AWS
Black Belt
Online Seminar

【AWS Black Belt Online Seminar】 AWS AppSync

アマゾン ウェブ サービス ジャパン株式会社
ソリューションアーキテクト　塚越 啓介

2018.05.23

Who am I ?

- 塚越 啓介（つかごし けいすけ）
  - Specialist Solution Architect
    @Amazon Web Services Japan
  - Mobile / DevOps / Serverless
- Background = アプリケーション開発
  - Webサービスの開発・運用
  - リアクティブプログラミング
  - Agile開発のコンサルティング
- 好きなサービス： AWS AppSync

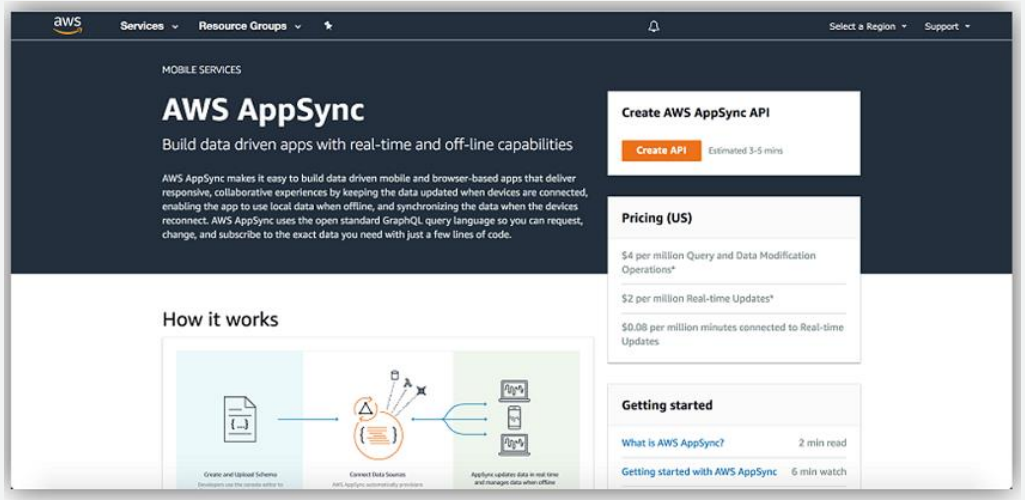
内容についての注意点

- 本資料では2018年5月16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.

AWS AppSync とは

リアルタイム機能とオフライン機能を備えた
フルマネージド GraphQL サービス

管理やスケールを気にせずに使える



Overview

Rest APIの次のパラダイムとして注目されているGraphQL。
AWS AppSyncというマネージドサービスを利用することで
簡単にGraphQLを使い始めることが可能。

「 GraphQL vs REST 」という形で紹介されることが多く
REST APIの置き換えという使われ方が多い

リアルタイム機能とオフライン機能を簡単に実装できる

ユースケース

- 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など、さまざま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

データを扱う際のデベロッパーの課題

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

GraphQLの概要

GraphQL ?

- GraphQL 、はAPI 用のクエリ言語であり、TypeSystem を使用してクエリを実行するためのサーバー側のランタイム。

- GraphQL は、クライアントがサーバーからデータをQuery(取得)、Mutation(変更)、Subscription(購読)できるようにするためのデータ言語。

- GraphQL クエリでは、クライアントがレスポンスの形式を指定する。これにより、クライアントは必要なデータのみを必要な形式で照会することが可能。

aws

GraphQLはどう動くか?

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

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

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

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

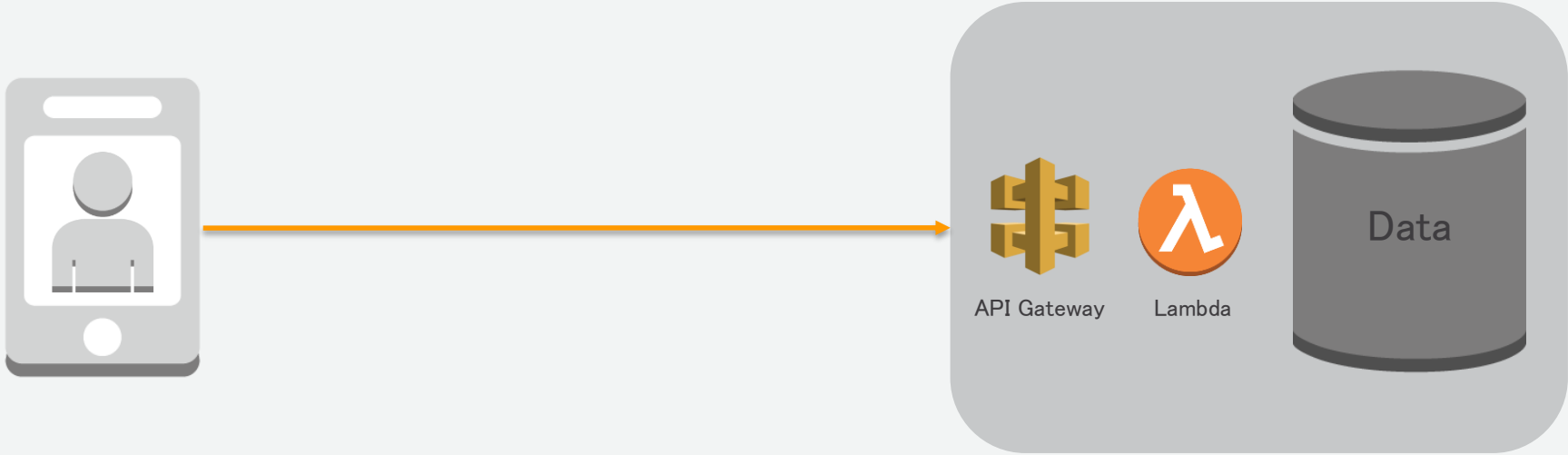
アプリのスキーマとモデルデータ

クライアントが必要なものだけをリクエスト

リクエストしたデータだけが返される

aws

GraphQLで変わること



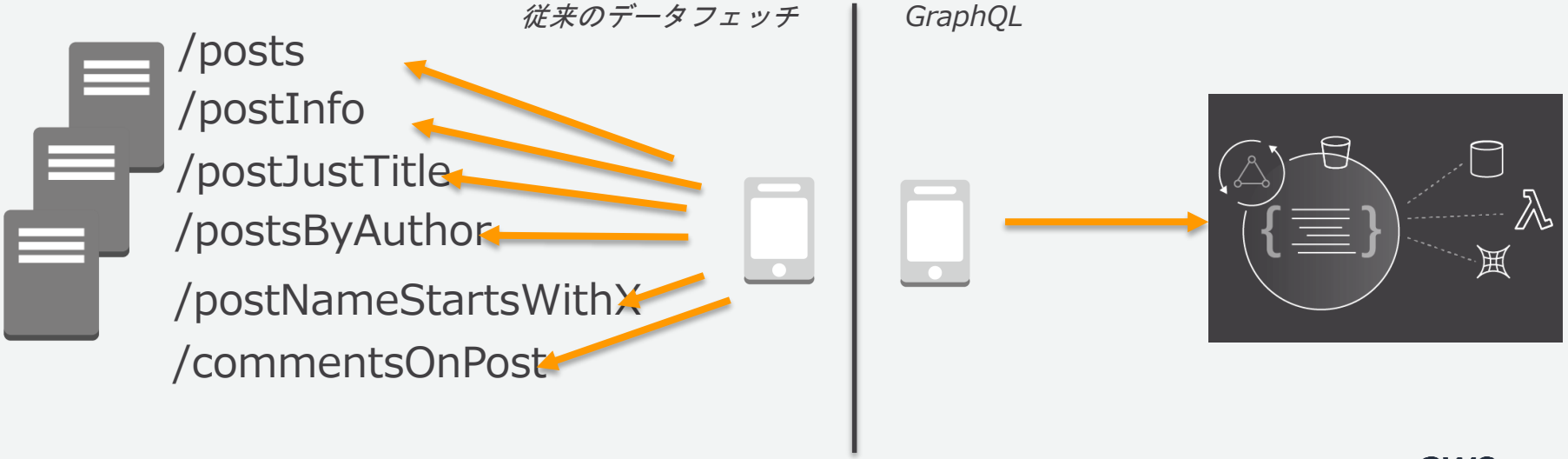
API Gateway と Lambda でDBにアクセスする
ケースをAppSyncでシンプルに

GraphQLで変わること

オープンで宣言的なデータ取得の仕様

!= Graph database

NoSQL, Relational, HTTP, etc.



GraphQL Subscription

ほぼリアルタイムでのデータを購読

Mutationをトリガーにしたイベントベースモード
- スケーラブルによくあるユースケースを想定して設計

AppSyncの任意のデータソースを使える
- Lambda, DynamoDB, Elasticsearch

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

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

GraphQL サブスクライブ ハンドシェイク

```
Subscription NewPostSub {
        addedPost{…}
}
```

WebSocket URL and Connection Payload

Secure Websocket Connection (wss://)

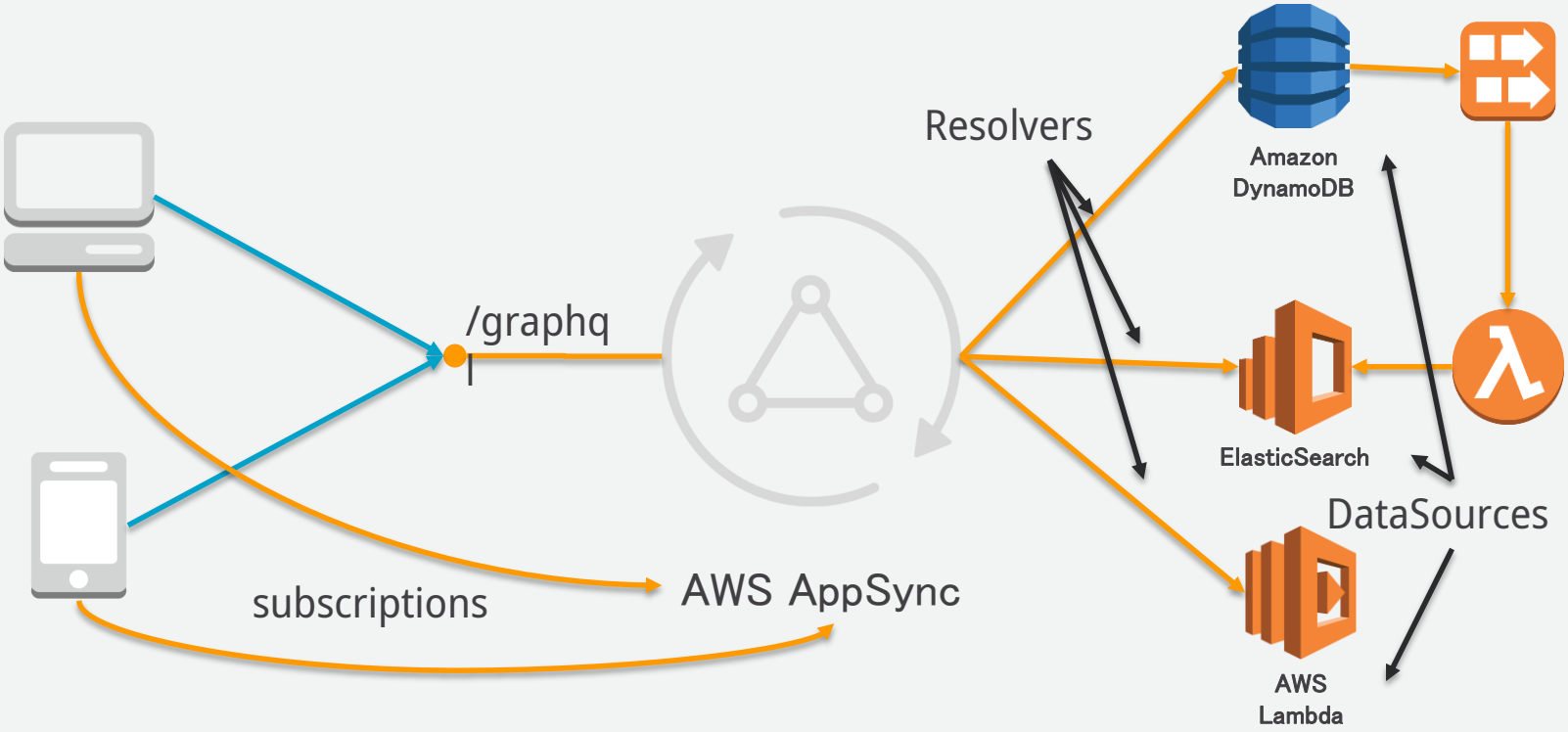
aws

AppSyncの概要

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
- **Data Source** : DynamoDB / Elastic Search / Lambda
- **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
- **Action** : GraphQL から Subscriber への通知
- **Resolver** : リクエスト / レスポンスの処理を記述する関数

AppSync Overview



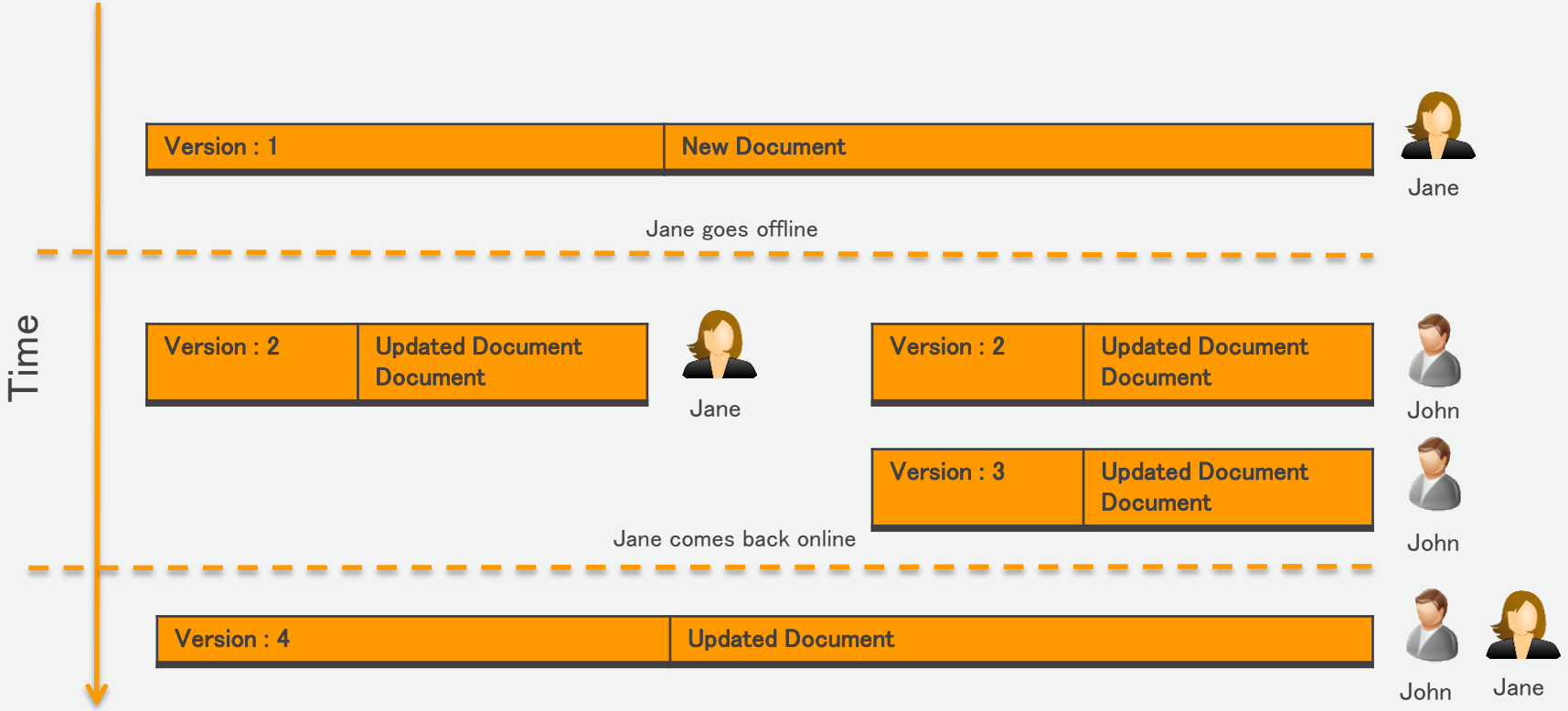
dynamoDBとAmazonES

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、e.t.c

LambdaとThirdPartyAPI

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

オフライン mutations



コンフリクトの解消と同期

クラウドでのコンフリクト解消

1. サーバー優先
2. サイレントリジェクト
3. カスタムロジック (AWS Lambda)

- バージョンチェック
- カスタムチェック

オプション

- クライアントコールバックでエラーハンドリングも可能

Example: Check that an ID doesn't already exist:

```
{
   "version" : "2017-02-28",
   "operation" : "PutItem",
   "key" : {
      "id" : { "S" : "1" }
   },
   "condition" : {
      "expression" : "attribute_not_exists(id)"
   }
}
```

Run Lambda if version wrong:

```
"condition" : {
    "expression" : "someExpression"
    "conditionalCheckFailedHandler" : {
        "strategy" : "Custom",
        "lambdaArn" : "arn:..."
    }
}
```

画像やリッチコンテンツ

```
type S3Object {
   bucket: String!
   key: String!
   region: String!
}
```

```
type Profile {
   name: String!
   profilePic: S3Object!
}
```

```
input S3ObjectInput {
   bucket: String!
   key: String!
   region: String!
   localUri: String!
}
```

```
type Mutation {
   updatePhoto(name: String!,
               profilePicInput: S3ObjectInput!): Profile
}
```


aws

認証

認証の種類

- **API_KEY**
  - API キーを使用する場合。
- **AWS_IAM**
  - AWS IDおよびIAMのアクセス許可を使用する場合。
- **AMAZON_COGNITO_USER_POOLS**
  - Amazon Cognito　User Pools を使用する場合。

API KEY 認証

- 認証されていない GraphQL エンドポイントを作成すると使用
- AppSync によって生成されるアプリケーション内の固定値

AWS_IAM 認証

Amazon Cognito Federated Identities の一時トークンを使用してアクセス。IAM Policy を使って認可

```
{
    "Version": "2012-10-17",
    "Statement": [ {
        "Effect": "Allow",
        "Action": [ "appsync:GraphQL" ],
        "Resource": [
            "arn:aws:appsync:us-west-2:1234:apis/#ApiId#/types/Query/fields/<Field-1>",
            "arn:aws:appsync:us-west-2:1234:apis/#ApiId#/types/Query/fields/<Field-2>",
            "arn:aws:appsync:us-west-2:1234:apis/#ApiId#/types/Mutation/fields/<Field-1>",
            "arn:aws:appsync:us-west-2:1234:apis/#ApiId#/types/Subscription/fields/<Field-1>",
        ]
     ] }
}
```

COGNITO USER POOL 認証

ユーザーグループを利用可能

```
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"])
}
```

スケーラビリティ

開発における**拡張性**が非常に高い

簡単に試すことができます

スキーマの定義

簡単に試すことができます

スキーマ定義

- 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ために使用されます。

- GraphQL API は1つの GraphQL スキーマで定義され、SDL(**Schema Definition Language**)によって記述さる。

aws

スキーマの書き方

ルートスキーマを定義

Query : 取得
Mutation : 更新
Subscription : 購読

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


aws

タイプの書き方

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!]

```
type Todo {
  id: ID!
  name: String
  description: String
  status: TodoStatus
}

type Query {
  getTodos: [Todo]
}

enum TodoStatus {
  done
  pending
}
```


aws

Subscription Schema

```
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

リゾルバーマッピング テンプレート

簡単に試すことができます

リゾルバーマッピングテンプレートとは？

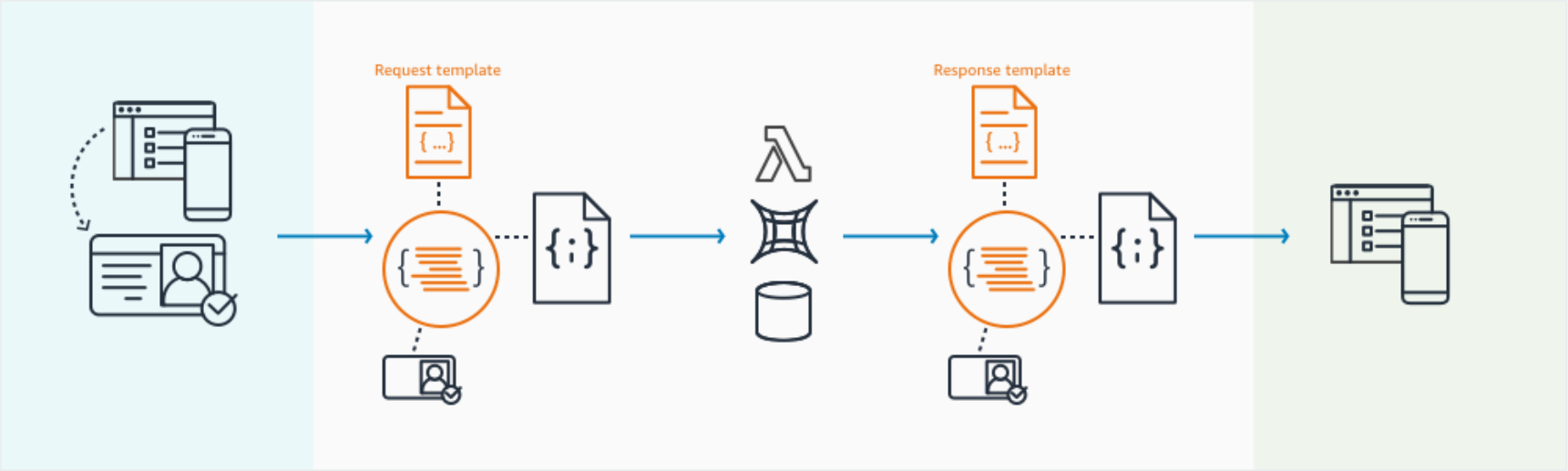
- マッピングテンプレートは、GraphQLリクエストをデータソースの命令に変換する方法と、データソースからの応答をGraphQLレスポンスに変換する方法を定義する
- Apache Velocity Template Language（VTL）
- プログラミングガイド
    - https://docs.aws.amazon.com/appsync/latest/devguide/resolver-mapping-template-reference-programming-guide.html

リゾルバーマッピングテンプレートでやることの例

- 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

マッピングテンプレートは２種類

- リクエストテンプレート
- レスポンステンプレート



データソースとリゾルバーマッピングテンプレート

リゾルバー、データソース間の通信はデータソース固有のパラメータを介して行われる

リゾルバ	パラメーター
Lambda リゾルバー	ペイロード
DynamoDB リゾルバー	キー
Elasticsearch リゾルバー	インデックス+クエリ操作

マッピングテンプレートで使えるcontext

```
“arguments” : {}, // GraphQLの引数
“source” : { // 親フィールドの情報
   "createdAt" : "2017-02-28T18:12:37Z",
   "title" : "A new post",
   "content" : "A long time ago, in a thread far far away",
   "postId" : "1234",
   "authorId" : "34521”
},
“result” : {  // リゾルバの結果　レスポンスマッピングテンプレートのみ
   "name" : "Steve",
   "joinDate" : "2017-02-28T18:12:37Z",
   "id" : "34521”
},
“identity” : { // 呼び出し元の情報 認証 e.t.c
   "sourceIp" : "x.x.x.x",
   "userArn" : "arn:aws:iam::123456789012:user/appsync",
   "accountId" : "123456789012",
   "user" : "AIDAAAAAAAAAAAAAAAAA”
}
```

例：リクエストマッピングテンプレート

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

例 : レスポンスマッピングテンプレート

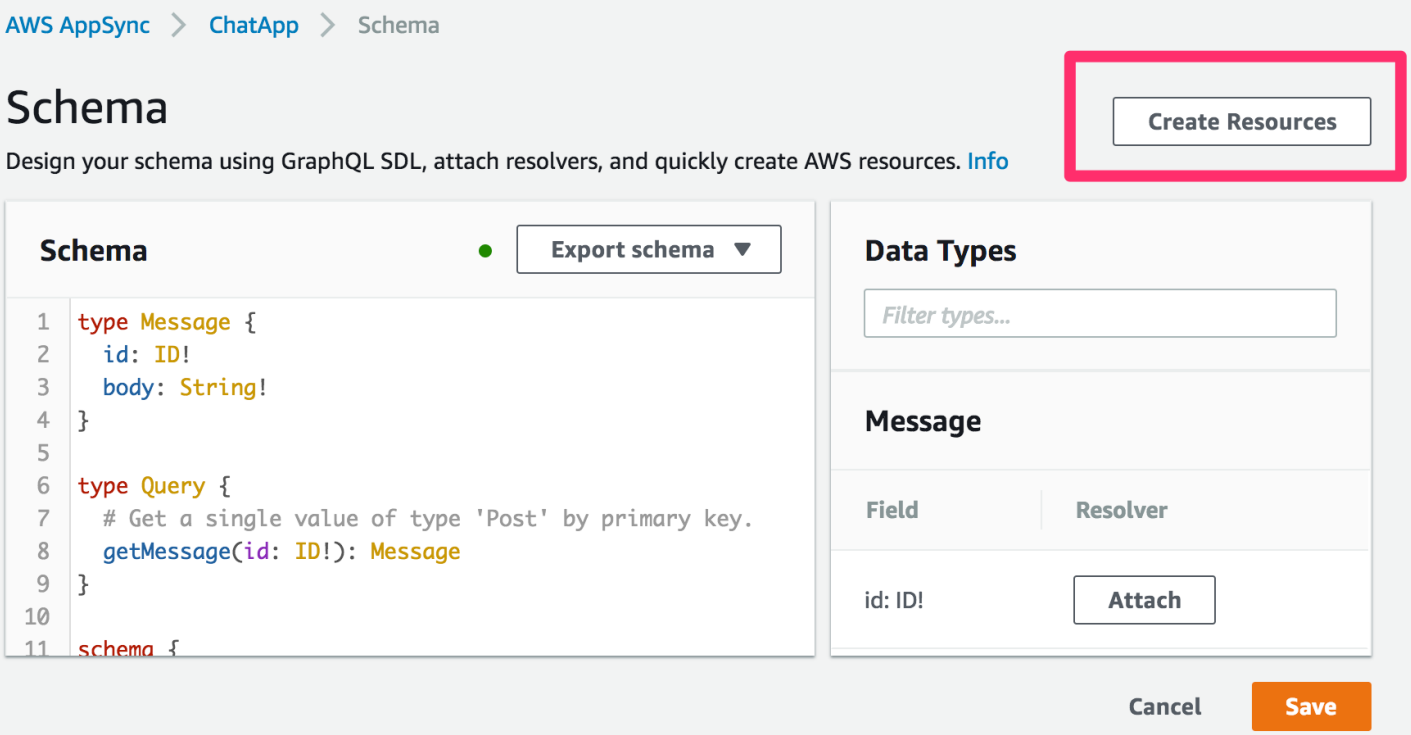
デフォルトで返す場合

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

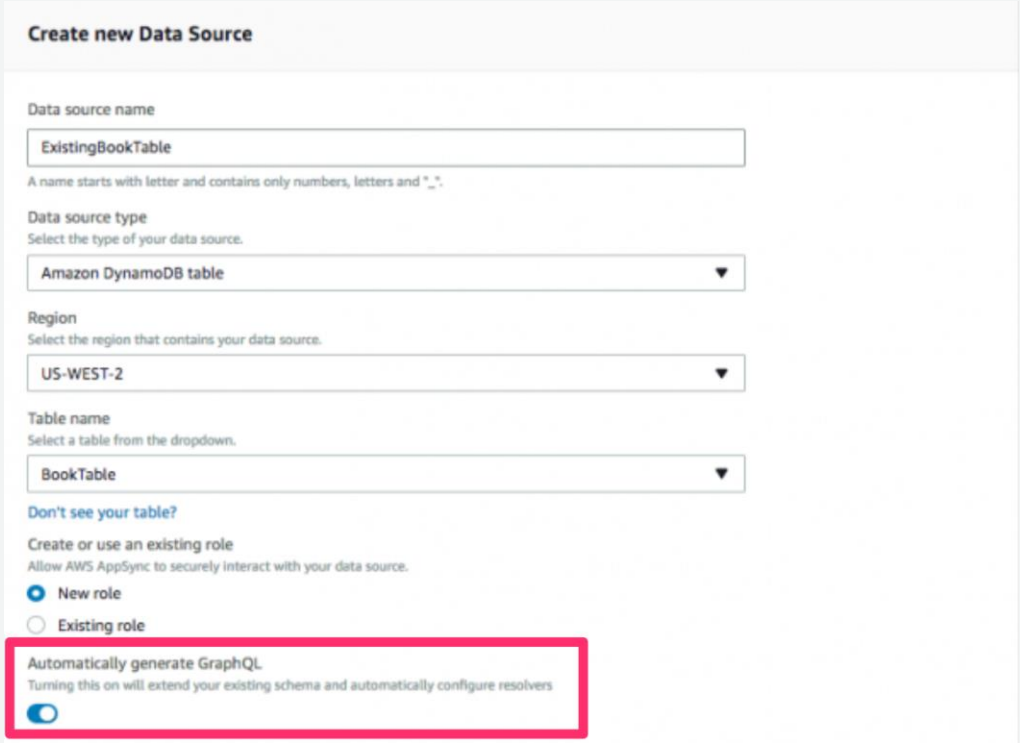
データを結合する場合

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

スキーマからジェネレート



DynamoDBからジェネレート

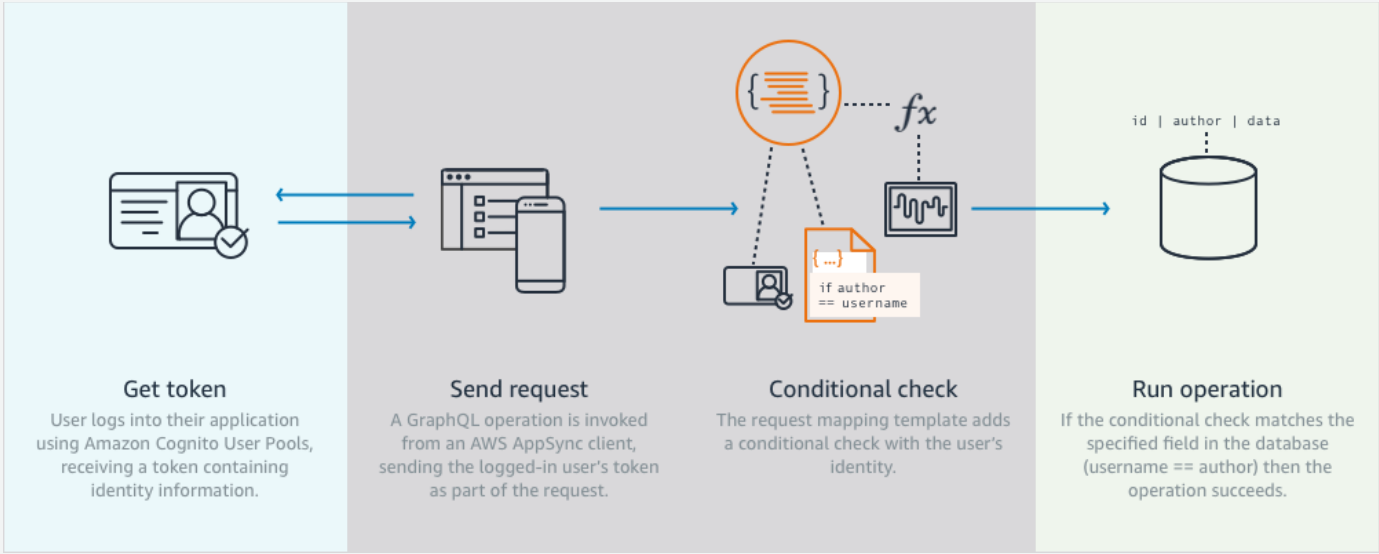


ファイングレイン
アクセスコントロール

AppSyncのファイングレイン・アクセスコントロール

特定の条件のユーザーだけにデータアクセスさせる場合

リゾルバーマッピングテンプレートを使って解決



アクセスコントロールの例

認可メタデータをリソースと共に格納する必要がある
メタデータ+Contextを使ってアクセスコントロールを実施

ID	Data	PeopleCanAccess	GroupsCanAccess	Owner
123	{my: data,...}	[Mary, Joe]	[Admins, Editors]	Nadia

```
#if($context.result["Owner"] == $context.identity.username)
    $utils.toJson($context.result);
#else
    $utils.unauthorized()
#end
```


https://docs.aws.amazon.com/appsync/latest/devguide/resolver-context-reference.html#aws-appsync-resolver-mapping-template-context-reference

アクセスコントロールの例

owner情報を書き込む

```
“operation” : “PutItem”,
"key" : {
    "postId" : { "S" : "${context.arguments.id}" } },
"attributeValues" : {
    ”owner" : {"S" : "${context.identity.username}"}
    #foreach( $entry in $context.arguments.entrySet() )
        #if( $entry.key != "id" )
            ,"${entry.key}" : { "S" : "${entry.value}" }
        #end
    #end },
"condition" : { "expression" : "attribute_not_exists(postId)" }
```


https://docs.aws.amazon.com/appsync/latest/devguide/resolver-context-reference.html#aws-appsync-resolver-mapping-template-context-reference

Resolver Helper

Resolver helper functions

- Resolverを簡単にするためのヘルパー
- Resolver Mapping Template Programming Guide
- https://docs.aws.amazon.com/appsync/latest/devguide/resolver-mapping-template-reference-programming-guide.html
- Resolver リファレンス
- https://docs.aws.amazon.com/appsync/latest/devguide/resolver-context-reference.html

aws

Resolver helper functions の例 (1/2)

データ検証

$utils.isNull()

リストの処理

$utils.list.copyAndRemoveAll()

正規表現

$utils.matches()

自動採番

$utils.autoId()

Resolver helper functions の例 (2/2)

Timestampの処理

  $utils.time

DynamoDB関連の処理

  $utils.dynamodb

画像やビデオなどのコンテンツの扱い

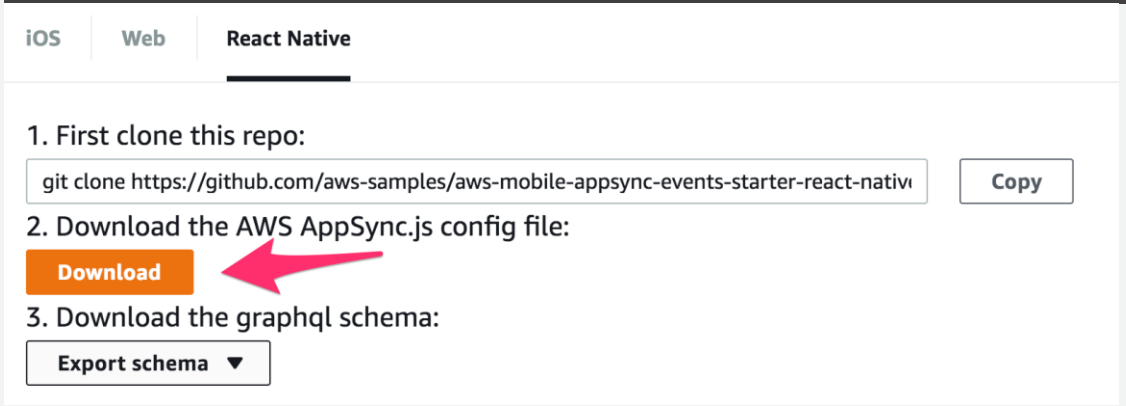
  $utils.dynamodb.toS3Object()

クライアントの設定

簡単に試すことができます

AppSyncへの接続情報

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



aws

Clientの設定

```
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

Clientの認証

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

Clientの認証

```
//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
}
```

query

```
export default gql`
query ListEvents {
    listEvents{
      items{
        __typename
        id
        name
        where
        when
      }
    }
}`;
```

compose

```
const AllEventWithData = compose(
    graphql(ListEvents, {
        options: {
            fetchPolicy: 'cache-and-network'
        },
        props: (props) => ({
            events: props.data.listEvents ?
                    props.data.listEvents.items : [],
        })
    }),
)(AllEvents);
```

render

```
render() {
  const { events, error } = this.props;
    return (
      <View style={styles.allEventPageStyle}>
        <ScrollView contentContainerStyle={styles.scroller}
          refreshing={true}>
          <View style={styles.container}>
            {[].concat(events).sort((a, b) => moment.utc(b.when) -
moment.utc(a.when)).map(this.renderEvents)}
          </View>
        </ScrollView>
      </View>
    );
}
```

Real time UI updates

```
const AllPostsWithData = compose(
  graphql(AllPostsQuery, { options: { fetchPolicy: 'cache-and-network‘ },
    props: (props) => ({
      posts: props.data.posts,
      subscribeToNewPosts: params => {
          props.data.subscribeToMore({
              document: NewPostsSubscription,
              updateQuery: (prev, { subscriptionData: { newPost } }) => ({
                 ...prev,
                 posts: [newPost, ...prev.posts.filter(post => post.id !== newPost.id)]
              })
            });
        });
    …//more code
```


aws

利用料金

	Free Tier	Standard Cost
Queries	250,000	$4 / million
Real-time Updates	250,000	$2 / million
Real-time Connection-minutes	600,000	$0.08 / million

- 最初の１２ヶ月は無料枠有
- データ転送量＋データベースに注意
- 最新の価格はAWSの料金ページを参照

料金の例

- 2,500ユーザーのチャットアプリの例
  - Average user connects for 1,500 minutes
  - 送信1,000 と 受信1,000のメッセージ
  - 2.5M クエリ と2.5Mのリアルタイムの更新

AppSync Query	2.5M x $4/million = $10.00
AppSync Real-time	2.5M x $2/million = $5.00
AppSync Minutes	2,500 x 1,500 x $0.08/million = $0.30
Data Transfer	1KB x 2.5M = 2.4GB x $0.09 = $0.21
DynamoDB Database	Free Tier (as long as store < 25Gb)
Total	$15.51

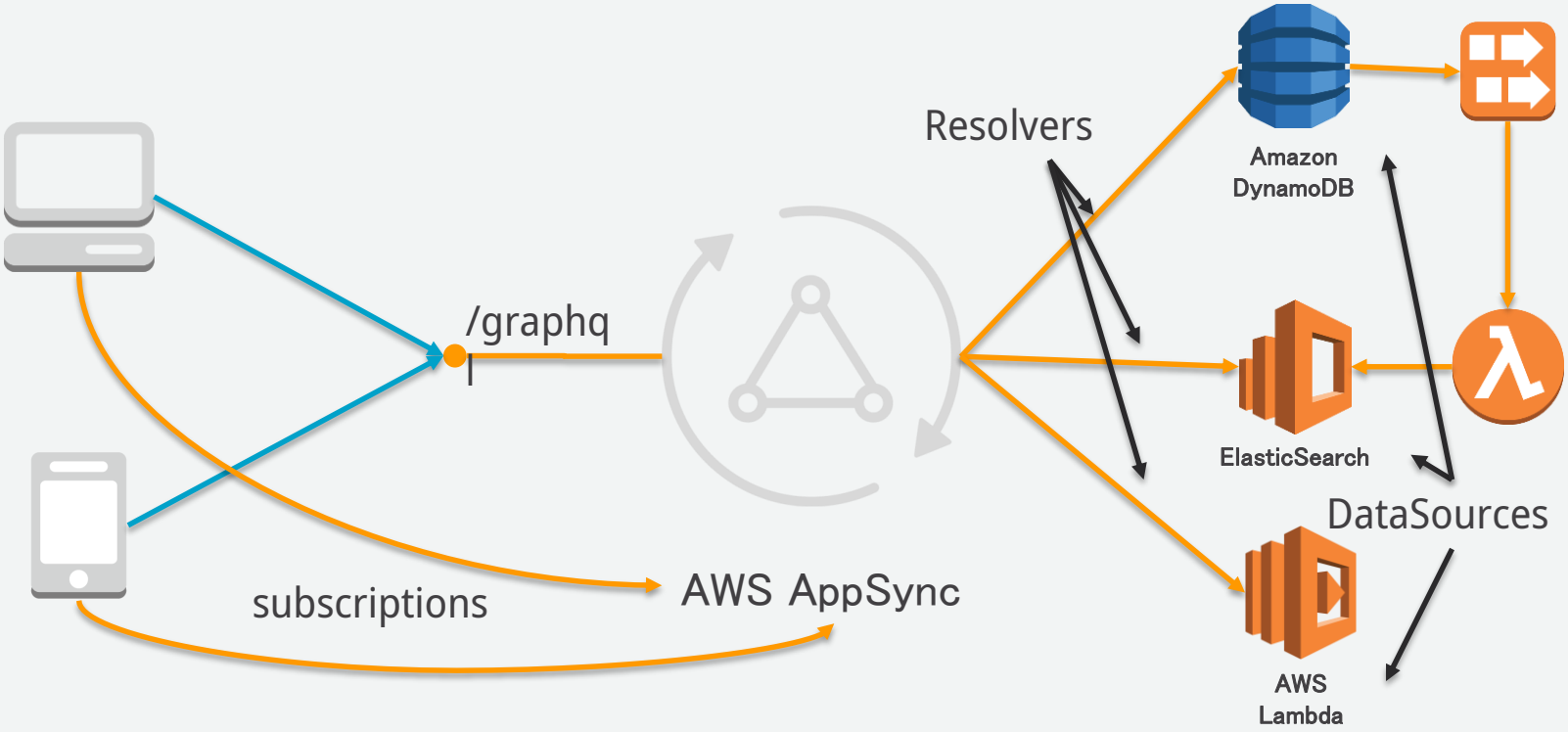
まとめ

- スケーラブルな GraphQL　API を簡単にデプロイできます

- AWS AppSyncを利用することでリアルタイム処理、オフライン処理を簡単にサポートできます

- サーバーアプリケーションとクライアントアプリケーションの
インターフェースをシンプルにできます

Happy coding with AppSync!



オンラインセミナー資料の配置場所

AWS クラウドサービス活用資料集

- https://aws.amazon.com/jp/aws-jp-introduction/

Amazon Web Servicesブログ (「AWS Japan Blog」で検索)

- 最新の情報、セミナー中のQ&A等が掲載されています。
- https://aws.amazon.com/jp/blogs/news/

公式Twitter/Facebook
AWSの最新情報をお届けします

@awscloud_jp

もしくは
http://on.fb.me/1vR8yWm

最新技術情報、イベント情報、お役立ち情報、
お得なキャンペーン情報などを日々更新しています！

お問い合わせ先

AWS導入に関しての問い合わせ
http://aws.amazon.com/jp/contact-us/aws-sales

(ご利用者様向け)課金・請求内容、アカウントに関するお問い合わせ
https://aws.amazon.com/jp/contact-us/

AWS技術サポート
https://aws.amazon.com/jp/premiumsupport/