[AWS Black Belt Online Seminar]
AWS AppSync

サービスカットシリーズ

Solutions Architect, Enterprise
布村 純也
2019/8/21

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive

自己紹介

布村 純也 (Atsuya, Nunomura)

- **所属：**

  Solution Architect, Enterprise

- **Background**

  Web/Mobile アプリケーション、API・ Web Service開発、Apollo

- **好きな AWS のサービス：**

  AWS Amplify　　AWS AppSync　　AWS Lambda

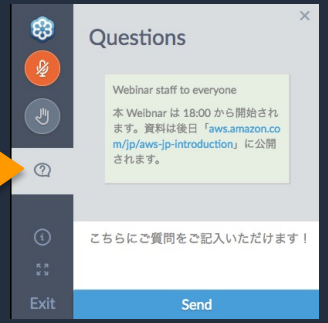
AWS Black Belt Online Seminar とは

「サービス別」「ソリューション別」「業種別」のそれぞれのテーマに分かれて、アマゾン ウェブ サービス ジャパン株式会社が主催するオンラインセミナーシリーズです。

質問を投げることができます！

- 書き込んだ質問は、主催者にしか見えません
- 今後のロードマップに関するご質問は
  お答えできませんのでご了承下さい

① 吹き出しをクリック
② 質問を入力
③ Sendをクリック



Twitter ハッシュタグは以下をご利用ください
#awsblackbelt

内容についての注意点

- 本資料では2019年8月21日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。

- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。

- 価格は税抜表記となっています。日本居住者のお客様が東京リージョンを使用する場合、別途消費税をご請求させていただきます。

- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

アジェンダ

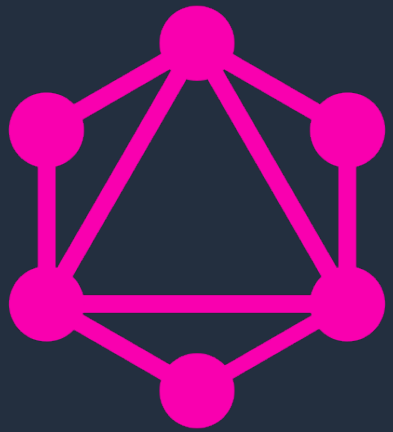
- AWS AppSyncとは！？
- 先ずはGraphQL ..
- 改めて AWS AppSyncとは
- 料金
- 重要機能アップデート
- まとめ

AWS AppSyncとは！？

AWS AppSync

AWS AppSync は GraphQL マネージド・サービス

GraphhQL



!?

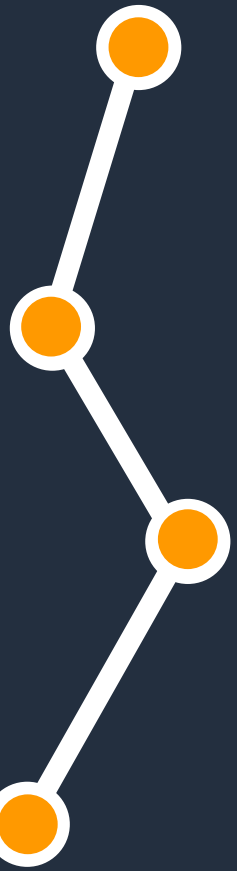
Managed

AWS AppSync

aws

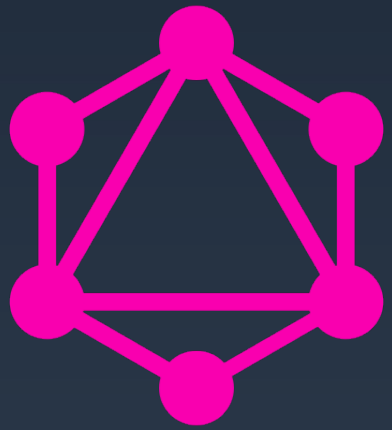
代表的なWebAPI Model

{ REST-API }



aws

先ずはGraphQL ..

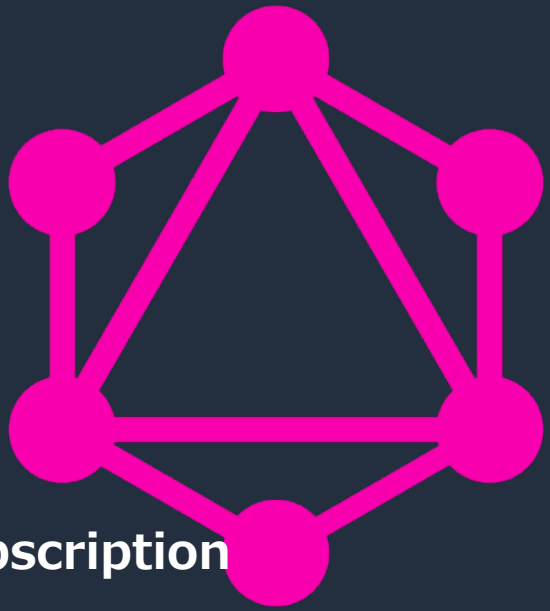


aws

GraphQL とは！？

ー TL;DR ー

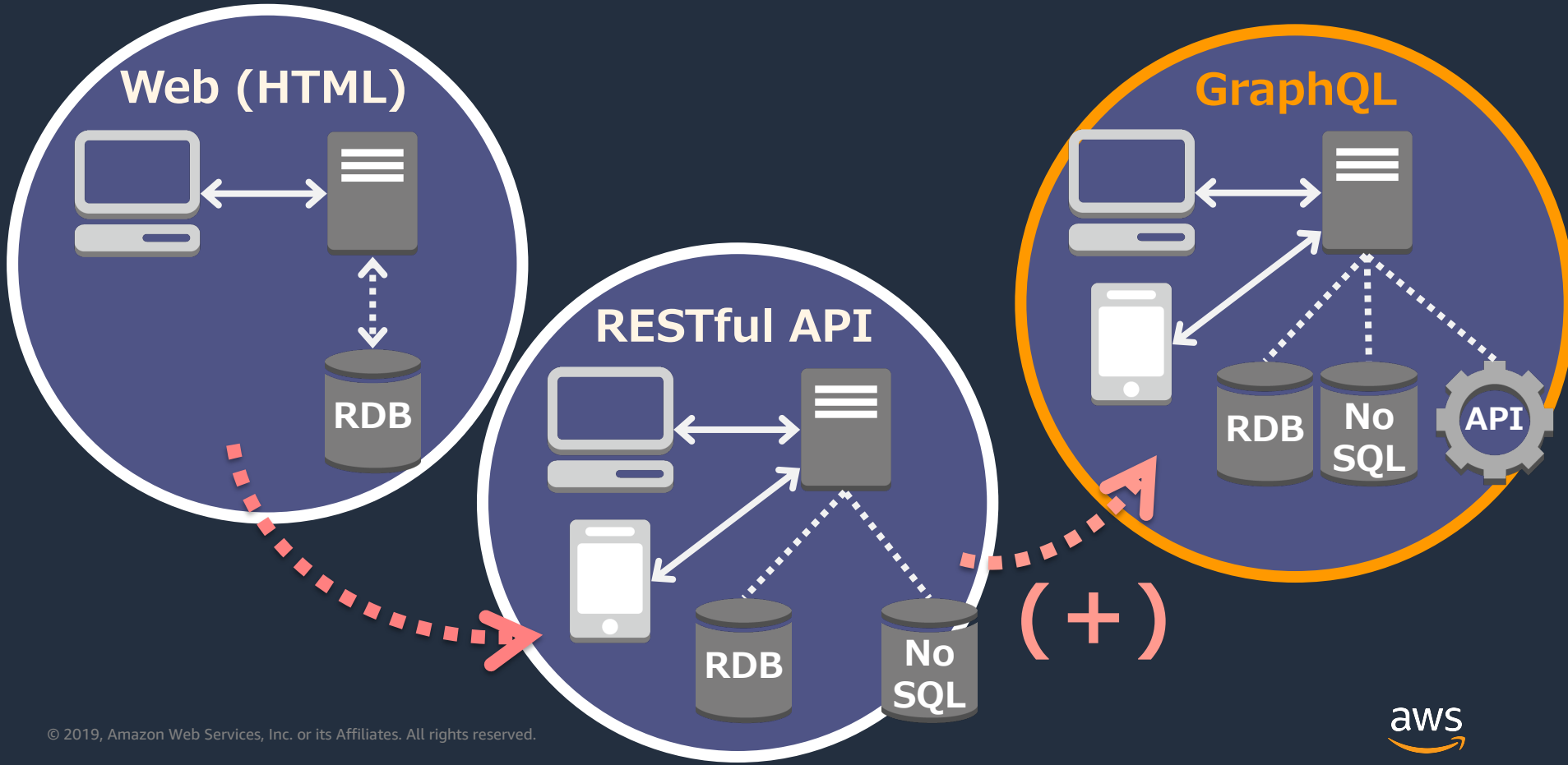
- GraphQL API用の OSS **QUERY言語**
  （*A query language for your API*）
- 定義に従ってQUERYを書きサーバーからJSONを取得
- 処理形態は取得/Query、変更/Mutation、購読/Subscription



https://graphql.org

aws

登場までの変遷



なぜ登場した！？

(T_T)

■ REST API開発者、利用者の課題

・API仕様のドキュメント管理が大変
・APIの叩き方を理解するのが大変
・APIのドキュメントと実装がズレてケンカ

■ クライアント開発者からの不満

・１ページ表示するのに幾つもAPIを叩かないといけない
・折角イベントドリブンに作ってもサーバーとの接続は結局Request / Responseの形が残る

aws

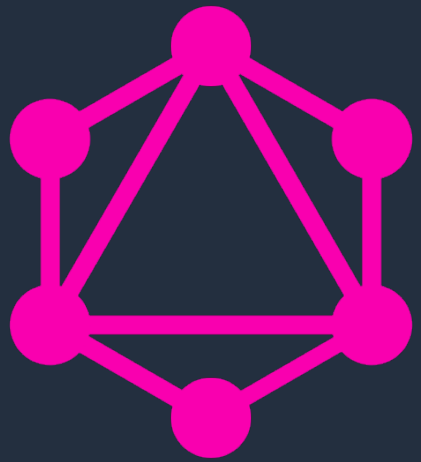
GraphQLによって得られるメリット

(^ ^)

- クライアント/サーバー間のインタフェースがクリーンになる
- 通信オーバーヘッドが削減される
- APIドキュメント作成に費やす時間が不要になる
- APIを理解するのに費やす時間が削減される

aws

なぜGraphQLが注目されているか？　- GraphQLの特徴 -



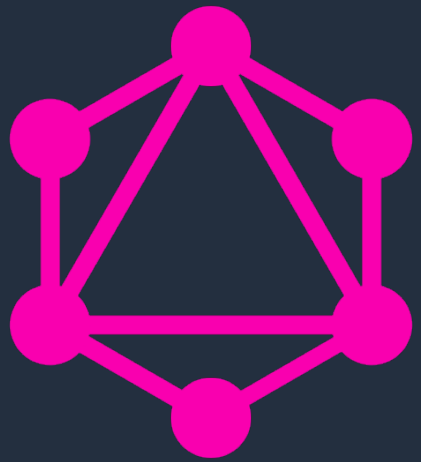
1．型指定されたスキーマ

2．クライアントからのレスポンス形式の指定

3．サブスクリプションを利用したリアルタイム処理

aws

なぜGraphQLが注目されているか？　- GraphQLの特徴 -



1. **型指定されたスキーマ**
   APIドキュメントを手動で記述する必要が無くなり、
   APIを定義したスキーマをベースに自動生成
2. クライアントからのレスポンス形式の指定
3. サブスクリプションを利用したリアルタイム処理

aws

スキーマ定義

```
type Query {
    getTodos: [Todo]
}

type Todo {
    id: ID!
    name: String
    description: String
    priority: Int
    duedate: String
}
```

スカラー型、オブジェクト型、
列挙型などを利用可能

Not Nullは感嘆符で表現

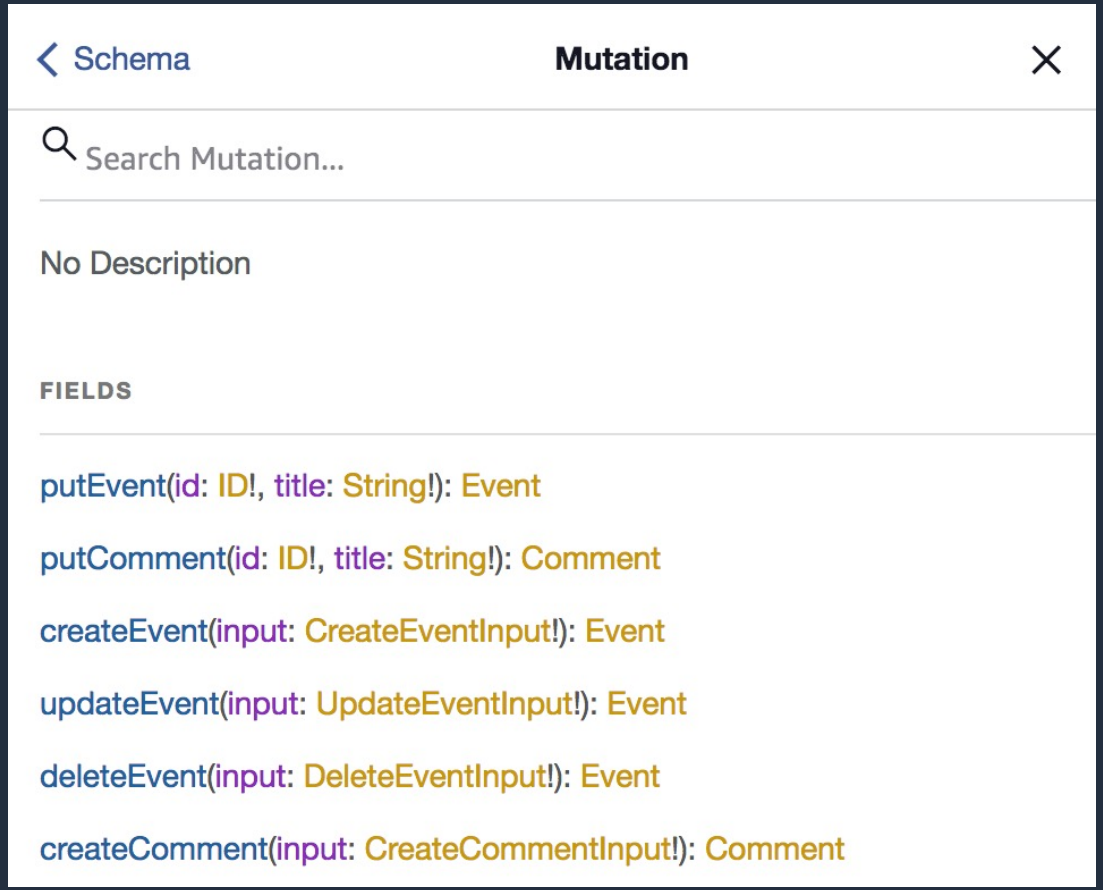
ID!

リストは角カッコで表現

[String!]

aws

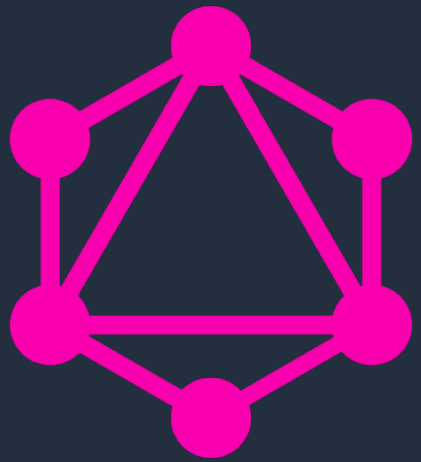
ドキュメント自動生成



Query実行環境


Docs
}
}
query list {
listPyconData(filter:{
title:{
contains:"pycon"
}
"dev day"
}
){
items{
id
title
content
}
}
}
{
"data": {
"listPyconData": {
"items": [
{
"id": "6688c20b-0539-444b-b5d1-b190ef418706",
"title": "p
"content": "dev day 2018" graphql"
}
]
}
}
}
QUERY VARIABLES
LOGS

なぜGraphQLが注目されているか？　- GraphQLの特徴 -



1. 型指定されたスキーマ
2. クライアントからのレスポンス形式の指定
   - オーバーフェッチ、アンダーフェッチが無くなる
   - クリーンなインタフェース
3. サブスクリプションを利用したリアルタイム処理

aws

クライアントがレスポンス形式を指定

```
type Query {
    getTodos: [Todo]
}

type Todo {
    id: ID!
    name: String
    description: String
    priority: Int
    duedate: String
}
```

Schemaとモデルデータ (Type)

```
query {
  getTodos {
    id
    name
    priority
  }
}
```

クライアントが必要なものだけをリクエスト

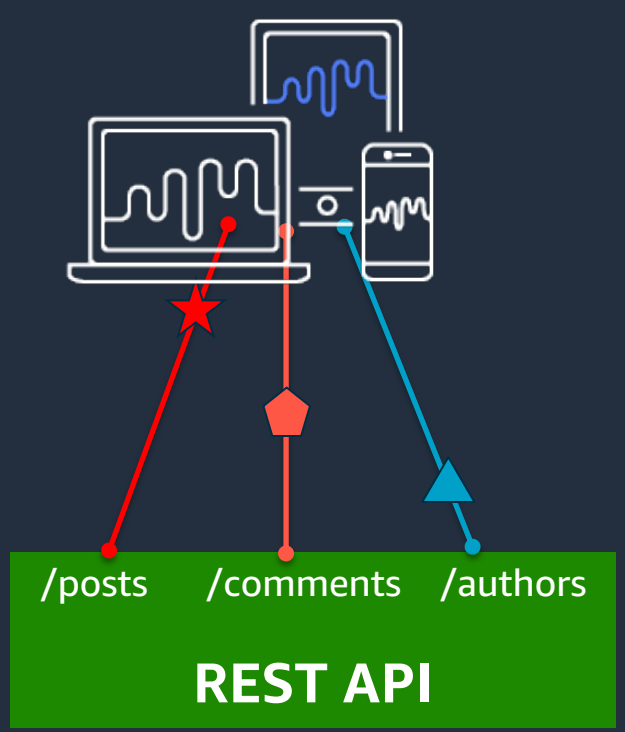
```
{
    "id": "1",
    "name": "Get Milk",
    “priority": "1"
},
{
    "id": “2",
    "name": “Go to gym",
    “priority": “5"
},…
```

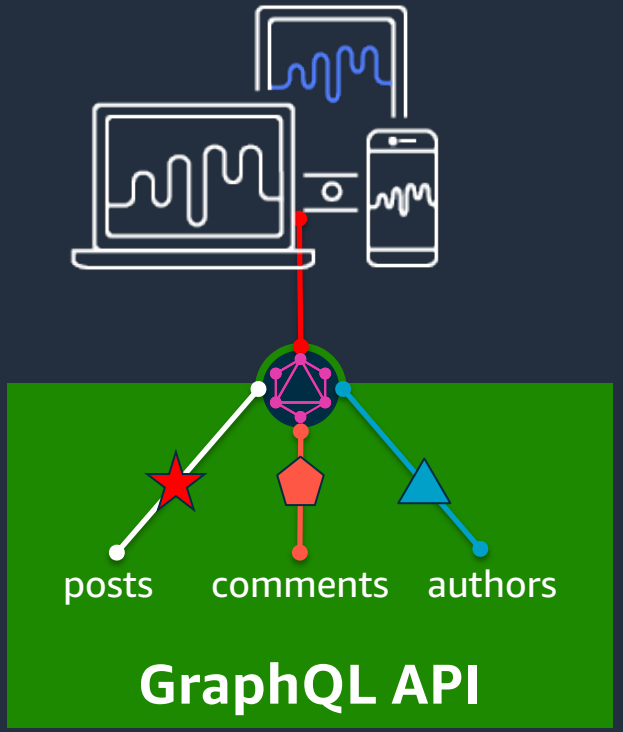
リクエストしたデータだけが返される

aws

クリーンなインタフェース

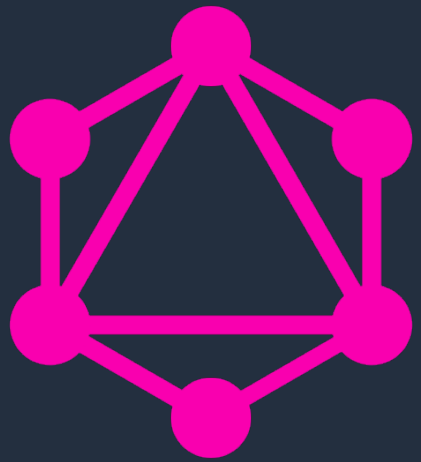
RESTとGraphQLのAPI アーキテクチャ





aws

なぜGraphQLが注目されているか？　- GraphQLの特徴 -



1. 型指定されたスキーマ
2. クライアントからのレスポンス形式の指定
3. サブスクリプションを利用したリアルタイム処理

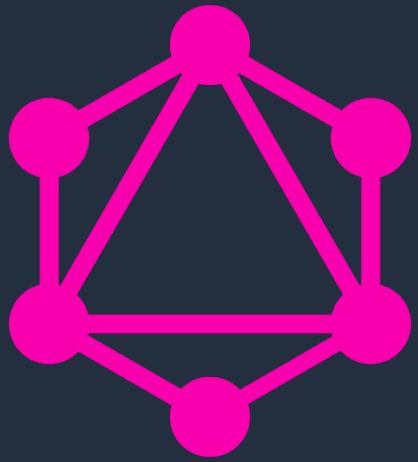
   クライアントはデータをサブスクライブする事で
   イベント・ドリブンに処理を実装可能

aws

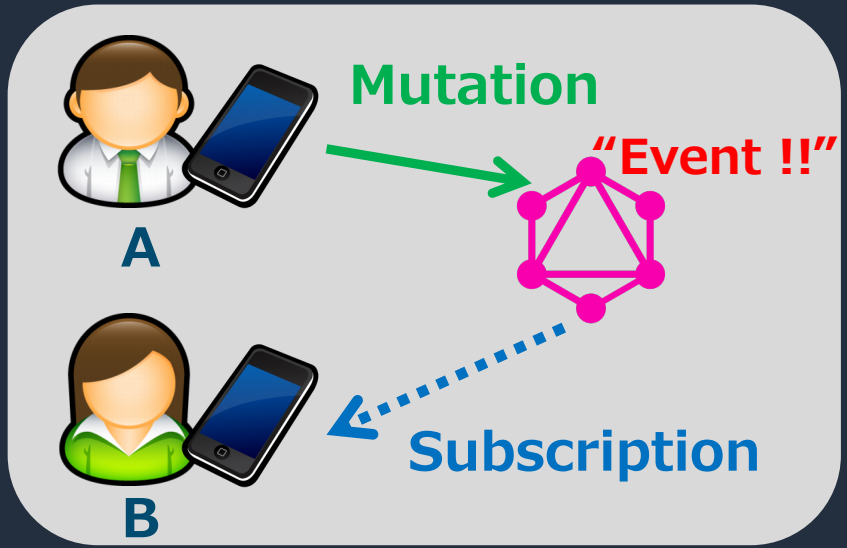
GraphQLの処理形態

Query：取得

Mutation：変更



Subscription：購読



aws

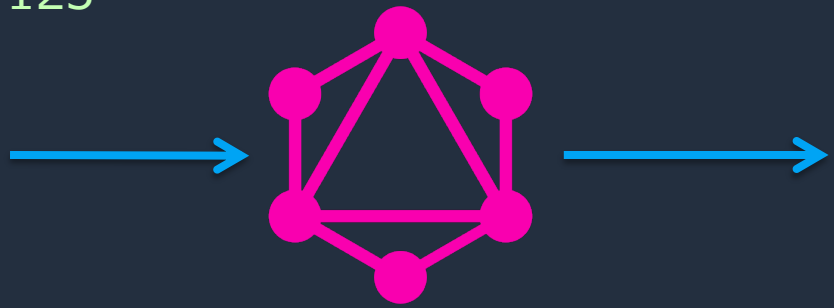
GraphQL Subscription

リアルタイムでのデータ購読
Mutationをトリガーとしたイベントベースモード

A

B

```
mutation addPost( id:123
  title: "New post!"
  author: "Nadia"){
  id
  title
  author
}
```



```
data: [{
  id:123,
  title:"New Post!"
  author:"Nadia"
}]
```


aws

スキーマ定義構成

```
type Subscription {
    addedPost: Post
    @aws_subscribe(mutations: ["addPost"])
    deletedPost: Post
    @aws_subscribe(mutations: ["deletePost"])
}


type Mutation {
    addPost(id: ID! author: String! title:
        String content: String): Post!
    deletePost(id: ID!): Post!
}
```

```
subscription NewPostSub {
    addedPost {
        __typename
        version
        title
        content
        author
        url
    }
}
```

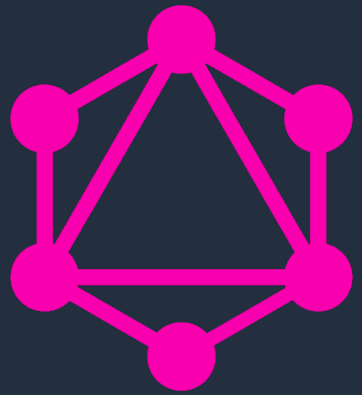

aws

GraphQL Subscription ハンドシェイク

Subscription NewPostSub {
    addedPost{…}
}

WebSocket URL and Connection Payload

Secure Websocket Connection (wss://)



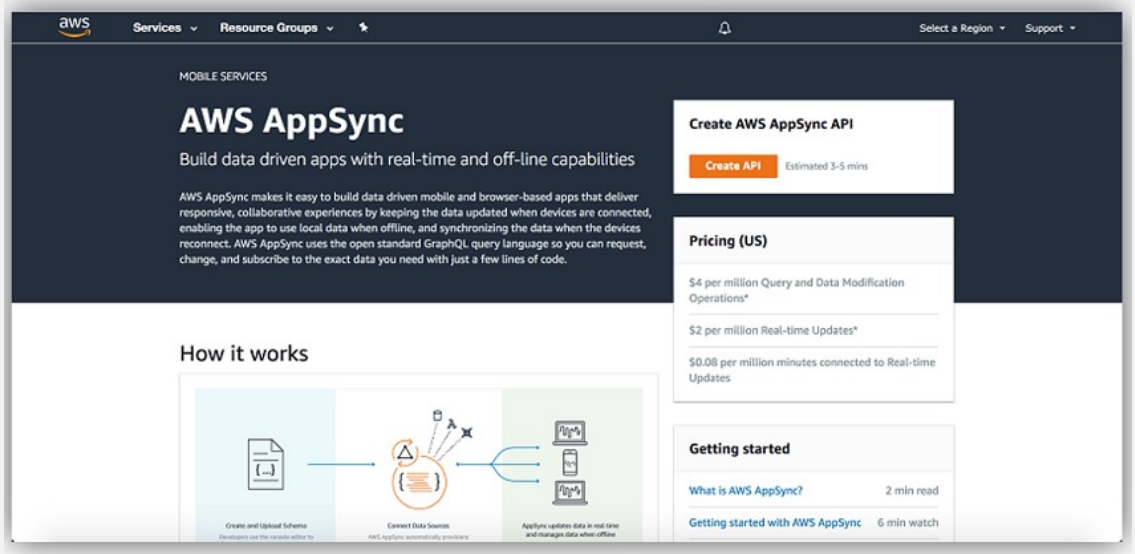
aws

改めて AWS AppSyncとは

AWS AppSync

GraphQL マネージド・サービス

直ぐにGraphQLの利用を始められます



AWS AppSync

GraphQL
マネージド・サービス

アカウントの
DataSourceに接続

データ同期、リアルタイム、
オフライン機能

GraphQL ファサード

競合の検出と解決

セキュリティ
(API Key, IAM, Cognito, OIDC)

aws

開発者の課題解決

リアルタイム
コラボレー
ション

同期を考慮し
たオフライン
プログラミン
グ

必要なデータ
のみの取得

複数のデータ
ソースへの
アクセス

アクセス制御

aws

Usecase

■ リアルタイム

- 最新の情報をウォッチするダッシュボード
- ほぼリアルタイムでデータを更新

■ コラボレーション

- 複数ユーザーが共同編集を行うアプリケーション
- ドキュメント、画像、テキストメッセージ等、様々なコンテンツタイプを自動更新

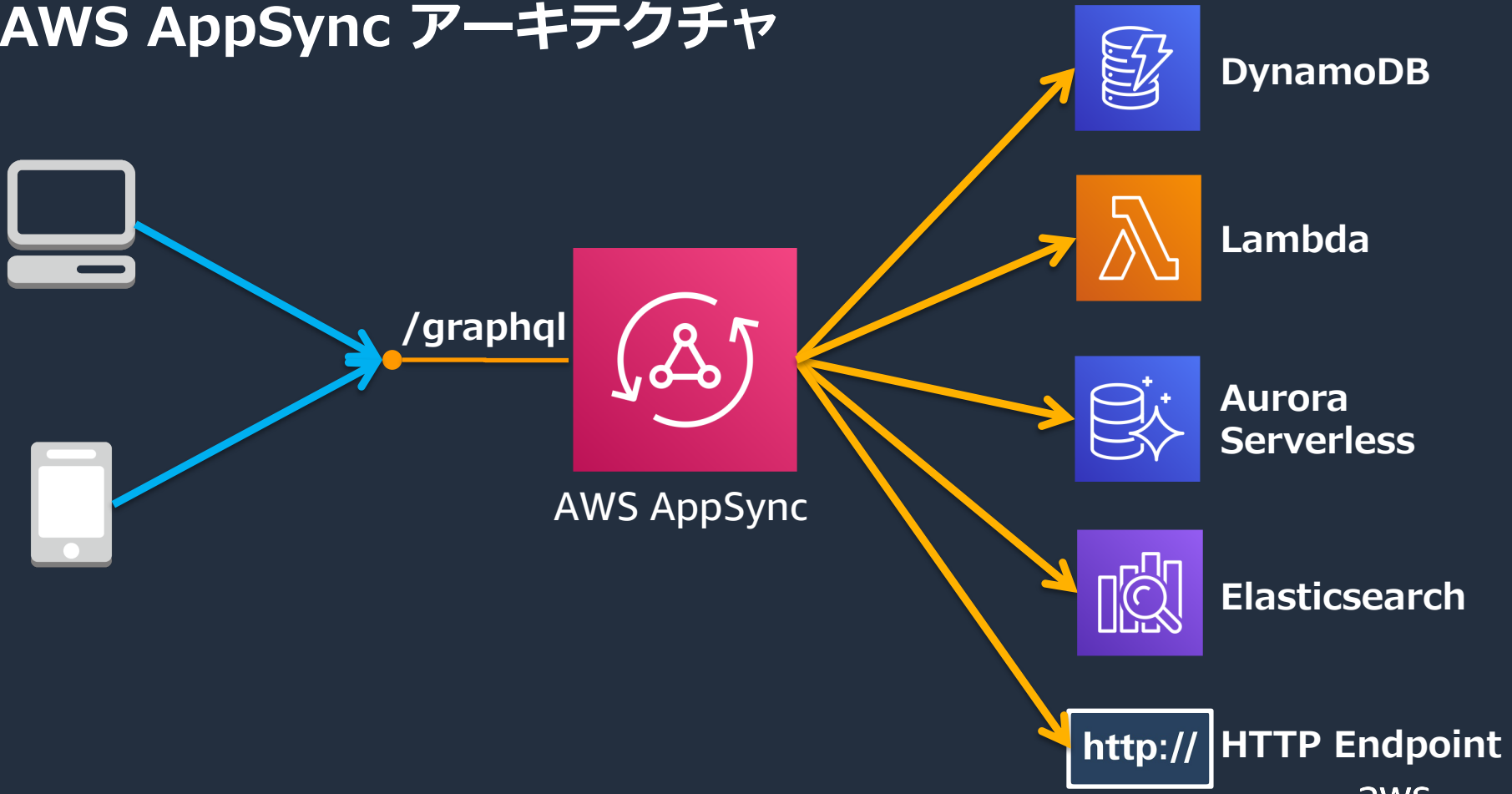
■ ソーシャルメディア

- ソーシャルメディアやチャット
- 複数ユーザー間でのメッセージング管理をサポート
- オフライン時でもアプリケーションを操作でき、再接続時に自動 Sync

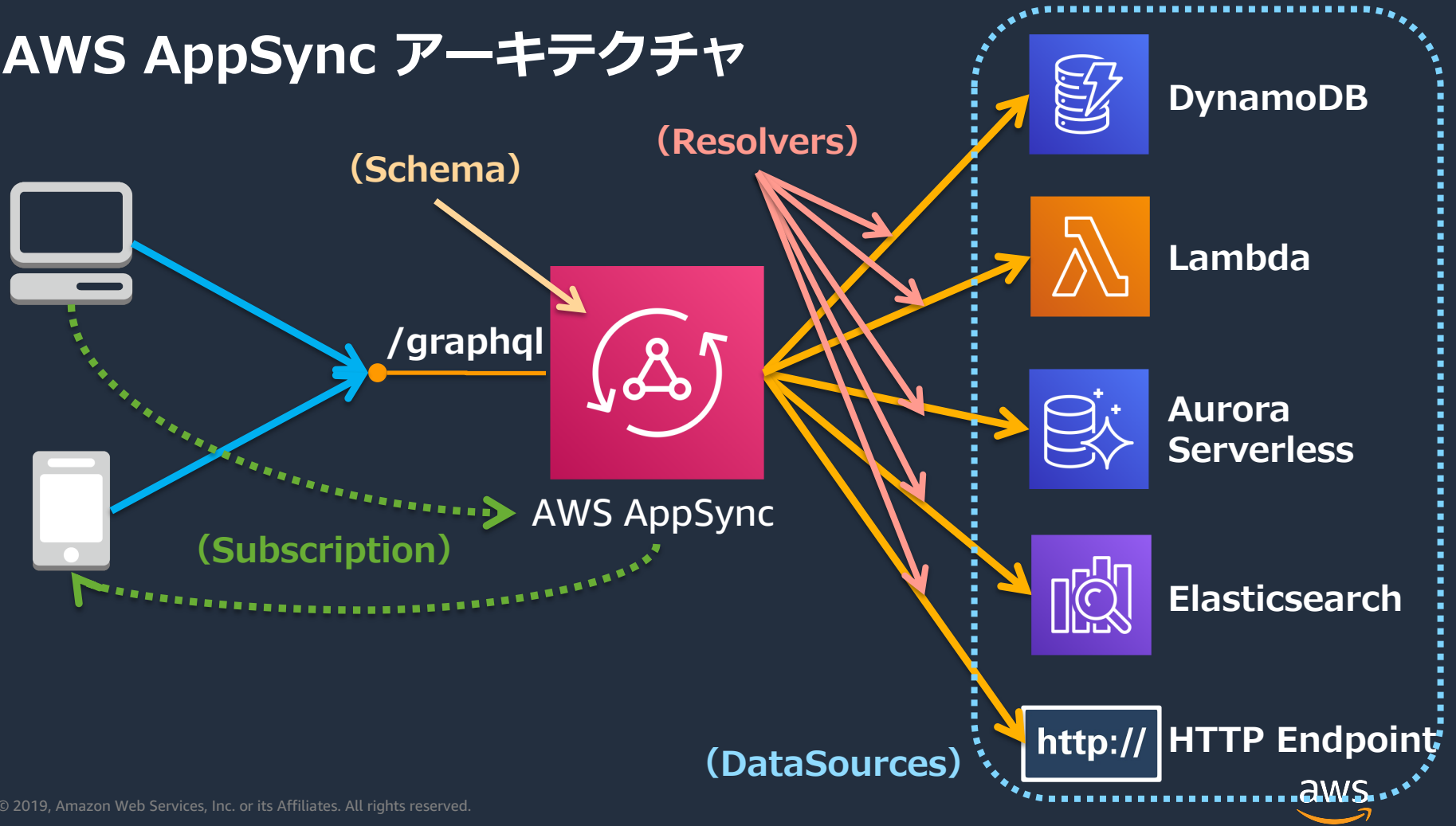
AWS AppSync のコンセプト

- **AWS App Sync Client**
  認証、オフラインロジックなどを含んだClient
- **Resolver**
  リクエスト / レスポンスの処理を記述する関数
- **Data Source**
  DynamoDB / Lambda / Aurora Serverless / Elasticsearch / HTTP Endpoint
- **Identity**
  GraphQL Proxy へのリクエストの認証
- **GraphQL Proxy**
  リクエストのマッピング、コンフリクトのハンドリング、アクセスコントロール
- **Operation**
  Query / Mutation / Subscription など GraphQL のオペレーション

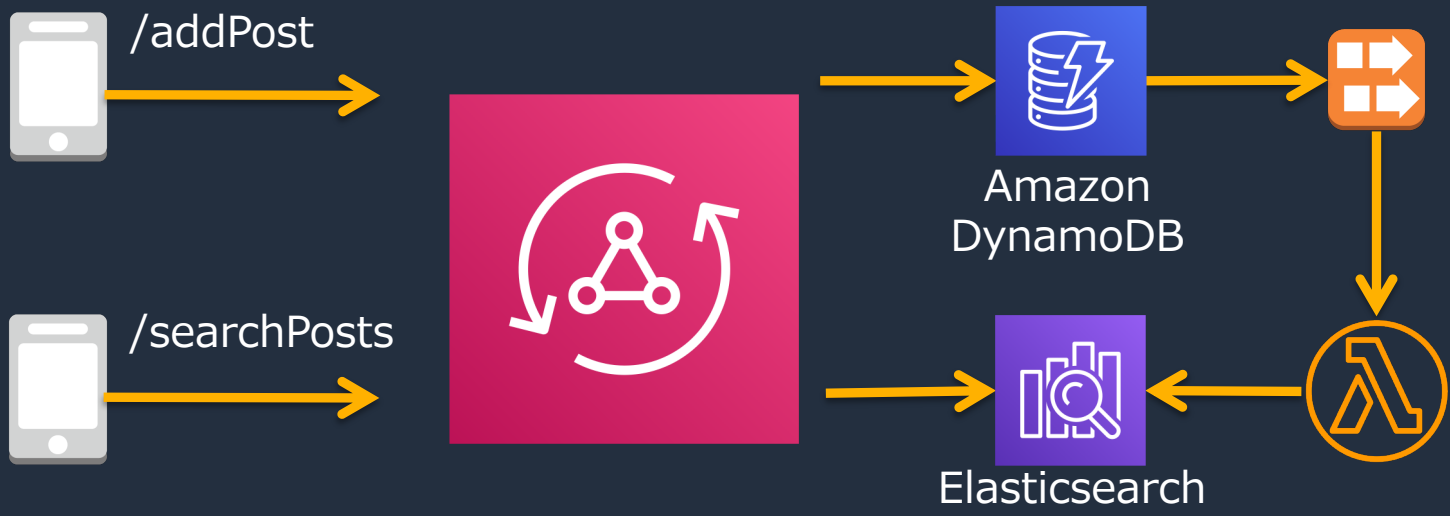
AWS AppSync アーキテクチャ



AWS AppSync アーキテクチャ



DataSource 例 (DynamoDB + Elasticsearch)



データソースを組み合わせることで高度な検索にも対応可能。
キーワード検索、ファジー検索、地理空間検索…

DataSource 例 (Lambdaと3rd Party API)

LambdaをDataSourceとして扱えるため、なんでもできる
外部のWebAPIを叩くことも可能

AWS AppSync

AppSyncバックグラウンドの
Resolver、DataSourceの組合せは
開発者の自由

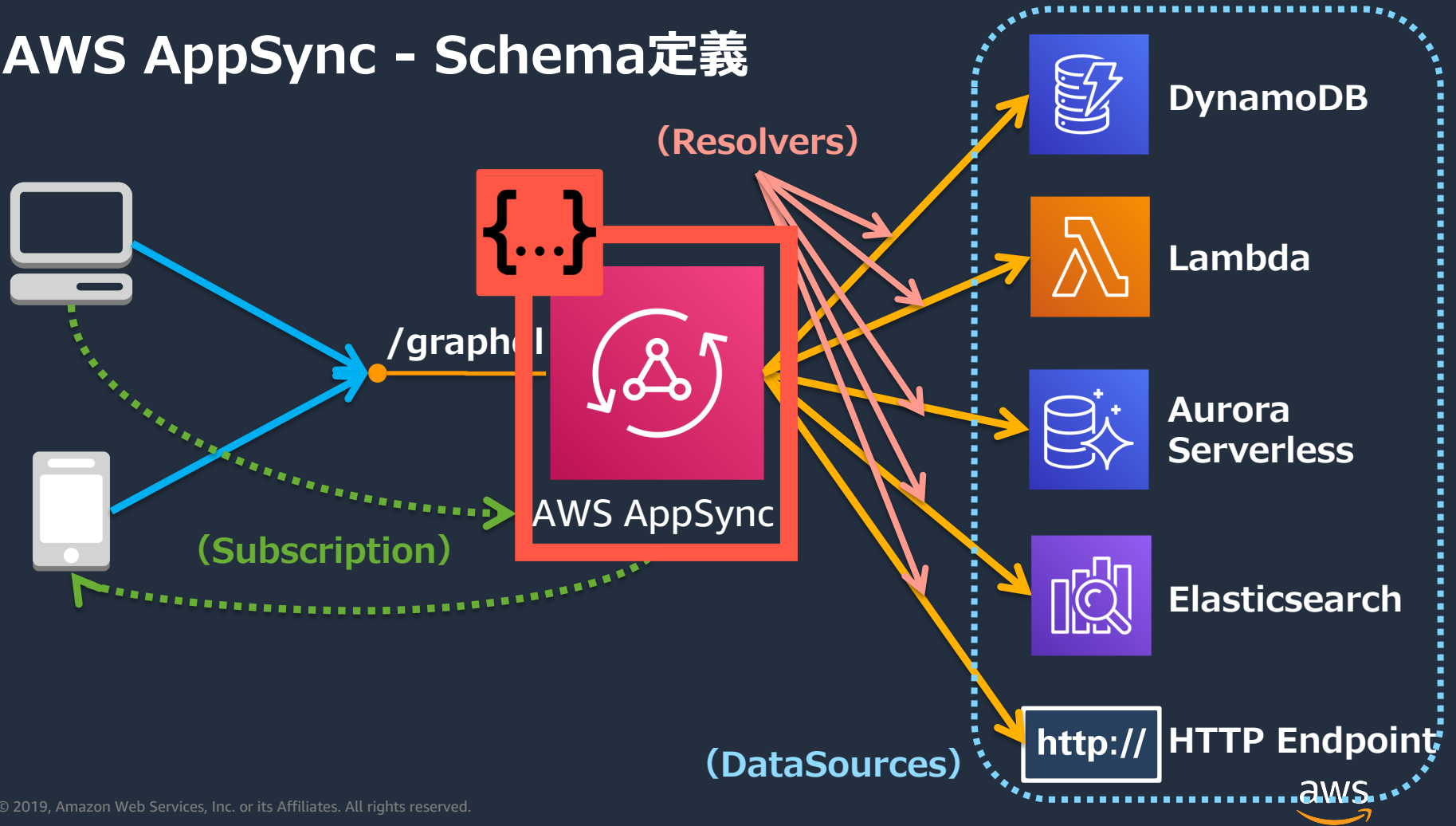
開発における**拡張性**が非常に高く
迅速なプロトタイピングが可能
(もちろん自由が高い分、事前の設計はより重要に)

迅速なプロトタイピング

AWS AppSync試行ステップ

迅速なプロトタイピング

AWS AppSync - Schema定義



Schema定義

- スキーマはサーバの機能を記述し、クエリが有効かどうかを判断する為に使用される。
- GraphQL API は1つの GraphQL Schema で定義され、SDL(Schema Definition Language)によって記述さる。

aws

Schemaの書き方 – Root Schema

Query：取得

Mutation：更新

Subscription：購読

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

Schemaの書き方 - Type

```
type Query {
    getTodos: [Todo]
}
type Todo {
    id: ID!
    name: String
    description: String
    status: TodoStatus
 }
enum TodoStatus {
    done
    pending
}
```

スカラー型、オブジェクト型、
列挙型などを利用可能

Not Nullは感嘆符で表現
ID!

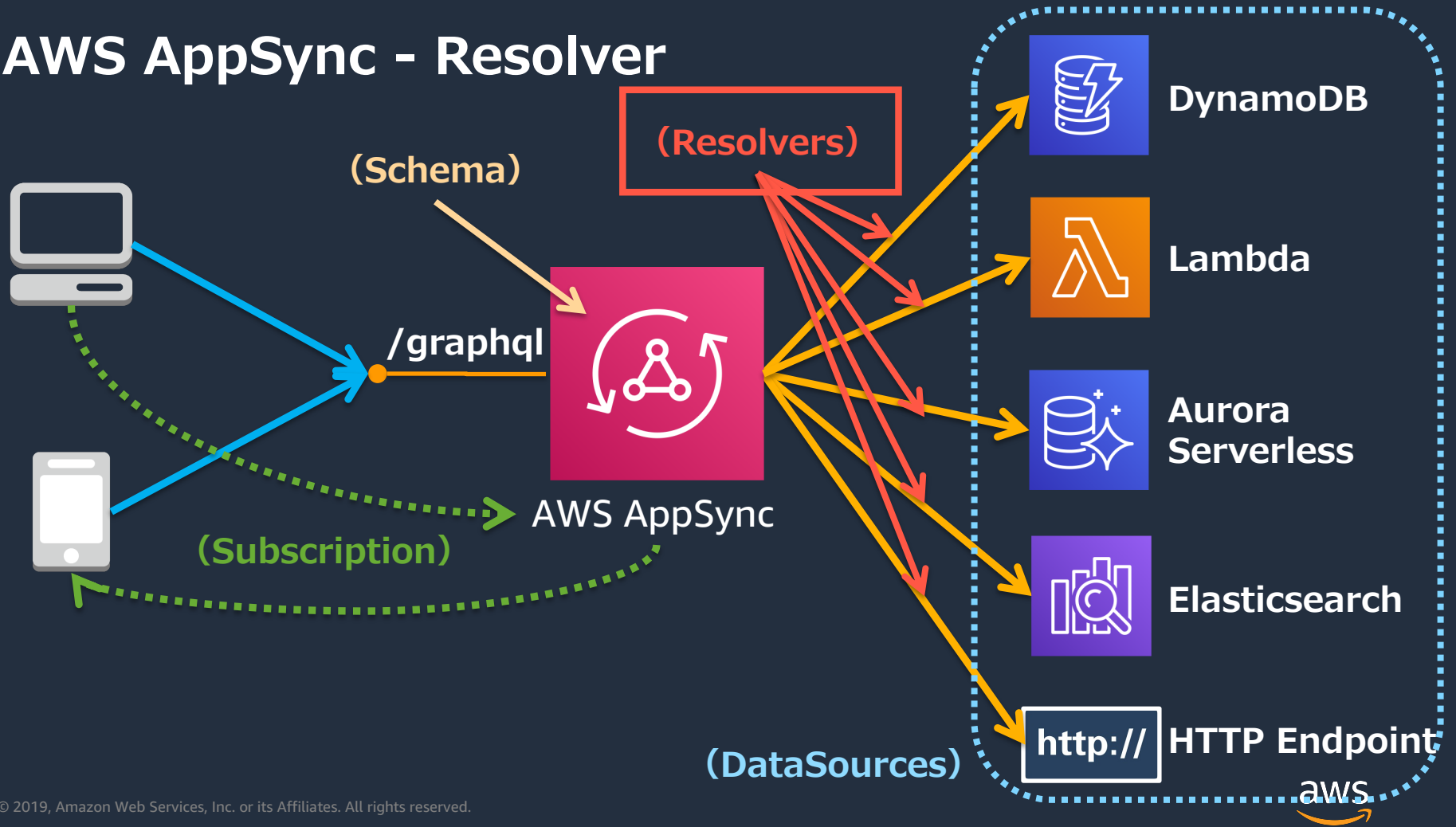
リストは角カッコで表現
[String!]

aws

迅速なプロトタイピング

aws

AWS AppSync - Resolver



Resolver マッピングテンプレート

- マッピングテンプレートは、GraphQL リクエストをデータソースの命令に変換する方法と、データソースからの応答をGraphQL レスポンスに変換する方法を定義する
- Resolver マッピングテンプレートは、VTL (Velocity Template Language)によって記述さる。

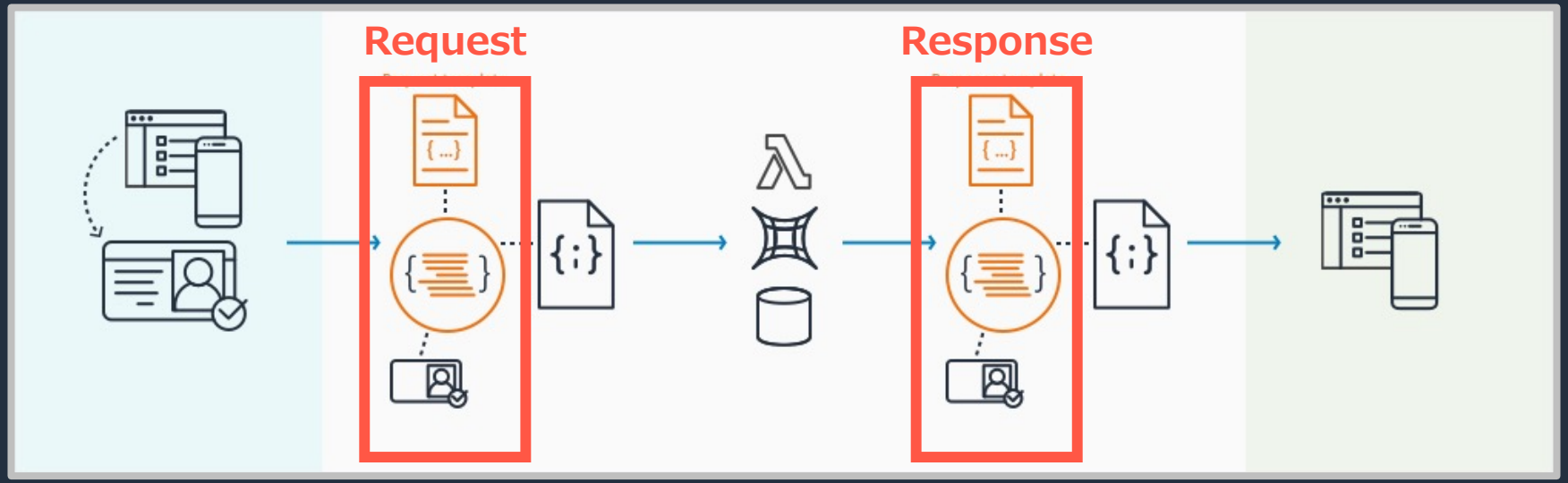
aws

Resolver マッピングテンプレートで実施する例

- アクセスコントロール
- 新規アイテムのデフォルト値
- 入力のバリデーション、フォーマット
- データの変換と整形
- リスト、マップの加工
- ユーザーIDに基づいたレスポンスのフィルタリング/変更
- 複雑な権限チェック

２種類のマッピングテンプレート

- Request テンプレート（Request → DataSource命令）
- Responseテンプレート（DataSource応答 → GraphQL Response）



Request マッピングテンプレート 例

```
{
    "version" : "2017-02-28",
    "operation" : "GetItem",
    "key" : {
        "id" : { "S" : "${context.arguments.id}" }
    }
}
```

Response マッピングテンプレート 例

- デフォルトで返す場合：

```
$utils.toJson($context.result)
```

- データを結合する場合：

```
{
    "id" : ${context.data.id},
    "title" : "${context.data.theTitle}",
    "content" : "${context.data.body1} ${context.data.body2}"
}
```

Request マッピングテンプレート 例（詳細）

```
#set($friendIds = [])
#set($friendships = $ctx.prev.result.items)

#foreach($tmp in $friendships)
      $util.qr($friendIds.add($tmp.friendId))
#end
{
   "version" : "2018-05-29",
   "operation" : "Scan",
   "filter" : {
      "expression": "contains(:fIds, userId) AND begins_with(registtime, :trgtMo)",
      "expressionValues" : {
         ":fIds" : $util.dynamodb.toDynamoDBJson($friendIds),
         ":trgtMo" : {S" : "$util.time.nowFormatted("yyyyMM", "Asia/Tokyo")"
```

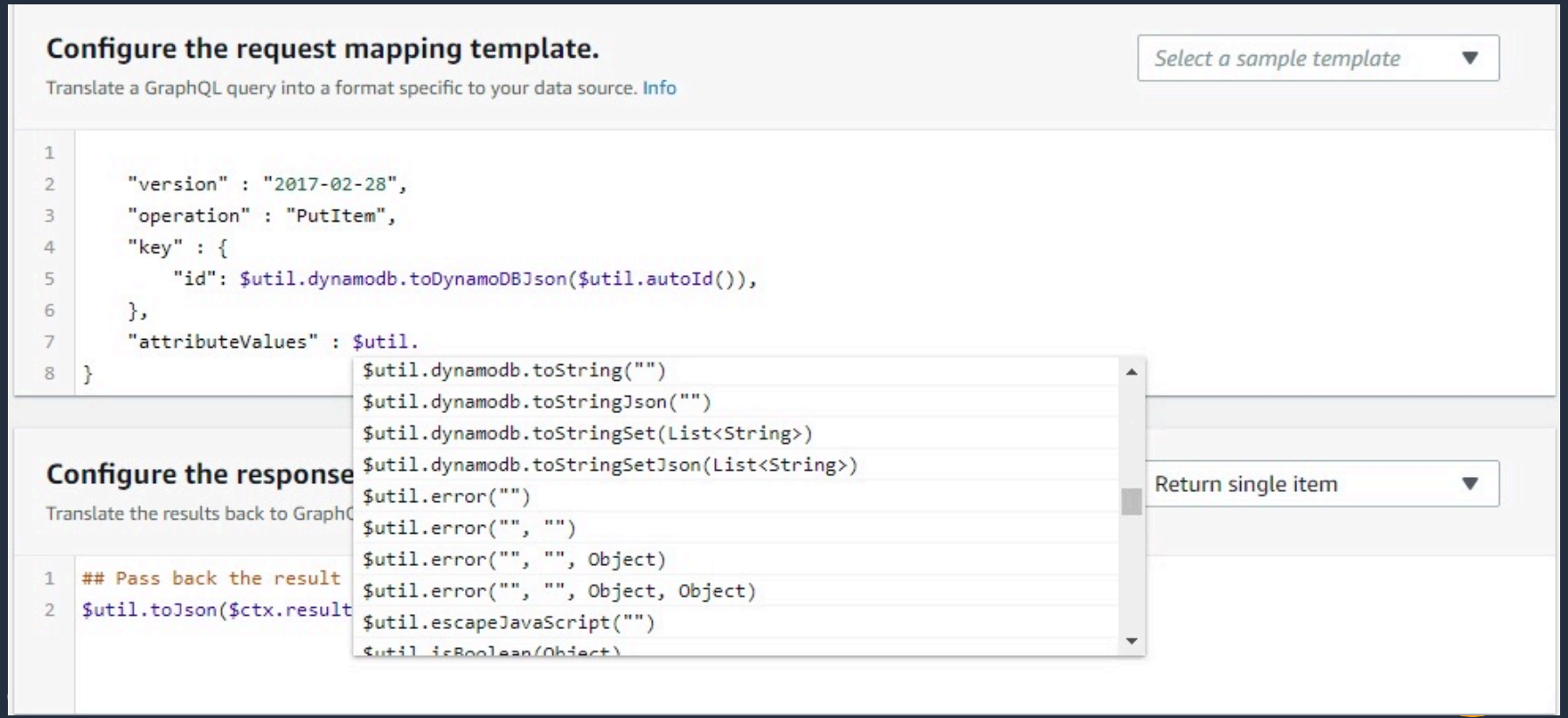
- DataSourceはDynamoDB
- 配列を作成
- 配列に前処理の結果Itemsを格納
- DynamoDBに対してScan実行

Response マッピングテンプレート 例（詳細）

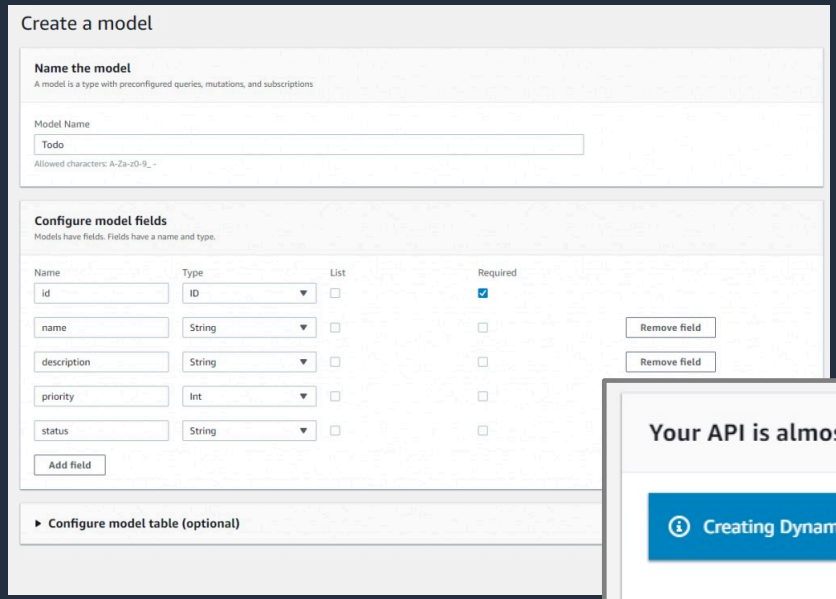
```
## Raise a GraphQL field error in case of a datasource invocation error
#if($ctx.error)
    $util.error($ctx.error.message, $ctx.error.type)
#end
## Pass back the result from DynamoDB. **
$util.toJson($ctx.result)
```

エラーが無ければJSON Parseした結果を返す

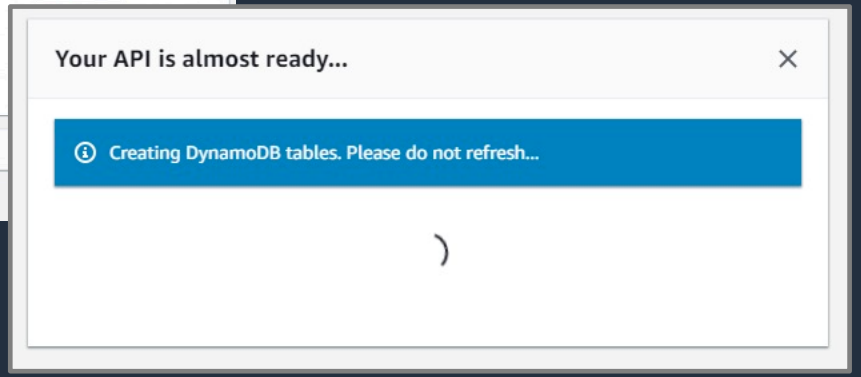
マッピングテンプレート記述方法
― 直接記述（自動補完） ―



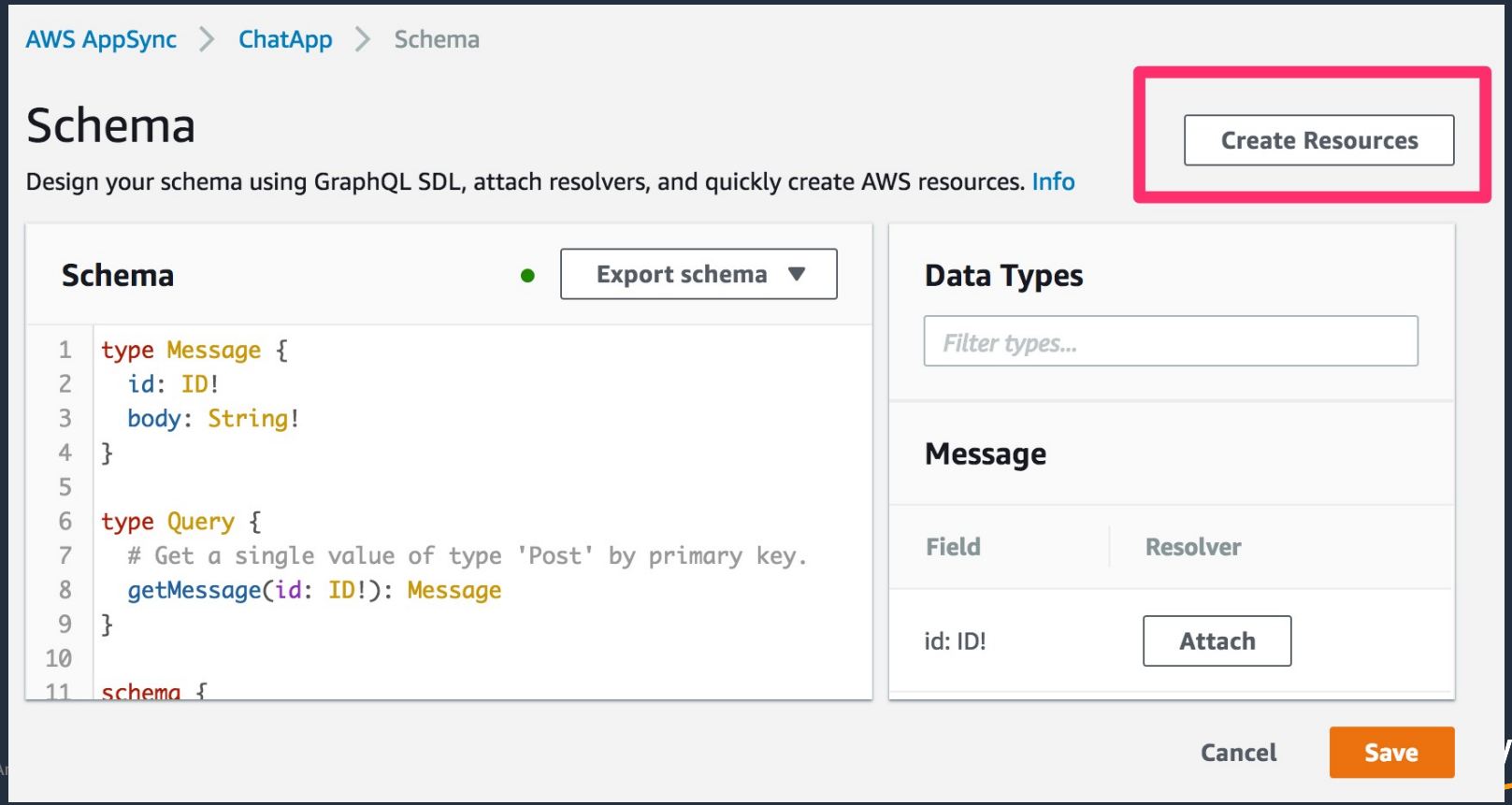
マッピングテンプレート記述方法
― No-code GraphQL API Builderを利用 ―



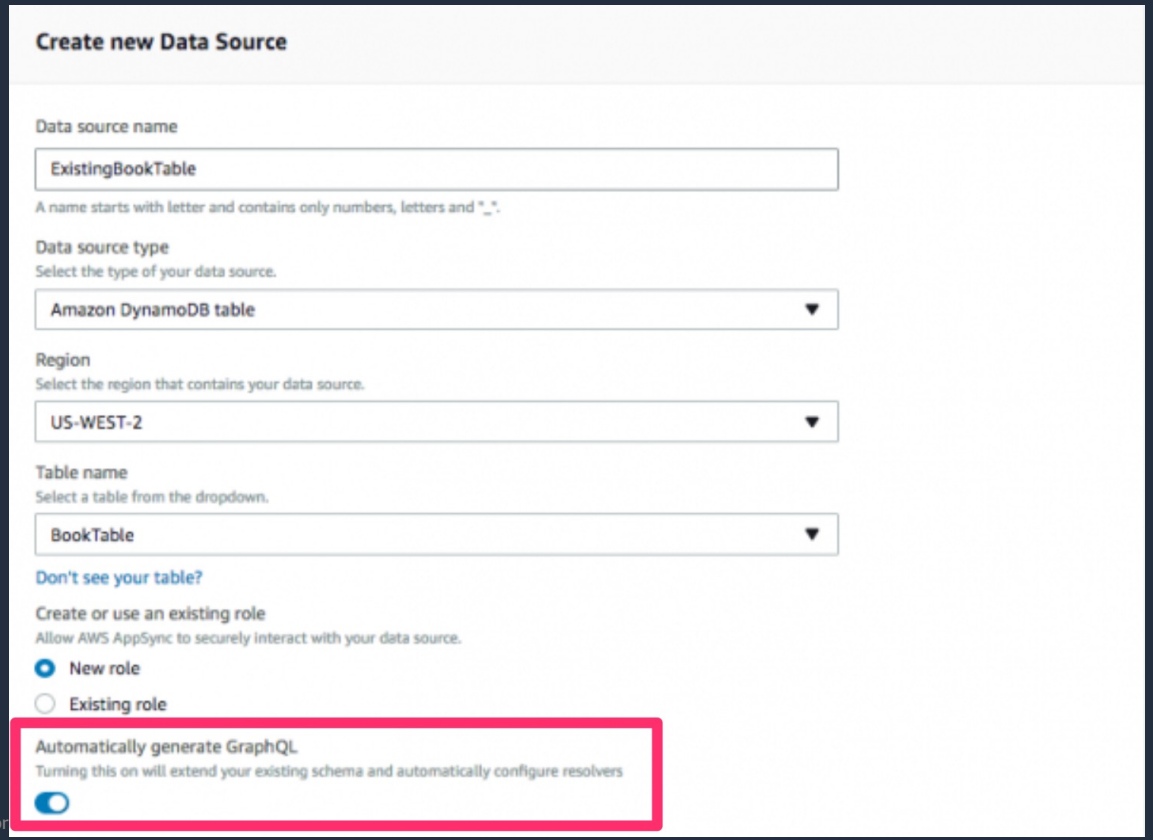
- スキーマ内の定義済みのTypeから DynamoDB テーブルをプロビジョニング
- リゾルバを利用してフィルタ、検索、比較などを簡単に組み込み



マッピングテンプレート記述方法
― Schemaからジェネレート ―



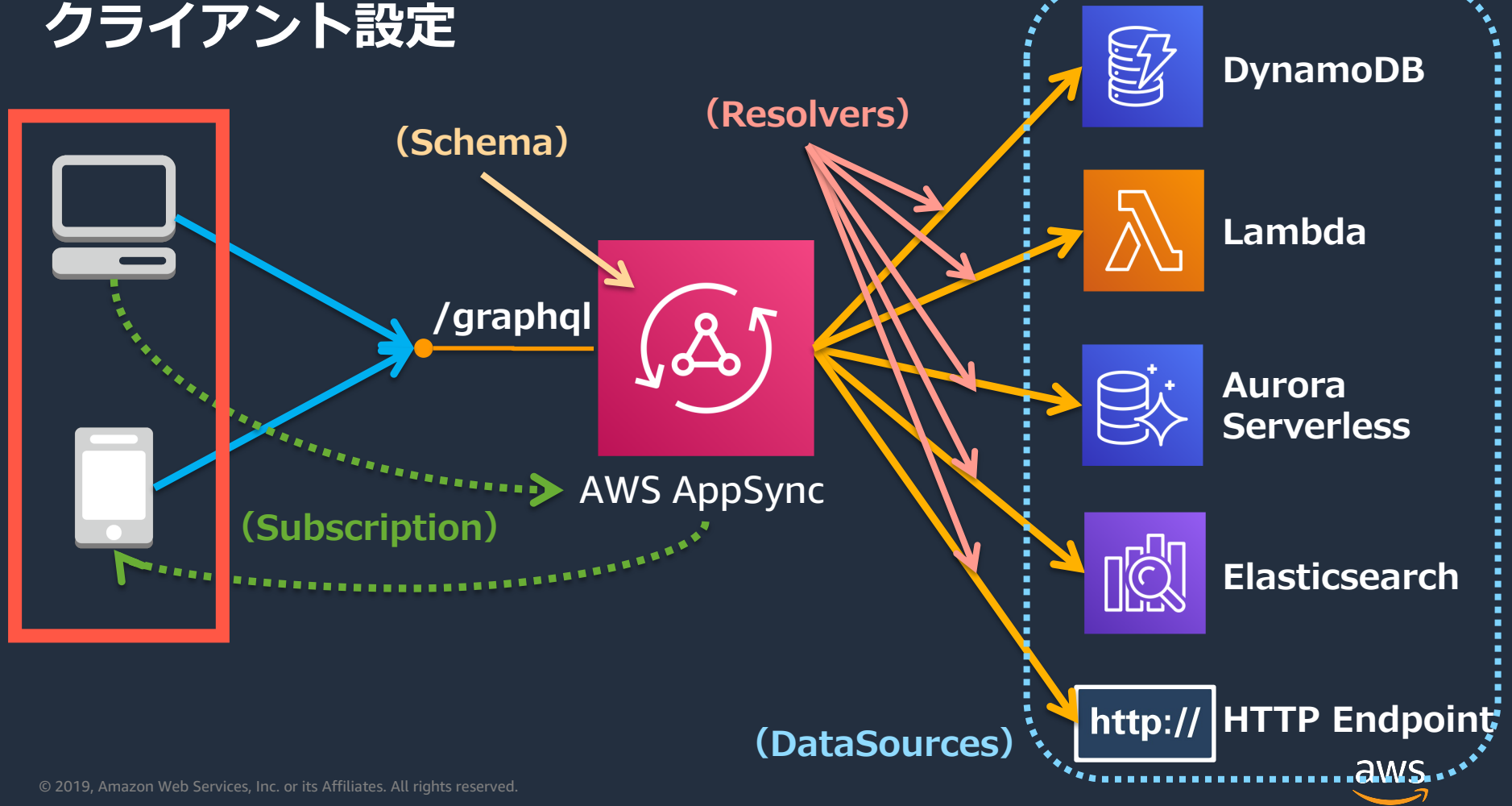
マッピングテンプレート記述方法
ー DynamoDBからジェネレート ー



迅速なプロトタイピング

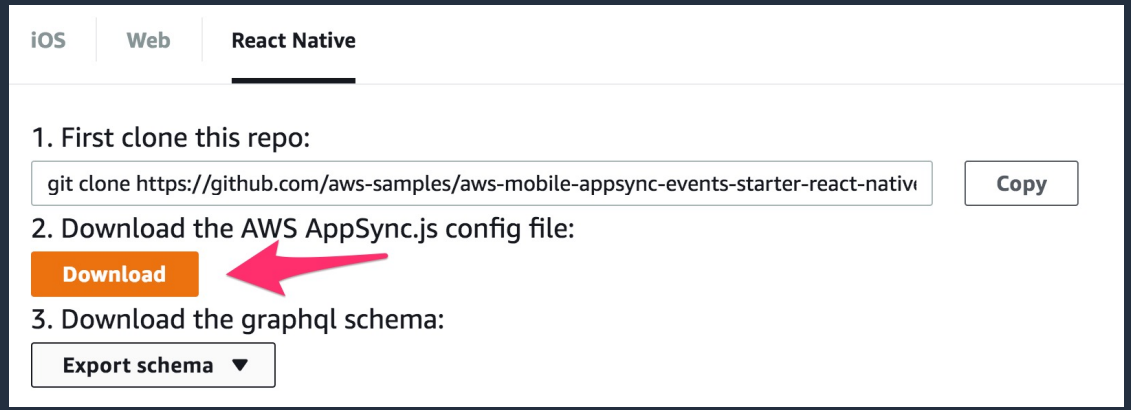
aws

クライアント設定



GraphQL Endpointへの接続情報

```
export default {
    "graphqlEndpoint": "https://**.appsync-api.**.amazonaws.com/graphql",
    "region": "us-east-1",
    "authenticationType": ”API_KEY",
    "apiKey": ”***"
}
```



クライアント設定

```
const client = new AWSAppSyncClient({
    url: awsconfig.ENDPOINT,
    region: AWS.config.region,
    auth: { type: AUTH_TYPE.AWS_IAM, credentials: Auth.currentCredentials() }
});

const WithProvider = () => (
    <ApolloProvider client={client}>
        <Rehydrated>
            <AppWithData />
        </Rehydrated>
    </ApolloProvider>
);
```

https://aws.github.io/aws-amplify/

自動でオフライン利用が可能に

aws

クライアント認証例

```
//API Key
const client = new AWSAppSyncClient({
  url: awsconfig.ENDPOINT,
  region: awsconfig.REGION,
  auth: { type: AUTH_TYPE.API_KEY, apiKey: awsconfig.apiKey}
});
```

クライアント認証例

```
//IAM認証
auth: { type: AUTH_TYPE.AWS_IAM,
    credentials: Auth.currentCredentials()
}

//Cognito User Pool 認証
auth: { type: AUTH_TYPE.AMAZON_COGNITO_USER_POOLS,
    jwtToken: Auth.currentSession().accessToke.jwtToken
}
```

料金

料金

	無料枠 (月) *	価格
クエリとデータ変更操作	250,000 件	$4.00 / 100万件
リアルタイム更新	250,000 件	$2.00 / 100万件
リアルタイム更新 接続時間 （分）	600,000 接続-分	$0.08 / 100万分

* AWSアカウントのサインアップから12ヶ月間

重要機能アップデート

重要機能アップデート（1）

1. Pipeline Resolver / Delta Sync

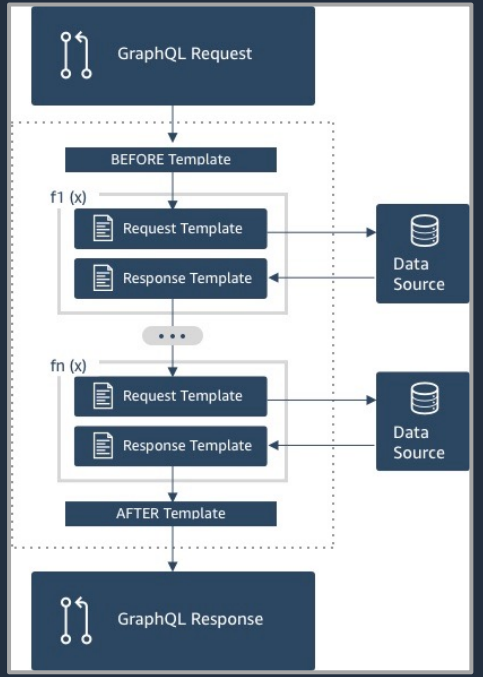
- 複数ResolverをFunctionとして束ねて実行する事が可能に
- Function間のデータ受け渡しもVTLで記述可能
- Delta Syncでキャッシュ2層に分割した差分同期が可能に (Pipeline Resolverとの併用を推奨)

2. DataSourceにAurora Serverlessを追加

- DataSourceにAurora Serverlessが利用可能に

3. AWS Amplify FrameworkがAurora Serverless、GraphQL、OAuthに対応

- DataSourceとしてAurora Serverlessを指定可能に
- GraphQL変換ライブラリにてOAuth対応



重要機能アップデート（2）

4. AppSyncがGraphQL APIのタグ付けサポート

タグ付けによりQuery、Mutation、Subscription等の操作に対するコスト配分及び追跡する事が可能に

5. AppSyncでGraphQLオペレーションのパフォーマンスと状態の可視性が向上

- AppSync単独でのログ出力は元々設定で可能
- CloudWatch Logs Insights、Amazon Elasticsearch、その他ログ分析ソリューションへのシームレスな統合が可能に
- GraphQLへのRequestのパフォーマンスやSchema Fieldの使用状況の把握が容易に

重要機能アップデート（３）

６. Amplify FrameworkにてLambda関数、DynamoDB カスタムインデックスをサポート

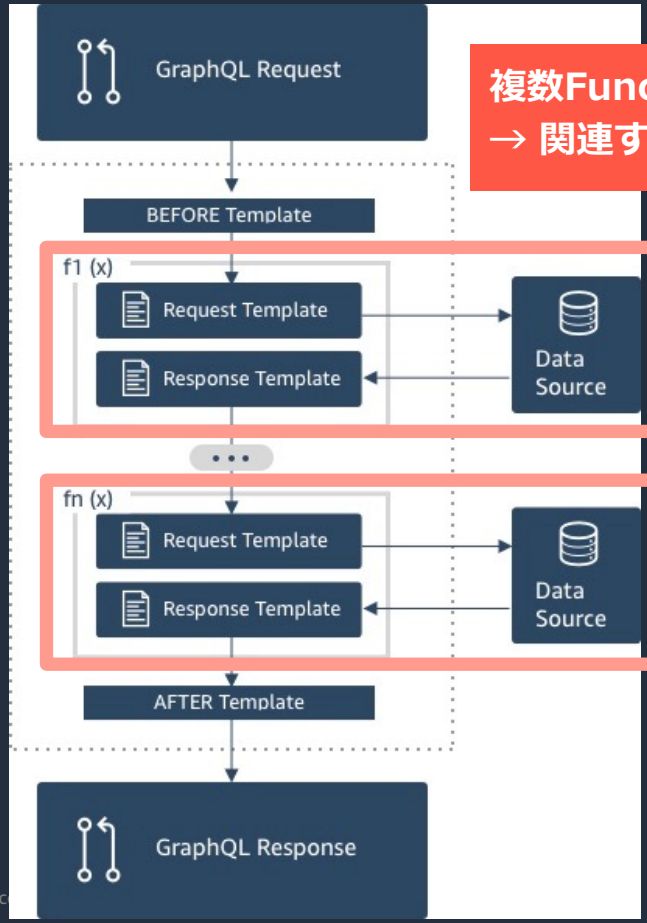
- GraphQL Transformer及び新しい@functionディレクティブを使用してLambda関数を追加する事が可能に

７. Multi-Auth対応

- 複数の認証タイプを組合せて同時に設定可能に
  { API Key、IAM、Cognito User Pool、OIDC }
- 複数認証プロバイダ対応が可能に
- Dev/Prod環境での認証方式の変更等が可能に

aws

1. Pipeline Resolver / Delta Sync



Pipeline Resolver

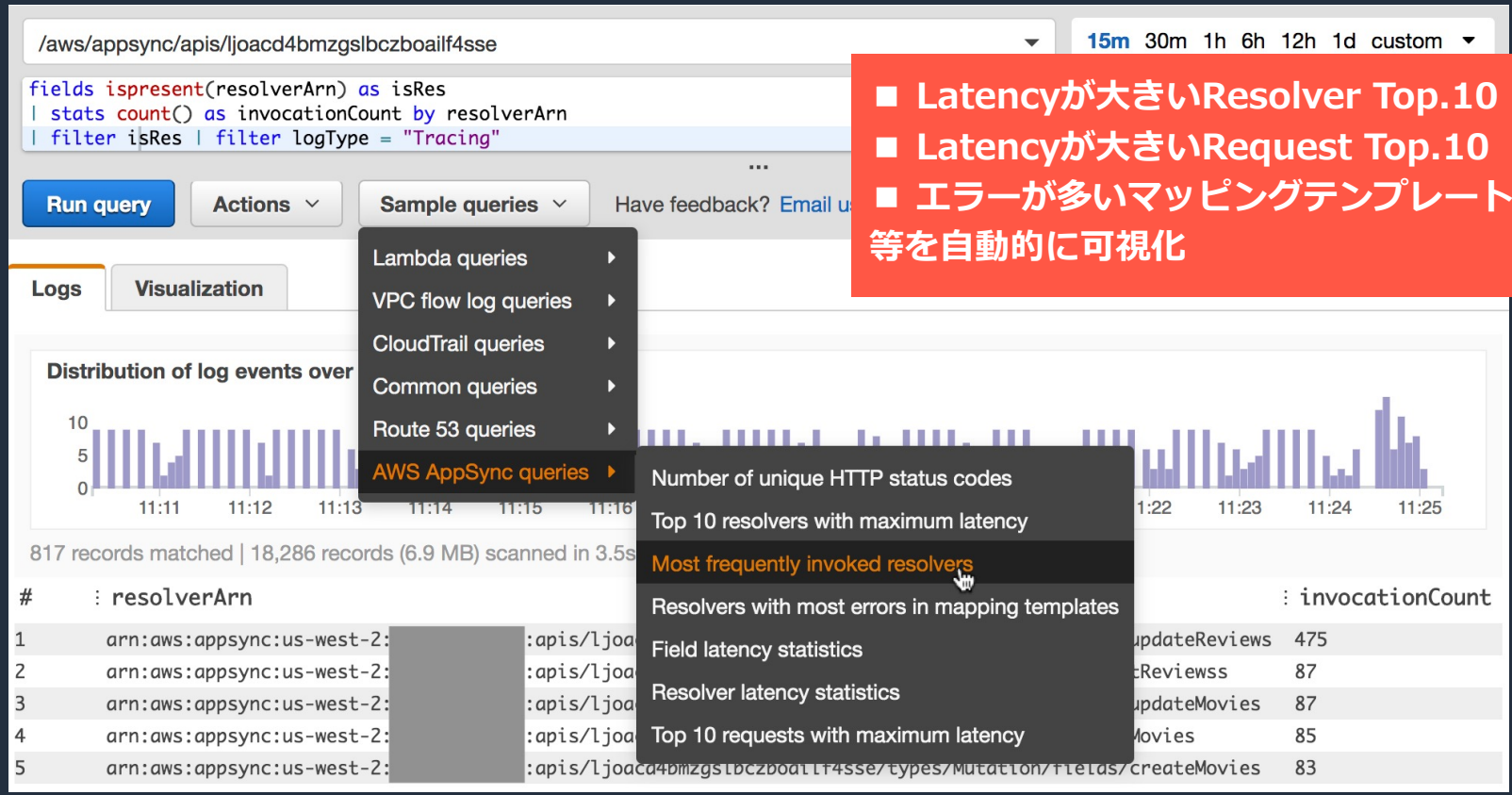
複数Functionで構成された1Resolverとして動作する
→ 関連する連続処理を纏めて1つのResolverとして動作可能

Function1
単独Resolverとして実行

Function２
単独Resolverとして実行
(Function1の結果取得も可能)

aws

5. AppSyncでGraphQLオペレーションのパフォーマンスと状態の可視性が向上

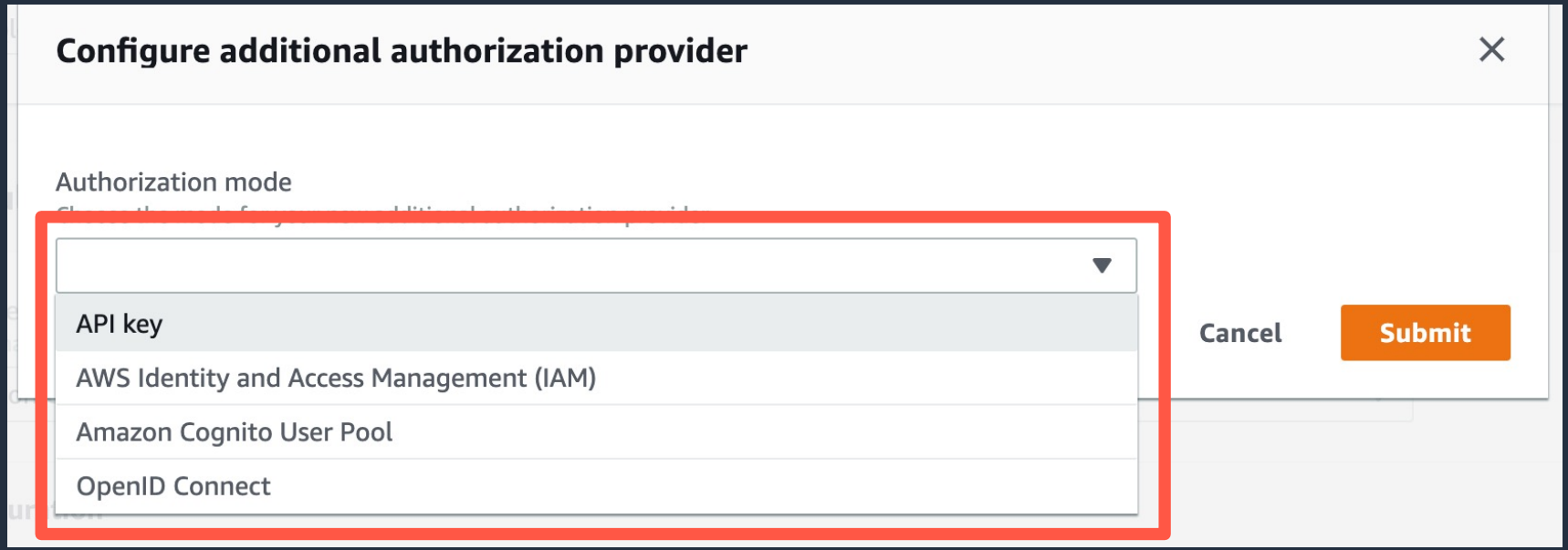


- Latencyが大きいResolver Top.10
- Latencyが大きいRequest Top.10
- エラーが多いマッピングテンプレート

等を自動的に可視化

5. AppSyncでGraphQLオペレーションのパフォーマンスと状態の可視性が向上

7. Multi-Auth対応



API Key、IAM、Cognito User Pool、OIDCから複数設定可能になった事により、Dev/Prod環境での認証使い分けや、複数認証プロバイダを利用するより柔軟な認証が可能に。

まとめ

まとめ

1. 型指定されたスキーマ

APIドキュメントを手動で記述する必要が無くなり、
APIを定義したスキーマをベースに自動生成

2. クライアントからのレスポンス形式の指定

・オーバーフェッチ、アンダーフェッチが無くなる
・クリーンなインタフェース

3. サブスクリプションを利用したリアルタイム処理

クライアントはデータをサブスクライブする事でイベント・
ドリブンに処理を実装可能

aws

Happy coding with AWS AppSync

aws

Q&A

いただいたご質問については
AWS Japan Blog 「https://aws.amazon.com/jp/blogs/news/」にて
後日掲載します。

8 月の Black Belt Online Seminar 配信予定

https://amzn.to/JPWebinar

~~08/06 (火) 12:00-13:00 AWS Glue~~
~~08/13 (火) 12:00-13:00 実践的 Serverless セキュリティプラクティス~~
~~08/14 (水) 18:00-19:00 AWS Serverless Application Model~~
~~08/20 (火) 12:00-13:00 Serverless モニタリング~~
08/21 (水) 18:00-19:00 AWS AppSync
08/28 (水) 18:00-19:00 Amazon Aurora with PostgreSQL Compatibility

AWS の日本語資料の場所「AWS 資料」で検索

https://amzn.to/JPArchive

第7回
Amazon SageMaker 事例祭り

2019 年 8 月 29 日（木）
アマゾン新目黒オフィス
目黒セントラルスクエア21F で開催

詳細・お申込みはこちらから>>
https://amzn.to/2MtH514

aws

アジェンダ

- ❑ AWSの機械学習サービス概要
- ❑ Amazon SageMaker の基礎
- ❑ Amazon SageMaker Ground Truthの概要とデモンストレーション

- ❑ **お客様事例**
  - 株式会社CACクロア 様
    「世に出ている医薬品の副作用情報を管理する業務で、どのようにAI/MLを活用しているか?」
  - サントリーシステムテクノロジー株式会社 様
    「技術検証プロジェクトにおけるSageMaker活用例」

DEV DAY
TOKYO

今年もやります！イノベーションをリードするDeveloperのための秋の祭典「**AWS DevDay Tokyo 2019**」
- 今おさえておくべき技術領域を網羅し、新たな時代に活躍するDeveloperとなるための知識とスキルを身に着ける2日間
- 豪華スピーカーによるゼネラルセッション
- 50にのぼるセッションとワークショップで構成。CFP枠を拡大し、より実践的なコンテンツへ

日程：2019年10月3日（木）ー4日（金）
会場：神田明神ホール（東京都千代田区 神田明神内）
主催：アマゾン ウェブ サービス ジャパン株式会社
協賛：インテル株式会社

今年の会場は「神田明神」

aws

Save the dateはこちらへ
https://amzn.to/devdaytokyo2019

ご視聴ありがとうございました

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive