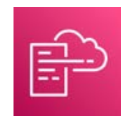
サーバーレスアプリケーションの開発とデプロイ

IaCを含む、スマートな環境構築の旅

Sebastian Rettig

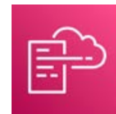
2021/12/18



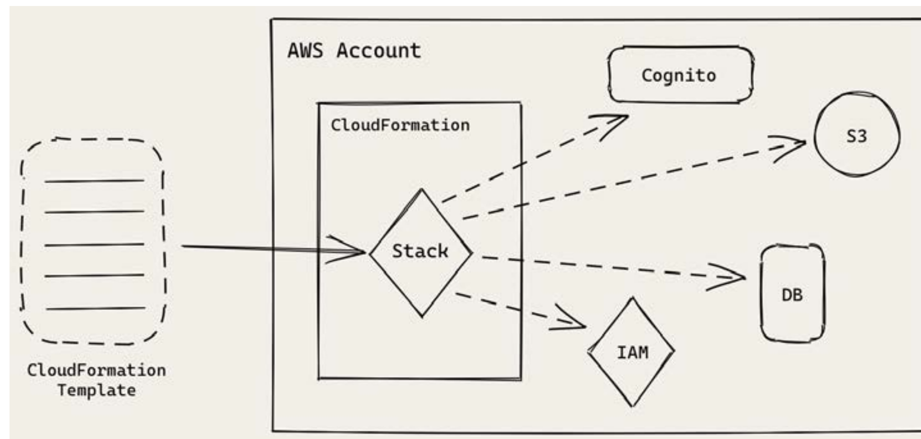
Outline

- Infrastructure as Code (IaC) について
- AWS SAMについて
- AWS Amplifyについて
  - AWS CDKの交換について
- AWS CDKについて
  - CDK Pipelineについて
- Serverless Stack (SST) について
- トレンド

Infrastructure as Code (IaC) について



- アプリ・システムのインフラをテンプレートやコードで定義する
- AWS CloudFormation
  - リソース自動作成
  - リソース簡単に削除
  - リソース更新の場合、変更の一覧作成する
  - Template
    - YAMLやJSONフォーマット
    - Nested Templates作成可能
      - VPC, DB, API, 等
  - Stack
    - 関連するリソースを1つのユニットとして管理します
  - Change Set
    - 変更の一覧
    - リソースにより変更方法が異なる
      - 例：RDS DBの名前を変更すると、新インスタンス構築、古インスタンス削除
- AWS CLI対応
- CloudFormation Designer UI



Source: ServerlessStack - What Is Infrastructure as Code

Infrastructure as Code (IaC) について

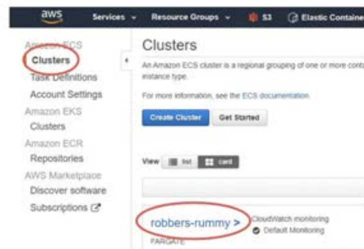
Pro

- インフラ組織のエラーチェック可能
- インフラ簡単にコマンドで自動再現できる
- 複数環境自動作成できる
  - 開発環境
  - テスト環境
  - 本番環境
- インフラ手動作成より、エラーが発生しにくい
- リソース構築の時間削減できる

Create a new revision of the ECS Task Definition

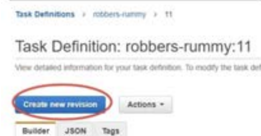
Open the ECS section of the AWS Console

On the Amazon ECS page click **Clusters** and select the cluster

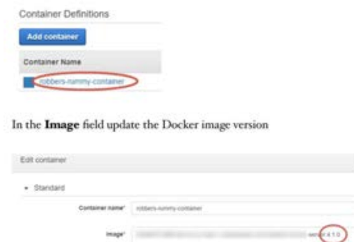


On the **Services** tab click the **Task Definition**

On the Task Definition page click the **Create new revision** button



Scroll down to the **Container Definitions** section select the container definition



Click the **Update** button at the bottom of the Container page

Click the **Create** button at the bottom of the Task Definition page

A new task definition revision has been created

Source: [Pinter - Deploy a new version of a task in an ECS Fargate cluster]

Infrastructure as Code (IaC) について

Contra

- テンプレートでは、アプリ・システムに必要なすべてのリソースを定義する必要
- テンプレートが長くなる可能性が高い
- テンプレート作成の勉強が時間かかる
- リソースを定義する方法を理解するため、ドキュメントの確認が時間かかる

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: sample script to demo dynamodb streams functionality

Resources:

  DynamoDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      TableName: !Ref APIName
      ProvisionedThroughput:
        WriteCapacityUnits: 5
        ReadCapacityUnits: 5
      AttributeDefinitions:
        - AttributeName: "pk1"
          AttributeType: "S"
        - AttributeName: "sk1"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "pk1"
          KeyType: "HASH"
        - AttributeName: "sk1"
          KeyType: "RANGE"

  RoleAppSyncCloudWatch:
    Type: AWS::IAM::Role
    Properties:
      ManagedPolicyArns:
        - "arn:aws:iam::aws:policy/service-role/AWSAppSyncPushToCloudWatchLogs"
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - sts:AssumeRole
            Principal:
              Service:
                - appsync.amazonaws.com

  RoleAppSyncDynamoDB:
    Type: AWS::IAM::Role
    Properties:
      ManagedPolicyArns:
        - !Ref PolicyDynamoDB
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - sts:AssumeRole
            Principal:
              Service:
                - appsync.amazonaws.com

  PolicyDynamoDB:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      Path: /service-role/
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - dynamodb:Query
              - dynamodb:GetItem
              - dynamodb:PutItem
              - dynamodb:DeleteItem
            Resource:
              - !Sub arn:aws:dynamodb:${AWS::Region}:${AWS::AccountId}:table/$

  GraphQLApi:
    Type: AWS::AppSync::GraphQLApi
    Properties:
      Name: !Ref APIName
      AuthenticationType: API_KEY
      LogConfig:
        CloudWatchLogsRoleArn: !GetAtt RoleAppSyncCloudWatch.Arn
        ExcludeVerboseContent: FALSE
        FieldLogLevel: ALL
```

```yaml
GraphQLApiSchema:
  Type: AWS::AppSync::GraphQLSchema
  Properties:
    ApiId: !GetAtt GraphQLApi.ApiId
    Definition: |
      schema {
        query: Query
        mutation: Mutation
      }

      type Data {
        data: [AWSJSON]
        pk1: String
        sk1: String
      }

      type DataCollection {
        items: [Data]
        nextToken: String
      }

      input WriteDataInput {
        pk1: String!
        sk1: String!
        data: [AWSJSON]!
      }

      input UpdateDataInput {
        pk1: String!
        sk1: String!
        data: [AWSJSON]!
      }

      type Mutation {
        writeData(input: WriteDataInput!): Data
        updateData(input: UpdateDataInput!): Data
        deleteData(pk1: String!, sk1: String!): Data
      }

      type Query {
        readData(pk1: String!, sk1: String!): Data
        readAllPKData(pk1: String!): DataCollection
      }
```

Source: [AWS Community Builders - Build a Serverless Data API with AppSync and DynamoDB]

There are solutions, but which fit the best?
(解決方法あるけど、どれが最適？)

AWS SAMについて

- **S**erverless **A**pplication **M**odel (SAM)
- Open Source
- Components
  - AWS SAM template specification
  - AWS SAM command line interface (AWS SAM CLI)

- Features
  - Single-deployment configuration
  - Extension of AWS CloudFormation
  - Built-in best practices
    - Code Deploy (with Canary Deployments)
    - AWS X-Ray
  - Local debugging and testing
    - Docker environment
    - AWS Toolkit (PyCharm, VSCode, etc)
  - Deep integration with development tools
    - CodeBuild, CodePipeline, AWS Toolkit, etc.

MEET SAM.

USE SAM TO BUILD TEMPLATES THAT DEFINE YOUR SERVERLESS APPLICATIONS.

DEPLOY YOUR SAM TEMPLATE WITH AWS CLOUDFORMATION.

Source: [Github - serverless-application-model]

AWS SAM：ワークフローについて

- sam-app/
  - my_function1/
    - app.js
    - package.json
  - template.yaml

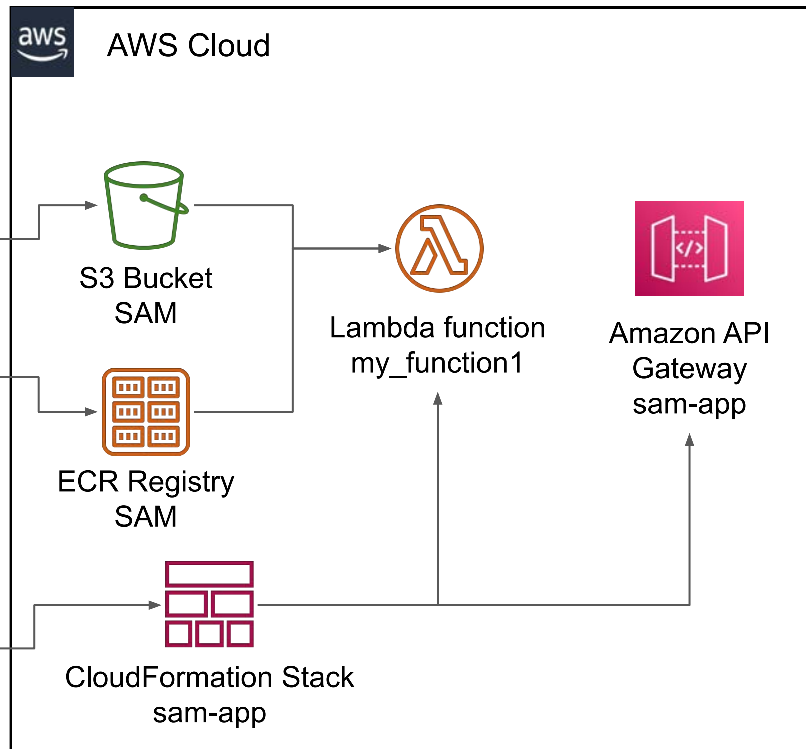
template.yaml

```yaml
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      PackageType: Zip
  FoobarFunction:
    Type: AWS::Serverless::Function
    Properties:
      PackageType: Image
```

PackageType: zip
- install modules
- zip folder

PackageType: image
- docker build

CloudFormation Template



AWS SAM：CLIについて

- コマンド（概要）
  - **sam init**
    - 新SAMプロジェクト作成
    - 10クイックスタートのテンプレートから選択する (Web Backend, SQS, S3, SNS, Step Functions)
    - zipやdockerイメージのデプロイ選択
    - 複数開発言語対応：js, java, python, .net, ruby, go
  - **sam build**
    - 依存関係の構築
  - **sam deploy --guided**
    - **--guided** インタラクティブCLI（Q＆Aパターン）
    - AWS CloudFormationのディプロイ
    - Lambdaのソースをzipに圧縮し、S3バケットへアップロードする
    - Dockerイメージ作成し、ECR リポジトリへアップロードする
  - **sam local start-api**
    - API GatewayがDockerコンテナ内でローカル環境で実行する
  - **sam local invoke**
    - Lambda FunctionがDockerコンテナ内でローカル環境で実行する
  - **sam pipeline init --bootstrap**
    - CI/CDパイプラインの環境とリソース作成する

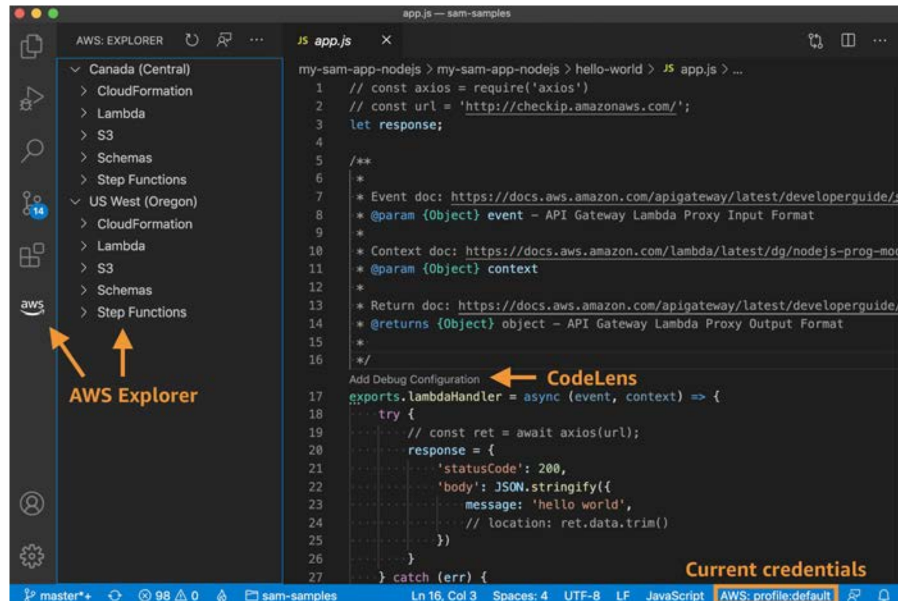
AWS SAMについて

Pro:

- Lambdaレイヤ対応
- Lambdaのデバッグ・ローカル実行
- CI/CD パイプライン デプロイメント
- 同じテンプレートでSAMリソースと CloudFormationリソース入れてOK
- AWS Toolkit for Visual Studio Code
  - プロジェクト作成
  - Lambdaのデバッグ・ローカル実行
  - デプロイ

Contra:

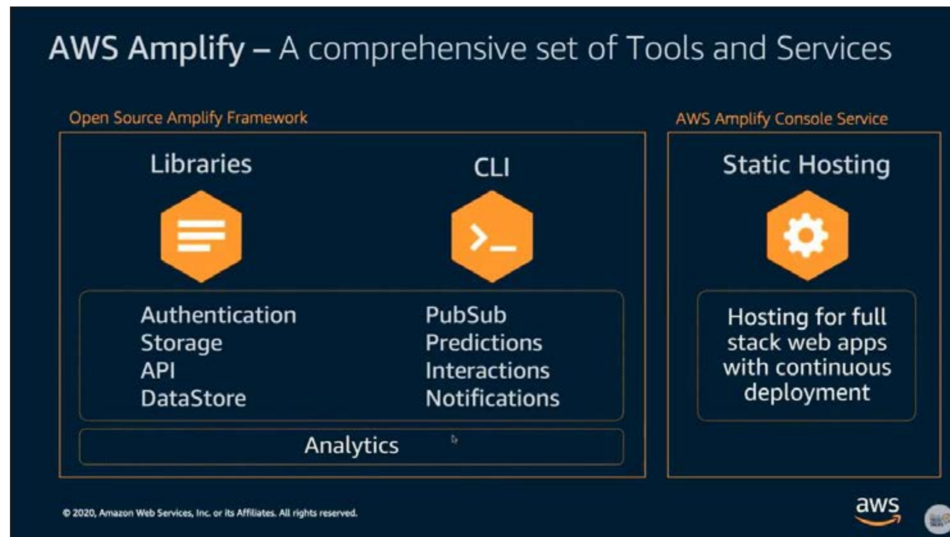
- Typescript未対応
  - 個別の設定が必要
- CloudFormationテンプレート知識必要
- API Gateway (REST)対応ですが、AppSync (GraphQL)未対応
- ローカル実行と他サービスの連携
  - 他サービス (例：SQS)をmockが必要



Source: Visual Studio Marketplace - AWS Toolkit

AWS Amplifyについて

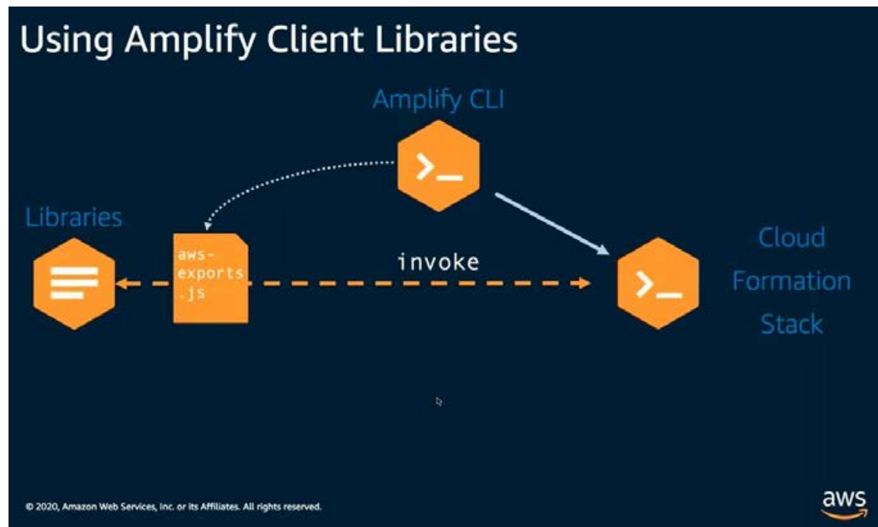
- Quickly and easily build full-stack mobile and web applications on AWS
- Open Source Framework
    - Amplify CLI:
        - toolchain to create, integrate, and manage the AWS cloud services
        - GraphQL Transform library generate resolvers
        - local mocking
    - Amplify Libraries
        - easy to use interfaces to cloud resources
    - Amplify UI Components
        - prebuilt UI components to use
- Console Service
    - CloudFront + S3ウェブサイトホスティング



Source: [Rapid Application Development Deep Dive with AWS Amplify and AWS CDK - AWS Online Tech Talks]

AWS Amplify：CLIについて

- インタラクティブCLI（Q＆Aパターン）
- コマンド (概要)
  - **amplify init**
    - 新amplifyプロジェクト作成
  - **amplify add** <resource>
    - AWSリソースを追加する
    - 例：Cognito User Pool 作成する
amplify add auth
  - **amplify push**
    - AWS CloudFormationのディプロイ
  - **amplify publish**
    - Amplify Hostingへディプロイ
  - **amplify mock**
    - ローカル開発環境作成
  - **amplify codegen**
    - GraphQL Transform ライブラリー 実行



Source: [Rapid Application Development Deep Dive with AWS Amplify and AWS CDK - AWS Online Tech Talks]

AWS Amplify：Resourcesについて

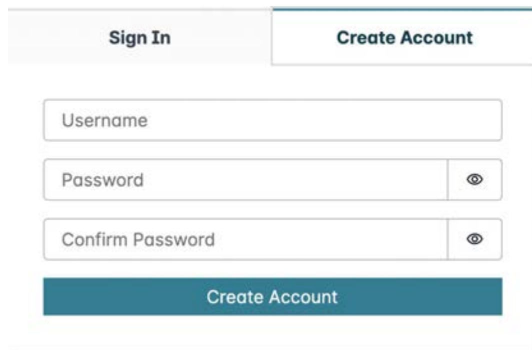
- **auth** (Amazon Cognito)
- **storage** (Amazon S3 & Amazon DynamoDB)
- **function** (AWS Lambda)
- **api** (AWS AppSync & Amazon API Gateway)
- **analytics** (Amazon Pinpoint)
- **hosting** (Amazon S3とAmazon CloudFront)
- **notifications** (Amazon Pinpoint)
- **interactions** (Amazon Lex)
- **predictions** (Amazon Rekognition, Amazon Textract, Amazon Translate, Amazon Polly, Amazon Transcribe, Amazon Comprehend, and Amazon SageMaker)

AWS Amplify：Librariesについて

- Client Library
  - フロントエンドとバックエンドの接続
    - React, Angular, Vue, 等
  - Cognito認証フロー完全に統合された
  - トークン管理（トークンの自動更新）
  - リクエストに署名する（v4）
  - アクセストークンのAPIリクエスト生成
  - 対応プラットフォーム
    - ネイティブモバイル（iOS, Android）
    - ウェブ（JavaScript）
    - Flutter
- UI Library
  - 簡単に統合できる UIコンポーネント
    - Authenticator
    - AmplifyS3Image
    - AmplifyChatbot
    - 等

React Client Library：Authのログインフロー

```
import { Auth } from 'aws-amplify';

async function signIn() {
  try {
    const user = await Auth.signIn(username, password);
  } catch (error) {
    console.log('error signing in', error);
  }
}
```



Source: [Amplify UI Components - Authenticator]

AWS Amplifyについて

Pro:

- Easy to start prototyping new applications
- Easy and fast development of AppSync (GraphQL) backed with DynamoDB
- Auto-Generate CloudFormation Templates
- fast local development with amplify mock
- best practice security solutions
  - restricted bucket access
  - API authorization rules generation
- **Client Library** allows an easy connection of frontend with backend
- **UI Library** allows reuse of UI components
  - Authenticator contains the complete authentication flow including signup, singin, reset password, etc.
- Next.js hosting possible
- hosting has integrated CD pipeline

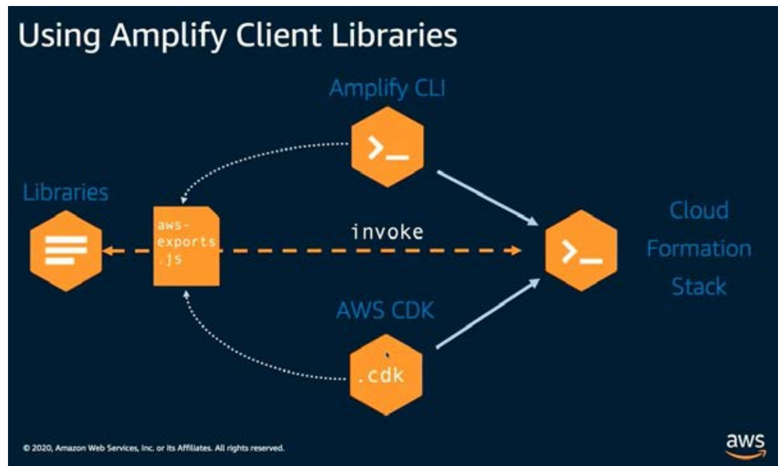
Contra:

- Resources to add are limited
  - How to add other resources?
- Limited control over resources
  - How to add tags, name resources or do detailed configuration?
- Interactive CLI is hard to integrate into other pipelines (CI/CD)
  - How to handle the CLI automatically?
- Amplify mock do not cover resources completely
  - Local mocked code fails in the could
- Major CLI bugs can affect the production environment
- Integrated authorization scheme do not support multi-tenant

対策：Amplify CLIがAWS CDKに交換する

- バックエンド
  - AWS CDKで作成する
- フロントエンド（例：Reactアプリ）
  - create-react-appで作成
  - Amplify Client Librariesそのまま使う
  - Amplify UI Library使える
- バックエンドとフロントエンドの接続
  - CDK Stackからaws-exports.jsを作成
- 問題：
  - **amplify mockが使えなくなる**
  - **ローカルのテスト・実行・デバッグ出来なくなる**

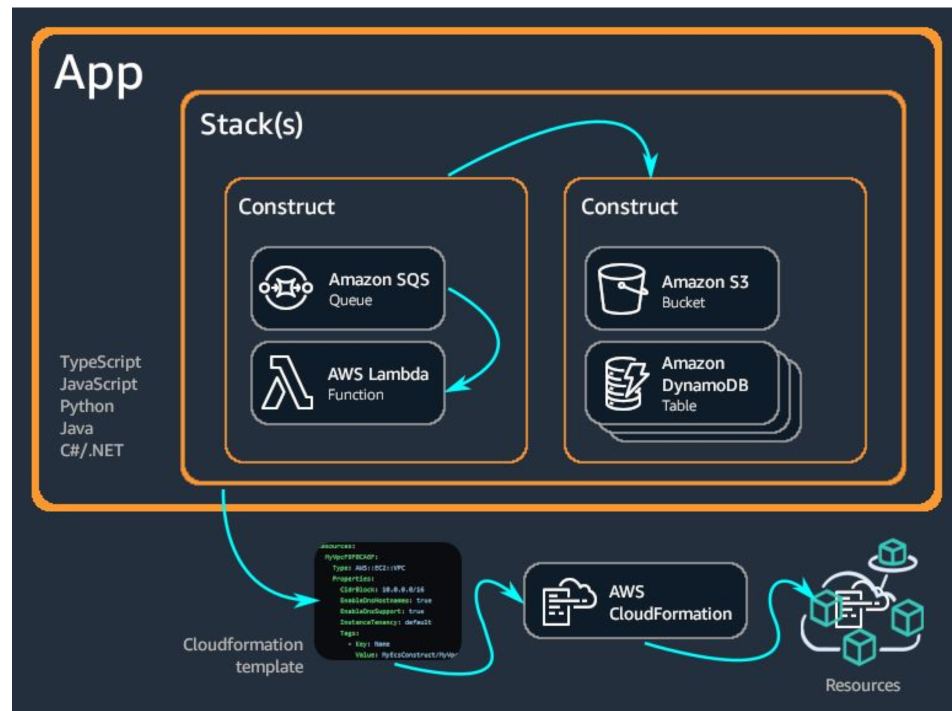
- **my_app/**
  - **backend_cdk/** ← CDKで作成
  - **frontend_react/** ← 例：create-react-appで作成



Source: Rapid Application Development Deep Dive with AWS Amplify and AWS CDK - AWS Online Tech Talks

AWS CDKについて

- プログラミング言語で使用される IaC
- 対応言語: TypeScript, JavaScript, Python, Java, C#/.Net, Go
- コンセプト（概要）
  - Construct (コンストラクト)
  - App (アプリ)
  - Stack (スタック)
  - Environment (環境)
  - Permission (許可)
  - Assets (アセット)
  - Bootstrap (ブートストラップ)
- ツールキット
  - AWS CDK Toolkit (cdk コマンド)
  - AWS Toolkit for Visual Studio Code
  - SAM CLI



Source: [AWS Cloud Development Kit (CDK) v2 - Developer Guide]

AWS CDK：Constructについて

- Construct
  - Layer 1 (L1) Construct
    - CloudFormationリソース
    - プレフィックスCfn
    - 例：**CfnBucket = AWS::S3::Bucket**
  - Layer 2 (L2) Construct
    - CloudFormationリソース
    - + ベストプラクティスのデフォルト
    - + ボイラープレートコード
    - + 接続のロジック
  - Layer 3 (L3) Construct
    - よく使われているタスクを構築する
    - 複数CloudFormationリソース
    - + 接続のロジック
    - + タスクのデフォルト

```
// L1 Construct creating a S3 bucket.
const bucket = new s3.CfnBucket(this, "MyBucket", {
  bucketName: "MyBucket"
});

// L2 Construct creating a S3 bucket AWS KMS encryption
// and static website hosting enabled. The construct
new s3.Bucket(this, 'MyEncryptedBucket', {
  encryption: s3.BucketEncryption.KMS,
  websiteIndexDocument: 'index.html'
});

// L3 Construct creating an AWS Fargate container cluster
// with public Application Load Balancer (ALB).
const repository_name = "amazon/amazon-ecs-sample";
new ecs_patterns.ApplicationLoadBalancedFargateService(this,
  "MyFargateService", {
    cluster: cluster, // Required
    cpu: 512, // Default is 256
    desiredCount: 2, // Default is 1
    taskImageOptions: {
      image: ecs.ContainerImage.fromRegistry(repository_name)
    },
    memoryLimitMiB: 2048, // Default is 512
    publicLoadBalancer: true // Default is false
  }
);
```

AWS CDK：App, Stack, Environment, Permissionについて

- App
  - AWS CDK アプリ（ルート）
  - スタックのインスタンス作成
- Environment
  - デプロイするAWSアカウントとリージョン
  - 1アプリで複数Environmentが可能（Multi-Account deployment）
- Stack
  - デプロイメントユニット
  - CloudFormationのスタック
  - 1アプリで複数スタックが可能
- Permission
  - L2 Constructから簡単なアクセス権限
  - grantRead()
  - grantReadWrite()

```
import { App, Stack, StackProps } from 'aws-cdk-lib';
import * as s3 from 'aws-cdk-lib/aws-s3';
import * as iam from 'aws-cdk-lib/aws-iam';

// 2 Environments
const envEU  = { account: '2383838383', region: 'eu-west-1' };
const envUSA = { account: '8373873873', region: 'us-west-2' };

// Stack creating a bucket and a group and grant read access
class MyFirstStack extends Stack {
  constructor(scope: App, id: string, props?: StackProps) {
    super(scope, id, props);

    const myBucket = new s3.Bucket(this, 'MyFirstBucket', {
      versioned: true
    });
    const dataScience = new iam.Group(this, 'data-science');
    // By default grant all objects
    myBucket.grantRead(dataScience);
  }
}

// App deploys the stack to 2 environments
const app = new App();

new MyFirstStack(app, 'first-stack-us', { env: envUSA });
new MyFirstStack(app, 'first-stack-eu', { env: envEU });

app.synth();
```

AWS CDK：Assetsについて

- Amazon S3 Assets

```
import { Asset } from 'aws-cdk-lib/aws-s3-assets';

// Archived and uploaded to Amazon S3 as a .zip file
const directoryAsset = new Asset(this, 'SampleZippedDirAsset', {
  path: path.join(__dirname, 'sample-asset-directory')
});
// Uploaded to Amazon S3 as-is
const fileAsset = new Asset(this, 'SampleSingleFileAsset', {
  path: path.join(__dirname, 'file-asset.txt')
});
```

- Docker Image Assets

```
import { DockerImageAsset } from 'aws-cdk-lib/aws-ecr-assets';

const asset = new DockerImageAsset(this, 'MyBuildImage', {
  directory: path.join(__dirname, 'my-image')
});
```

AWS CDK：Bootstrappingについて

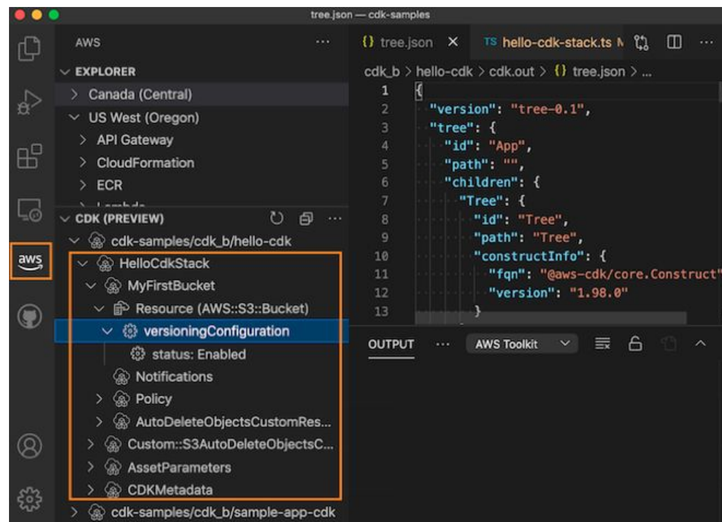
- AWS CDK デプロイメントスタックを作成と設定する
  - デプロイメントため、必要な AWSリソースを構築する（ S3バケット、IAMロール、等）
  - 目的のEnvironment毎（アカウント＋リージョン）に設定が必要
  - 既定スタック名 : CDKToolkit
- テンプレートの２種類：
  - legacy: **CDK v1 default**
    - シングルアカウントのデプロイ
    - Bootstrapのユーザーの権限を使う
    - 作成リソース: S3バケット
  - modern: **CDK v2 default**
    - マルチアカウントのデプロイ
    - Bootstrapに提供された権限を使う
    - 作成リソース: S3バケット、ECRリポジトリー、IAMロール、等

AWS CDK：ツールキットについて

- AWS CDK Toolkitのcdkコマンド（概要）
  - **cdk init app --language LANG**
    - **LANG**: typescript, javascript, python, java, csharp, go
    - 新CDKアプリを作成する
  - **cdk synth**
    - スタックのCloudFormationテンプレート生成
  - **cdk ls**
    - アプリのスタックの一覧
  - **cdk deploy**
    - スタックをCloudFormationでデプロイ
  - **cdk destroy**
    - CloudFormationからスタック削除
  - **cdk bootstrap**
    - CDKのデプロイメントスタック作成
- AWS Toolkit for Visual Studio Code
  - CDK Explorer: L2とL3コンストラクトの内容調査ツール
- SAM CLI
  - Lambda Functionのローカル実行・デバッグ

新CDKアプリを作成

```
mkdir my-project
cd my-project
cdk init app --language typescript
```



Source: Visual Studio Marketplace - AWS Toolkit

AWS CDK：Lambda function Example

- Code
  - **fromAsset(path)**
    - 直接ローカルのファイル・フォルダー使用
  - **fromAssetImage(directory)**
    - 直接ローカルの Dockerfileからビルド
  - **fromBucket(bucket, key)**
    - S3バケットからコード取得
  - **fromEcrImage(repository)**
    - ECRリポジトリーから Image使用

```
// Lambda function in folder hello/index.js

exports.handler =  async function(event, context) {
  return 'Hello World!';
}
```

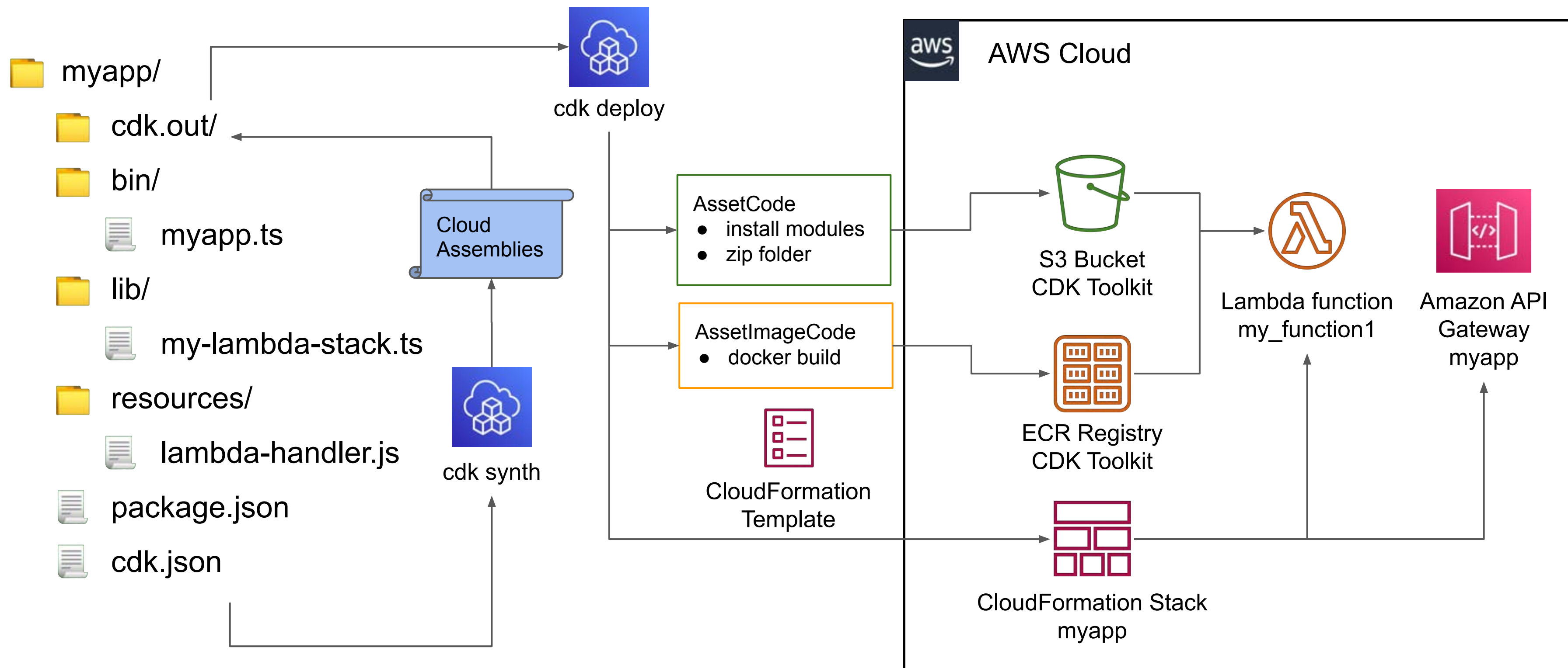
```
// AWS CDK stack creates a lambda function.
// The handler code asset will be zipped

import * as cdk from 'aws-cdk-lib';
import { Constructs } from 'constructs';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as path from 'path';

export class MyLambdaStack extends cdk.Stack {
  constructor(scope: Construct, id: string,
              props?: cdk.StackProps) {
    super(scope, id, props);

    const assetFolder = path.join(__dirname, 'hello');

    new lambda.Function(this, 'myLambdaFunction', {
      code: lambda.Code.fromAsset(assetFolder),
      runtime: lambda.Runtime.NODEJS_14_X,
      handler: 'index.handler'
    });
  }
}
```

AWS CDK：ECS task definition example

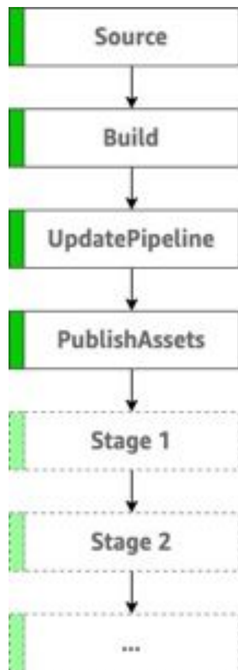
- ContainerImage
  - **fromAsset(directory)**
    - 直接ローカルの Dockerfileからビルド
  - **fromDockerImageAsset(asset)**
    - ビルド済みの Image使用
  - **fromEcr(repository)**
    - ECRリポジトリーから Image使用
  - **fromTarball(tarballFile)**
    - tarballに入っている Image使用

```
import * as ecs from 'aws-cdk-lib/aws-ecs';
import * as path from 'path';
import { DockerImageAsset } from 'aws-cdk-lib/aws-ecr-assets';

// build docker image and upload to ECR repository:
const asset = new DockerImageAsset(this, 'my-image', {
  directory: path.join(__dirname, "..", "demo-image")
});

const taskDefinition = new ecs.FargateTaskDefinition(this,
  "TaskDef", {
    memoryLimitMiB: 1024,
    cpu: 512
  }
);

// deploy-time attributes:
// - asset.repository
// - asset.imageUri
taskDefinition.addContainer("my-other-container", {
  image: ecs.ContainerImage.fromEcrRepository(
    asset.repository,
    asset.imageUri.split(":").pop()
  )
});
```

AWS CDK：ワークフローについて



AWS CDK：CDK Pipelineについて



- continuous integration and delivery (CI/CD) of AWS CDK applications
- automatically build, test, and deploy your new version of the CDK App
- APIが2つあります:
  - original:
    - Developer Preview
    - only in CDKv1 and deprecated
    - still a lot of blog posts existing
  - modern:
    - CDKv2
    - mostly AWS blogs and Developer Guide

```
// original API use CdkPipeline construct
import { CdkPipeline } from 'aws-cdk/pipelines';
```

```
// modern API use CodePipeline construct

// CDKv1
import { CodePipeline } from 'aws-cdk/pipelines';
// CDKv2
import { CodePipeline } from 'aws-cdk-lib/pipelines';
```

AWS CDK：CDK Pipeline導入について

- パイプラインのビルドStage作成
  - 例：my-pipeline-app-stage.ts
  - ビルドしたいStackを追加する
- パイプラインのStack作成
  - 例：my-pipeline-stack.ts
  - パイプラインのConstructを使う
  - input（リポジトリ）を設定する
    - Githubリポジトリ
    - CodeCommitリポジトリ
  - パイプラインのビルドコマンドを設定する
    - **npm ci**: 依存関係モジュールのインストール
    - **npm run build**: transpile ts → js
    - **npx cdk synth**: cloud assemblyの生成
  - パイプラインにビルドStage追加

```
// lib/my-pipeline-app-stage.ts
export class MyPipelineAppStage extends cdk.Stage {
  constructor(scope: Construct, id: string, props?:
cdk.StageProps) {
    super(scope, id, props);
    const lambdaStack = new MyLambdaStack(this, 'LambdaStack');
  }
}
```

```
// lib/my-pipeline-stack.ts
export class MyPipelineStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?:
cdk.StackProps) {
    super(scope, id, props);

    const pipeline = new CodePipeline(this, 'Pipeline', {
      pipelineName: 'MyPipeline',
      synth: new ShellStep('Synth', {
        input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
        commands: ['npm ci', 'npm run build', 'npx cdk synth']
      })
    });

    // add a build stage
    pipeline.addStage(new MyPipelineAppStage(this, "test", {
      env: { account: "111111111111", region: "eu-west-1" }
    }));
  }
}
```

AWS CDK：CDK Pipeline導入について

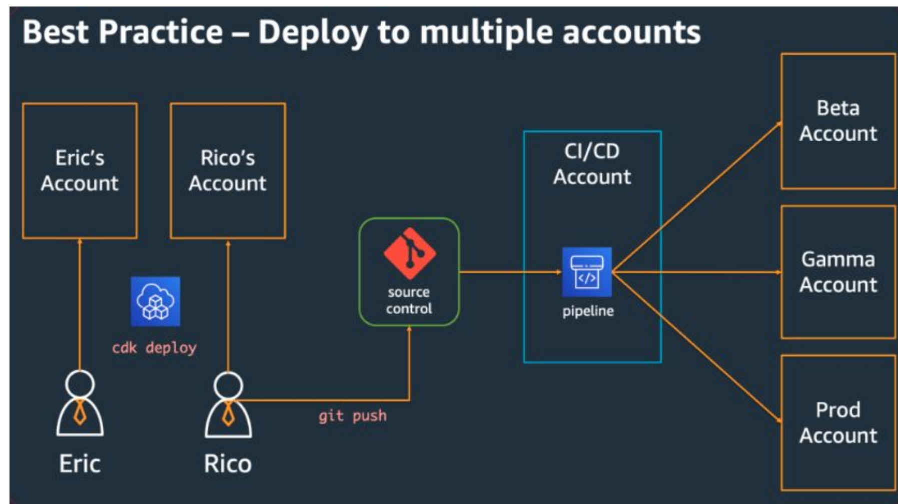
- アプリを修正する
  - 例：myapp.ts
  - 元ビルドしたStackを抜く
  - パイプラインのStack追加
- ソースをcommitする
- パイプラインをデプロイする
  - **cdk deploy**
  - 手動デプロイは一回だけ！
  - パイプラインが自動更新する

```
// bin/myapp.ts
import * as cdk from 'aws-cdk-lib';
import { MyPipelineStack } from '../lib/my-pipeline-stack';

const app = new cdk.App();
new MyPipelineStack(app, 'MyPipelineStack', {
  env: {
    account: '111111111111',
    region: 'eu-west-1',
  }
});
app.synth();
```

- myapp/
  - bin/
    - myapp.ts
  - lib/
    - my-lambda-stack.ts
    - my-pipeline-app-stage.ts
    - my-pipeline-stack.ts
  - resources/
    - lambda-handler.js

AWS CDK：Best Practiceについて

- 複数Environment（アカウント）構築する
  - AWS OrganizationsやAWS Control Towerで管理する
- 開発者アカウント
  - サーバーレスの環境構築がコスト掛からない
  - cdk deployコマンドでリソース環境構築
  - cdk watchコマンドでソース変更がすぐ開発者環境に反映
- CI/CDアカウント
  - CDK Pipelineがソース管理のソース更新で実行される
  - CDK Pipelineがデプロイする
    - Test Stage (Beta)
    - Staging Stage (Gamma)
    - Production Stage (Prod)
- Stage毎アカウント作成
  - テストや開発者アカウントのリソース変更が本番アカウントに影響なし
  - Free Tierがアカウント毎有効になる
    - AWSのコスト削減



Source: AWS Cloud Development Kit (CDK) v2 - Developer Guide

AWS CDKについて

Pro:

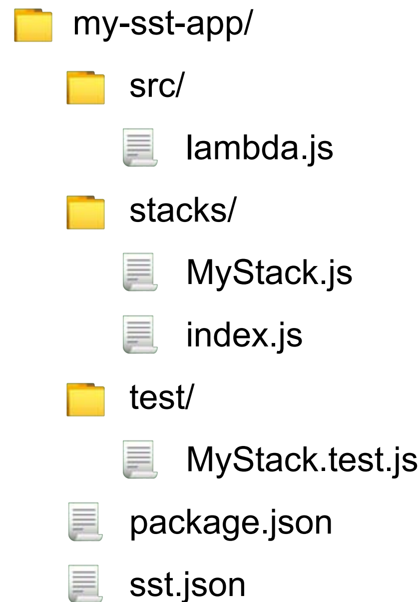
- 好きな開発言語でAWSリソース作成
- 細かいCloudFormationの知識が必要ない
- AssetCodeの自動圧縮、
S3へアップロードと管理
- AssetImageCodeの自動ビルド、
ECRへアップロードと管理
- L2 Construct
  - default設定～詳細設定対応する
  - default設定がbest practice対応
- L3 Construct
  - よく使われているタスク簡単に構築できる
  - 開発時間の削減できる
- L2 Constructの接続Permissionが簡単に追加
- カスタムConstruct作成可能
- 複数アカウントへデプロイ可能
- CDK Pipelinesでデプロイメント時間の削減簡単に導入
- ユニットテストできる
  - 開発言語のテストフレームワーク

Contra:

- ロカル実行・デバッグが難しい
  - 現在AWS SAMが必要
  - 修正後synthが必要
- Bootstrapの手動ステップが必要
- Assetの個別管理できるか？
- Constructのデバッグが難しい
- Constructの種類がAWSリソースより多い
  - 最適なConstructの調査の時間がかかる

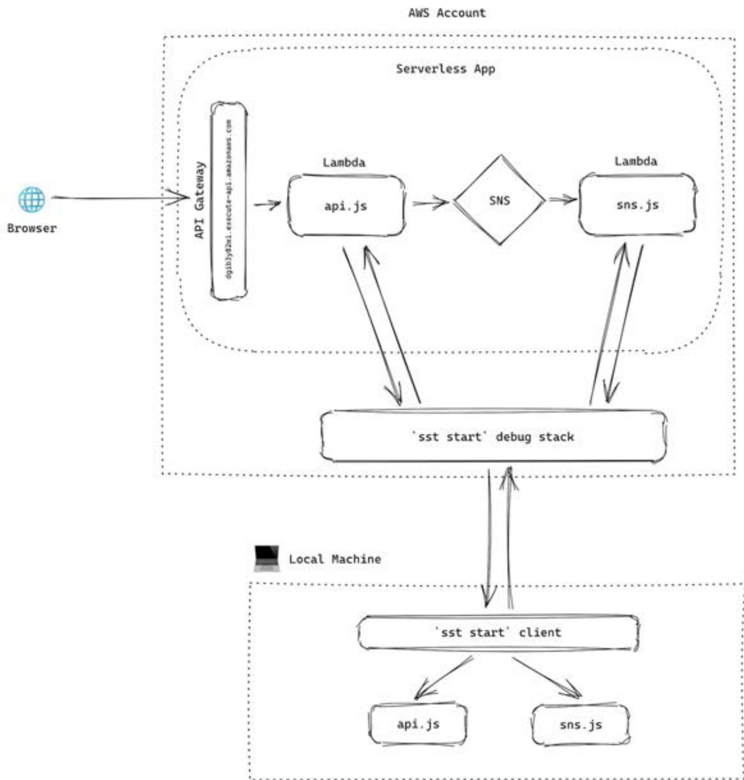
Serverless Stack (SST) について

- open-source framework
- AWSで**サーバーレス**アプリケーションを簡単に構築
- L3 Constructs提供
  - よく使われているサーバーレスタスクを構築する
- AWS CDKとSSTのConstructが使える
- デプロイメントがAWS CDKについて
- Live Lambda Development
- Lambdaデバッグ
- 現在対応言語:
  - JavaScript, TypeScript
- ユニットテストがJest Frameworkで実装
- SST CLI
  - **sst build**
    - Cloud Assemblies生成
  - **sst start**
    - Live Lambda Development環境起動
  - **sst test**
    - Unit testのデバッグモード
  - **sst deploy**
    - CloudFormationへデプロイ



SST：Live Lambda Developmentとデバッグについて

- Lambda用ローカルの開発環境
- CLIコマンド
  - **sst start**
- Debug Stackをデプロイする
  - Serverless WebSocket API
  - DynamoDB Table
  - S3 Bucket
- Lambdaをstub Lambdaを交換する
- Local WebSocket client 起動し、Debug StackのWebSocket APIと接続する
- 他サービス連携の Mockが必要ない



Source: [ServerlessStack - What Is Infrastructure as Code]

SST：VS Codeデバッグについて

- VSCode内のTerminalでsst start起動
- インフラ組織変更 watcher
- **--increase-timeout**
  - デバッグ情報送るためLambdaのTimeoutを臨時に上がる

.vscode/launch.json

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Debug SST Start",
      "type": "node",
      "request": "launch",
      "runtimeExecutable": "${workspaceRoot}/node_modules/.bin/sst",
      "runtimeArgs": ["start", "--increase-timeout"],
      "console": "integratedTerminal",
      "skipFiles": ["<node_internals>/**"]
    },
    {
      "name": "Debug SST Tests",
      "type": "node",
      "request": "launch",
      "runtimeExecutable": "${workspaceRoot}/node_modules/.bin/sst",
      "args": ["test", "--runInBand", "--no-cache",
"--watchAll=false"],
      "cwd": "${workspaceRoot}",
      "protocol": "inspector",
      "console": "integratedTerminal",
      "internalConsoleOptions": "neverOpen",
      "env": { "CI": "true" },
      "disableOptimisticBPs": true
    }
  ]
}
```

Source: [ServerlessStack - Debugging With Visual Studio Code]

SST:タスクの例

```
// Create a Cognito User Pool and Identity Pool

this.auth = new sst.Auth(this, "Auth", {
  cognito: {
    userPool: {
      // Users can login with their email and password
      signInAliases: { email: true },
    },
  },
});

this.auth.attachPermissionsForAuthUsers([
  // Allow access to the API
  api,
  // Policy granting access to a specific folder in the
bucket
  new iam.PolicyStatement({
    actions: ["s3:*"],
    effect: iam.Effect.ALLOW,
    resources: [
      bucket.bucketArn +
"/private/${cognito-identity.amazonaws.com:sub}/*",
    ],
  }),
]);
```

```
// AWS Appsync (GraphQL) Backend

new sst.Api(this, "Api", {
  routes: {
    "GET /notes": "src/list.main",
    "GET /notes/{id}": "src/get.main",
    "PUT /notes/{id}": "src/update.main"
  }
});
```

```
// AWS API Gateway (REST) Backend

new sst.AppSyncApi(this, "Api", {
  graphqlApi: { schema: "schema.graphql" },
  resolvers: {
    "Query get": "src/get.main",
    "Query list": "src/list.main",
    "Mutation update": "src/update.main"
  }
});
```

Serverless Stack (SST) について

Pro:

- L3 Construct
  - サーバーレスタスク簡単に構築できる
- AWS CDKが同時に使用可能
- Lambda Live Development
- Local Execution
  - 他サービスのMockが必要ない
- IDEでデバッグ可能
- マイグレーション方法が難しくない
  - Serverless Framework → SST
  - AWS CDK → SST
- ユニットテストできる

Contra:

- CI/CD Pipeline
  - 個別サービスSEEDで実装された
  - CDK Pipelineでも実現できるかと確認中
- Local ExecutionとLambda Live Development使用すると、少しAWSの料金かかる可能性がある

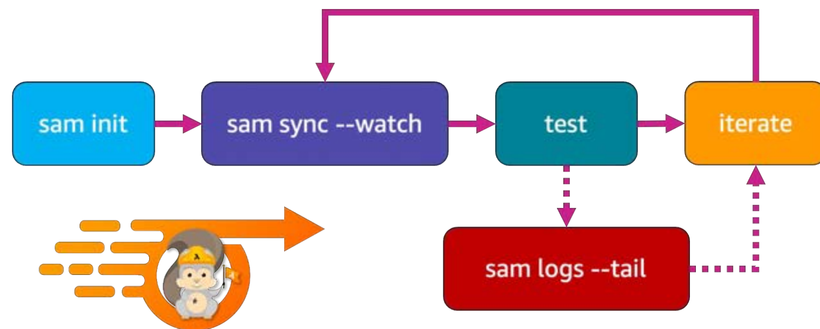
トレンド

AWS SAM:トレンドについて

- AWS Cloud Development Kit (CDK) support
  - 2021/04/29 public previewリリース
  - AWS SAM CLIでCDKアプリのビルドとテスト
  - ローカル実行
  - 開発中:AWS SAM CLIでCDKデプロイ

- AWS SAM Accelerate
  - 2021/10/27 public previewリリース
  - 自動デプロイメント
  - デプロイメント速度改善

Source: AWS Compute Blog - Better together: AWS SAM and AWS CDK



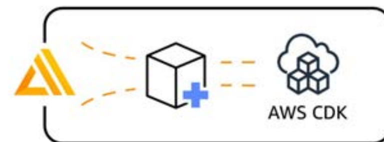
Source: AWS Compute Blog - Accelerating serverless development with AWS SAM Accelerate

AWS Amplify：トレンドについて

- Extend Backend with CDK Constructs
  - 2021/11/16リリース
  - CDKリソース導入可能
- Export Backend to CDK Stack
  - 2021/11/19リリース
  - AmplifyのStackをCDKでデプロイ可能
- GraphQL Transformers v2
  - 2021/11/23リリース
  - Lambda Functionでカスタム承認ルール作成可能
    - AppSyncでマルチテナント認証作成可能？
- Amplify UI
  - Amplify UI Componentsが個別プロジェクト[Amplify UI]に移動された
  - UI Webcomponents提供
- Amplify Studio
  - 2021/12/02リリース
  - ビジュアル開発環境
  - 最小限のコーディングでウェブアプリ作成（フロント・バックエンド）

NEW: Export Amplify projects as CDK and use in your existing CI/CD pipelines

Extend Amplify backends with custom AWS resources using AWS CDK or AWS CloudFormation



Announcing AWS Amplify's new GraphQL Transformer v2

① deny-by-default authorization
② pipeline resolvers support
③ override GraphQL cloud resources
④ more powerful OpenSearch
⑤ simpler data modeling DX
... and so much more

Source: [AWS Compute Blog - Amplify]

AWS CDK：トレンドについて

- CDK v2
  - 2021/12/02リリース
  - **aws-cdk-lib**: コアライブラリとAWSコンストラクトライブラリの安定したコンストラクト
  - **constructs**: Construct ベースクラスが個別ライブラリーに移動された。（CDK以外プロジェクトに導入可能）
- CDK Watch
  - 2021/12/02リリース
  - 新コマンド：cdk watch
  - 開発速度向上
    - アセットとコードのバックグラウンド監視
    - 自動デプロイ
  - **hotswap**
    - CloudFormationスタックをバイパスし、AWSリソースを直接更新（アセットのみ）
    - 開発版環境のみ

Serverless Stack (SST) :トレンドについて

- CDK v2 対応
  - 2022年1月リリース予定
- Next.js Hosting
  - 2021/09/22リリース
  - S3 Bucket + CloudFront CDN

References

- AWS CloudFormation
  - AWS CloudFormation - User Guide
- AWS SAM
  - AWS Serverless Application Model - Developer Guide
  - AWS Compute Blog - Better together: AWS SAM and AWS CDK
  - AWS Compute Blog - Accelerating serverless development with AWS SAM Accelerate
- AWS Toolkit for Visual Studio Code
  - AWS Toolkit for VS Code - User Guide
- AWS Amplify
  - AWS Amplify Studio のご紹介
  - AWS Amplify Studio – Figma to Fullstack React App With Minimal Programming
  - Amplify UI
  - Export Amplify backends to CDK and use with existing deployment pipelines
  - Extend Amplify backend with custom AWS resources using AWS CDK or CloudFormation
- AWS CDK
  - AWS Cloud Development Kit (CDK) v2 - Developer Guide
  - AWS What's New - AWS Cloud Development Kit (AWS CDK) v2 is now generally available
  - AWS Developer Tools Blog - Increasing development speed with CDK Watch
  - CDK Pipelines: AWS CDK アプリケーションの継続的デリバリ
- Serverless Stack (SST)
  - Serverless Stack - Docs
  - Serverless Stack - Guide
  - Update CDK to v2