

**Serverlessを使った匿名  
でGitHub Issueを立て  
るAPIを作った**

# 自己紹介

- Name : **azu**
- Twitter : [@azu\\_re](#)
- Website: [Web scratch](#), [JSer.info](#)



# やりたいこと: **API Gateway && Lambda**

- API Gatewayで POST リクエストを受け取る
- lambdaに POSTの body を渡す
- ... 何かのバリデーション ...
- lambdaで GitHub APIを叩きIssueの作成を代行する

# Serverless Framework

*SERVER*⚡*LESS*  
***FRAMEWORK***  
*VERSION 1.0*

# Serverless Framework

- API Gatewayとlambdaを管理するツール
  - AWS以外も対応してる

# Serverless Frameworkセッティング

<https://github.com/serverless/serverless/blob/master/docs/02-providers/aws/01-setup.md>

## 1. serverless-adminのIAM Userを作る

1. Serverless FrameworkはCloudFormationを使ってすべてやる

## 2. IAM UserのAPIkeyをダウンロード

## 3. ~/.aws/credentialsに profile(ここではserverless)の設定を置く

```
[serverless]
aws_access_key_id = ADFGHJKFGHJKFGHJ
aws_secret_access_key = FGAHJSJKDHAJKHSDJHASDH
```

# Service: ping を作る例

- ServerlessではAPI Gatewayとlambdaなどをまとめたserviceという単位でプロジェクトを作る
  - 1 serviceに1ディレクトリ作るイメージ

```
$ serverless create --template aws-nodejs --name ping --path ping
# ping/ に 設定が作られる

$ ls ping
event.json      handler.js      serverless.yml
```



# Serviceにprofileの設定

- まずDeployできるように `serverless-admin` のAPIキーを設定する
  - AWSでは複数のIAM Userを扱うことがよくあるので profile 単位で管理
- 先ほどの `serverless` を profile として設定する
  - `prod`と`dev`の2環境つくり `-s prod`とかで分けられるようにしておく

```
service: ping
```

```
provider:
```

```
  name: aws
```

```
  runtime: nodejs4.3
```

```
  stage: ${opt:stage, self:custom.defaultStage}
```

```
  region: ap-northeast-1
```

```
  profile: ${self:custom.profiles.${self:provider.stage}}
```

```
custom:
```

```
  defaultStage: dev # デフォルトはdev
```

```
  profiles:
```

```
    dev: serverless # -s dev
```

```
    prod: serverless # -s prod
```

```
functions:
```

```
  create:
```

```
    handler: handler.create
```

# Serviceのfunctionとpath(API Gateway)を関連づけ

```
service: ping

provider:
  name: aws
  runtime: nodejs4.3
  stage: ${opt:stage, self:custom.defaultStage}
  region: ap-northeast-1
  profile: ${self:custom.profiles.${self:provider.stage}}
custom:
  defaultStage: dev
  profiles:
    dev: serverless
    prod: serverless

functions:
  create:
    handler: handler.create # lambda
    events:
      - http:
          path: ping/create # API Gateway
          method: post
          cors: true
```

# POST /ping/create が完成！

```
functions:
  create:
    handler: handler.create # lambda
    events:
      - http:
          path: ping/create # API Gateway
          method: post
          cors: true
```

# lambdaでPost body を受け取る

```
// handler.create
module.exports.create = (event, context, cb) => {
  const body = event.body;
  if (!body) {
    return cb(new Error("No body"));
  }
  // bodyを使った処理
  // デフォルトはJSONを受け取るので オブジェクト が入ってる
};
```

## デプロイ

CloudFormationががちゃがちゃやってくれる

```
$ serverless deploy
# prod
# slsというのはalias
$ sls deploy -s prod
```

# デプロイする **API Gateway**や **lambda**ができる

手動でAPI Gatewayのロットリングは設定した方が良い

```
➤ sls deploy
Serverless: Packaging service...
Serverless: Removing old service versions...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading service .zip file to S3...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
.....
Serverless: Stack update finished...

Service Information
service: ping
stage: dev
region: ap-northeast-1
api keys:
  None
endpoints:
  POST - https://example.execute-api.ap-northeast-1.amazonaws.com/dev/ping/create
functions:
  ping-dev-create: arn:aws:lambda:ap-northeast-1:122233:function:ping-dev-create
```

# 作ったもの

- [JSer.info 300回目 && https化 && ユーザー投稿機能 - JSer.info](#)
- [jser/serverless: JSer.info serverless side](#) ソースコード

# まとめ

- Serverless 1.0(RC)は結構簡単になった
- CORSとかPOST APIが簡単
- 手動でやる場合は異常な面倒くさかった
  - CORSの設定とか手動でやってた