

# Level Up! AWS Amplify

## ～ 爆速かつスケーラブルなフルスタック Web / モバイルアプリケーションの開発 ～

鈴木 貴博

技術統括本部 エンタープライズ技術本部 ストラテジックエンタープライズ ソリューション部  
アマゾン ウェブ サービス ジャパン合同会社

# 自己紹介

鈴木 貴博 (Takahiro Suzuki)



## 所属

アマゾン・ウェブ・サービスジャパン合同会社  
技術統括本部  
ソリューションアーキテクト

## 好きなAWSサービス



AWS Amplify



Amazon EKS



AWS Fargate



# はじめに

## このセッションの想定視聴者

- ・ 個人開発、スタートアップ、エンタープライズを問わず、フロントエンド含めたフルスタックアプリケーションの構築を検討されている方
- ・ 既に Amplify を使った開発をされているが、最新版の機能をキャッチアップしたい方
- ・ AWS Amplify における GraphQL API のスキーマ設計に悩まれている方

## このセッションでお伝えしたいこと

- ・ 用途・規模に応じた Amplify の各機能の使い分け
- ・ GraphQL Transformer v2 を使用したスキーマ設計



# 本日のご紹介する内容

- AWS Amplify の主要機能 (CLI, Libraries, Studio, Hosting)
- GraphQL Transformer v2 によるスキーマ設計

# AWS Amplify の主要機能



## Amplify CLI

Web やモバイルアプリケーションを一般的なユースケースベースのガイド付きワークフローでバックエンドを簡単に作成、管理するツール

バックエンド

ローカル実行

Infrastructure as Code



## Amplify Studio

AWS 上に最小限のコーディングでフロントからバックまでのアプリケーションを作成できるビジュアルな開発環境

デザイン (Figma)

フロントエンド

ヘッドレス CMS



## Amplify Libraries

Web やモバイルアプリケーションと AWS を統合するためのユースケース中心のライブラリ

フロントエンド

バックエンド連携



## Amplify Hosting

継続的デプロイメントを管理し、モダンな Web アプリケーションをビルト、テスト、デプロイ、そしてホスティングするための AWS サービス

CI/CD

Web ホスティング

E2E テスト

SSR

ISR

SSG



# Amplify CLI



Amplify CLI

Web やモバイルアプリケーションを一般的なユースケースベースのガイド付きワークフローでバックエンドを簡単に作成、管理するツール

バックエンド

ローカル実行

Infrastructure as Code



Amplify Studio

AWS 上に最小限のコーディングでフロントからバックまでのアプリケーションを作成できるビジュアルな開発環境

デザイン (Figma)

フロントエンド

ヘッドレス CMS



Amplify Libraries

フロントエンド

バックエンド連携

継続的デプロイメントを管理し、モダンな Web アプリケーションをビルド、テスト、デプロイ、そしてホスティングするための AWS サービス



Amplify Hosting

CI/CD

Web ホスティング

E2E テスト

SSR

ISR

SSG



# Amplify CLI

AWS AppSync や Amazon DynamoDB 等の各サービスを1から設定する必要がないため、各 AWS サービスの詳細なパラメータを意識することなく、汎用的な構成をカテゴリベースで構築可能

```
$ amplify add api  
? Please select from one of the below mentioned services  
> GraphQL  
    REST  
  
? Choose the default authorization type for the API  
> API key  
    Amazon Cognito User Pool  
    IAM  
    OpenID Connect  
  
? Choose a schema template:  
> Single object with fields (e.g., "Todo" with ID, name, description)  
    One-to-many relationship (e.g., "Blogs" with "Posts" and "Comments")
```

```
$ amplify add function  
? Select which capability you want to add: Lambda function  
? Choose the runtime that you want to use  
.NET Core 3.1  
Go  
Java  
> NodeJS  
Python
```

File /amplify/backend/api

- build
- resolvers
- stacks
- cli-inputs.json
- parameters.json
- schema.graphql

File /amplify/backend/function

- src
- event.json
- index.js
- package.json
- custom-policies.json



AWS AppSync



Resolver



Amazon DynamoDB

\$ amplify push



AWS Lambda



Lambda Layer



Execution Role

# Amplify CLI で生成したリソースの拡張

Amplify CLI で生成されたバックエンドリソースを AWS Cloud Development Kit (AWS CDK) を使用してカスタマイズすることが可能

```
$ amplify override api
```



```
import { AmplifyApiGraphQLResourceStackTemplate } from '@aws-amplify/cli-extensibility-helper';

export function override(resources: AmplifyApiGraphQLResourceStackTemplate){
    resources.api.GraphQLAPI.xrayEnabled = true; // AWS X-Ray の有効化

    resources.opensearch.openSearchDomain.elasticsearchClusterConfig = {
        ...resources.opensearch.OpenSearchDomain.elasticsearchClusterConfig,
        instanceCount: 6
    }; // OpenSearch のインスタンス数の変更

    resources.models["Todo"].modelDDBTable.timeToLiveSpecification = {
        attributeName: "ttl",
        enabled: true
    } // @modelディレクティブで自動生成された DynamoDB Table のTTLを有効化
}
```

認証済みおよび未認証のユーザーロールに影響があるので、他のリソースがまだ追加されていないプロジェクトの開始時にのみ、プロジェクトの開始時のみこれらのリソースを変更することが推奨

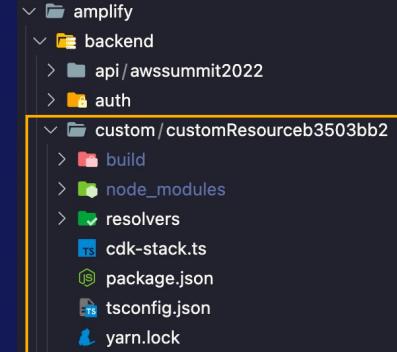
TS amplify/backend/api/<resource-name>/overrides.ts

# カスタムリソースの追加

amplify add custom によって AWS CDK または AWS CloudFormation を使用し、Amplify が作成したバックエンドに任意 AWS サービスを追加することが可能

```
$ amplify add custom  
? How do you want to define this custom resource?  
> AWS CDK  
AWS CloudFormation
```

CDK or CloudFormation のテンプレートを生成し、Amplify のカテゴリとしてデプロイすることが可能



```
const resolver = new appsync.CfnResolver(this, "getTeamMemberResolver", {  
    apiId: api.apiId,  
    fieldName: "listGroupMember",  
    typeName: "Query",  
    kind: "PIPELINE",  
    requestMappingTemplate: appsync.MappingTemplate.fromFile("req.vtl").renderTemplate(),  
    responseMappingTemplate: appsync.MappingTemplate.fromFile("res.vtl").renderTemplate(),  
    pipelineConfig: {  
        functions: [listGroupFunction.attrFunctionId, batchGetUsers.attrFunctionId]  
    }  
});
```

Amplify CLI で作成したリソースを参照しが可能

TS amplify/backend/custom/xxx/cdk-stack.ts

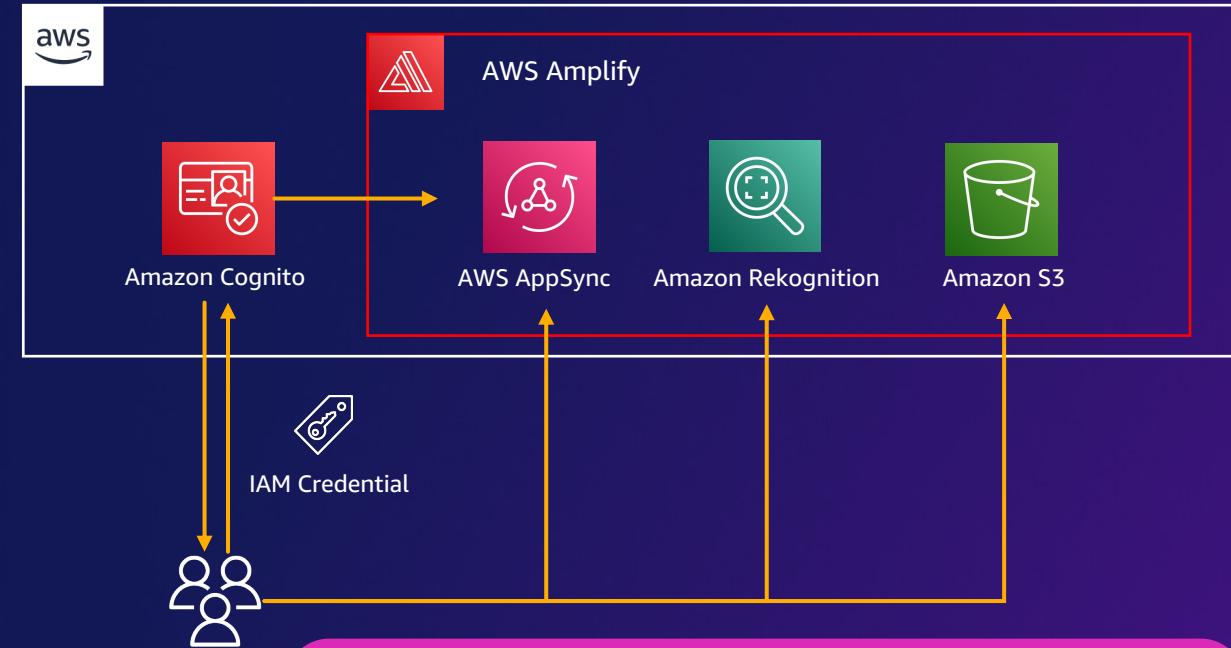
# Amplify CLI による既存リソースのインポート

Amplify CLI で作成したリソースだけでなく、既存の Amazon Cognito リソース等も Amplify プロジェクトにインポートすることが可能

```
$ amplify import auth  
? what type of auth resource do you want to import? ...  
> Cognito User Pool and Identity Pool  
Cognito User Pool only  
✓ Select the User Pool you want to import: xxxxxxxx  
✓ Select a web client to import: xxxxxxxx  
✓ Select a Native client to import: xxxxxxxx
```

```
const awsmobile = {  
  "aws_project_region": "ap-northeast-1",  
  "aws_cognito_identity_pool_id": "YOUR_IDENTITY_POOL_ID",  
  "aws_cognito_region": "YOUR_REGION",  
  "aws_user_pools_id": "YOUR_USER_POOL_ID",  
  "aws_user_pools_web_client_id": "YOUR_USER_POOL_WEB_CLIENT_ID",  
  "oauth": {}  
};
```

 aws-exports.js



指定された既存の Cognito ユーザープールまたは ID プールで認証されるようにすべての Amplify プロビジョニングされたリソース (GraphQL API、S3 バケットなど) を自動的に設定

# Amplify CLI によるローカルモックテスト

AWS CloudFormation でバックエンドリソースをプロビジョニングすることなくローカル環境でデバッグ可能

```
$ amplify mock api  
$ amplify mock function  
$ amplify mock storage
```

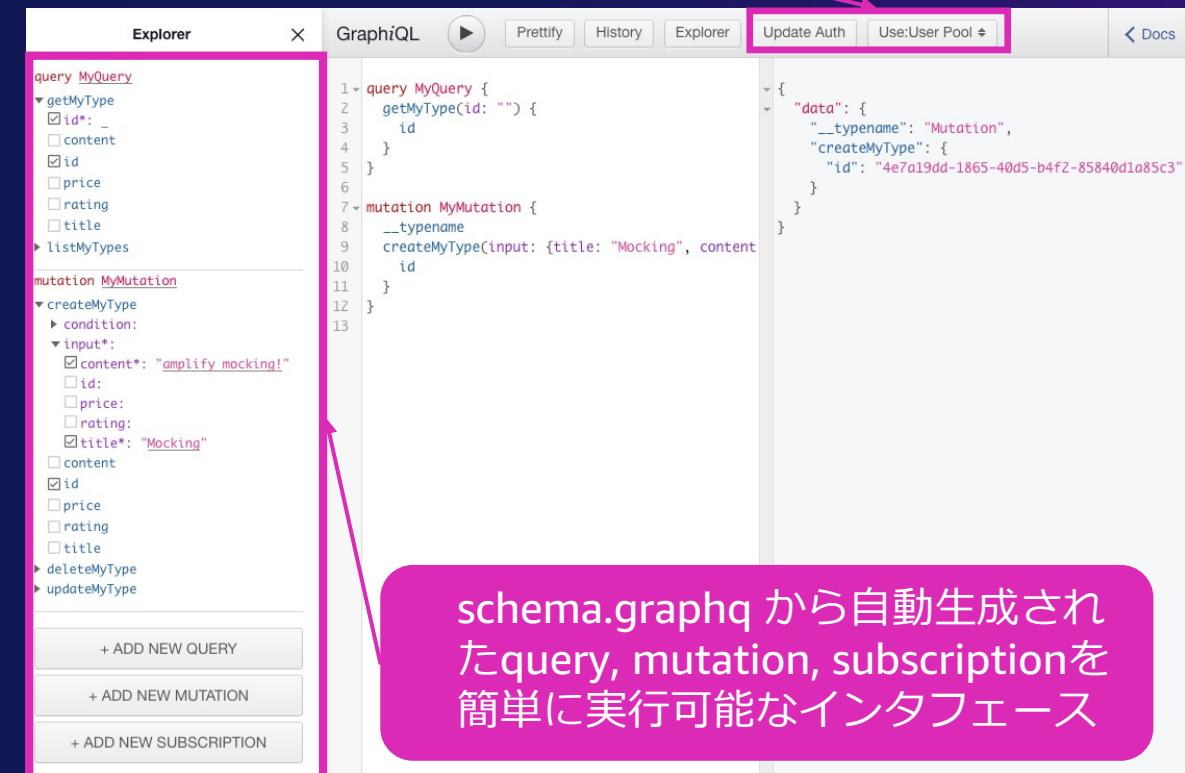
## aws-exports.js (Mock 起動時)

```
const awsmobile = {  
  "aws_appsync_graphqlEndpoint":  
    "http://192.168.10.102:20002/graphql",  
}
```

## aws-exports.js

```
const awsmobile = {  
  "aws_appsync_graphqlEndpoint":  
    "https://xx.appsync-api.ap-northeast-1.amazonaws.com/graphql ",  
}
```

Update AuthでAPI(GraphQL)の認証方法(Amazon Cognito User Pool/OIDC/API KEY/IAM)を切り替え可能



schema.graphq から自動生成されたquery, mutation, subscriptionを簡単に実行可能なインターフェース

# 他のAWSリソースへのアクセスを含めたモックテスト

Mock 時は Amplify が使用している IAM Credential が自動的に参照されるため、Lambda 関数から他の AWS リソースに対してアクセスが必要な処理に関してもローカル環境でモック可能

```
$ amplify mock api  
$ amplify mock function --event "<path to event JSON file>"
```

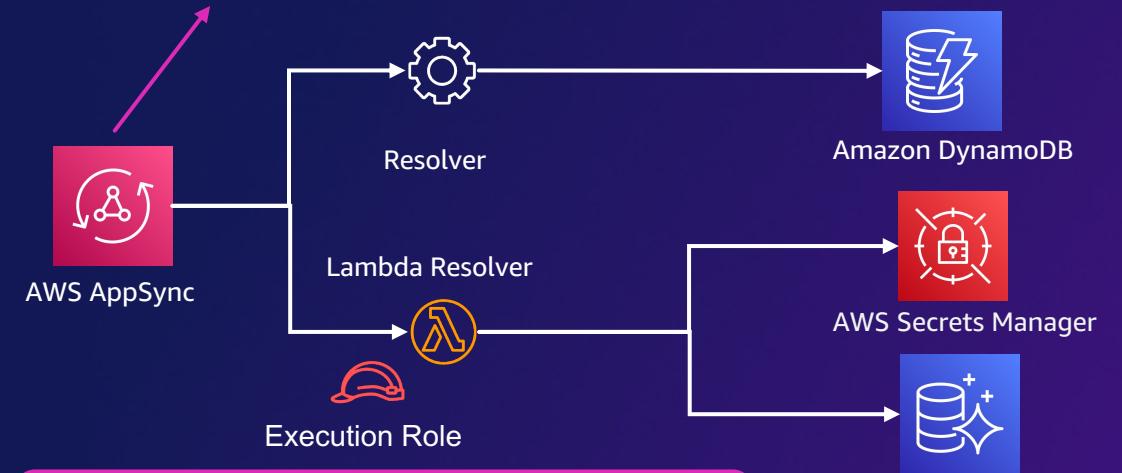


```
IS_MOCK=true  
DB_ENDPOINT=http://localhost:62224  
DB_NAME=mydb
```

モック時には自動的に .env に記載した環境変数に上書きされる

```
const connectionConfig = process.env.IS_MOCK ? {  
  host: process.env['DB_ENDPOINT'],  
  user: 'admin',  
  database: process.env['DB_NAME'],  
} : undefined;  
  
connection = mysql2.createConnection(connectionConfig);
```

```
type Todo @model {  
  id: ID!  
  name: String!  
  description: String  
  owner: User @function(name: "getUser-${env}")  
}
```



モック時には自動的に Amplify で使用している IAM Credential を参照

<https://docs.amplify.aws/cli/usage/mock/#storage-mocking-setup>



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

# Amplify Libraries



Amplify CLI

Web やモバイルアプリケーションを一般的なユースケースベースのガイド付きワークフローでバックエンドを簡単に作成、管理するツール

バックエンド

ローカル実行

Infrastructure as Code



Amplify Libraries

フロントエンド

バックエンド連携



Amplify Studio

AWS 上に最小限のコーディングでフロントからバックまでのアプリケーションを作成できるビジュアルな開発環境

デザイン (Figma)

フロントエンド

ヘッドレス CMS



Amplify Hosting

継続的デプロイメントを管理し、モダンな Web アプリケーションをビルド、テスト、デプロイ、そしてホスティングするための AWS サービス

CI/CD

Web ホスティング

E2E テスト

SSR

ISR

SSG



# Amplify Libraries

カテゴリベースの API が提供されているため、AWS SDK を使用して各サービス毎にAPIを呼び出す必要がない



JavaScript/Android/iOS/Flutter



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.



AWS AppSync



Amazon Pinpoint



Amazon Personalize



Amazon Pinpoint



Amazon API Gateway



Amazon Kinesis Data Streams



Amazon Kinesis Data Firehose



AI / ML Predictions



Amazon Polly



Amazon Rekognition



Amazon Cognito



Amazon Location Service



Amazon Translate



Amazon Transcribe



Amazon S3



AWS IoT Core



Amazon Lex

# Amplify CLI を使用せずに Amplify Libraries だけを使用する場合

既存の AWS AppSync や Amazon API Gateway 等のバックエンドリソースがある場合でも、各エンドポイントやリソース名など指定することで、Amplify Libraries を使用することが可能

TS

```
import Amplify from 'aws-amplify';

const myAppConfig = {
    // ...
    'aws_appsync_graphqlEndpoint': 'https://xxxxxxxxxxxx/graphql',
    'aws_appsync_region': 'ap-northeast-1',
    'aws_appsync_authenticationType': 'AMAZON_COGNITO_USER_POOLS',
    // ...
}

Amplify.configure(myAppConfig);
```

既存の AWS AppSync のエンドポイントや認証方法を指定することで、Amplify Libraries が適切なヘッダーを設定しリクエストする



AWS Serverless Application Model (AWS SAM)



AWS AppSync



Lambda Resolver



Amazon DynamoDB

TS

```
import { API, graphqlOperation } from 'aws-amplify';
import { createTodo, updateTodo, deleteTodo } from './graphql/mutations';

const todo = { name: "My first todo", description: "Hello world!" };

await API.graphql(graphqlOperation(createTodo, {input: todo}));
await API.graphql(graphqlOperation(deleteTodo, { input: { id: todoId } }));
```

Amplify Libraries 側の実装は既存のリソースか CLI で作成したリソースかに関わらず、同様に実装可能

<https://docs.amplify.aws/lib/graphqlapi/create-or-re-use-existing-backend/q/platform/js/>



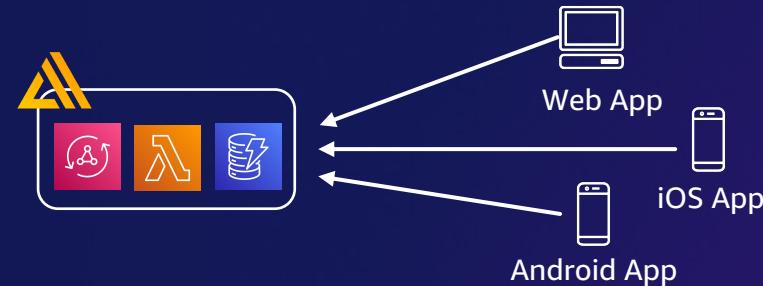
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

# Amplify Env - Multiple Frontends

複数のフロントエンド・ユーザーで同じバックエンドリソースを共有する場合

```
$ amplify pull // 既存のバックエンド環境のImport  
? which app are you working on?  
> project1  
project2  
? Do you plan on modifying this backend?: No
```

```
{  
  "dev": {  
    "awscloudformation": {  
      "AuthRoleName": "xxxx",  
      "UnauthRoleArn": "arn:aws:iam::xxxx:",  
      "AuthRoleArn": "arn:aws:iam::yyyy:",  
      "Region": "ap-northeast-1",  
      "DeploymentBucket": "xxxxxxxxxxxxxx",  
      "UnauthRoleName": "arn:aws:iam::yyyy:",  
      "StackName": "amplify-project",  
      "StackId": "arn:aws:cloudformation:ap-northeast-1:xxxxxx:stack/amplify-project/xxxxxx",  
      "AmplifyAppId": "xxxxxx"  
    }  
  }  
}
```



- team-provider-info.json と aws-exports.js のみが生成される
- クライアントからはバックエンドの設定が見えない
- amplify env コマンドを使用して、Amplify Libraries から参照するバックエンドを切り替え可能

```
const awsmobile = {  
  "aws_project_region": "ap-northeast-1",  
  "aws_appsync_graphqlEndpoint": "https://xxxx/graphql",  
  "aws_appsync_region": "ap-northeast-1",  
  "aws_appsync_authenticationType": "API_KEY",  
  "aws_appsync_apiKey": "xxxx"  
};  
  
export default awsmobile;
```

```
$ amplify env checkout dev
```



# Amplify Libraries の設定を上書きする場合

Amplify CLI によって自動生成された Amplify Libraries の設定を上書きして、トークンの管理方法などをカスタマイズすることが可能

```
import awsmobile from './aws-exports';

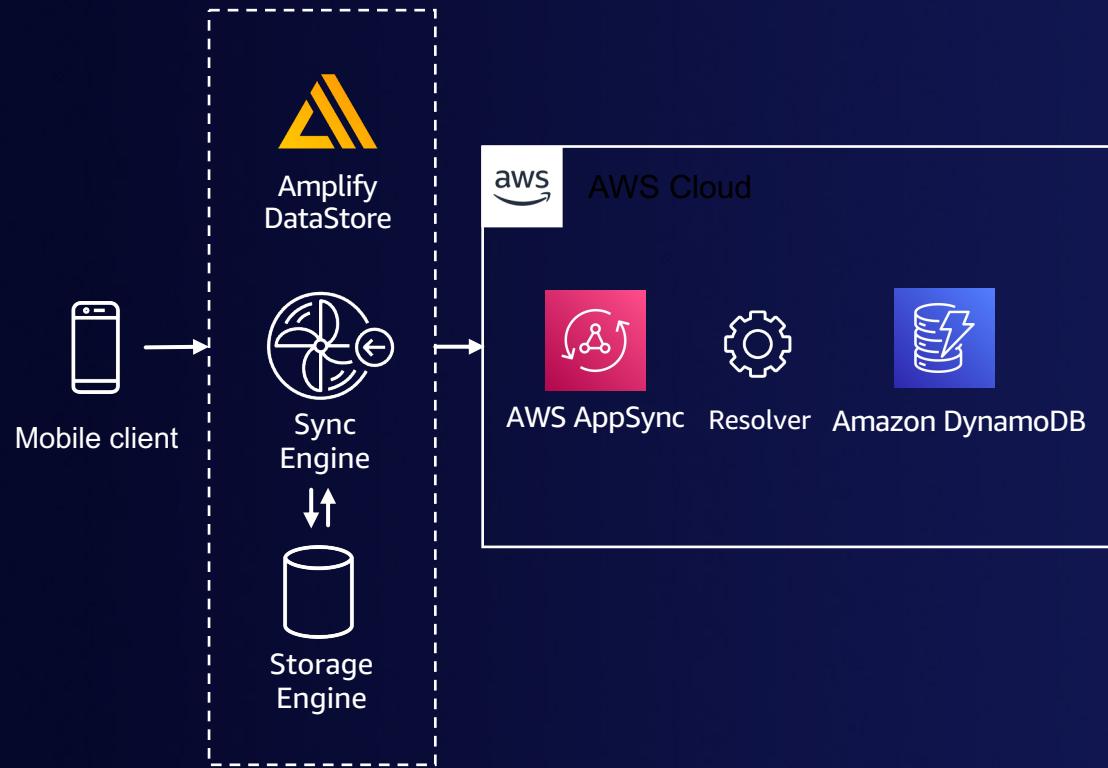
Amplify.configure({
  ...awsmobile,
  Auth: {
    // トークンの保存場所をデフォルトの localstorage ではなく cookie にする場合
    cookieStorage: {
      domain: '.yourdomain.com',
      path: '/',
      expires: 365,
      sameSite: "strict",
      secure: true
    },
    // Session Storage 等の任意のストレージオブジェクトを指定することも可能
    storage: window.sessionStorage
  },
  geo: {...},
  Analytics: {...},
  Storage: {...},
  API: {...},
  ...
});
```

Amplify CLI によって自動生成されたクライアント側の設定を上書き

既存の AWS AppSync と同様に他のリソースに関しても、Amplify CLI を使用せずに、Amplify Libraries から参照することが可能

# Amplify DataStore

オフラインとオンラインのシナリオで追加のコードを書くことなく共有・分散データを活用するための API



## リアルタイム処理

ローカルの Storage Engine 及びリモートの AWS AppSync に対して実行される Query や Mutation に対してリアルタイムに各モデルの変更をサブスクライブ

## オフライン処理

ネットワーク接続がオフライン時はローカルのデータストアを使用し、オンライン時には AWS AppSync と自動的に同期

## バージョン管理

ローカルとサーバーで同じアイテムに対してコンフリクトが発生した場合、自動的に競合を検知してマージすることが可能

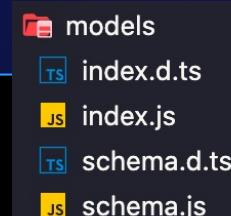
# Amplify DataStore

スキーマさえ定義してしまえば GraphQL を意識することなくデータ操作が可能

```
type Order @model {  
  orderId: ID! @primaryKey  
  customerId: ID!  
  status: String!  
  amount: Int!  
}
```

\$ amplify codegen models

```
export declare class Order {  
  readonly orderId: string;  
  readonly cusomerId: string;  
  readonly status: string;  
  readonly createdAt?: string;  
  readonly updatedAt?: string;  
  constructor(init: ModelInit<Order, OrderMetaData>);  
  static copyof(  
    source: Order,  
    mutator: (draft: MutableModel<Order, OrderMetaData>  
      => MutableModel<Order, OrderMetaData> | void): Order;  
  )  
}
```



## Query

```
const orders = await DataStore.query(Order, o =>  
  o.status("eq", "OPEN"));
```

## Delete

```
await DataStore.delete(Order, o => o.status("eq", "OPEN"));
```

## Real time

```
const subscription = await DataStore.observe(Order,  
  "id") .subscribe(msg => console.log(msg))
```

# Amplify Studio



Amplify CLI

Web やモバイルアプリケーションを一般的なユースケースベースのガイド付きワークフローでバックエンドを簡単に作成、管理するツール



Amplify Libraries

Web やモバイルアプリケーションと AWS を統合するためのユースケース中心のライブラリ

バックエンド

ローカル実行

Infrastructure as Code



Amplify Studio

AWS 上に最小限のコーディングでフロントからバックまでのアプリケーションを作成できるビジュアルな開発環境

デザイン (Figma)

フロントエンド

ヘッドレス CMS



Amplify Hosting

継続的デプロイメントを管理し、モダンな Web アプリケーションをビルド、テスト、デプロイ、そしてホスティングするための AWS サービス

CI/CD

Web ホスティング

E2E テスト

SSR

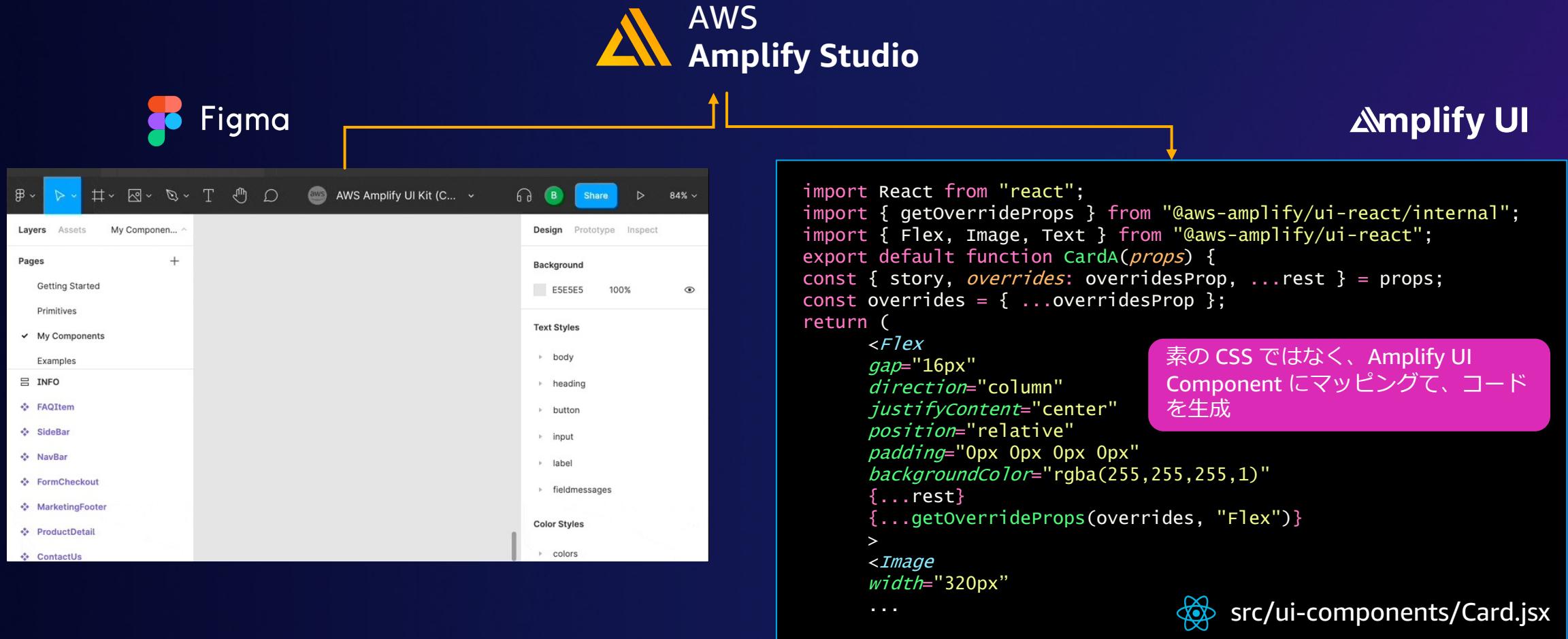
ISR

SSG



# Amplify Studio

Figma で作成されたデザインをAmplify UI Component (React コンポーネント)に変換可能



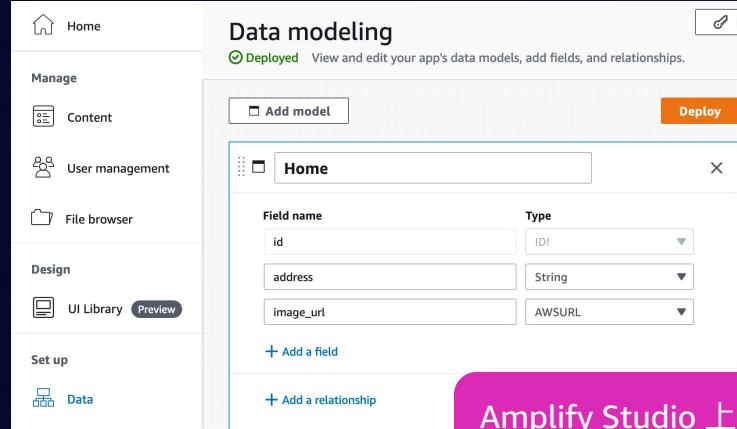
<https://aws.amazon.com/blogs/mobile/aws-amplify-studio-figma-to-fullstack-react-app-with-minimal-programming/>

# Amplify Studio で DataStore のモデルと UI をマッピング

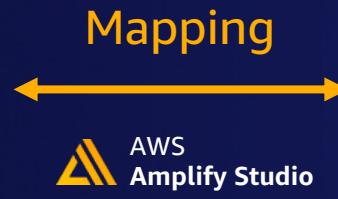
Amplify DataStore を使用して定義したデータモデルとUI Componentのアトリビュートをマッピング可能

```
type Home @model {
  id: ID! @primaryKey
  address: String!
  image_url: AWSURL!
}
```

or



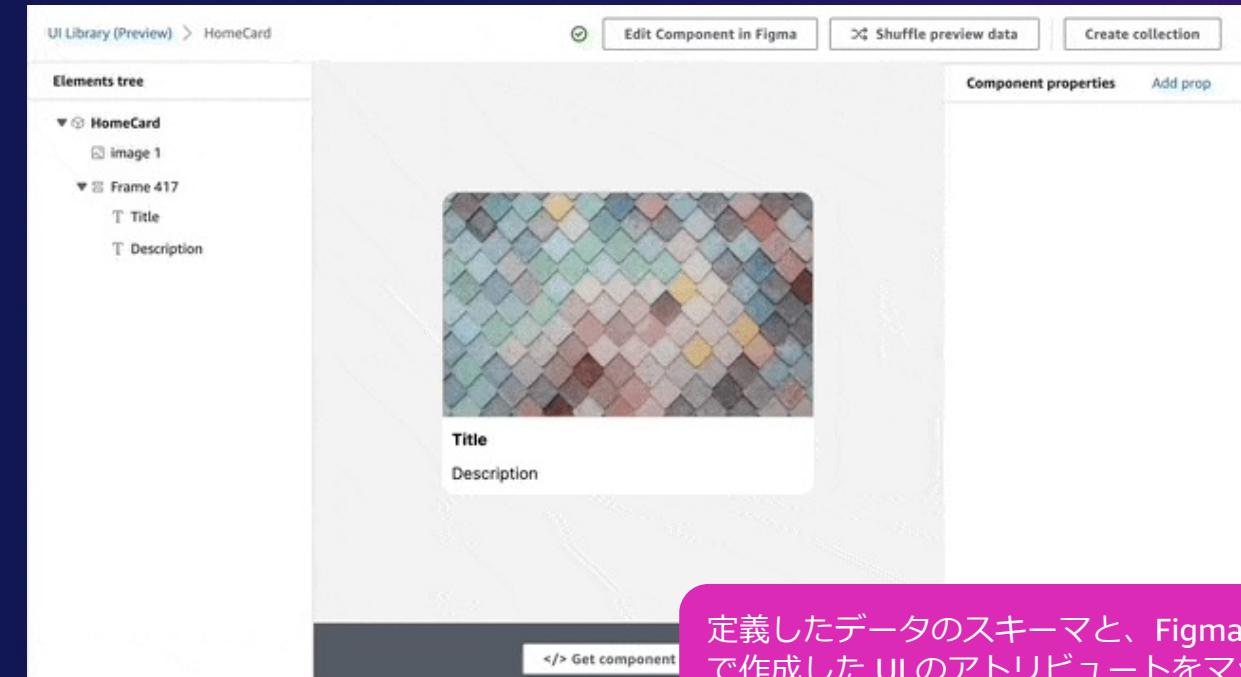
Amplify Studio 上の GUI でスキーマ  
を定義することも



AWS  
Amplify Studio

Figma

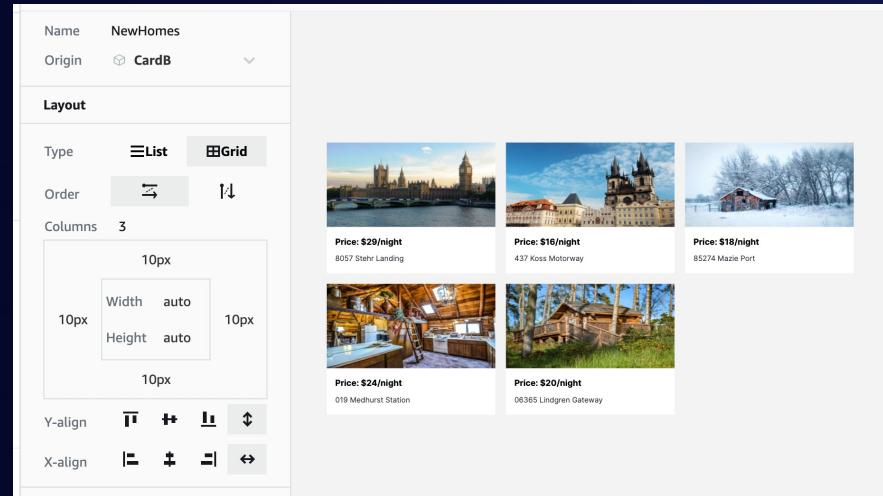
▼ Sync



定義したデータのスキーマと、Figma  
で作成した UI のアトリビュートをマッ  
ピングすることが可能

# Amplify Studio で DataStore のモデルと UI をマッピング

アイテム一覧を取得する DataStore API を含めた Collection Component を自動生成することも可能



```
import React from "react";
import { Post } from "../models";
import {
  getOverrideProps,
  useDataStoreBinding,
} from "@aws-amplify/ui-react/internal";
import Card from "./Card";
import { Collection } from "@aws-amplify/ui-react";
export default function VListCollection(props) {
  const { post, items: itemsProp, overrides: overridesProp, ...rest } = props;
  const overrides = { ...overridesProp };
  const items =
    itemsProp !== undefined
      ? itemsProp
      : useDataStoreBinding({
        type: "collection",
        model: Post,
      }).items;
  return (
    <Collection
      type="list"
      direction="column"
      justifyContent="stretch"
      items={items || []}
      {...rest}
      {...getOverrideProps(overrides, "Collection")}
    >
      {(item, index) => (
        <Card
          post={item}
          key={item.id}
          {...getOverrideProps(overrides, "Collection.VList[0]")}
        ></Card>
      )}
    
```

Amplify DataStore を使ってマッピングしたデータを取得する部分も自動生成

取得したデータを作成したコンポーネントを使って一覧表示する部分も自動生成



src/ui-components/CardCollection.jsx



# Amplify UI

ログインやファイル操作等のロジックを含んだ UI コンポーネントを提供

```
import { Amplify } from 'aws-amplify';
import { Authenticator } from '@aws-amplify/ui-react';
import '@aws-amplify/ui-react/styles.css';

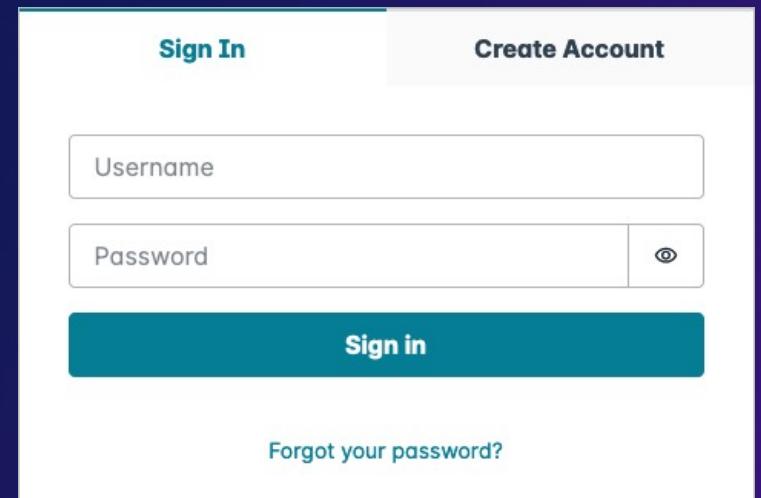
import awsExports from './aws-exports';

Amplify.configure(awsExports);

export default function App() {
  return (
    <Authenticator>
      {({ signOut, user }) => (
        <main>
          <h1>Hello {user.username}</h1>
          <button onClick={signOut}>Sign out</button>
        </main>
      )}
    </Authenticator>
  );
}
```



Amplify UI



React/Angular/Vue/Flutter に対応

Sign In、Sign Up、パスワードリセットなどの Amplify Libraries によるロジックを含んだ UI コンポーネントも利用可能



# Amplify Hosting



Amplify CLI

Web やモバイルアプリケーションを一般的なユースケースベースのガイド付きワークフローでバックエンドを簡単に作成、管理するツール



Amplify Libraries

Web やモバイルアプリケーションと AWS を統合するためのユースケース中心のライブラリ

バックエンド

ローカル実行

Infrastructure as Code



Amplify Studio

AWS 上に最小限のコーディングでフロントからバックまでのアプリケーションを作成できるビジュアルな開発環境

デザイン (Figma)

フロントエンド

ヘッドレス CMS



Amplify Hosting

継続的デプロイメントを管理し、モダンな Web アプリケーションをビルト、テスト、デプロイ、そしてホスティングするための AWS サービス

CI/CD

Web ホスティング

E2E テスト

SSR

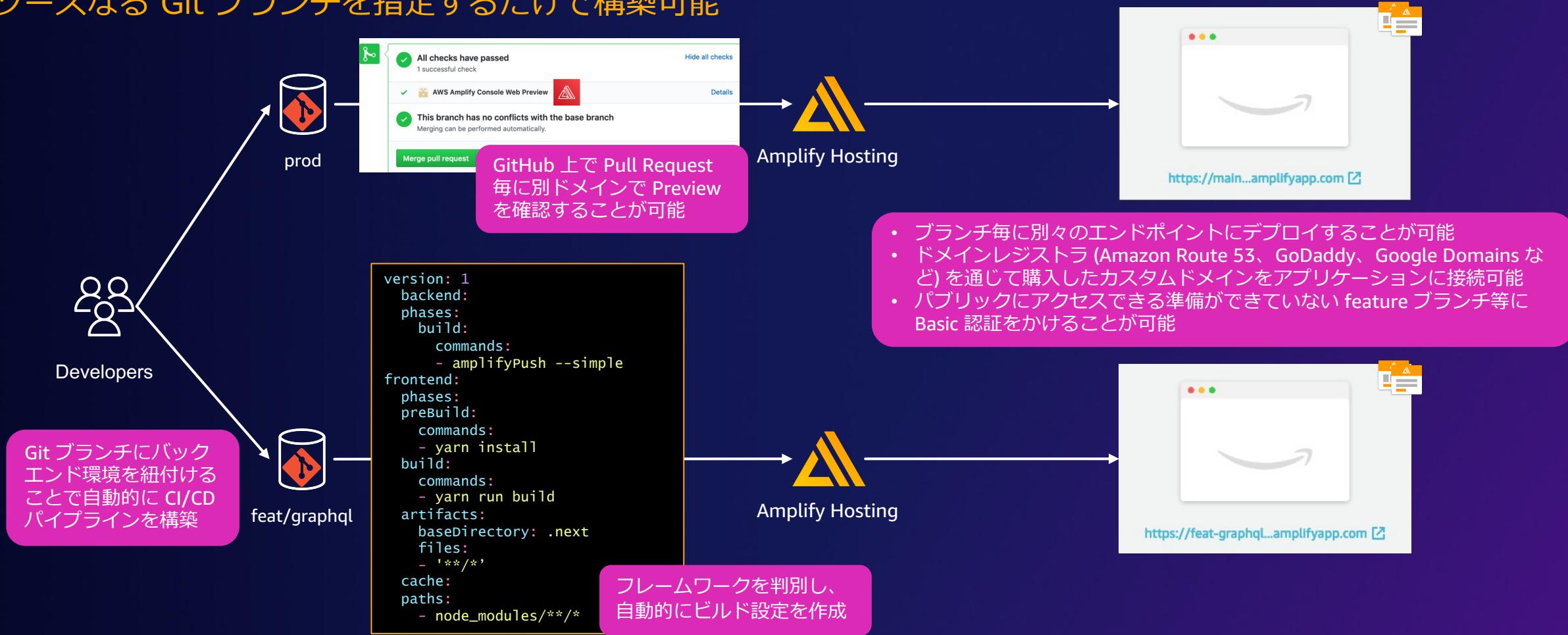
ISR

SSG



# Amplify Hosting

フロントエンドの SPA アプリケーションのホスティング、及びバックエンドリソースの CI/CD パイプラインをソースなる Git ブランチを指定するだけで構築可能



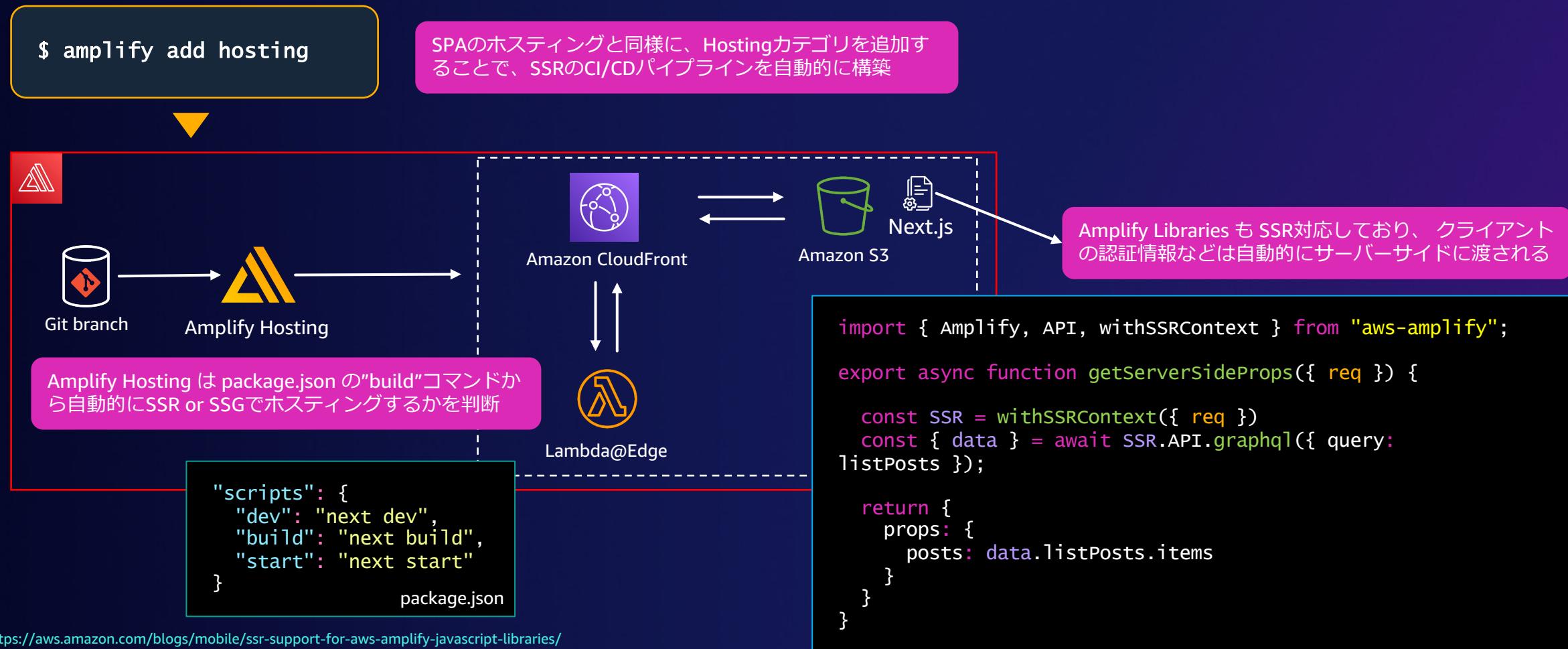
<https://aws.amazon.com/jp/amplify/hosting/>



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

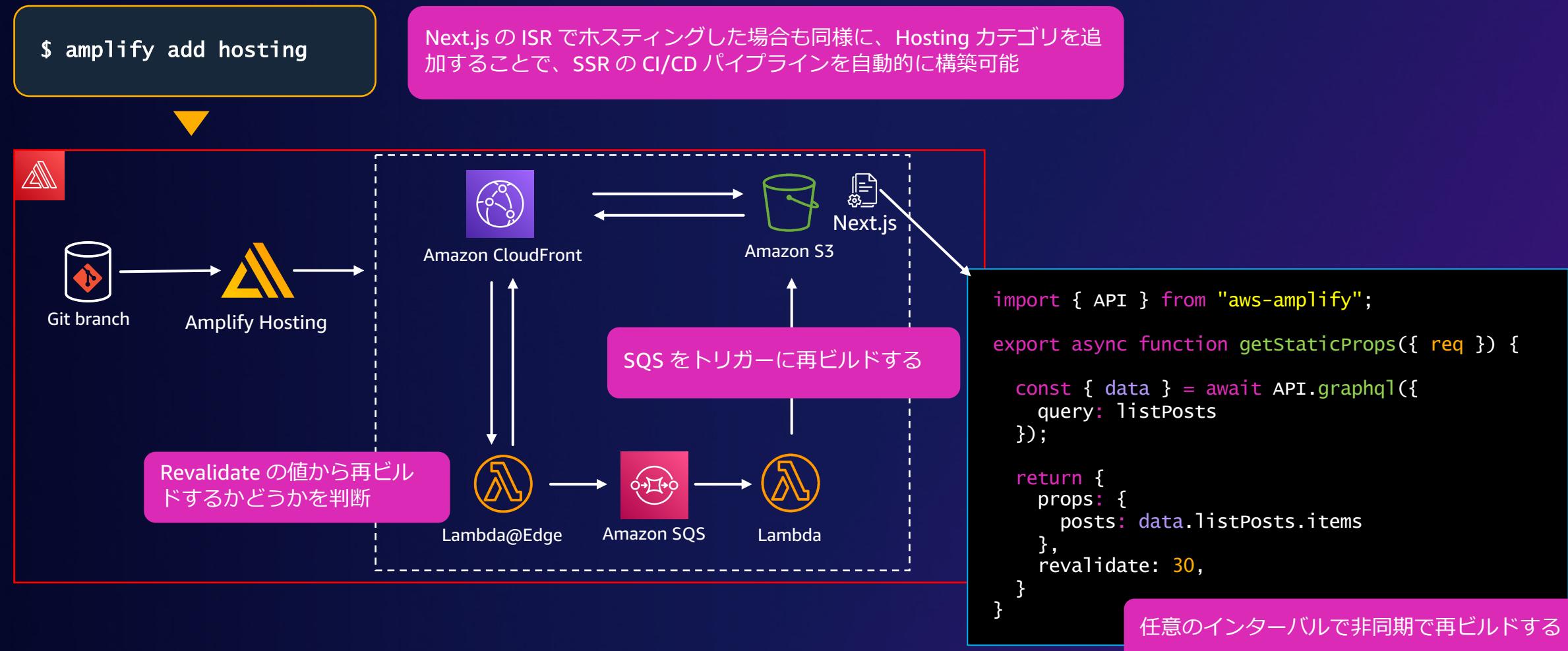
# Amplify Hosting - Server Side Rendering (SSR)

Next.js で作成されたアプリケーションの SSR に Amplify Hosting を使用することで、CI/CD パイプライン・SSR によるホスティングまで Amplify だけで完結することが可能



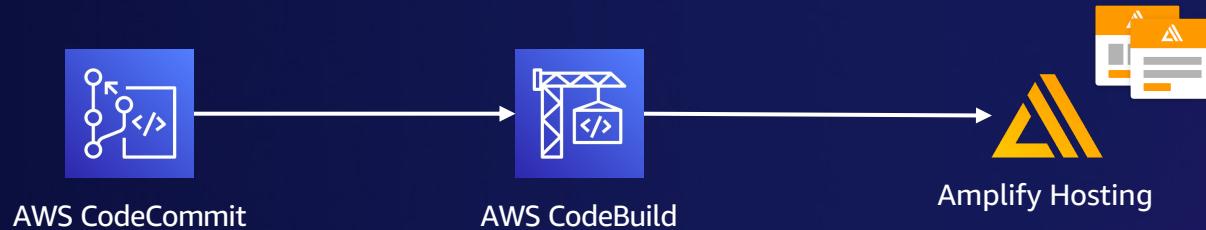
# Amplify Hosting – Incremental Static Regeneration (ISR)

Next.js で実装した ISR アプリケーションを Amplify Hosting によってサーバーレス構成でホスティング可能



# Amplify Hosting

Amplify Libraries、Amplify CLI の使用有無に関係なく、マネージドな SPA のホスティング用途として Amplify Hosting のみを使用することも可能



CI/CD パイプラインは独自に構築して、ホスティングの用途として Amplify Hosting を選択することも可能

```
version: 0.2
phases:
  install:
    runtime-versions:
      nodejs: latest
    commands:
      - yarn add global @aws-amplify/cli
      - yarn install
      - amplifyPush --simple
  build:
    commands:
      - yarn run build
      - amplify publish -c --yes
```

buildspec.yml

Amplify Hosting が提供する Helper Script によって aws-exports.js を動的に生成



# GraphQL Transformer v2 を使用 したスキーマ設計



# GraphQL Transformer v2 による DynamoDB のテーブル定義

より直感的に DynamoDB のインデックス及び、モデル間のリレーションを定義することが可能に

```
type Order @model {  
    orderId: ID! @primaryKey  
    customerId: ID! @index(name: "customerByStatusByDate", sortKeyFields: ["status", "date"])  
    status: String!  
    amount: Int!  
    date: String!  
}
```



AWS AppSync

- Query.getOrder.req.vtl
- Query.listOrdesr.req.vtl
- Mutation.createOrdes.req.vtl
- Mutation.updateOrdes.req.vtl

Primary Key	Attributes			
	customerId	status	amount	data
orderId(PK) xxx	aaa	OPEN	2	2022-5-12

```
export declare class Order {  
    readonly orderId: string;  
    readonly cusomerId: string;  
    readonly status: string;  
    readonly createdAt?: string;  
    readonly updatedAt?: string;  
    constructor(init: ModelInit<Order, OrderMeta>, options?: ModelOptions<Order, OrderMeta>);  
    static copyOf(source: Order, mutator: (draft: MutableModel<Order, OrderMeta>, options: ModelOptions<Order, OrderMeta>) => MutableModel<Order, OrderMeta> | void): Order;  
}
```

```
graphql  
  mutations.ts  
  queries.ts  
  subscriptions.ts  
models  
  index.d.ts  
  index.js  
  schema.d.ts  
  schema.js  
  API.ts
```



# GraphQL Transformer v2 - @index、@primaryKey -

@index、@primaryKey を使用することで DynamoDB テーブルを直感的に定義可能

```
type Order @model {  
    orderId: ID! @primaryKey @index(name: "byOrderByAmount", sortKeyFields: ["amount"])  
    customerId: ID! @index(name: "byCustomerByStatusByDate", sortKeyFields: ["status", "date"])  
    status: String!  
    amount: Int!  
    date: String!  
}
```

Primary Key (Partition Key +  
Sort key) の指定



@primaryKey を該当フィールドに付与し、ソートキーを指定する  
場合は sortKeyFields に対象とアトリビュートを指定

Global Secondary Index の指定



@index を @primaryKey を付与していないフィールドに付与

Local Secondary Index の指定



@index を @primaryKey を付与しているフィールドに付与

# @index Directiveを使ったGSIの定義

規模が大きくなった場合、DynamoDB の GSI を活用してパフォーマンス・コストを改善するパターン

```
type Todo @model {  
  id: ID! @primaryKey  
  name: String!  
  description: String  
}
```



```
const fetchMyTodos = async (name: string) => {  
  const listTodosVariables: ListTodosQueryVariables = {  
    filter: { name: { eq: name } }  
  }  
  const todos = await API.graphql({  
    query: listTodos,  
    variables: listTodosVariables  
  })  
}
```



複数アイテムを取得するための、listXXXXXというコードが自動生成される

```
{  
  "operation" : "Scan",  
  "filter" : {  
    "expression" : "name =:name",  
    "expressionValues" : {  
      ":name" : $util.dynamodb.toDynamoDBJson($context.args)  
    },  
  },  
}
```



Scanで全件取得してから、Filterをかけることになる



# @index Directiveを使ったGSIの定義

規模が大きくなつた場合、DynamoDB の GSI を活用してパフォーマンス・コストを改善するパターン

```
type Todo @model {  
  id: ID! @primaryKey  
  name: String!  
  description: String  
}
```

```
const fetchMyTodos = async (name: string) => {  
  const listTodosVariables: ListTodosQueryVariables = {  
    filter: { name: { eq: name } }  
  }  
  const todos = await API.graphql({  
    query: listTodos,  
    variables: listTodosVariables  
  })  
}
```

複数アイテムを取得するための、listXXXXXというコードが自動生成される

```
type Todo @model {  
  id: ID! @primaryKey  
  name: String! @index(name: "todosByName", queryField: "todosByName")  
  description: String  
}
```

NameにGSIを定義し、対応するQueryを自動生成

```
const fetchMyTodos = async (name: string) => {  
  const todosByNameVariables: TodosByNameQueryVariables = {  
    name: name  
  }  
  const todos = await API.graphql({  
    query: todosByName,  
    variables: todosByNameVariables  
  })  
}
```

```
{  
  "operation" : "Scan",  
  "filter" : {  
    "expression" : "name =:name",  
    "expressionValues" : {  
      ":name" : $util.dynamodb.toDynamoDBJson($context.argu...)  
    },  
  }  
}
```

Scanで全件取得してから、Filterをかけることになる

```
{  
  "operation" : "Query",  
  "query" : {  
    "expression" : " todosByName =:todosByName ",  
    "expressionValues" : {  
      ":todosByName" : $util.dynamodb.toDynamoDBJson($context.argu...)  
    },  
  }  
}
```

Queryを使って対象のItemだけを取得する

# GraphQL Transformer v2 - モデル間のリレーション

@hasOne、@hasMany、@manyToMany等を使用して直感的にリレーションモデルを構築することが可能

```
type Project @model {  
  id: ID!  
  name: String  
  team: Team @hasOne  
}  
  
type Team @model {  
  id: ID!  
  name: String!  
}
```

1 : 1

```
type Post @model {  
  id: ID!  
  title: String!  
  comments: [Comment] @hasMany  
}  
  
type Comment @model {  
  id: ID!  
  content: String!  
}
```

1 : N

```
type Post @model {  
  id: ID!  
  title: String!  
  content: String  
  tags: [Tag] @manyToMany(relationName: "PostTags")  
}  
  
type Tag @model {  
  id: ID!  
  label: String!  
  posts: [Post] @manyToMany(relationName: "PostTags")  
}
```

N : M

```
query QueryAllPosts {  
  listPosts() {  
    posts {  
      items {  
        title  
        content  
        updatedAt  
        tags {  
          items {  
            label  
            posts {  
              items {  
                title  
              }  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

VTL を記述することなく、ネストしたモデルを定義可能

# GraphQL Transformer v2 - モデル間のリレーションの注意点

@hasOne、@hasMany等を使用してリレーションモデルを簡単に構築することができるが、N+1問題が発生

```
type Group @model {  
  id: ID!  
  name: String!  
  users: [User] @hasMany  
}  
  
type User @model {  
  id: ID!  
  username: String!  
  email: String!  
}
```

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

Group Table

Primary Key	Attributes	
Id(PK)		
xxx	name	
	aaa	

User Table

Primary Key	Attributes		
Id(PK)			
yyy	username	groupUserId(GSI)	
	bbb	xxx	

GSI

```
query ListGroupMember{  
  listGroups {
```

name

items {

users {

items {

user {

email

}

}

}

}

}

}

Groupテーブルから一覧取得

(取得したGroup一覧の各アイテムに対して、UserテーブルからgroupUserId(GSI)を使ってQuery) x N回



# GraphQL Transformer v2 - モデル間のリレーションの注意点

BatchGetItem を実行する Pipeline Resolver をカスタムカテゴリを使って実装することも有効な選択肢だが、開発速度を重視する場合や、データの規模によっては自動生成したものを使用した方がメリットがある

```
type Group {  
  id: ID!  
  name: String!  
  users: [User]  
}  
  
type User {  
  id: ID!  
  username: String!  
  email: String!  
}  
  
type GroupUsers @model {  
  pk: ID! @primaryKey(sortKeyFields: ["sk"])  
  sk: ID!  
  userName: String  
  groupName: String  
}  
  
type Query {  
  listGroupMember: [Group]  
}
```

\$ amplify add custom  
? How do you want to define  
this custom resource?

› AWS CDK  
AWS CloudFormation

Custom カテゴリを使用して、  
Pipeline Resolver を作成

```
query ListGroupMember{  
  listGroups {  
    name  
    items {  
      users {  
        items {  
          user {  
            email  
          }  
        }  
      }  
    }  
  }  
}
```

Groupテーブルから一覧取得

Group 1

Group 2

Group N

BatchGetItem APIにより、  
まとめてUser情報を検索

# GraphQL Transformer v2 - @auth -

@auth ディレクティブによって サインインしたユーザー・グループごとにアイテム・アトリビュート毎に認可ルールを Deny-by-Default の原則で定義することが可能

```
type Employee @model @auth(rules: [
  { allow: private, operations: [read]}, 
  { allow: owner }
]) {
  name: String
  email: String
  phone: String @auth(rules: [{ allow: owner }]) {}
}
```

```
$ amplify status api -acm Employee
```

private	Create	Read	Update	Delete
name	✗	✓	✗	✗
email	✗	✓	✗	✗
phone	✗	✗	✗	✗

owner	Create	Read	Update	Delete
name	✓	✓	✓	✓
email	✓	✓	✓	✓
phone	✓	✓	✓	✓

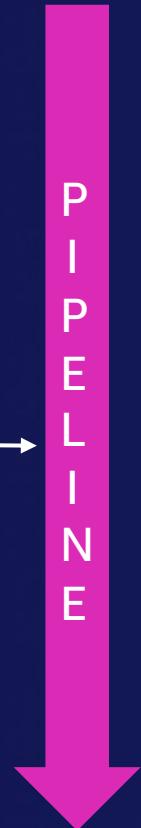
各モデルのアクセス権限を表形式で出力することが可能



# GraphQL Transformer v2 によるコード自動生成

TypeScript / JavaScript / Java / Swift のモデル・クエリ、及び Pipeline Resolver の自動生成

```
type Order @model {  
    orderID: ID! @primaryKey  
    customerId: ID!  
    status: String!  
    amount: Int!  
    date: String!  
}
```



- Query.getOrder.init.1.req.vtl
- Query.getOrder.preAuth.1.req.vtl
- Query.getOrder.auth.1.req.vtl
- Query.getOrder.postAuth.1.req.vtl
- Query.getOrder.preDataLoad.1.req.vtl
- Query.getOrder.req.vtl**
- Query.getOrder.postDataLoad.1.req.vtl
- Query.getOrder.finish.1.req.vtl



Amazon DynamoDB

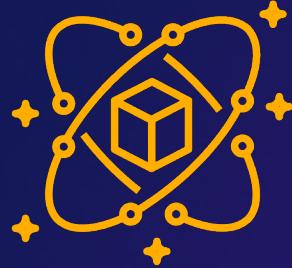
Pipeline Resolverが自動的に生成されるため、特定のロジックの実行後に任意の処理を挟むことが容易に

# まとめ



## モダンなアプリケーション開発のためには…

- フロントエンドとデザイナー、あるいはフロントエンドとバックエンドがシームレスに連携するためのツール
- SSR、ISRなどのモダンなホスティング手法をフロントエンドエンジニアの労力を最小限にマネージドにデプロイできるツール
- 220以上の多種多様なAWSサービスとシームレスに連携するためのツール



## Amplify Amplify

- Amplify Studio**による Figma デザインファイルを React Componentへの変換、及び **Amplify Data Store** で定義したデータモデルとUIのマッピング
- Amplify Hosting** によるマネージドな CI/CD パイプラインにより、ISR、SSR、SSG 等の各種ホスティング方法を容易に実現
- Amplify Libraries**、**Amplify CLI** により各サービスレベルではなく、機能ベースでサービスを組み合わせることが可能

# まとめ

## AWS Amplify

によって、デザイナー、フロントエンジニア、バックエンドエンジニア問わずに、  
アプリケーション開発を効率化することが可能です

個人開発、スタートアップ、エンタープライズまで開発チームの規模を問わずに拡張性の高いアプリケーションを構築可能

ミッションクリティカル、大規模なバックエンドのスケールが必要なアプリケーション等、サービスの規模を問わずスケーラブルなアプリケーションを構築可能



AWS Amplify



AWS Device Farm



Amazon Location  
Service



AWS AppSync

アプリケーション・開発の規模を問わずに、  
様々なワークLOADの開発を Amplify を使って効率化

# Thank you!

鈴木貴博 (Takahiro Suzuki)

[tkasuz@amazon.co.jp](mailto:tkasuz@amazon.co.jp)

 [kopkunka55](https://github.com/kopkunka55)



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.