

`Int`

Um valor inteiro entre $-(2^{31})$ e $2^{31}-1$.

`Float`

Um valor de ponto flutuante IEEE 754.

Boolean

Um valor booleano, `true` ou `false`.

AWS AppSync escalares

AWS AppSync define os seguintes escalares:

AWS AppSync lista de escalares

AWSDate

Uma string de [data ISO 8601](#) estendida no formato `YYYY-MM-DD`.

AWSTime

Uma string de [horário ISO 8601](#) estendida no formato `hh:mm:ss.sss`.

AWSDateTime

Uma string de [data e horário ISO 8601](#) estendida no formato `YYYY-MM-DDThh:mm:ss.sssZ`.

Note

Os escalares `AWSDate`, `AWSTime` e `AWSDateTime` podem incluir um [deslocamento de fuso horário](#). Por exemplo, os valores `1970-01-01Z`, `1970-01-01-07:00` e `1970-01-01+05:30` são todos válidos para `AWSDate`. O deslocamento do fuso horário deve ser `Z` (UTC) ou um deslocamento em horas e minutos (e, opcionalmente, segundos). Por exemplo, `±hh:mm:ss`. O campo de segundos no deslocamento de fuso horário é considerado válido mesmo que não faça parte do padrão ISO 8601.

AWSTimestamp

Um valor inteiro que representa o número de segundos antes ou depois de `1970-01-01-T00:00Z`.

AWSEmail

Um endereço de email no formato `local-part@domain-part` definido pela [RFC 822](#).

AWSJSON

Uma string JSON. Qualquer estrutura JSON válida é automaticamente analisada e carregada no código do resolvidor como mapas, listas ou valores escalares, em vez de strings de entrada literais. Strings sem aspas ou JSON inválido resultam em um erro de validação do GraphQL.

AWSPhone

Um número de telefone. Esse valor é armazenado como uma string. Os números de telefone podem conter espaços ou hífens para separar grupos de dígitos. Os números de telefone sem código de país são considerados números dos EUA/norte-americanos que aderem ao [Plano de Numeração norte-americano \(NANP\)](#).

AWSURL

Um URL conforme definido pela [RFC 1738](#). Por exemplo, `https://www.amazon.com/dp/B000NZW3KC/` ou `mailto:example@example.com`. URLs deve conter um esquema (`http`, `mailto`) e não pode conter duas barras (`//`) na parte do caminho.

AWSIPAddress

Um IPv6 endereço válido IPv4 ou. IPv4 os endereços são esperados em notação quad-pontilhada (`()`). `123.12.34.56` IPv6 os endereços são esperados em formato sem colchetes e separados por dois pontos (`()`). `1a2b:3c4b::1234:4567` Você pode incluir um sufixo CIDR opcional (`123.45.67.89/16`) para indicar a máscara de sub-rede.

Exemplo de uso de esquema

O exemplo de esquema do GraphQL a seguir usa todos os escalares personalizados como um "objeto" e mostra os modelos de solicitação e resposta do resolvidor para operações básicas de colocação, obtenção e lista. Por fim, o exemplo mostra como você pode usar isso ao executar consultas e mutações.

```
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
}

```

Esta é a aparência de um modelo de solicitação para `putObject`. Um `putObject` usa uma operação `PutItem` para criar ou atualizar um item na tabela do Amazon DynamoDB. Observe que esse trecho de código não tem uma tabela do Amazon DynamoDB configurada como fonte de dados. Ele está sendo usado apenas como exemplo:

```

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

```


O modelo de resposta para `putObject` retorna os resultados:

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

Esta é a aparência de um modelo de solicitação para `getObject`. Um `getObject` usa uma operação `GetItem` para retornar um conjunto de atributos para o item dada a chave primária. Observe que esse trecho de código não tem uma tabela do Amazon DynamoDB configurada como fonte de dados. Ele está sendo usado apenas como exemplo:

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

O modelo de resposta para `getObject` retorna os resultados:

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

Esta é a aparência de um modelo de solicitação para `listObjects`. Um `listObjects` usa uma operação `Scan` para retornar um ou mais itens e atributos. Observe que esse trecho de código não tem uma tabela do Amazon DynamoDB configurada como fonte de dados. Ele está sendo usado apenas como exemplo:

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

O modelo de resposta para `listObjects` retorna os resultados:

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

Veja a seguir alguns exemplos de uso deste esquema com consultas do GraphQL:

```
mutation CreateObject {
  putObject(email: "example@example.com"
    json: "{\"a\":1, \"b\":3, \"string\": 234}")
}
```

```
    date: "1970-01-01Z"
    time: "12:00:34."
    datetime: "1930-01-01T16:00:00-07:00"
    timestamp: -123123
    url: "https://amazon.com"
    phoneno: "+1 555 764 4377"
    ip: "127.0.0.1/8"
  ) {
    id
    email
    json
    date
    time
    datetime
    url
    timestamp
    phoneno
    ip
  }
}

query getObject {
  getObject(id: "0d97daf0-48e6-4ffc-8d48-0537e8a843d2"){
    email
    url
    timestamp
    phoneno
    ip
  }
}

query listObjects {
  listObjects {
    json
    date
    time
    datetime
  }
}
```

Interfaces e uniões no GraphQL

O sistema do tipo GraphQL oferece suporte a [interfaces](#). Uma interface expõe um determinado conjunto de campos que um tipo deve incluir para implementar a interface.

O sistema de tipos GraphQL também oferece suporte a [Uniões](#). As uniões são idênticas às interfaces, exceto que não definem um conjunto comum de campos. As uniões geralmente são preferidas em relação às interfaces quando os tipos possíveis não compartilham uma hierarquia lógica.

A seção a seguir é uma referência para digitação de esquema.

Exemplos de interface

Podemos representar uma interface `Event` que representa qualquer tipo de atividade ou reunião de pessoas. Alguns tipos de eventos possíveis são `Concert`, `Conference` e `Festival`. Esses tipos possuem características em comum, incluindo um nome, um local onde o evento está acontecendo e datas de início e término. Esses tipos também têm diferenças, uma `Conference` oferece uma lista de palestrantes e seminários enquanto um `Concert` contém uma banda em apresentação.

No SDL (Schema Definition Language), a interface `Event` é definida da seguinte forma:

```
interface Event {  
  id: ID!  
  name : String!  
  startsAt: String  
  endsAt: String  
  venue: Venue  
  minAgeRestriction: Int  
}
```

E cada um dos tipos implementa a interface `Event` da seguinte forma:

```
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]
}
```

As interfaces são úteis para representar elementos que podem ser de vários tipos. Por exemplo, podemos pesquisar todos os eventos que acontecem em um local específico. Vamos adicionar um campo `findEventsByVenue` ao esquema da seguinte forma:

```
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]
}
```

O `findEventsByVenue` retorna uma lista de `Event`. Como os campos da interface do GraphQL são comuns a todos os tipos de implementação, é possível selecionar qualquer campo na interface `Event` (`id`, `name`, `startsAt`, `endsAt`, `venue` e `minAgeRestriction`). Além disso, é possível

acessar os campos em qualquer tipo de implementação usando [fragmentos](#) do GraphQL, desde que o tipo seja especificado.

Vamos examinar um exemplo de uma consulta do GraphQL que usa a interface.

```
query {  
  findEventsAtVenue(venueId: "Madison Square Garden") {  
    id  
    name  
    minAgeRestriction  
    startsAt  
  
    ... on Festival {  
      performers  
    }  
  
    ... on Concert {  
      performingBand  
    }  
  
    ... on Conference {  
      speakers  
      workshops  
    }  
  }  
}
```

A consulta anterior produz uma única lista de resultados e o servidor pode, por padrão, classificar os eventos por data de início.

```
{  
  "data": {  
    "findEventsAtVenue": [  
      {  
        "id": "Festival-2",  
        "name": "Festival 2",  
        "minAgeRestriction": 21,  
        "startsAt": "2018-10-05T14:48:00.000Z",  
        "performers": [  
          "The Singers",  
          "The Screammers"  
        ]  
      },  
    ],  
  },  
}
```

```

    {
      "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"
      ]
    }
  ]
}

```

Como os resultados são retornados como uma única coleção de eventos, o uso de interfaces para representar características comuns é muito útil para classificar os resultados.

Exemplos de uniões

Conforme mencionado anteriormente, as uniões não definem conjuntos comuns de campos. Um resultado de pesquisa pode representar muitos tipos diferentes. Usando o esquema `Event`, defina uma união `SearchResult` da seguinte forma:

```

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

```

Neste caso, para consultar qualquer campo na nossa união `SearchResult`, deve-se usar fragmentos.

```
query {  
  search(query: "Madison") {  
    ... on Venue {  
      id  
      name  
      address  
    }  
  
    ... on Festival {  
      id  
      name  
      performers  
    }  
  
    ... on Concert {  
      id  
      name  
      performingBand  
    }  
  
    ... on Conference {  
      speakers  
      workshops  
    }  
  }  
}
```

Resolução de texto em AWS AppSync

A resolução de tipo é o mecanismo pelo qual o mecanismo do GraphQL identifica um valor resolvido como um tipo de objeto específico.

Voltando ao exemplo de pesquisa da união, assumindo que a consulta gerou resultados, cada item na lista de resultados deve se apresentar como um dos tipos possíveis definidos pela união `SearchResult` (isto é, `Conference`, `Festival`, `Concert` ou `Venue`).

Como a lógica para identificar um `Festival` de uma `Venue` ou uma `Conference` depende dos requisitos do aplicativo, o mecanismo do GraphQL deve receber uma dica para identificar os tipos possíveis nos resultados brutos.

Com AWS AppSync, essa dica é representada por um metacampo chamado `__typename`, cujo valor corresponde ao nome do tipo de objeto identificado. `__typename` é necessário para tipos de retorno que sejam interfaces ou uniões.

Exemplo de resolução de tipo

Vamos reutilizar o esquema anterior. Você pode acompanhar navegando até o console e adicionando o seguinte na página Esquema:

```
schema {  
  query: Query  
}  
  
type Query {  
  # Retrieve Events at a specific Venue  
  findEventsAtVenue(venueId: ID!): [Event]  
  # Search across all content  
  search(query: String!): [SearchResult]  
}  
  
union SearchResult = Conference | Festival | Concert | Venue  
  
type Venue {  
  id: ID!  
  name: String!  
  address: String  
  maxOccupancy: Int  
}  
  
interface Event {  
  id: ID!  
  name: String!  
  startsAt: String  
  endsAt: String  
  venue: Venue  
  minAgeRestriction: Int  
}  
  
type Festival implements Event {  
  id: ID!  
  name: String!  
  startsAt: String  
  endsAt: String
```

```
    venue: Venue
    minAgeRestriction: Int
    performers: [String]
  }

type Conference implements Event {
  id: ID!
  name: String!
  startsAt: String
  endsAt: String
  venue: Venue
  minAgeRestriction: Int
  speakers: [String]
  workshops: [String]
}

type Concert implements Event {
  id: ID!
  name: String!
  startsAt: String
  endsAt: String
  venue: Venue
  minAgeRestriction: Int
  performingBand: String
}
```

Vamos anexar um resolvedor ao campo `Query.search`. Na `Resolvers` seção, escolha `Anexar`, crie uma nova fonte de dados do tipo `NENHUMA` e, em seguida, nomeie-a `StubDataSource`. Para esse exemplo, vamos imaginar que os resultados foram obtidos de uma fonte externa e codificar os resultados no modelo de mapeamento da solicitação.

No painel do modelo de mapeamento da solicitação, insira o seguinte:

```
{
  "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 Screammers"]
    },
    {
      "id": "Concert-3",
      "name": "Concert 3",
      "performingBand": "The Jumpers"
    },
    {
      "id": "Conference-4",
      "name": "Conference 4",
      "speakers": ["The Storytellers"],
      "workshops": ["Writing", "Reading"]
    }
  ]
}

```

Se o aplicativo retornar o nome do tipo como parte do campo `id`, a lógica de resolução de tipo deverá analisar o campo `id` para extrair o nome do tipo e, em seguida, adicionar o campo `__typename` a cada um dos resultados. Você pode executar essa lógica no modelo de mapeamento da resposta da seguinte forma:

Note

Observação: também é possível executar essa tarefa como parte da função do Lambda se você estiver usando a fonte de dados do Lambda.

```

#foreach ($result in $context.result)
  ## Extract type name from the id field.
  #set( $typeName = $result.id.split("-")[0] )
  #set( $ignore = $result.put("__typename", $typeName))
#end
$util.toJson($context.result)

```

Execute a seguinte consulta:

```

query {

```

```
search(query: "Madison") {  
  ... on Venue {  
    id  
    name  
    address  
  }  
  
  ... on Festival {  
    id  
    name  
    performers  
  }  
  
  ... on Concert {  
    id  
    name  
    performingBand  
  }  
  
  ... on Conference {  
    speakers  
    workshops  
  }  
}
```

A consulta gera os seguintes resultados:

```
{  
  "data": {  
    "search": [  
      {  
        "id": "Venue-1",  
        "name": "Venue 1",  
        "address": "2121 7th Ave, Seattle, WA 98121"  
      },  
      {  
        "id": "Festival-2",  
        "name": "Festival 2",  
        "performers": [  
          "The Singers",  
          "The Screammers"  
        ]  
      }  
    ]  
  }  
}
```

```
    },  
    {  
      "id": "Concert-3",  
      "name": "Concert 3",  
      "performingBand": "The Jumpers"  
    },  
    {  
      "speakers": [  
        "The Storytellers"  
      ],  
      "workshops": [  
        "Writing",  
        "Reading"  
      ]  
    }  
  ]  
}  
}
```

A lógica da resolução de tipo varia dependendo do aplicativo. Por exemplo, você pode ter uma lógica de identificação diferente que verifica a existência de determinados campos ou até mesmo uma combinação de campos. Ou seja, é possível detectar a presença do campo `performers` para identificar um `Festival` ou a combinação dos `speakers` e dos campos `workshops` para identificar uma `Conference`. De forma geral, cabe a você definir a lógica que deseja usar.

Solução de problemas e erros comuns em AWS AppSync

Esta seção discute alguns erros comuns e como solucioná-los.

Mapeamento de chave do DynamoDB incorreto

Se a operação do GraphQL retornar a seguinte mensagem de erro, pode ser porque a estrutura do modelo de mapeamento da solicitação não corresponde à estrutura da chave do Amazon DynamoDB:

```
The provided key element does not match the schema (Service: AmazonDynamoDBv2; Status Code: 400; Error Code
```

Por exemplo, se a tabela do DynamoDB tem uma chave de hash chamada "id" e o modelo diz "PostID", conforme o exemplo a seguir, isso resulta no erro anterior, pois "id" não corresponde a "PostID".

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

Resolver ausente

Se você executar uma operação do GraphQL, como uma consulta, e obter uma resposta nula, isso pode acontecer por não ter um resolver configurado.

Por exemplo, se você importar um esquema que define um campo `getCustomer(userId: ID!)`, e não tiver configurado um resolver para esse campo, então ao executar uma consulta como `getCustomer(userId:"ID123"){...}`, você receberá uma resposta como a seguinte:

```
{
  "data": {
    "getCustomer": null
  }
}
```

```
}  
}
```

Erros do modelo de mapeamento

Se o modelo de mapeamento não estiver configurado corretamente, você receberá uma resposta do GraphQL em que `errorType` é `MappingTemplate`. O campo `message` deve indicar onde o problema está no modelo de mapeamento.

Por exemplo, caso não tenha um campo `operation` no modelo de mapeamento da solicitação, ou se o nome do campo `operation` estiver incorreto, você receberá uma resposta como a seguinte:

```
{  
  "data": {  
    "searchPosts": null  
  },  
  "errors": [  
    {  
      "path": [  
        "searchPosts"  
      ],  
      "errorType": "MappingTemplate",  
      "locations": [  
        {  
          "line": 2,  
          "column": 3  
        }  
      ],  
      "message": "Value for field '$[operation]' not found."  
    }  
  ]  
}
```

Tipos de retorno incorretos

O tipo de retorno da fonte de dados deve corresponder ao tipo definido de um objeto no esquema, caso contrário, poderá receber um erro do GraphQL como:

```
"errors": [  
  {
```

```
    "path": [
      "posts"
    ],
    "locations": null,
    "message": "Can't resolve value (/posts) : type mismatch error, expected type LIST,
got OBJECT"
  }
]
```

Por exemplo, isso pode ocorrer com a seguinte definição de consulta:

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

Que espera uma LISTA de objetos [Posts]. Por exemplo, caso tenha uma função do Lambda no Node.JS com algo semelhante ao seguinte:

```
const result = { data: data.Items.map(item => { return item ; }) };
callback(err, result);
```

Isso lançaria um erro uma vez que `result` é um objeto. Seria necessário alterar o retorno de chamada para `result.data` ou alterar o esquema para não retornar uma LISTA.

Processamento de solicitações inválidas

Quando AWS AppSync não conseguir processar e enviar uma solicitação (devido a dados impróprios, como sintaxe inválida) para o resolvedor de campo, a carga de resposta retornará os dados do campo com valores definidos como `null` e quaisquer erros relevantes.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.