AWS SAM で始める サーバーレスアプリケーション開発

2017.6.2

株式会社セクションナイン 吉田真吾

吉田真吾

- バックグラウンド
  - 証券システム基盤開発
    - 基盤開発、Oracleチューニングなど
  - エバンジェリスト
    - 講演年間113回(2013年実績)
    - AWS設計・構築・移行(2014-2015)
- 現在のしごと
  - (株) セクションナイン 代表取締役社長
    - AWSコンサルティング・設計・構築
    - DevOps、Dockerize、Serverless 支援
  - (株) Cloud Payment
    - 技術顧問
- 実績等
  - AWSウルトラクイズ 初代チャンピオン (2012年)
  - AWS Samurai 2014 / **2017** ←New!!
  - AWSエキスパート養成読本 執筆
  - AWS認定全資格(5種類コンプリート)
  - Oracle Database 11g (Gold, Performance Tuning)

amazon web services Certified
Solutions Architect - Professional

amazon web services Certified
DevOps Engineer - Professional

amazon web services Certified
Solutions Architect - Associate

amazon web services Certified
SysOps Administrator - Associate

amazon web services Certified
Developer - Associate

ORACLE MASTER GOLD

ORACLE
Certified Expert
Oracle Database 11g
Performance Tuning

AWS Samurai 2016 受賞

AWS Samurai

ユーザーコミュニティにおけるさまざまな継続的な活動において、ユーザーコミュニティの成長およびAWSクラウドの普及に大きく貢献または影響を与えた人たち

吉田 真吾 さん（JAWS-UG 横浜支部）

新たなクラウドのアーキテクチャとして注目されているサーバーレスや開発効率を向上させるための関連フレームワークなど、インフラエンジニアやアプリ開発者向けのイベントや各種勉強会をオーガナイザーとして多くの仲間を集め、東京や大阪地域で積極的に企画および実施しました。また、**JAWS-UG横浜をリブート**し、次世代のコミュニティリーダーの発掘・育成など、ユーザーグループが今後さらに成長する為の施策を進めました。アプリ／ウェブデベロッパーやインフラエンジニアをはじめとした開発・運用に関わるエンジニアの方々にサーバーレスアーキテクチャを知ってもらい、サービスを活用してイノベーションを起こしてもらえるよう、今後もファンや仲間を増やす活動を進めていただきたいと思います。

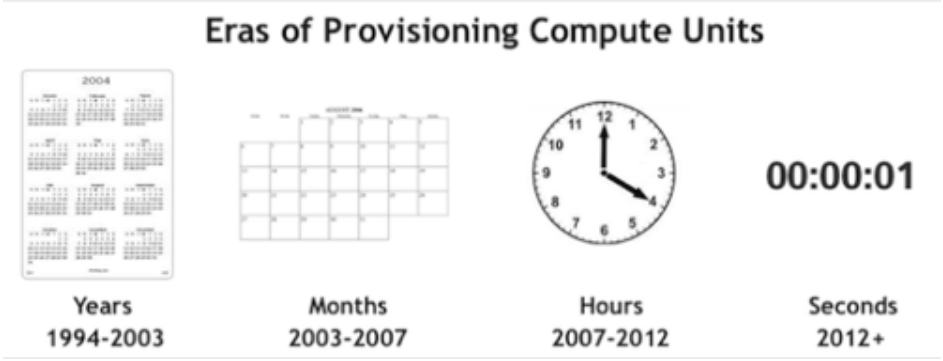
がんばります！

Serverless Community(JP)

- ServerlessConf Tokyo ’16
- Serverless Meetup (Tokyo|Osaka|Sapporo)

パラダイムシフト

- Why The Future Of Software And Apps Is Serverless
  by Ken Fromm, VP of Business Development at Iron.io
  - コンピューティングリソースの調達リードタイムの短縮
  - スタンダローンアプリからの変化（現在のMicroservices）
  - クラウドで柔軟にコンピューティングリソースをサービスとして利用することができる
  - サーバーが要らないということではなく、開発者はサーバーについて「考えなくてもよくなる」



http://readwrite.com/2012/10/15/why-the-future-of-software-and-apps-is-serverless/

サーバーレスエコシステム

- サーバー構築不要
- スケーラブル
- 従量課金　etc…

プラットフォーム事業者

- API定義と関数コードの一体管理
- チーム開発（テスト、デプロイ）　etc…

フレームワークやツール

- 企画→開発→デリバリーに集中
- サービスマネジメント
- etc…

アプリケーション開発者

Serverlessでの開発にどんなツールを使ってますか？

#	ツール名	用途	人数
1	Serverless Framework	開発フレームワーク・デプロイ(Node)	**35**
2	Apex	複数Lambda関数のデプロイ(Node)	**12**
3	Lamvery	複数Lambda関数のデプロイ(Python)	**9**
4	Swagger	API定義・設計	**8**
5	AWS Serverless Application Model(SAM)	開発フレームワーク・デプロイ (CloudFormation)	**7**
6	Postman	RESTのリクエスター・テストツール	**7**
7	Microsoft Visual Studio	統合開発環境(IDE)	**5**
8	AWS CLI	コマンドラインツール	**5**
9	Eclipse	統合開発環境(IDE)	**3**
10	Python Serverless Microframework for AWS (chalice)	開発フレームワーク・デプロイ(Python)	**2**

Serverless Community(JP) https://www.facebook.com/groups/813718382095265/

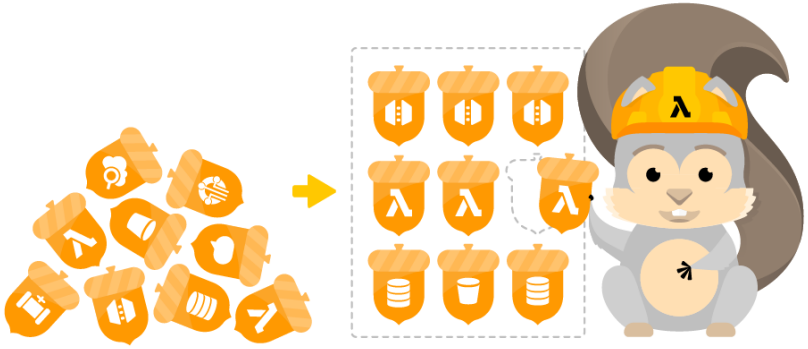
AWS Serverless Application Model (SAM)

- サーバーレスアプリケーションの管理フレームワーク
- アプリケーションを CloudFormation テンプレート(AWS SAMファイル)で統合管理できる
- 現在 Lambda、API Gateway、DynamoDB(簡易的) がサポートされているが、それ以外の設定を統合的に行いたい場合は既存のCFnと合わせて実行可能
- Apache 2.0 ライセンス

https://github.com/awslabs/serverless-application-model

AWS Serverless Application Model (SAM)

MEET SAM.



USE SAM TO BUILD TEMPLATES THAT DEFINE YOUR SERVERLESS APPLICATIONS.

DEPLOY YOUR SAM TEMPLATE WITH AWS CLOUDFORMATION.

https://github.com/awslabs/serverless-application-model

サンプルコード

- Lambda単体
- S3トリガ (S3イベント->Lambda)
- スケジュールイベント (CW Events->Lambda)
- APIバックエンド (API GW->Lambda->DynamoDB)
- Swagger定義から作るAPIバックエンド
- IoTバックエンド (IoT->Lambda->DynamoDB)
- ストリームプロセッシング (Kinesis Stream->Lambda)
- Alexaスキル

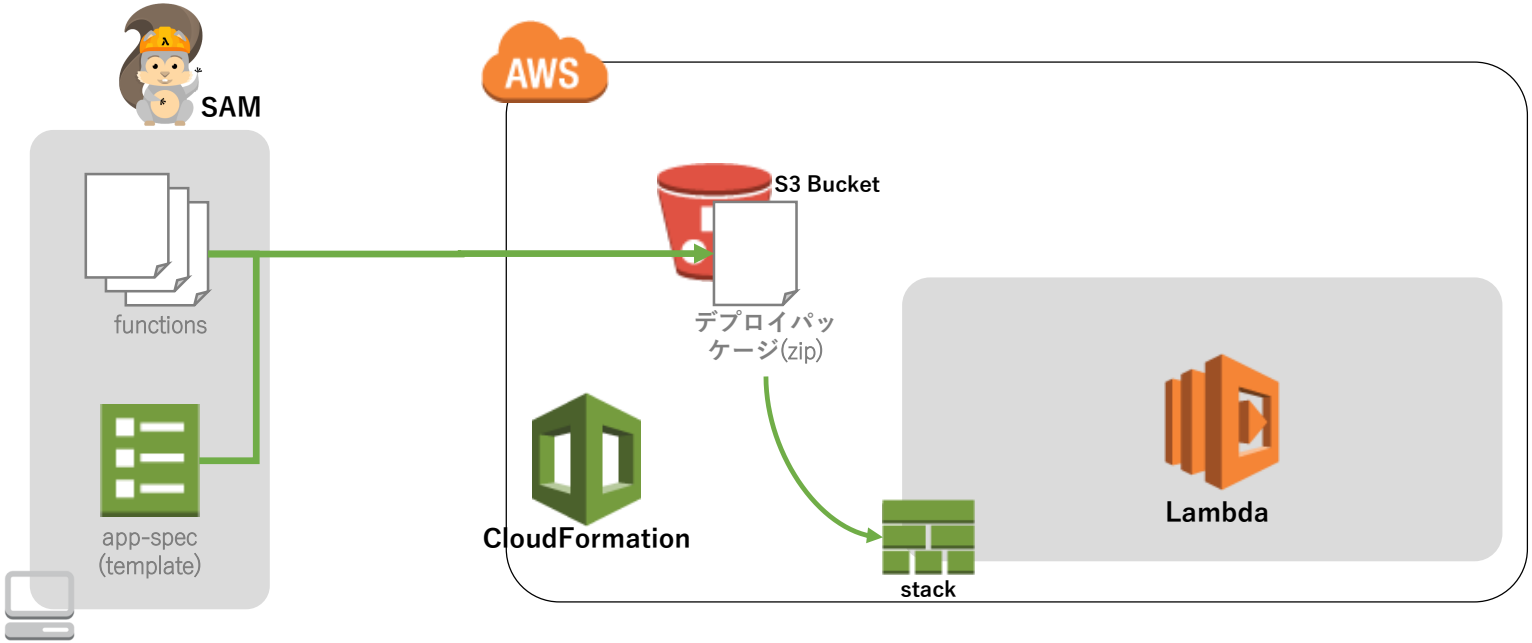
https://github.com/awslabs/serverless-application-model

サンプルコード

- **Lambda単体**
- S3トリガ (S3イベント->Lambda)
- スケジュールイベント (CW Events->Lambda)
- APIバックエンド (API GW->Lambda->DynamoDB)
- Swagger定義から作るAPIバックエンド
- IoTバックエンド (IoT->Lambda->DynamoDB)
- ストリームプロセッシング (Kinesis Stream->Lambda)
- Alexaスキル

https://github.com/awslabs/serverless-application-model

作成する構成

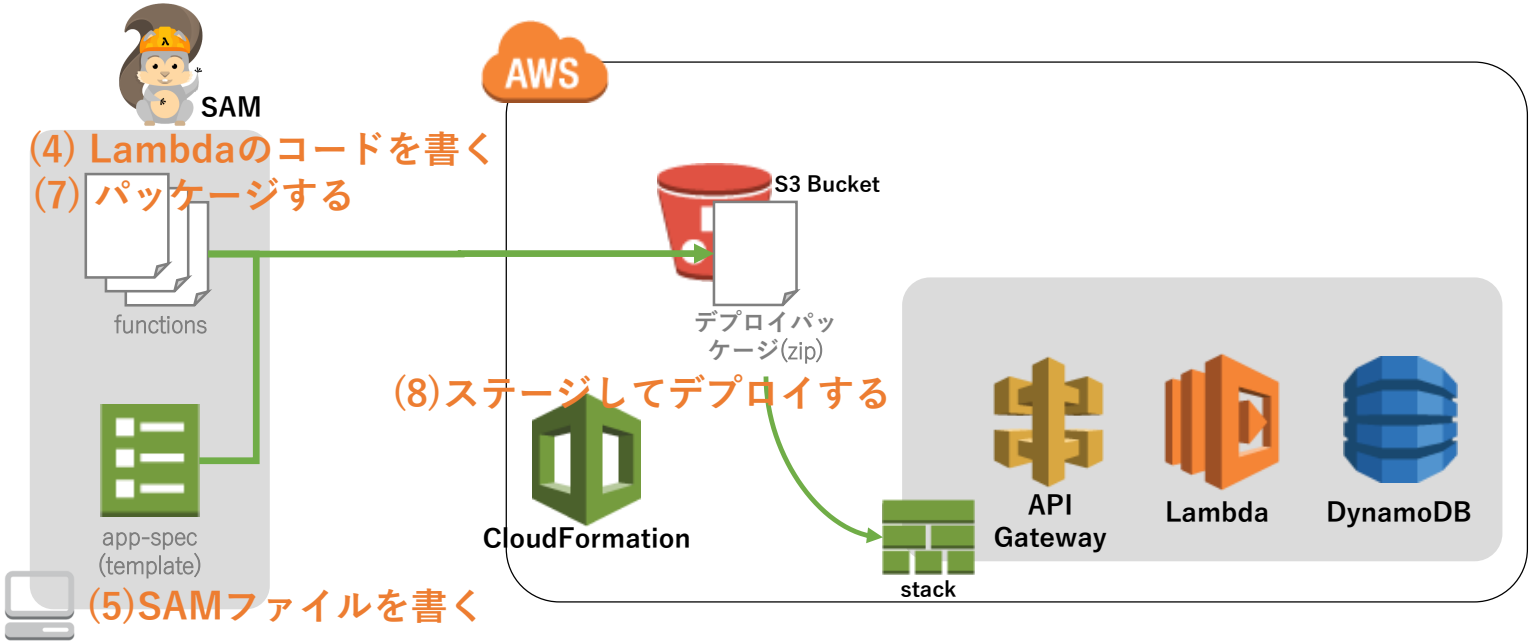

SAM
functions
app-spec
(template)
AWS
S3 Bucket
デプロイパッ
ケージ(zip)
CloudFormation
stack
Lambda

手順

- 準備編
  1. AWS CLIのインストール
  2. IAMユーザー作成とクレデンシャル取得
  3. リリースステージ用S3バケットの作成
- 開発編
  4. Lambdaのコードを書く
  5. AWS SAMファイルを書く
- デプロイ編
  6. Lambda関数のzip化
  7. パッケージする
  8. アーティファクトをステージしてデプロイする

準備編

手順


SAM
(4) Lambdaのコードを書く
(7) パッケージする
functions
app-spec
(template)
(5)SAMファイルを書く
AWS
S3 Bucket
デプロイパッケージ(zip)
(8)ステージしてデプロイする
CloudFormation
stack
API Gateway
Lambda
DynamoDB

準備編

AWS CLI のインストール

```
$ pip install awscli  [Mac]

あるいは インストーラー実行 [Win]
```

< SAMのパッケージやデプロイ作業は
aws cloudformation サブコマンドでおこなうよ

準備編

IAMユーザーのクレデンシャル作成

< aws cloudformation コマンドの実行や
実行中にAWSリソースや実行ロールを作成するための
権限が必要だよ

準備編

IAMユーザーのクレデンシャル作成

```
$ aws configure

AWS Access Key ID : AKI****************
AWS Secret Access Key : ****************
Default region name : ap-northeast-1
Default output format : json
```

< aws cloudformation コマンドの実行や
実行中にAWSリソースや実行ロールを作成するための
権限が必要だよ

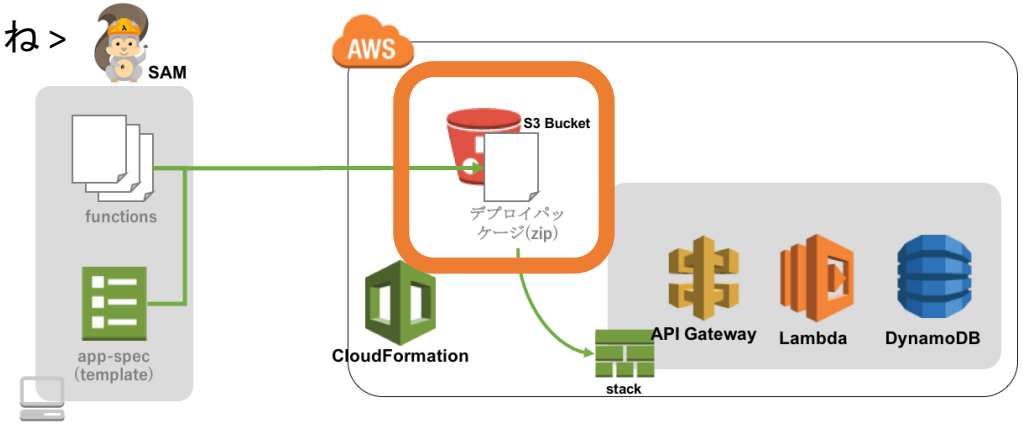
準備編

リリースステージ用のS3バケット作成

```
$ aws s3 mb s3://bucket-name --region ap-northeast-1
```

あるいはコンソールから作成

世界にひとつだけの名前をつけてね＞



開発

構成ファイル

```
$ tree
.
 ├── app-spec.yml        [AWS SAM ファイル]
 └── index.js            [Lambda 関数コード]
```

< シンプルだね！

開発

AWS SAMファイル

< ここでSAMにテンプレを切替え

< ここにリソース定義

“HelloWorldFunction”
というLambda Functionを
ハンドラーとランタイムを指定して
ローカルのapp.zipをデプロイするよ

```
AWSTemplateFormatVersion: '2010-09-09’
Transform: AWS::Serverless-2016-10-31
Description: A hello world application.
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs4.3
      CodeUri: app.zip
```

開発

Lambda関数コード

```
'use strict';
console.log('Loading function');

exports.handler = (event, context, callback) => {
    callback(null, 'Hello World!');
};
```

＜ハンドラーがコールされたら単純なレスポンスを返すよ

アーティファクトのパッケージ

```
$ zip app.zip index.js

$ aws cloudformation package --template-file app-spec.yml
 --output-template-file app-spec.deploy --s3-bucket bucket-name

$ tree
.
├── app-spec.yml
├── app-spec.deploy
├── index.js
└── app.zip
```

デプロイ

```
$ aws cloudformation deploy ¥
 --template-file app-spec.deploy ¥
 --stack-name stack-name ¥
 --capabilities CAPABILITY_IAM
```

< 最後のcapabilities指定はFunctionのサービスロールを
作成するためのIAMリソースの承認だよ、忘れずにね

※ ¥はバックスラッシュに読み替えてね

CloudFormation ▾ | スタック › スタックの詳細

hello-sam-stack

スタックの名前:	hello-sam-stack
スタック ID:	arn:aws:cloudformation:ap-northeast-1:633064615840:stack/hello-sam-stack/088347e0-4460-11e7-8e88-500c28b9749a
状況:	CREATE_COMPLETE
状況の理由:	
IAM ロール:	
説明:	

▸ 出力

▸ リソース

▾ イベント

2017-05-29	状況	タイプ	論理 ID	状況の理由
▸ 20:15:33 UTC+0900	CREATE_COMPLETE	AWS::CloudFormation::Stack	hello-sam-stack	
▸ 20:15:30 UTC+0900	CREATE_COMPLETE	AWS::Lambda::Function	HelloWorldFunction	
▸ 20:15:30 UTC+0900	CREATE_IN_PROGRESS	AWS::Lambda::Function	HelloWorldFunction	Resource creation Initiated
20:15:29 UTC+0900	CREATE_IN_PROGRESS	AWS::Lambda::Function	HelloWorldFunction	
▸ 20:15:25 UTC+0900	CREATE_COMPLETE	AWS::IAM::Role	HelloWorldFunctionRole	
▸ 20:15:08 UTC+0900	CREATE_IN_PROGRESS	AWS::IAM::Role	HelloWorldFunctionRole	Resource creation Initiated
20:15:07 UTC+0900	CREATE_IN_PROGRESS	AWS::IAM::Role	HelloWorldFunctionRole	
▸ 20:15:01 UTC+0900	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	hello-sam-stack	User Initiated
▸ 20:14:50 UTC+0900	REVIEW_IN_PROGRESS	AWS::CloudFormation::Stack	hello-sam-stack	User Initiated


Lambda > 関数 > hello-sam-stack-HelloWorldFunction-I6RVQ0CEEV6K
限定条件
テスト
アクション
コード
設定
トリガー
タグ
モニタリング
コード エントリ タイプ
コードをインラインで編集
1 'use strict';
2 console.log('Loading function');
3
4 exports.handler = (event, context, callback) => {
5 callback(null, 'Hello World!');
6 };
実行結果: 成功 (ログ)
関数の実行から返された結果が以下のエリアに表示されます。関数から結果を返す方法の詳
"Hello World!"
概要
コード SHA-256 /jLuZBbpz3+tXPQKrH7EBTyvuVh4YZALJVCzWcD45X0=
リクエスト ID fcf5c74a-4469-11e7-b558-bf8f1d5346de
所要時間 12.38 ms
課金期間 100 ms
設定済みリソース 128 MB
使用中の最大メモリ 18 MB

< 大成功！

< ローカルじゃなくてS3からデプロイする場合はCodeUriにS3のキー名(ファイルパス)を指定しよう

通常のCFnと同じようにParameter節で定義した変数を組込み関数を使って参照できる

```
AWSTemplateFormatVersion: '2010-09-09’
Transform: AWS::Serverless-2016-10-31
Description: A hello world application.
Parameters:
    BucketName:
        Type: String
        Default: ”bucket-name”
    CodeKey:
        Type: String
        Default: "app.zip"
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs4.3
      CodeUri:
        Bucket: !Ref BucketName
        Key: !Ref CodeKey
```

< さて、Lambdaだけじゃつまらないね
つぎは本格的にいくつかのサービスが
連携したスタックを丸ごと作ってみよう！

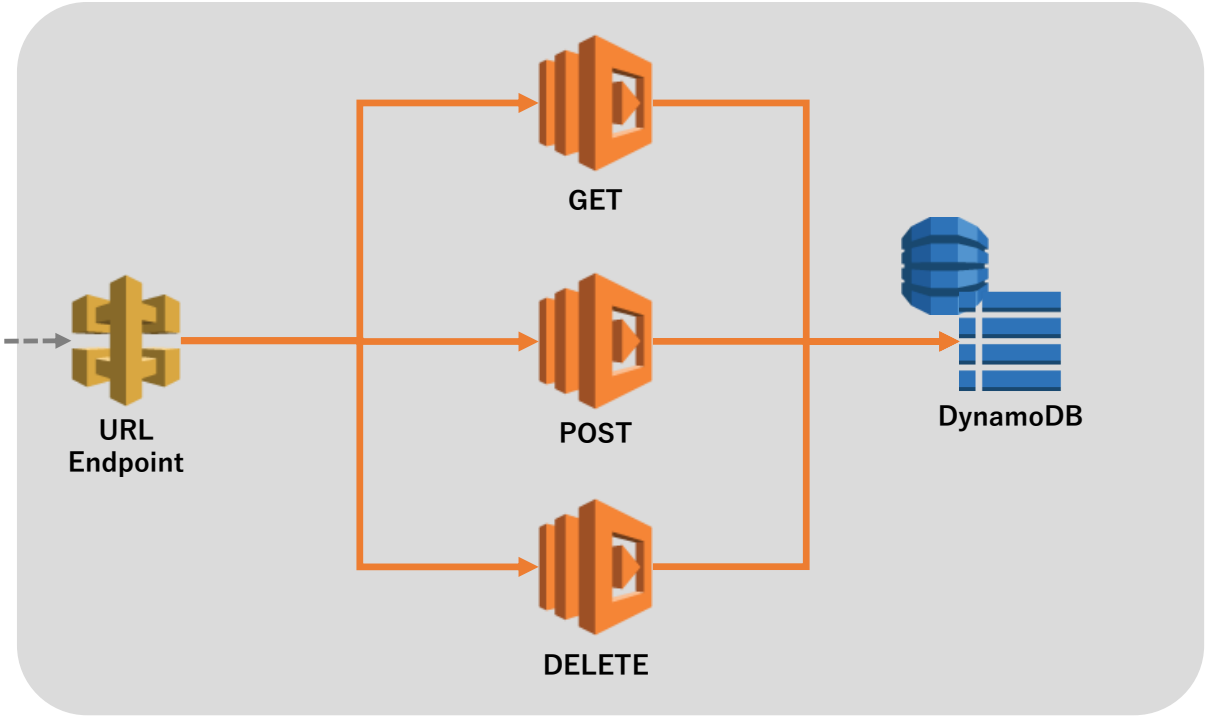
サンプルコード

- Lambda単体
- S3トリガ (S3イベント->Lambda)
- スケジュールイベント (CW Events->Lambda)
- **APIバックエンド (API GW->Lambda->DynamoDB)**
- Swagger定義から作るAPIバックエンド
- IoTバックエンド (IoT->Lambda->DynamoDB)
- ストリームプロセッシング (Kinesis Stream->Lambda)
- Alexaスキル

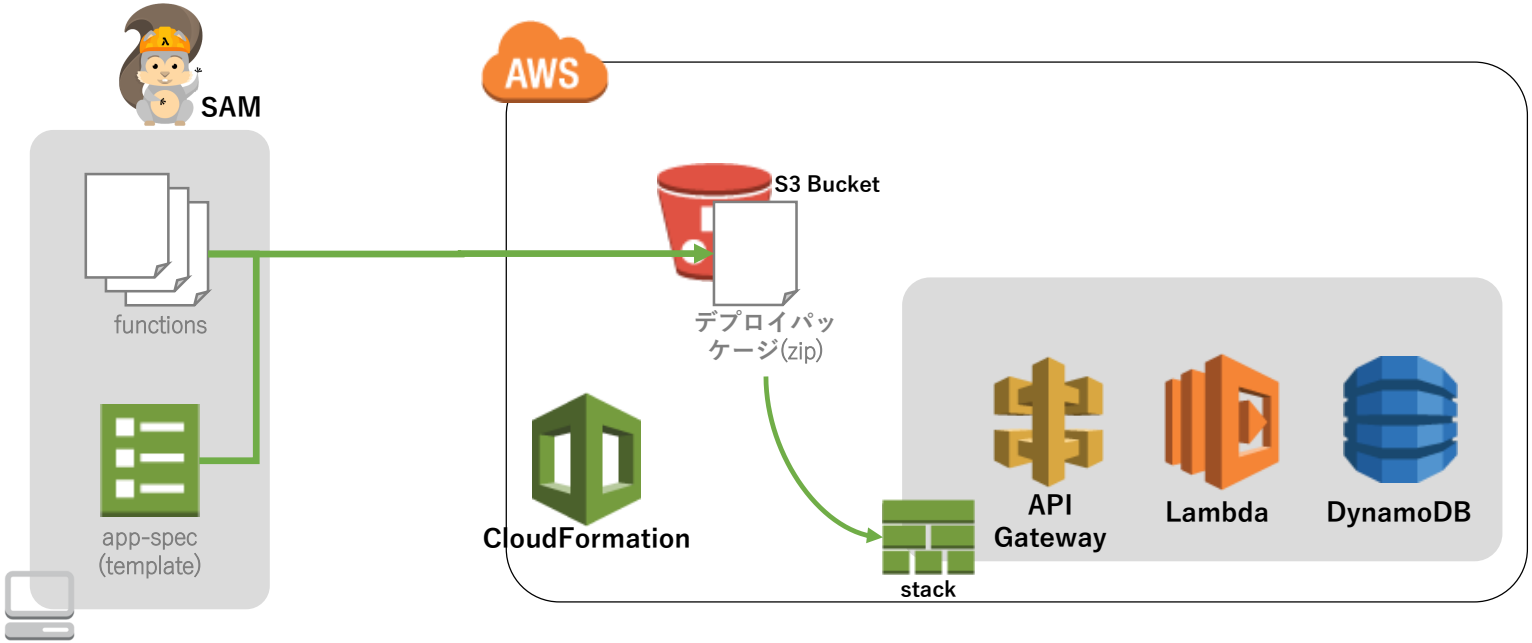
https://github.com/awslabs/serverless-application-model

作成する構成

<テーブルデータの
取得・追加・削除
をするAPIだよ



作成する構成


SAM
functions
app-spec
(template)
AWS
S3 Bucket
デプロイパッ
ケージ(zip)
CloudFormation
stack
API
Gateway
Lambda
DynamoDB

手順

- 準備編
  1. AWS CLIのインストール
  2. IAMユーザー作成とクレデンシャル取得
  3. リリースステージ用S3バケットの作成
- 開発編
  4. Lambdaのコードを書く
  5. AWS SAMファイルを書く
- デプロイ編
  6. Lambda関数のzip化
  7. パッケージする
  8. アーティファクトをステージしてデプロイする

開発

Lambda関数コード

```
'use strict';
console.log('Loading function');

let doc = require('dynamodb-doc');
let dynamo = new doc.DynamoDB();

const tableName = process.env.TABLE_NAME;

const createResponse = (statusCode, body) => {
    return {
        "statusCode": statusCode,
        "body": body
    }
};

exports.get = (event, context, callback) => {
    var params = {
        "TableName": tableName,
        "Key": {
            id: event.pathParameters.resourceId
        }
    };
    dynamo.getItem(params, (err, data) => {
        var response;
        if (err)
            response = createResponse(500, err);
        else
            response = createResponse(200, data.Item.doc);
        callback(null, response);
    });
};
```

```
exports.put = (event, context, callback) => {
    var item = {
        "id": event.pathParameters.resourceId,
        "doc": event.body
    };

    var params = {
        "TableName": tableName,
        "Item": item
    };
    dynamo.putItem(params, (err, data) => {
        var response;
        if (err)
            response = createResponse(500, err);
        else
            response = createResponse(200, null);
        callback(null, response);
    });
};

exports.delete = (event, context, callback) => {
    var params = {
        "TableName": tableName,
        "Key": {
            "id": event.pathParameters.resourceId
        }
    };
    dynamo.deleteItem(params, (err, data) => {
        var response;
        if (err)
            response = createResponse(500, err);
        else
            response = createResponse(200, null);
        callback(null, response);
    });
};
```

< get/put/delete
用のハンドラ

開発

AWS SAMファイル

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Simple CRUD webservice. State is stored in a SimpleTable (DynamoDB) resource.
Resources:
  GetFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.get
      Runtime: nodejs4.3
      Policies: AmazonDynamoDBReadOnlyAccess
      Environment:
        Variables:
          TABLE_NAME: !Ref Table
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /resource/{resourceId}
            Method: get

  PutFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.put
      Runtime: nodejs4.3
      Policies: AmazonDynamoDBFullAccess
      Environment:
        Variables:
          TABLE_NAME: !Ref Table
      Events:
        PutResource:
          Type: Api
          Properties:
            Path: /resource/{resourceId}
            Method: put
```

```yaml
  DeleteFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.delete
      Runtime: nodejs4.3
      Policies: AmazonDynamoDBFullAccess
      Environment:
        Variables:
          TABLE_NAME: !Ref Table
      Events:
        DeleteResource:
          Type: Api
          Properties:
            Path: /resource/{resourceId}
            Method: delete

  Table:
    Type: AWS::Serverless::SimpleTable
```

< それぞれのFunctionに対応する
APIの定義や環境変数(テーブル名)
や簡易的なテーブル作成ができる

開発

AWS SAMファイル

- 設定可能なリソースタイプ
  - AWS::Serverless::Function[Lambda]
  - AWS::Serverless::Api [API Gateway]
  - AWS::Serverless::SimpleTable [DynamoDB]
- 設定可能なイベントソースタイプ
  - S3
  - Kinesis
  - Api
  - CloudWatchEvent
  - AlexaSkill
  - SNS
  - DynamoDB
  - Schedule
  - IoTRule

https://github.com/awslabs/serverless-application-model/blob/howto_guide/versions/2016-10-31.md

アーティファクトのパッケージ

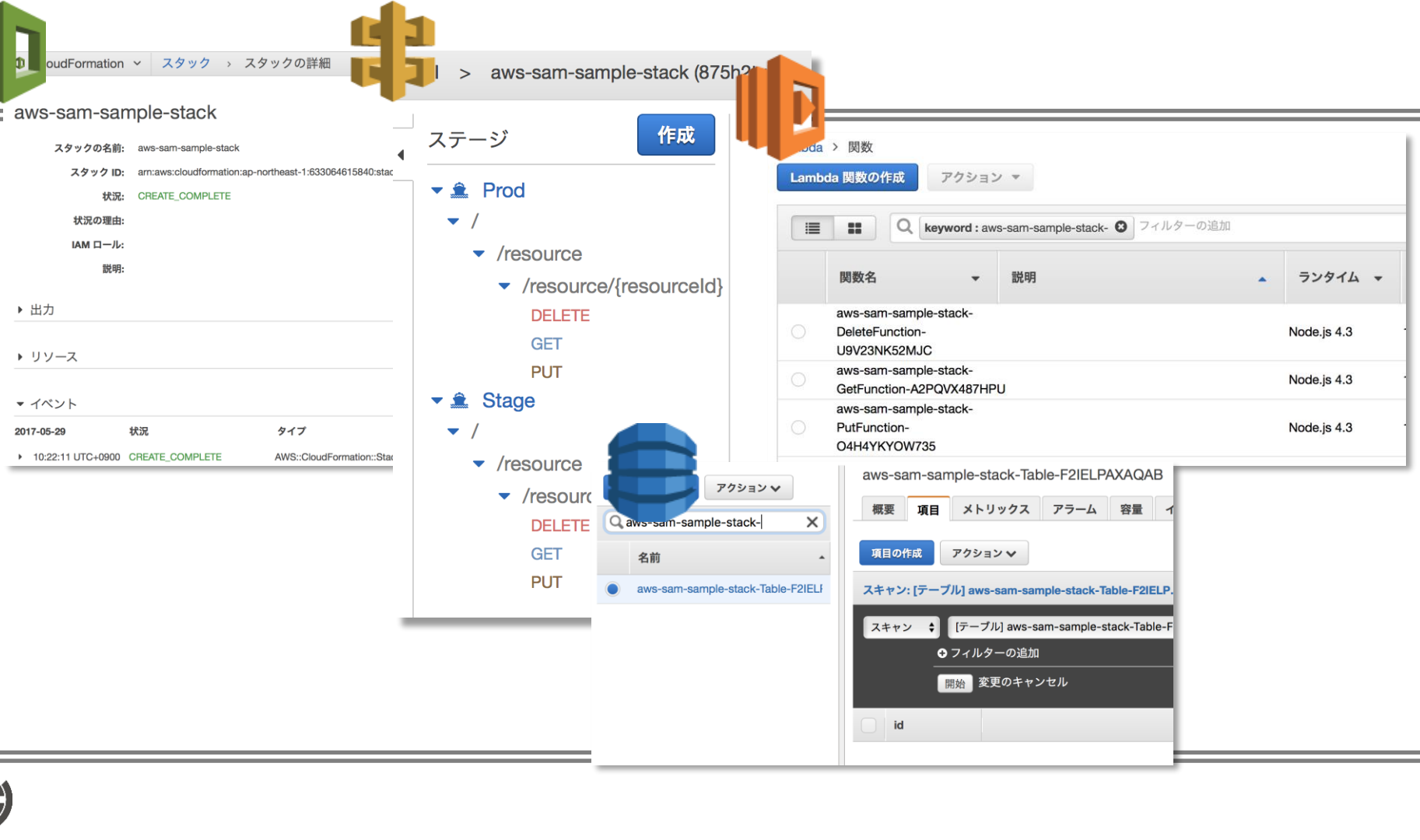
```
$ zip app.zip index.js

$ aws cloudformation package --template-file app-spec.yml
 --output-template-file app-spec.deploy --s3-bucket bucket-name

$ tree
.
├── app-spec.yml
├── app-spec.deploy
├── index.js
└── app.zip
```

デプロイ

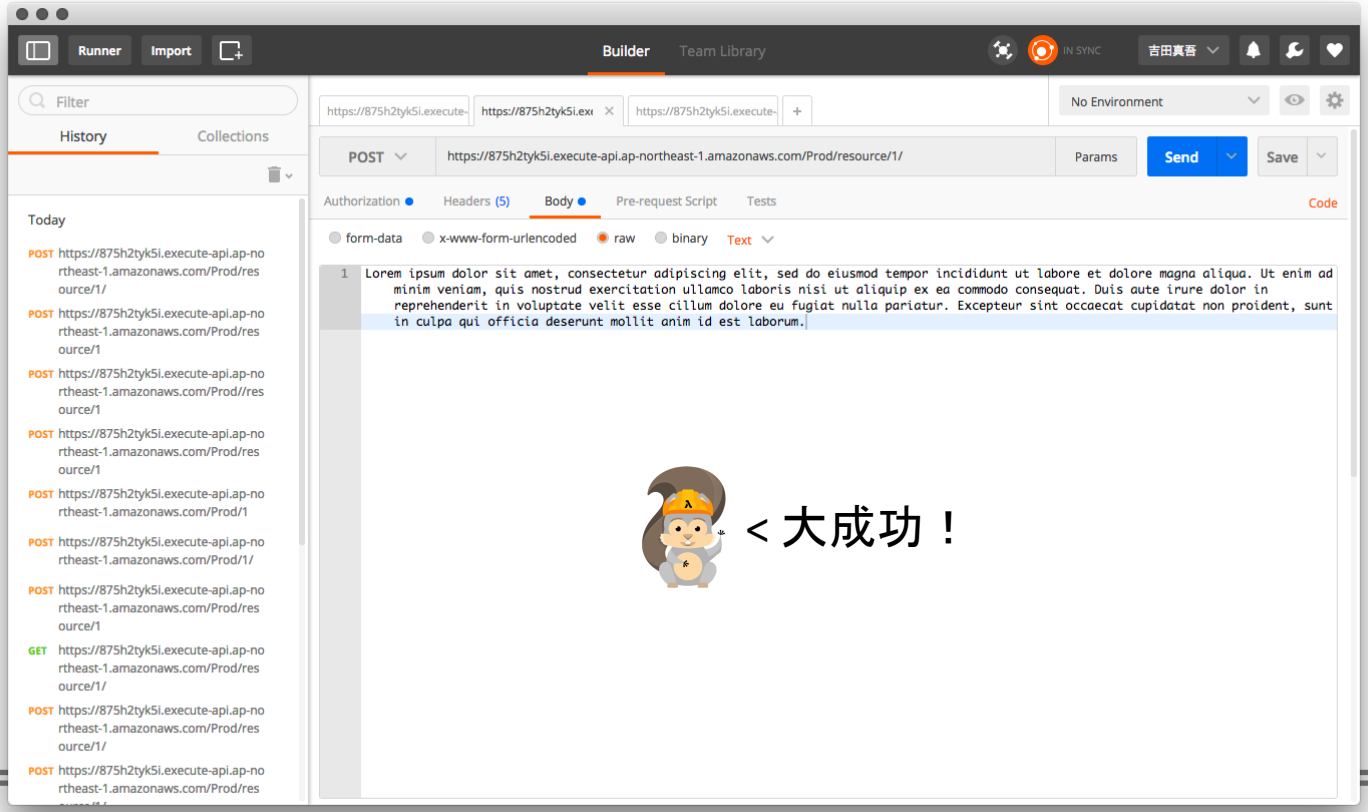
```
$ aws cloudformation deploy ¥
 --template-file app-spec.deploy ¥
 --stack-name stack-name ¥
 --capabilities CAPABILITY_IAM
```

< 最後のcapabilities指定はFunctionのサービスロールを
作成するためのIAMリソースの承認だよ、忘れずにね

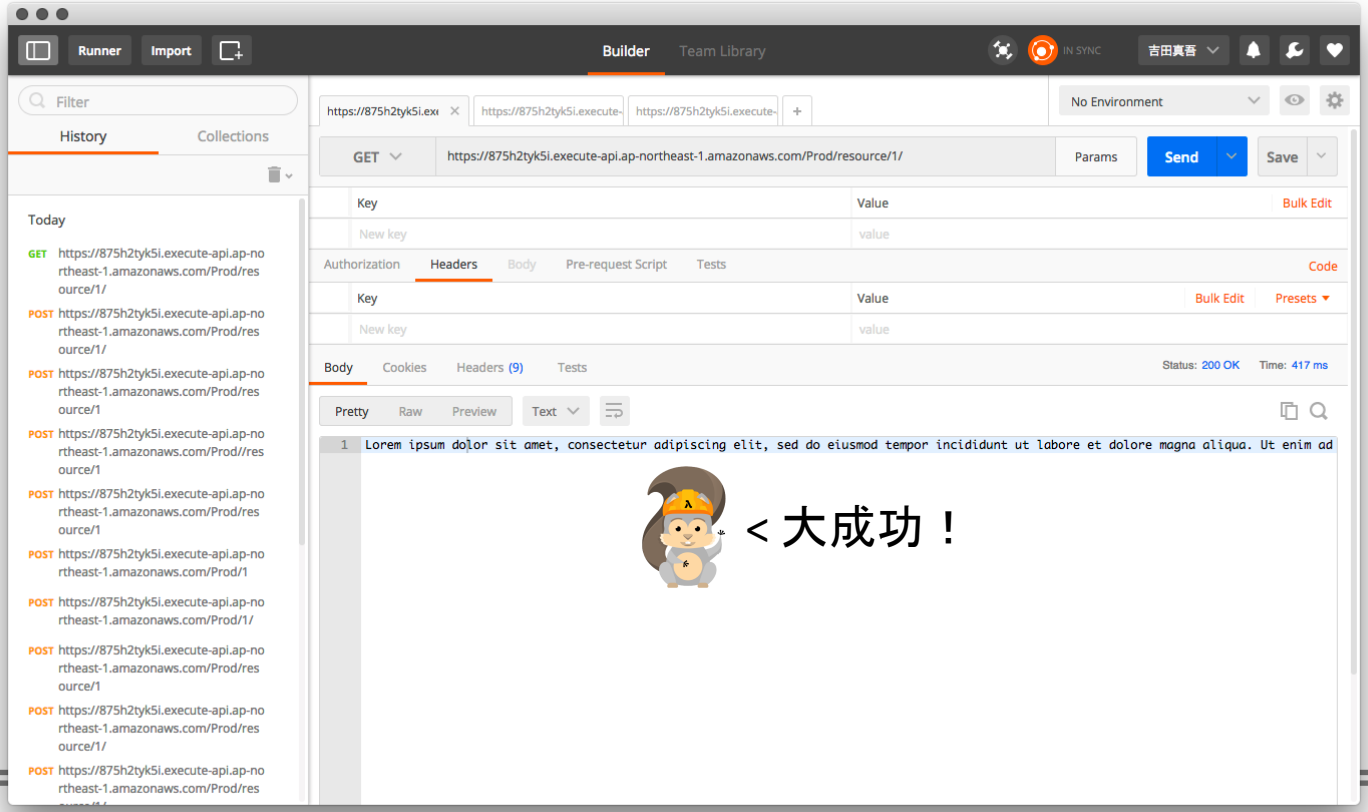
※ ¥はバックスラッシュに読み替えてね


aws-sam-sample-stack
スタックの名前: aws-sam-sample-stack
スタック ID: arn:aws:cloudformation:ap-northeast-1:633064615840:stac
状況: CREATE_COMPLETE
状況の理由:
IAM ロール:
説明:
出力
リソース
イベント
2017-05-29 状況 タイプ
10:22:11 UTC+0900 CREATE_COMPLETE AWS::CloudFormation::Stac
ステージ
作成
Prod
/
/resource
/resource/{resourceId}
DELETE
GET
PUT
Stage
/
/resource
DELETE
GET
PUT
関数
Lambda 関数の作成
アクション
keyword : aws-sam-sample-stack-
フィルターの追加
関数名
説明
ランタイム
aws-sam-sample-stack-DeleteFunction-U9V23NK52MJC
Node.js 4.3
aws-sam-sample-stack-GetFunction-A2PQVX487HPU
Node.js 4.3
aws-sam-sample-stack-PutFunction-O4H4YKYOW735
Node.js 4.3
アクション
aws-sam-sample-stack-
名前
aws-sam-sample-stack-Table-F2IELPAXAQAB
概要
項目
メトリックス
アラーム
容量
項目の作成
アクション
スキャン: [テーブル] aws-sam-sample-stack-Table-F2IELP.
スキャン
[テーブル] aws-sam-sample-stack-Table-F
フィルターの追加
開始
変更のキャンセル
id

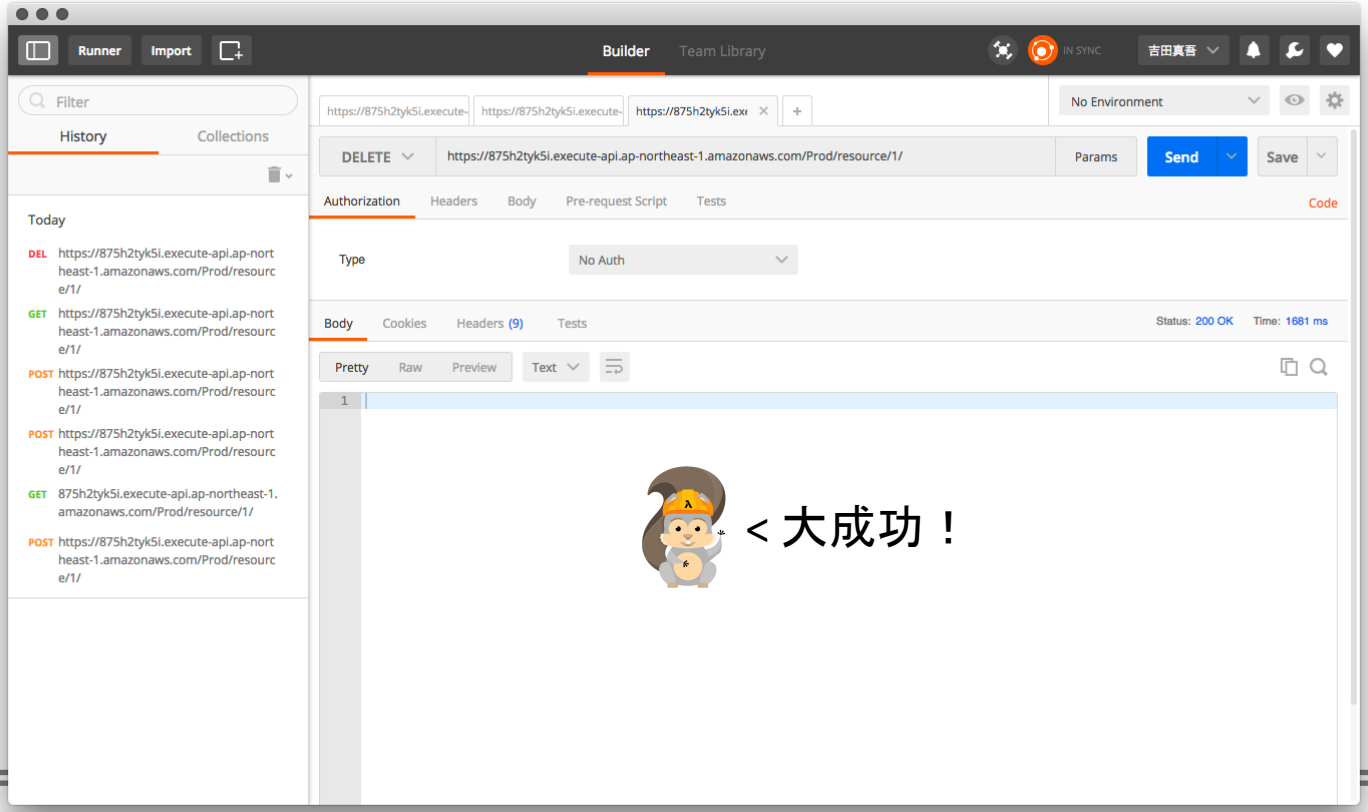
POST


Runner
Import
Builder
Team Library
IN SYNC
吉田真吾
Filter
History
Collections
Today
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod//resource/1
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/1
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1
GET https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/res
No Environment
POST
https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
Params
Send
Save
Authorization
Headers (5)
Body
Pre-request Script
Tests
Code
form-data
x-www-form-urlencoded
raw
binary
Text
1
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
< 大成功！

GET

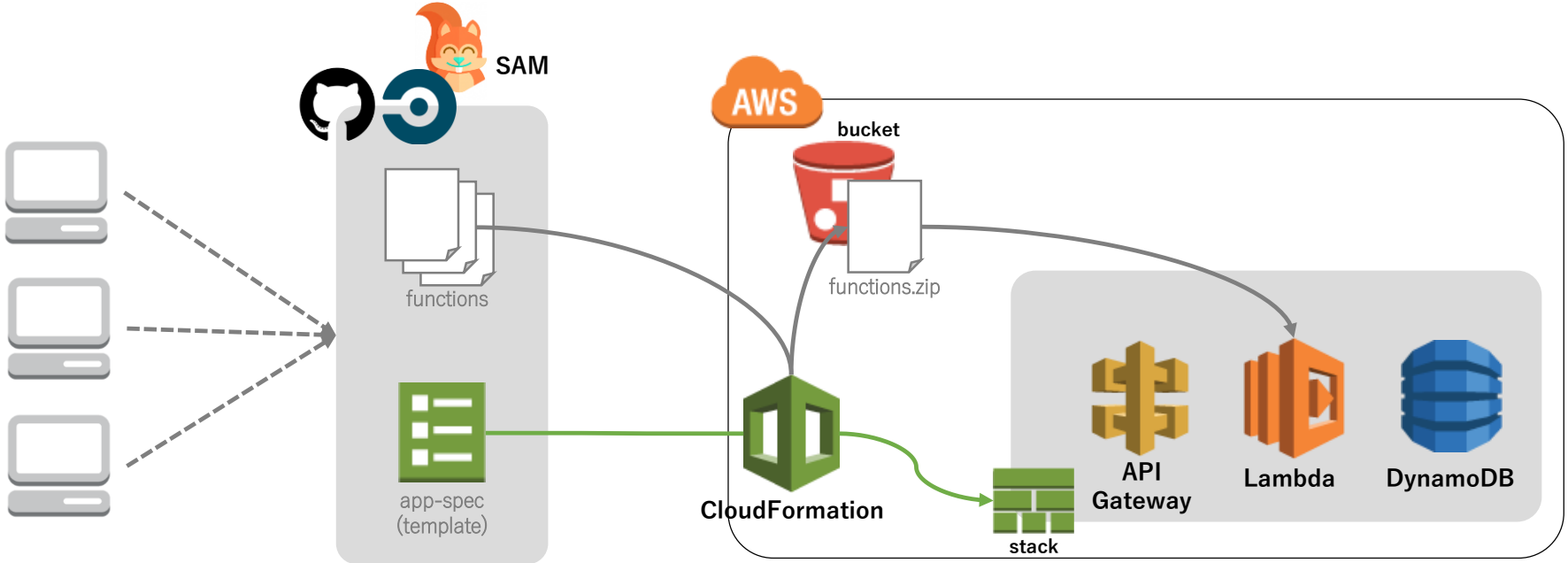

Runner
Import
Builder
Team Library
IN SYNC
吉田真吾
Filter
History
Collections
Today
GET https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod//resource/1
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/1
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/res
No Environment
GET
https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
Params
Send
Save
Key
Value
Bulk Edit
New key
value
Authorization
Headers
Body
Pre-request Script
Tests
Code
Presets
Cookies
Headers (9)
Status: 200 OK
Time: 417 ms
Pretty
Raw
Preview
Text
1
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
< 大成功！

DELETE


Runner
Import
Builder
Team Library
IN SYNC
吉田真吾
Filter
History
Collections
Today
DEL https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
GET https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
GET 875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
POST https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
No Environment
DELETE
https://875h2tyk5i.execute-api.ap-northeast-1.amazonaws.com/Prod/resource/1/
Params
Send
Save
Authorization
Headers
Body
Pre-request Script
Tests
Code
Type
No Auth
Body
Cookies
Headers (9)
Tests
Status: 200 OK
Time: 1681 ms
Pretty
Raw
Preview
Text
< 大成功！

< コマンドラインが長いので、
CI/CDパイプラインを早めに作って
おいてしまうと開発が楽だよ！

SAM+CircleCIで構築自動化



http://yoshidashingo.hatenablog.com/entry/2016/12/05/024120

circle.yml

```
# required Environment Variables
# - REGION
# - S3_BUCKET_NAME
# - STACK_NAME
machine:
  timezone: Asia/Tokyo

dependencies:
  override:
    - sudo pip install awscli
  post:
    - aws configure set region $REGION

deployment:
  production:
    branch: master
    commands:
      - zip app.zip index.js
      - aws cloudformation package --template-file app-spec.yml --output-template-file app-spec.deploy --s3-bucket $S3_BUCKET_NAME
      - aws cloudformation deploy --template-file app-spec.deploy --stack-name $STACK_NAME --capabilities CAPABILITY_IAM
```

< SAMで作れないAWSリソース(VPCやEC2,RDSなど)は
既存のCloudFormationテンプレートで構築することで
なんだってできるよ

その他

- 無変更->再デプロイした場合、AWS CLIの仕様でノンゼロ（255）が返ってくる
  https://github.com/awslabs/serverless-application-model/issues/71


the future will be
SERVERLE
SS


Section-9