
aws
DEVDAY
GLOBAL SERIES

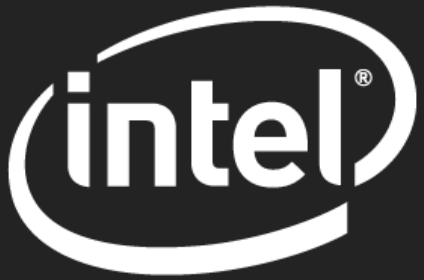
サーバレスで 王道 Web フレームワークを 使う方法

Amazon Web Services Japan K.K.
Solutions Architect, **Akihiro Tsukada**

2017.06.02

amazon web services

THANKS TO OUR FRIENDS AT:

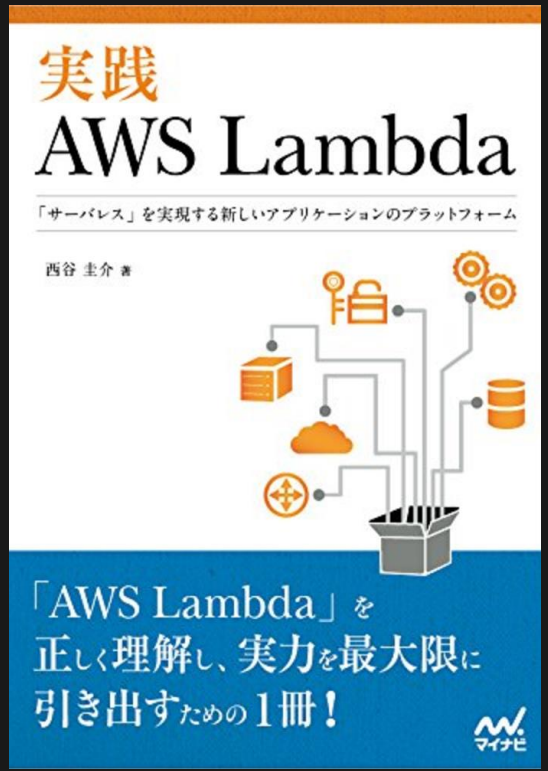


本セッションのFeedbackをお願いします

受付でお配りしたアンケートに本セッションの満足度やご感想などをご記入ください
アンケートをご提出いただきました方には、もれなく素敵なAWSオリジナルグッズをプレゼントさせていただきます

アンケートは各会場出口、パミール3FのEXPO展示会場内にて回収させて頂きます

AWS Lambdaの本が出ます



- AWS Lambdaを網羅した本を出します
- 2017年6月9日マイナビ出版より出版予定
- 3,240円（税込）
- Amazonで予約受け付け中
  http://amzn.asia/ew2WWPm
- AWS Summit 本会場のExpoエリア奥、
  マイナビ出版様ブースで先行販売中

@akitsukada
karaage.jackey

- つかだ あきひろ
- スタートアップ
  - モバイル
  - サーバレス
  - ブロックチェーン
- 健康診断全部A 😌

amazon web services

アジェンダ

- このセッションについて
- github/awslabs
- Express.js
- Spring Framework
- AWS CodeStar
- まとめ

amazon web services

このセッションについて

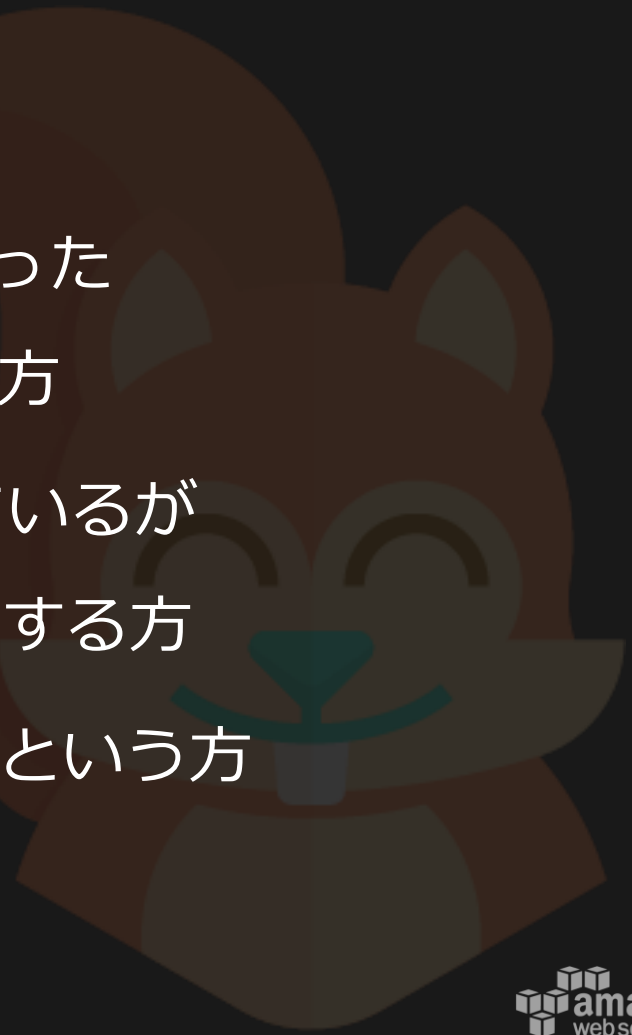
- 解決したい課題
- セッションの対象者
- このセッションで話さないこと

解決したい課題

- 「AWS Lambda と Amazon API Gateway で
  バックエンドを全部作ってしまえないものか」
- 「でも Lambda で大きい Web アプリを開発するのって
  大変なんじゃない？」
  - 多くの Lambda ファンクションをどう管理すれば？
  - デプロイや運用のベストプラクティスは？
- 「いつものフレームワークが使えれば楽なのに  」

このセッションの対象者

- 一般的な Web フレームワークを使った
  アプリケーション開発の経験がある方
- Lambda と API Gateway は知っているが
  Web アプリはいつも EC2 上に構築する方
- 便利なのはいいけど複雑なのは勘弁という方



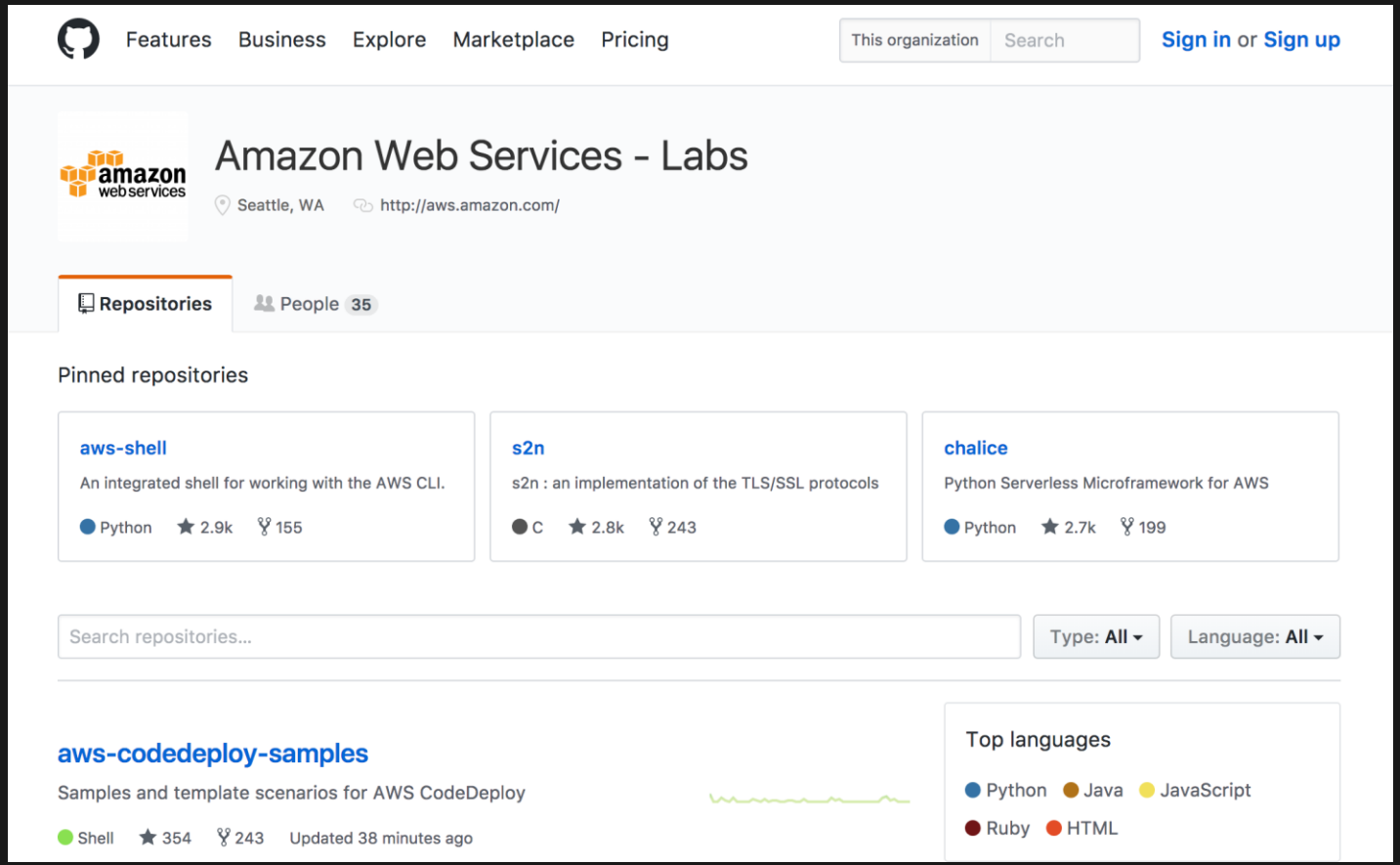
このセッションで話さないこと

- 各サービス・機能・フレームワーク等の基礎的な説明
  - AWS Lambda, Amazon API Gateway, AWS CloudFormation, Amazon S3 etc
  - Swagger, Express.js, Spring Framework
- 継続的な運用、テスト、CI/CDパイプラインの詳細
  - 本日の各セッションを参照のこと

github/awslabs

amazon web services

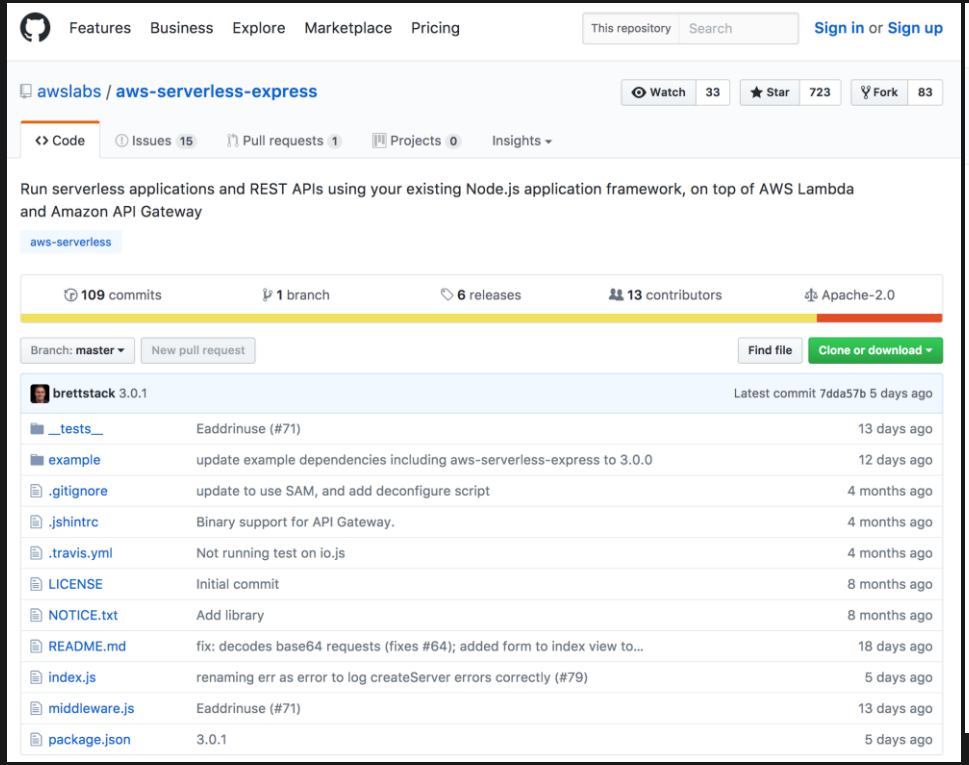
https://github.com/awslabs

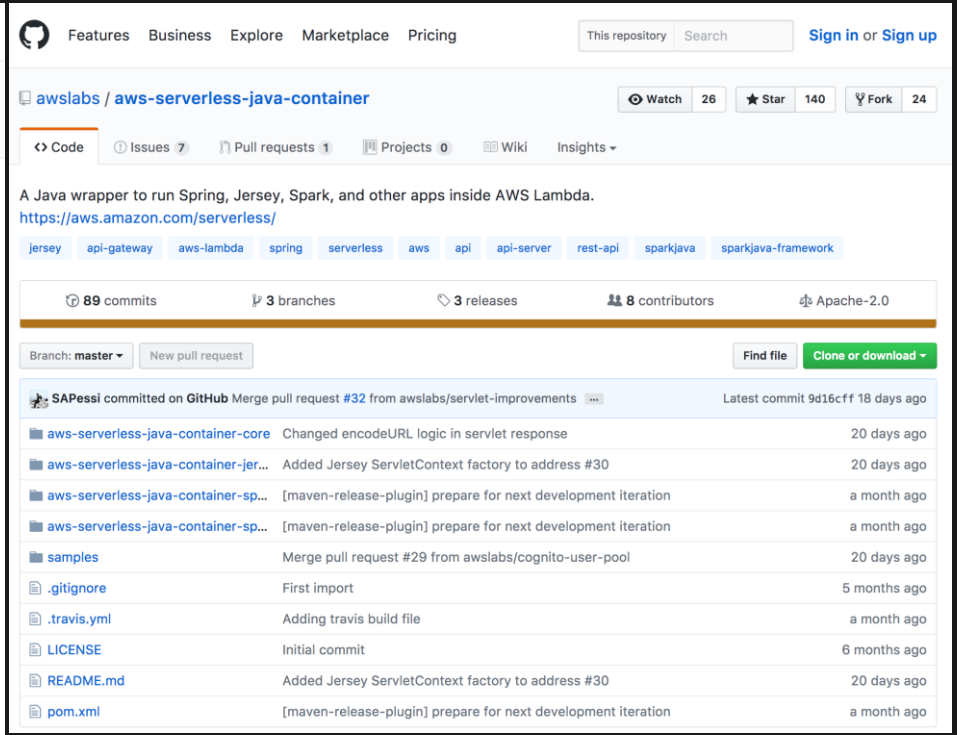


https://github.com/awslabs

- AWS が実験的・先進的なお役立ちツール、サンプル、デベロッパープレビューなライブラリなどを OSS として開発・公開している GitHub リポジトリ群
  - 2017年6月現在300強のリポジトリが存在
  - 対して github/aws はより Fixed なものの置き場所

Express.js と Java Container も awslabs に





aws-serverless-express

aws-serverless-java-container

amazon web services

awslabs にあるリポジトリ例

- https://github.com/awslabs/git-secrets
  - AWS Access Key や ~/.aws/credentials ファイルの誤コミットを防止
- https://github.com/awslabs/s2n
  - シンプルで高速な TLS/SSL 実装
- https://github.com/awslabs/aws-{platform}-sample
  - 各言語やフレームワークなどの実装サンプル集
  - swift, android, machinelearning, serverless…
- https://github.com/awslabs/chalice
  - 非常にコンパクトな Python 製の Serverless フレームワーク

Express.js
on Serverless

How to Get Started
始め方を知る

How It Works
動作原理を知る

amazon web services

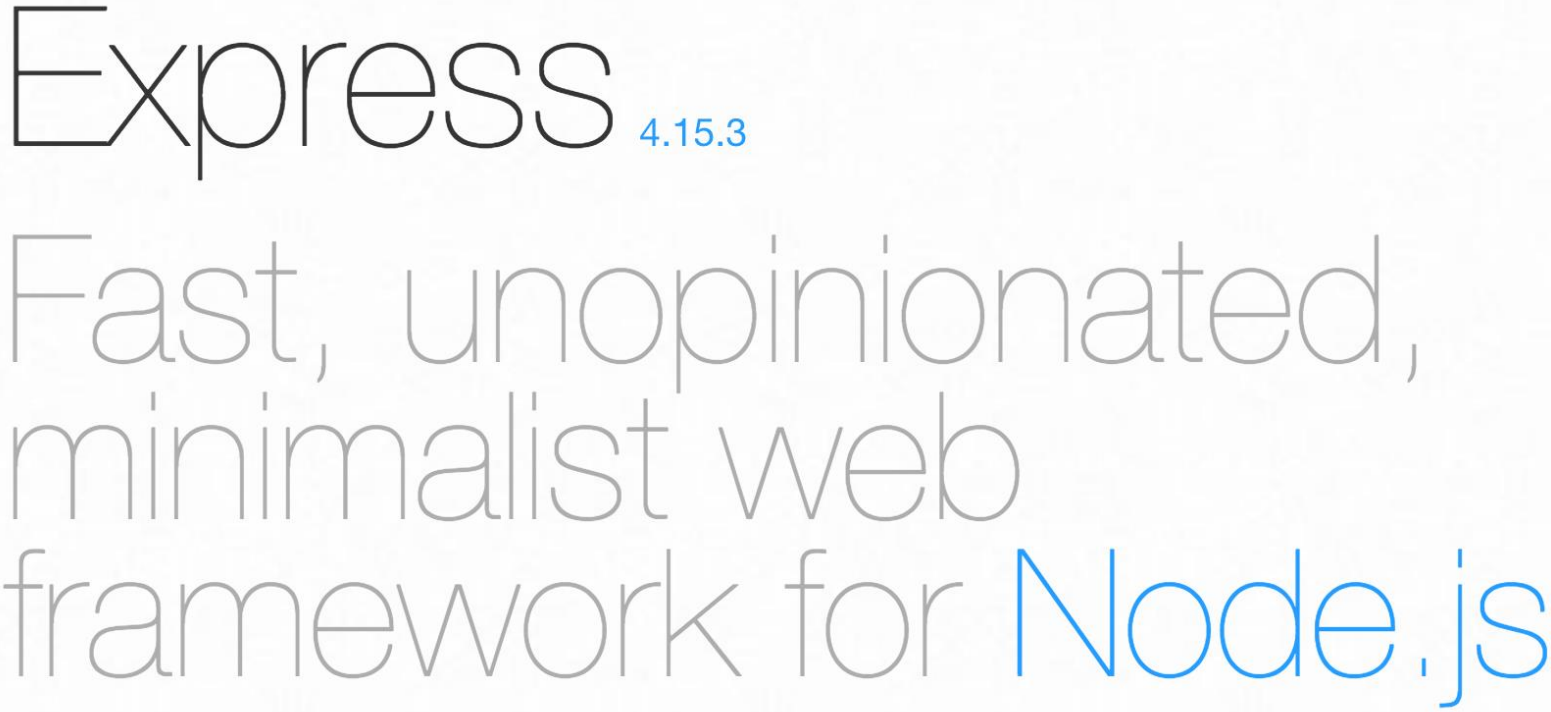
Express.js
on Serverless

How to Get Started
始め方を知る

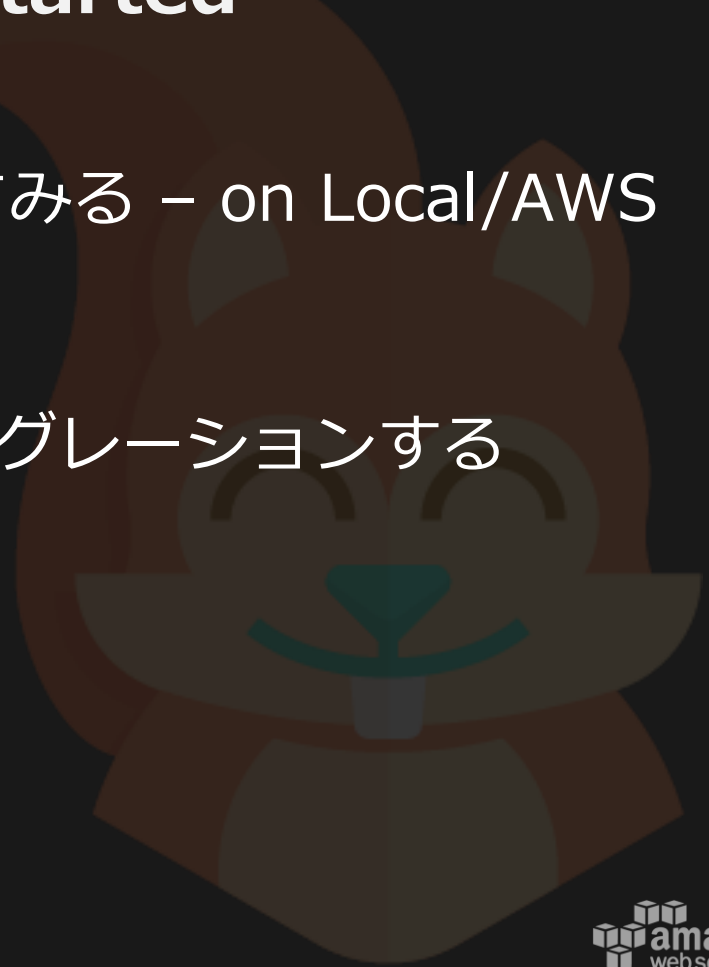
How It Works
動作原理を知る

amazon web services

https://expressjs.com/



Express.js – How to Get Started



- awslabs の example を走らせてみる – on Local/AWS
- 既存の Express プロジェクトを
  aws-serverless-express にマイグレーションする

amazon web services

Express.js – 1. example を走らせてみる

- ソースコードの取得

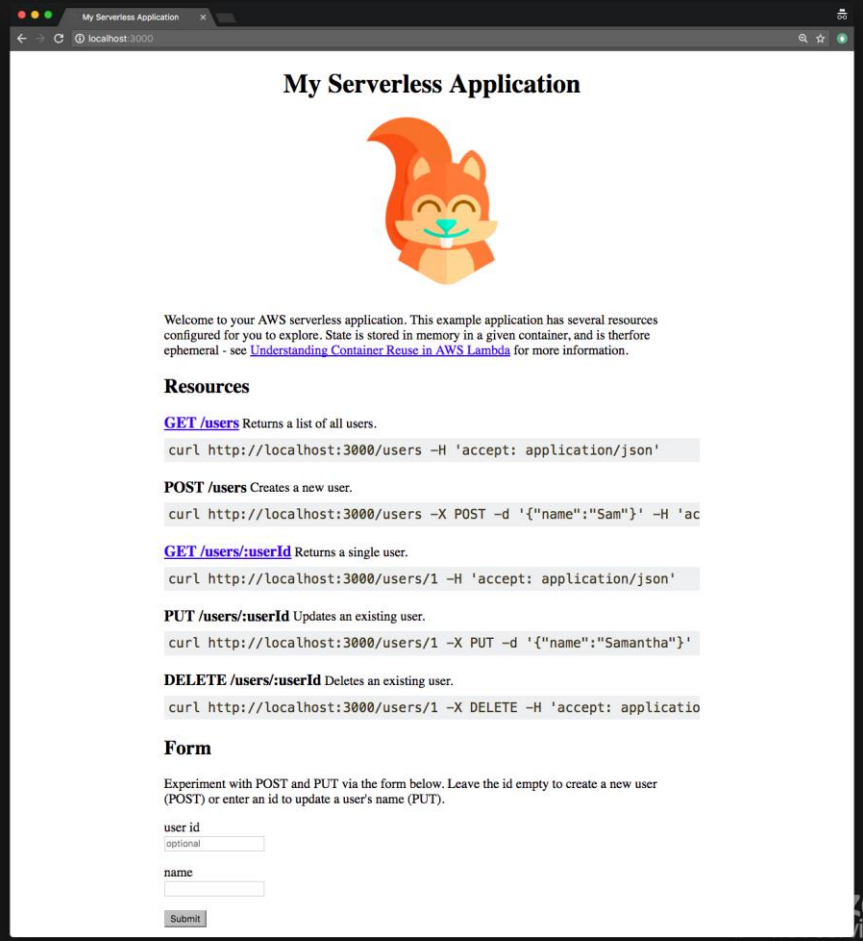
```
$ git clone https://github.com/awslabs/aws-serverless-express
$ cd aws-serverless-express/example
$ ls -1
README.md
api-gateway-event.json
app.js
app.local.js
cloudformation.yaml
lambda.jspackage.json
sam-logo.png
scripts
simple-proxy-api.yaml
vanilla-server.js
views
```

Express.js – 1. example を走らせてみる

- ローカルで実行

```
$ npm install && npm run start
(snip)
listening on http://localhost:3000
```

http://localhost:3000 =>



※ ローカルで開発したい場合は
Node.js >= 4 が必要

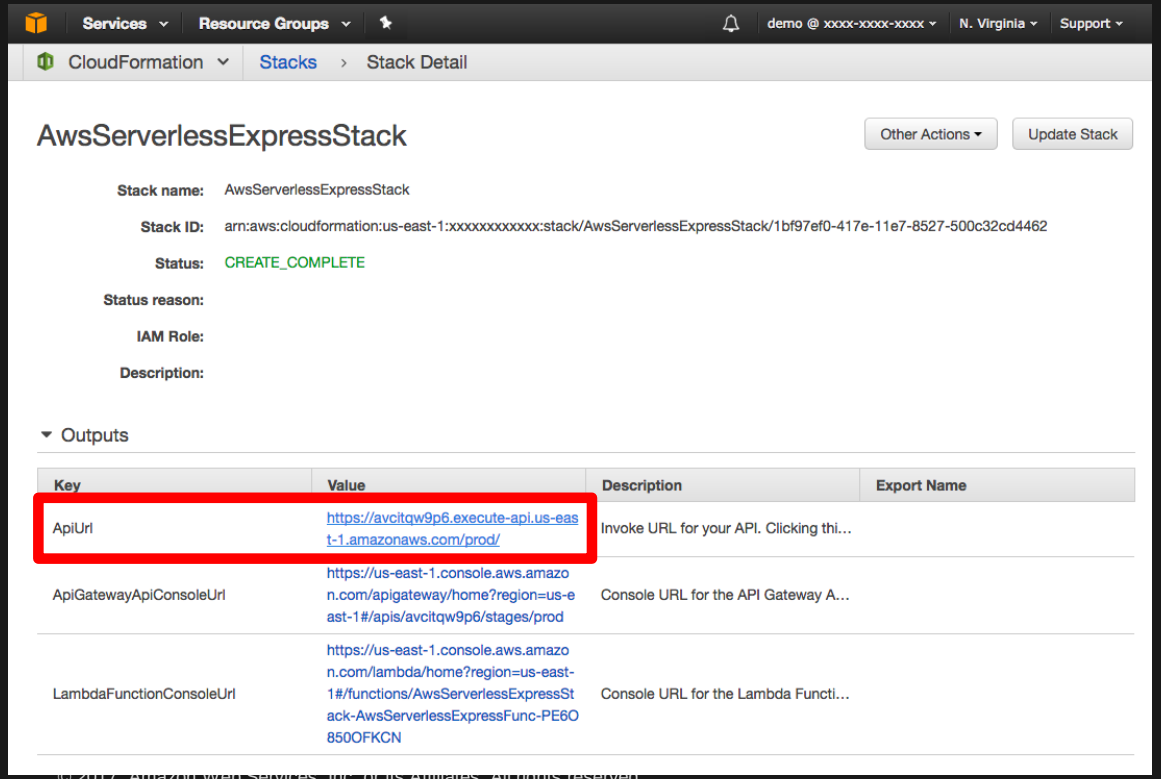
Express.js – 1. example を走らせてみる

- AWS上で実行

```
$ npm run config -- --account-id="<accountId>" ¥
  --bucket-name="<bucketName>" ¥
  [--region="<region>" --function-name="<functionName>"]
# region, function-name を 省略すると
# us-east-1, AwsServerlessExpressFunction になる

$ npm run setup #  Windowsの場合 npm run win-setup
(snip) Successfully created/updated stack – AwsServerlessExpressStack

# Tips:
# "ZIP does not support timestamps before 1980" なエラーが出たときは
# find . -mtime +1200 -exec touch {} ¥; のように回避できる
```

Express.js – 1. example を走らせてみる

- デプロイされた URL を確認して開く



CloudFormation Home

↓

AwsServerlessExpressStack

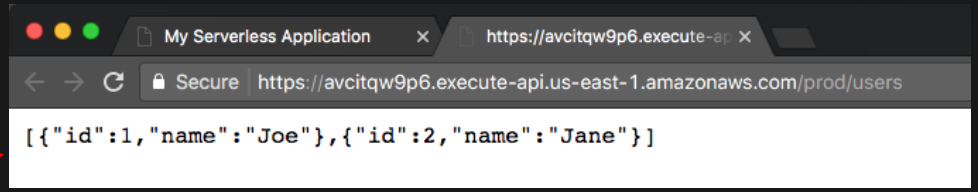
↓

Outputs

↓

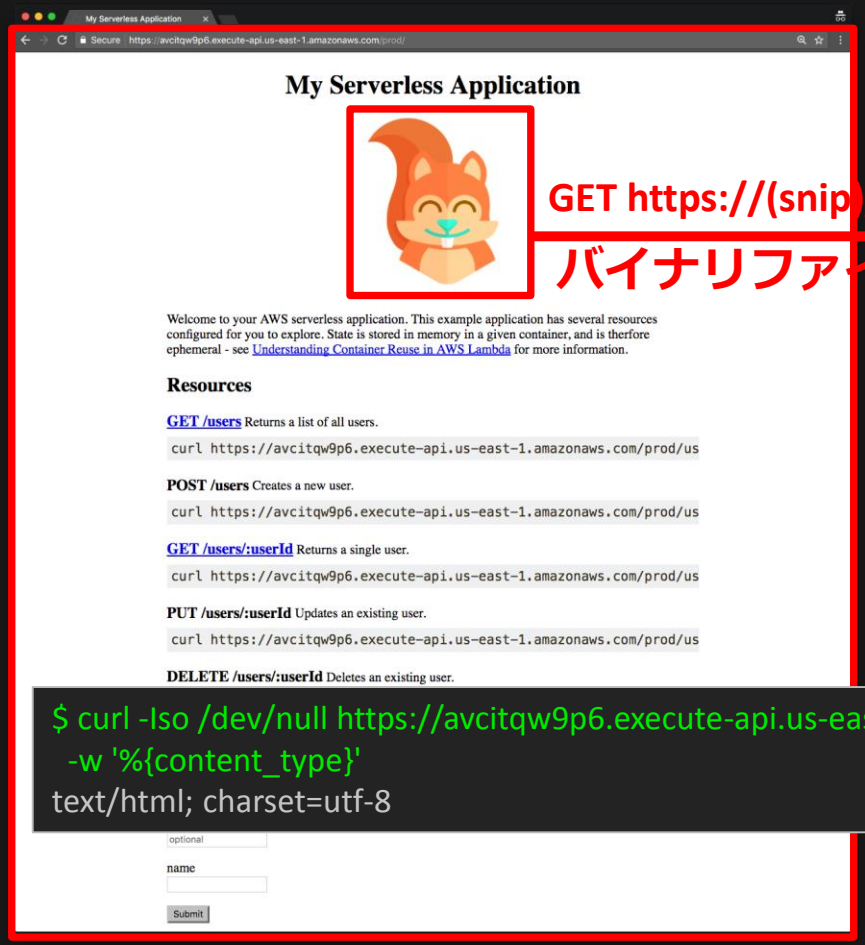
ApiUrl を選択

Express.js – 1. example を走らせてみる



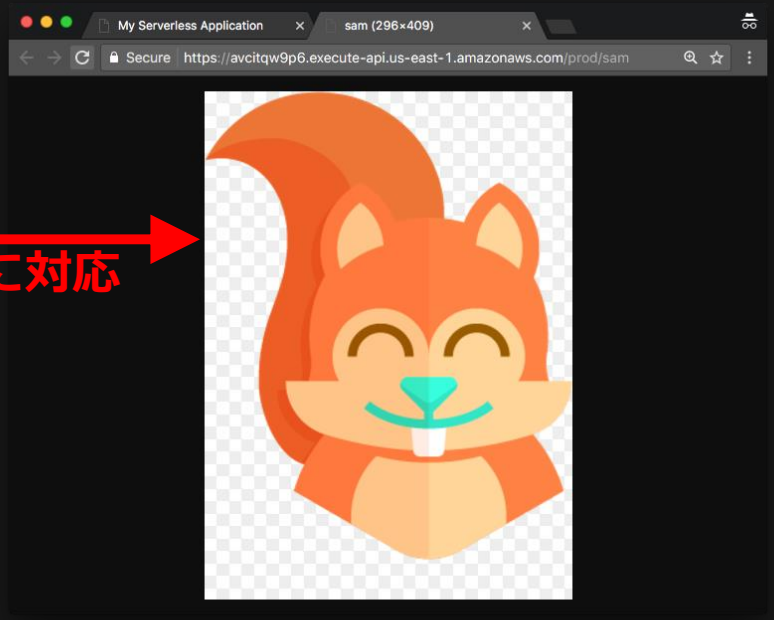

```
$ curl https://avcitqw9p6.execute-api.us-east-1.amazonaws.com/prod/users -H ¥
  'accept: application/json'
[{"id":1,"name":"Joe"},{"id":2,"name":"Jane"}]
```

Express.js – 1. example を走らせてみる



GET https://(snip)/sam

バイナリファイルに対応




```
$ curl -Iso /dev/null https://avcitqw9p6.execute-api.us-east-1.amazonaws.com/prod ¥
  -w '%{content_type}'
text/html; charset=utf-8
```

HTMLに対応

amazon web services

Express.js – How to Get Started

- ~~awslabs の example を走らせてみる – on Local/AWS~~
- 既存の Express プロジェクトを
  aws-serverless-express にマイグレーションする

amazon web services

Express.js – 2. 既存プロジェクトのマイグレート

1. example から必要なファイルをコピー

```
$ cp –R ¥
  api-gateway-event.json ¥ # ローカルテスト用のモックデータ
  cloudformation.yaml    ¥ # CFn テンプレート
  lambda.js              ¥ # Lambda ファンクションの handler
  simple-proxy-api.yaml  ¥ # Swagger の API GW 定義ファイル
  scripts                ¥ # 便利なスクリプト群
  $existing_express_dir    # 既存プロジェクトのディレクトリ
```

2. example の package.json から既存プロジェクトの package.jsonへ、scripts と config の記述をコピー

参考 https://github.com/awslabs/aws-serverless-express/tree/master/example

amazon web services

Express.js – 2. 既存プロジェクトのマイグレート

3. (未実行の場合) 設定スクリプトを実行

```
$ npm run config -- --account-id="<accountId>" ¥
  --bucket-name="<bucketName>" ¥
  [--region="<region>" --function-name="<functionName>"]
```

4. npm install の実行

```
$ npm install --save aws-serverless-express
```

5. デプロイの実行

```
$ npm run package-deploy
# Windows の場合 npm run win-package-deploy
```

参考 https://github.com/awslabs/aws-serverless-express/tree/master/example

amazon web services

Express.js – 2. 既存プロジェクトのマイグレート

Tips (1/2)

- 既存プロジェクトのメインファイル名が app.js でない場合、lamda.js の require を既存プロジェクトのファイル名に合わせる
  - 例: require('./app') → require('./server')
- ソースコードをトップ以外のディレクトリに配置している場合、cloudformation.yml 内の CodeUri で指定する
  - 例: CodeUri: ./ → CodeUri: ./src
  - Webpack 等の build ツールを使っている場合はそのアウトプットディレクトリを指定

Express.js – 2. 既存プロジェクトのマイグレート

Tips (2/2)

- ローカルの Node サーバで Lambda のテストをするには
  npm run local を実行
  - api-gateway-event.json のデータが event として渡される
  - Lambda とローカルの Node.js バージョンを合わせることを推奨
- API Gateway の構成を変更したい場合は
  simple-proxy-api.yaml を、
  AWS リソースの構成を変更したい場合は
  cloudformation.yaml を編集して
  npm run package-deploy を実行

amazon web services

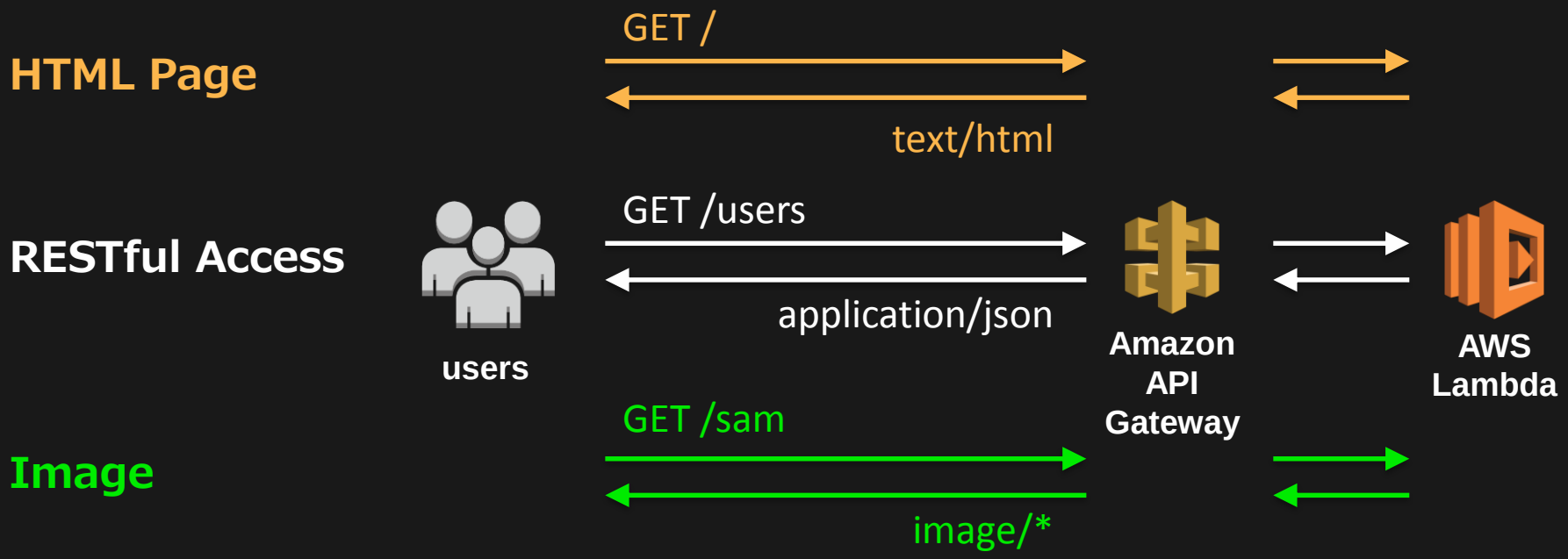
Express.js
on Serverless

✓ How to Get Started
始め方を知る

 How It Works
動作原理を知る

amazon web services

Express.js – aws-serverless-express の振る舞い



amazon web services

Express.js – aws-serverless-express の構造

/
ANY
OPTIONS
/{proxy+}
ANY
OPTIONS

☑ Lambda Integration Proxy

Binary media: */*

lambda.js (handler)

app.js

```
app.get('/', (req, res) => {
  res.render('index', {
    (snip)
  })
})

app.get('/sam', (req, res) => {
  res.sendFile(
    `${__dirname}/sam-logo.png`)
})

app.get('/users', (req, res) =>
{
  res.json(users)
})
```

Express.js – aws-serverless-express の構造

/ に対する
すべての
メソッド、
/* に対する
すべての
メソッド

/
ANY
OPTIONS
/{proxy+}
ANY
OPTIONS

☑ Lambda Integration Proxy

Binary media:
/

lambda.js
(handler)

app.js

```
app.get('/', (req, res) => {
  res.render('index', {
    (snip)
  })
})

app.get('/sam', (req, res) => {
  res.sendFile(
    `${__dirname}/sam-logo.png`)
})

app.get('/users', (req, res) =>
{
  res.json(users)
})
```

Express.js – aws-serverless-express の構造

レスポンスヘッダはLambdaで生成する

/
ANY
OPTIONS
/{proxy+}
ANY
OPTIONS

☑ Lambda Integration Proxy

Binary media: */*

lambda.js (handler)

app.js

```
app.get('/', (req, res) => {
  res.render('index', {
    (snip)
  })
})

app.get('/sam', (req, res) => {
  res.sendFile(
    `${__dirname}/sam-logo.png`)
})

app.get('/users', (req, res) =>
{
  res.json(users)
})
```

Express.js – aws-serverless-express の構造

/
ANY
OPTIONS
/{proxy+}
ANY
OPTIONS

☑ Lambda Integration Proxy

Binary media: */*

画像ファイルをレスポンスできるように

lambda.js (handler)

app.js

```
app.get('/', (req, res) => {
  res.render('index', {
    (snip)
  })
})

app.get('/sam', (req, res) => {
  res.sendFile(
    `${__dirname}/sam-logo.png`)
})

app.get('/users', (req, res) =>
{
  res.json(users)
})
```

Express.js – aws-serverless-express の構造

/
ANY
OPTIONS

Binary media:
/

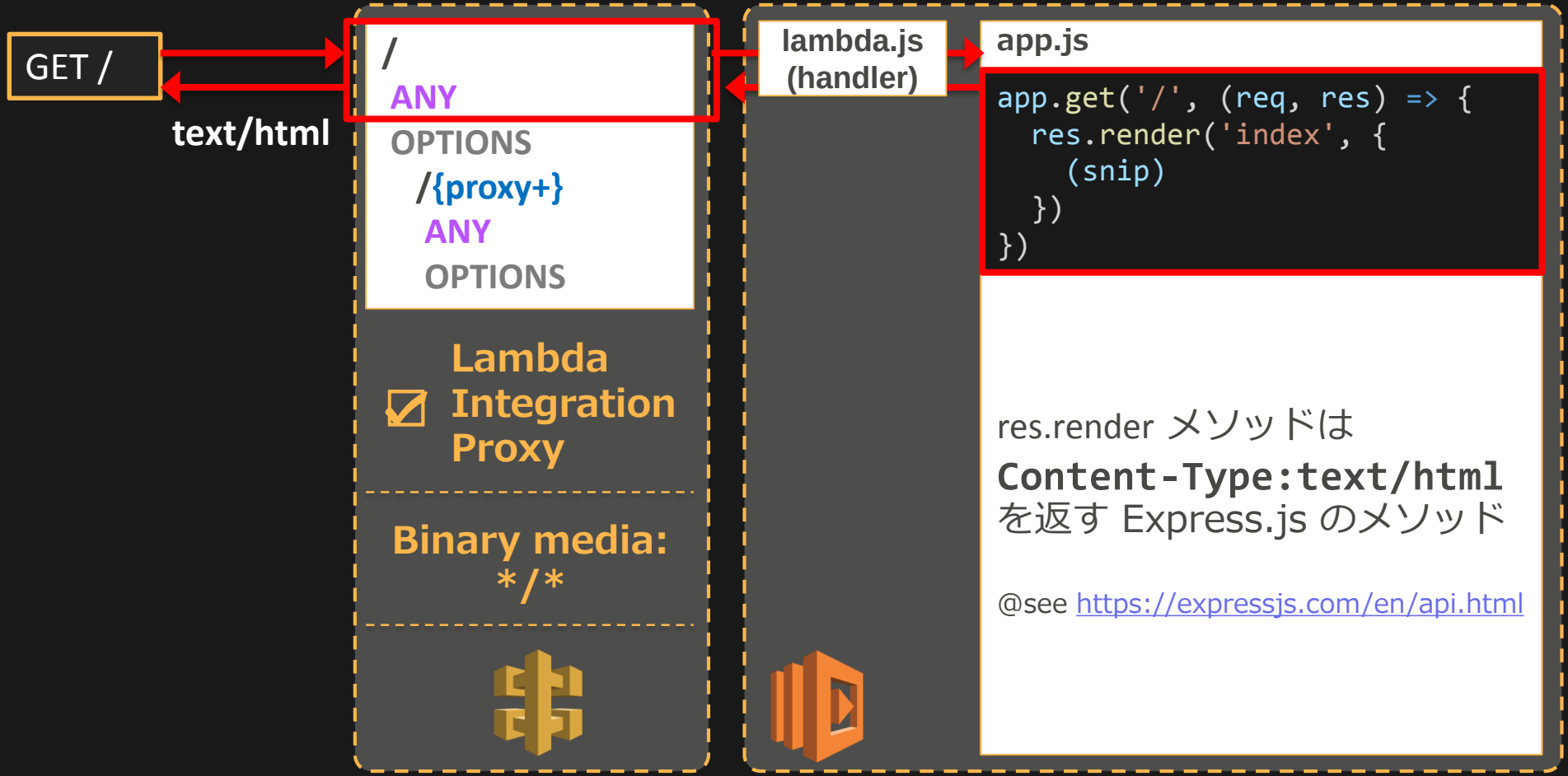
複数の .js ファイルを
含む一つの Lambda
ファンクション

lambda.js
(handler)

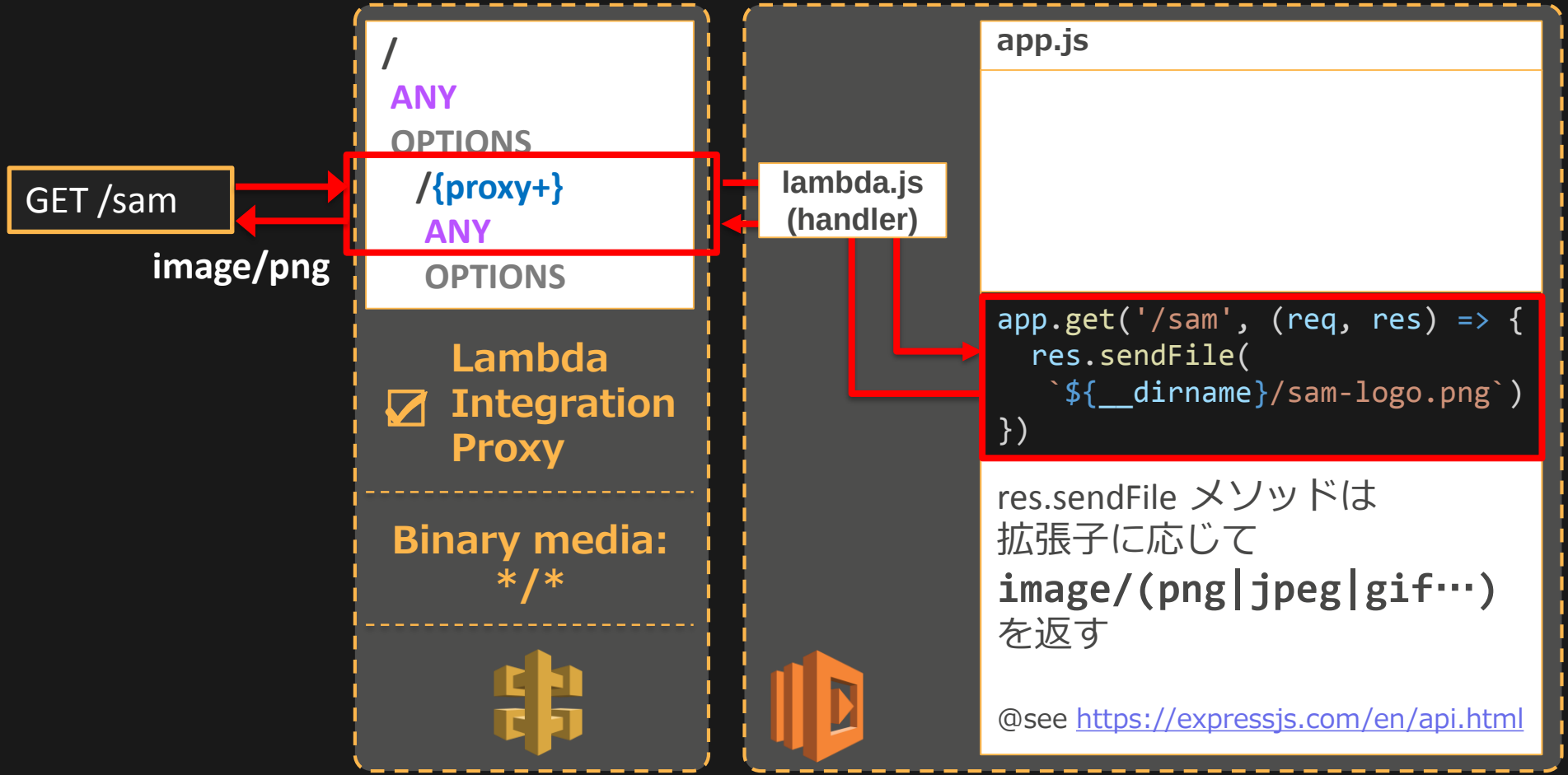
app.js

```
app.get('/', (req, res) => {
  res.render('index', {
    (snip)
  })
})

app.get('/sam', (req, res) => {
  res.sendFile(
   `${__dirname}/sam-logo.png`)
})

app.get('/users', (req, res) =>
{
  res.json(users)
})
```

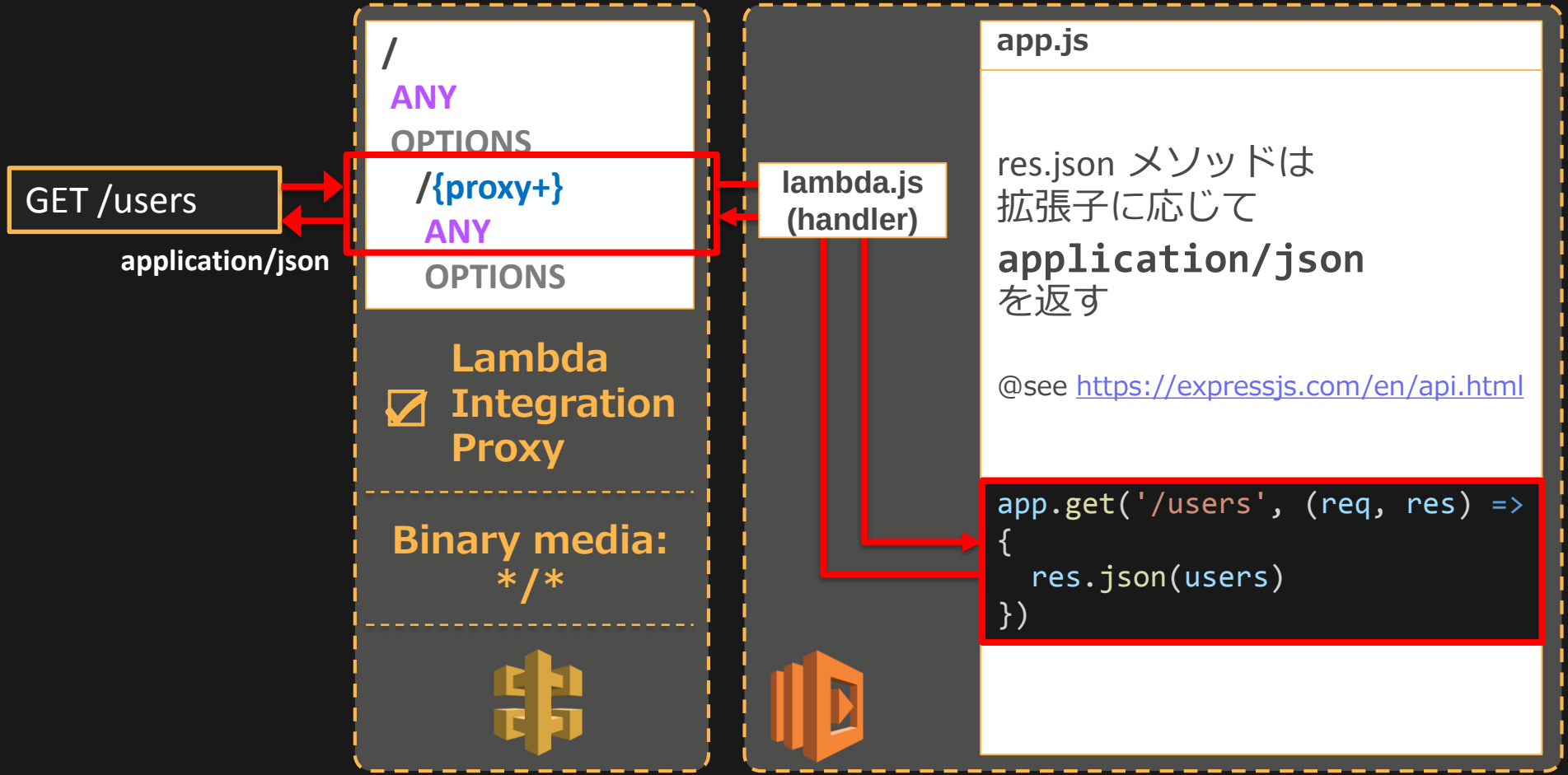
Express.js – GET / のシーケンス(HTML)



Express.js – GET /sam のシーケンス(Image)

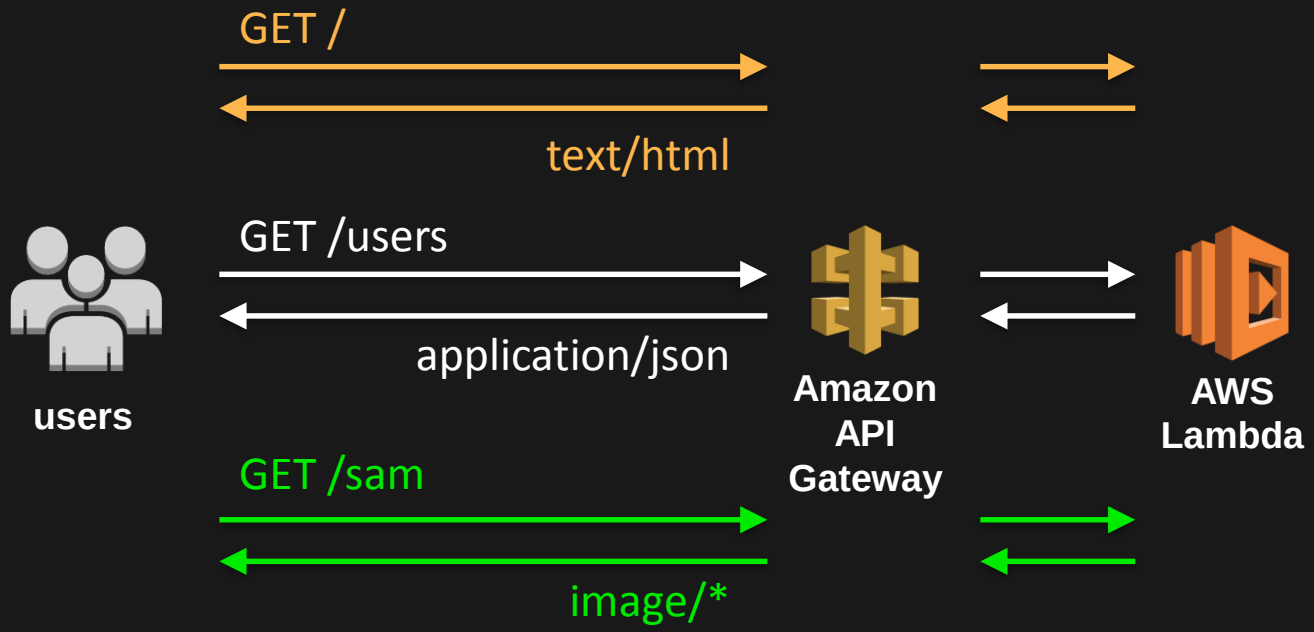


Express.js – GET /users のシーケンス(RESTful API)



Express.js – aws-serverless-express の振る舞い

- ✔ HTML Page
- ✔ RESTful Access
- ✔ Image



amazon web services

Express.js
on Serverless

✓ **How to Get Started**
始め方を知る

✓ **How It Works**
動作原理を知る

amazon web services

Spring Framework

- **How to Get Started**
  始め方を知る
- **How It Works**
  動作原理を知る

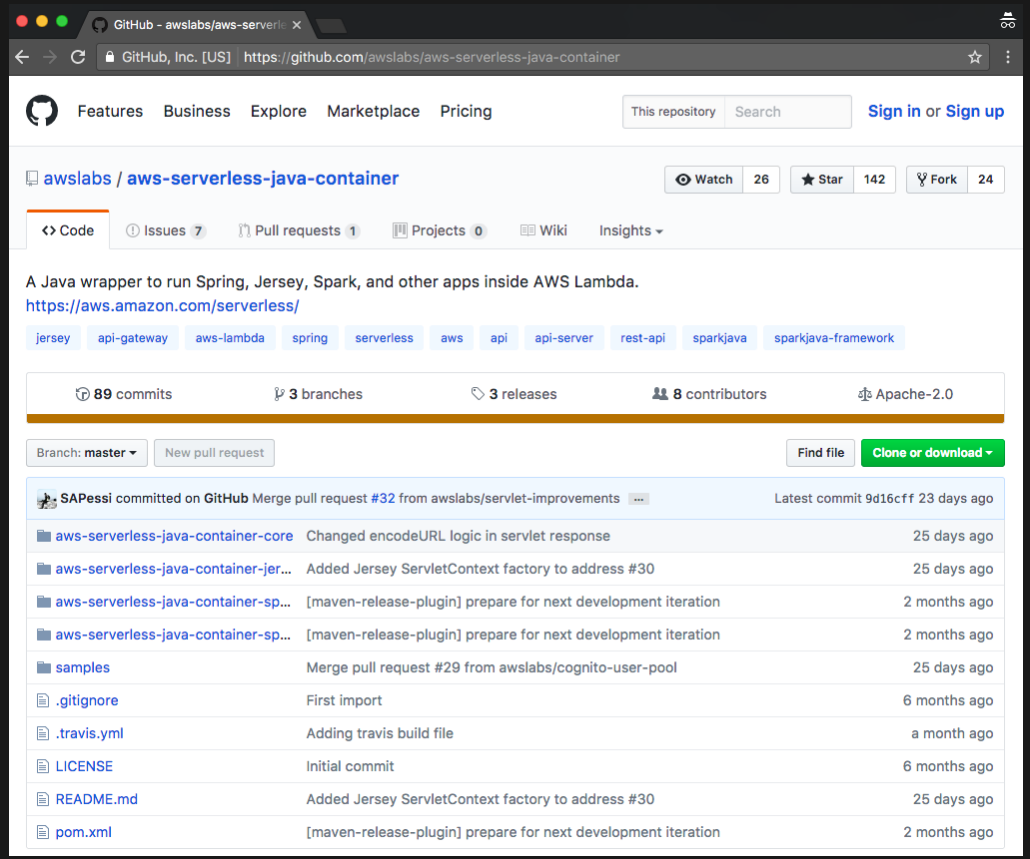
amazon web services

Spring Framework

- **How to Get Started**
  始め方を知る
- **How It Works**
  動作原理を知る

amazon web services

awslabs/aws-serverless-java-container



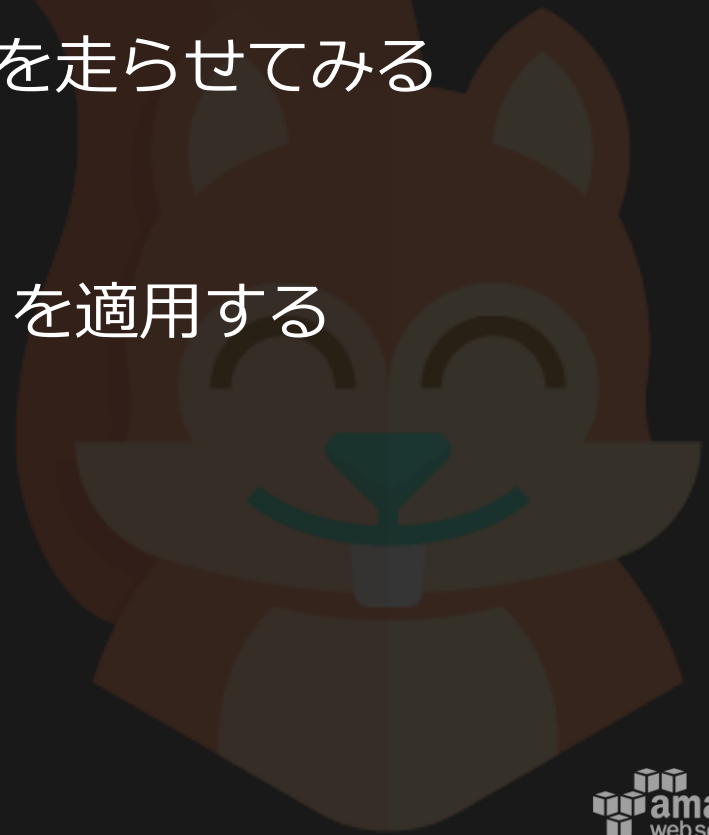
- API Gateway と Lambda で
  - Spring
  - Jersey
  - Spark
  - などを動作させるためのラッパーライブラリ

参考 https://github.com/awslabs/aws-serverless-java-container/wiki

amazon web services

Spring Framework – How to Get Started

- awslabs の sample(pet-store) を走らせてみる
- 既存の Spring プロジェクトに
  aws-serverless-java-container を適用する

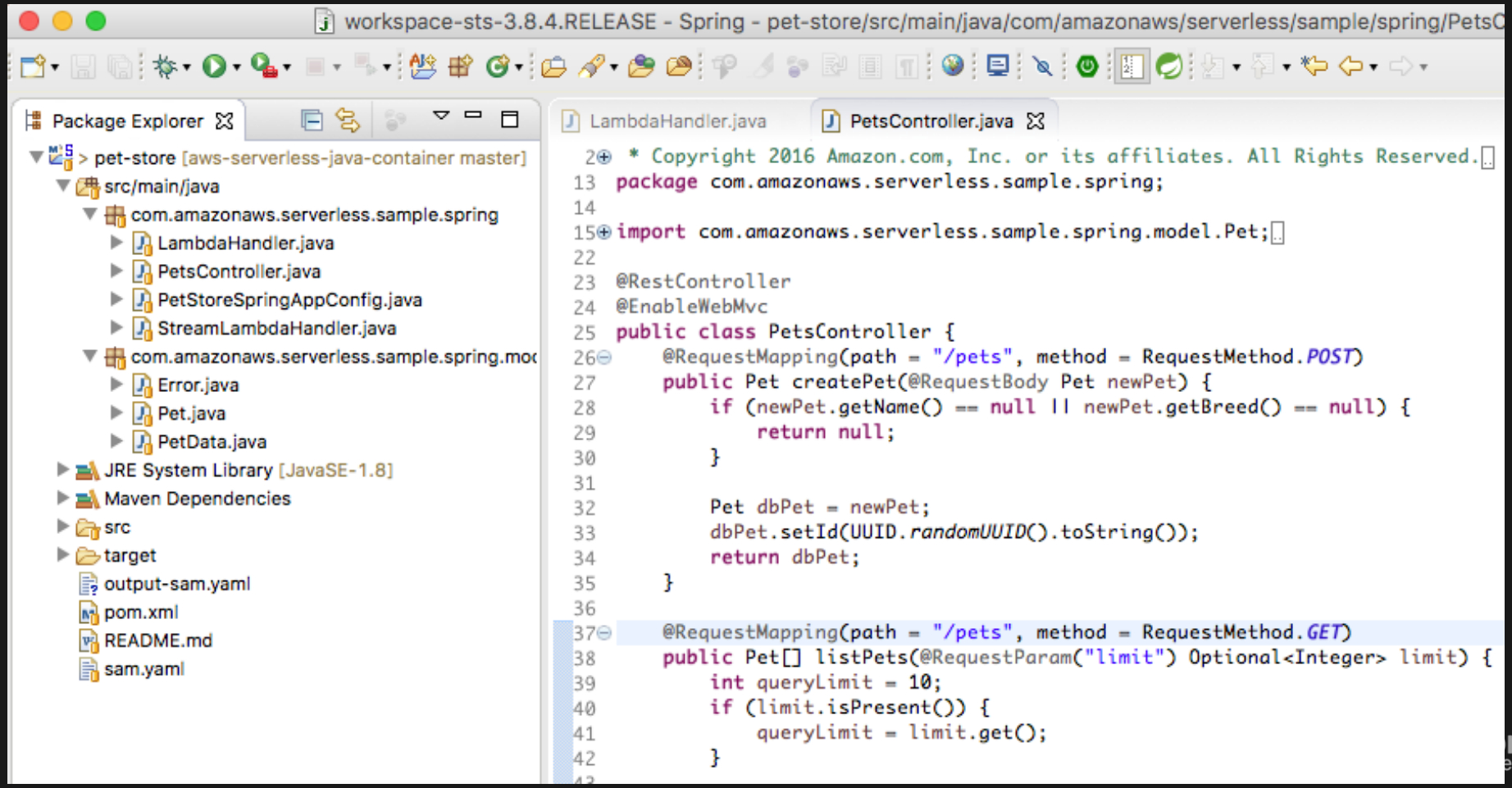


Spring Framework – pet-store on Serverless

ソースコードの取得

```
$ git clone https://github.com/awslabs/aws-serverless-java-container
$ cd aws-serverless-java-container/samples/spring/pet-store
$ ls -1
README.md
output-sam.yaml
pom.xml
sam.yaml
src
target
```

Spring Framework – pet-store on Serverless



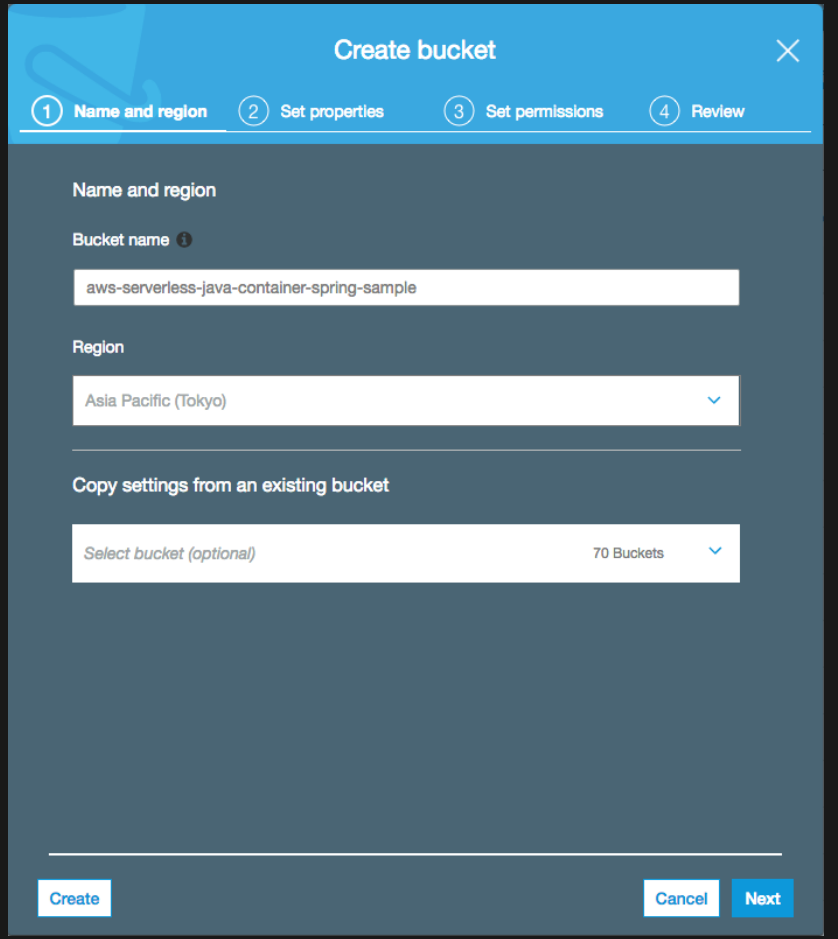
Spring Framework – pet-store on Serverless

- Maven で package を作成

```
$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------
[INFO] Building Spring example for the aws-serverless-java-container library 1.0-SNAPSHOT
...(snip)...
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 39.695 s
[INFO] Finished at: 2017-05-25T15:26:32+09:00
[INFO] Final Memory: 33M/243M
[INFO] ------------------------------------------------------------
```

Spring Framework – pet-store on Serverless

- CloudFormation の artifact を置く S3 バケットを作成



Spring Framework – pet-store on Serverless

CloudFormation package の作成

```
$ aws cloudformation package --template-file sam.yaml ¥
  --output-template-file output-sam.yaml ¥
  --s3-bucket <YOUR S3 BUCKET NAME>
Uploading to 26175c16dff8870d7dc34225dc7aa47a 7553653 / 7553653.0  (100.00%)
Successfully packaged artifacts and wrote output template to file output-sam.yaml.
Execute the following command to deploy the packaged template
aws cloudformation deploy --template-file /Path/to/output-sam.yaml --stack-name <YOUR STACK NAME>
```

デプロイ実行用のコマンドが示される

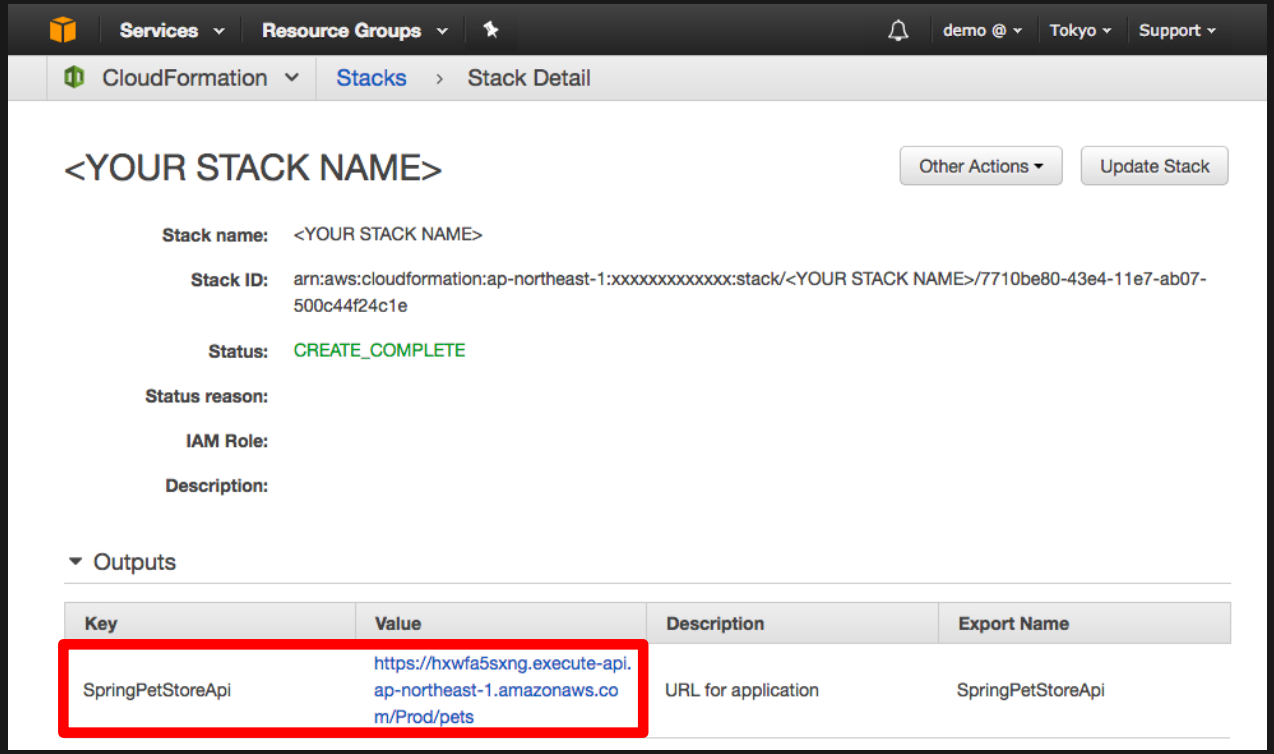
Spring Framework – pet-store on Serverless

デプロイ実行

```
$ aws --region ap-northeast-1 cloudformation deploy ¥
  --template-file /Path/to/output-sam.yaml ¥
  --stack-name <YOUR STACK NAME> [--capabilities CAPABILITY_IAM]
Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - <YOUR STACK NAME>
```

Spring Framework – pet-store on Serverless

- デプロイされた URL を確認して開く



CloudFormation Home

↓

<YOUR STACK NAME>

↓

Outputs

↓

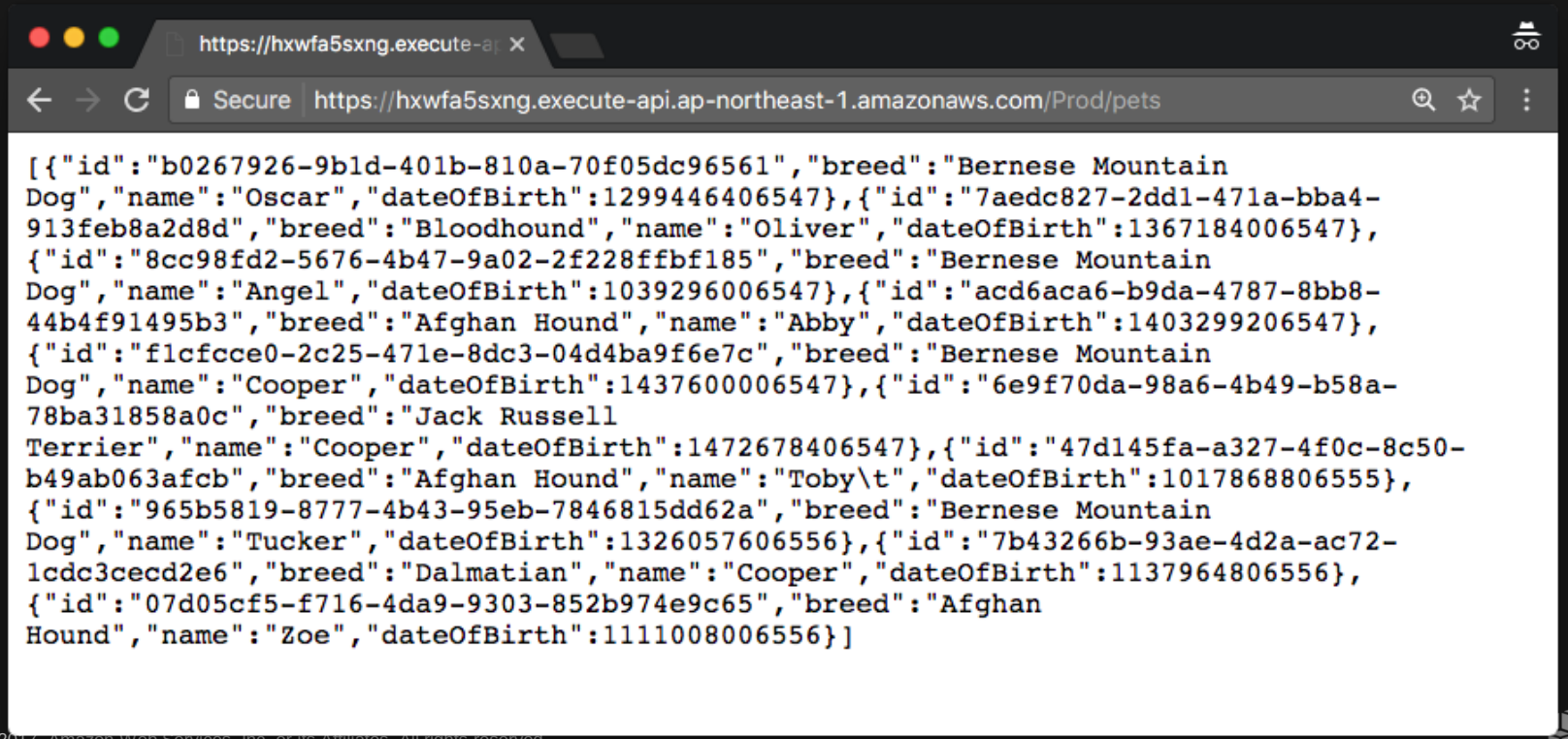
SpringPetStoreApi

を選択

amazon web services

Spring Framework – pet-store on Serverless

- デプロイされた URL を確認して開く



Spring Framework – How to Get Started

- ~~awslabs の sample(pet-store) を走らせてみる~~
- 既存の Spring プロジェクトに
  aws-serverless-java-container を適用する

amazon web services

Spring Framework – 既存プロジェクトへの適用

- 依存関係の設定 (pom.xml)

```
<dependency>
  <groupId>com.amazonaws.serverless</groupId>
  <artifactId>aws-serverless-java-container-spring</artifactId>
  <version>LATEST</version>
</dependency>
```

@see https://github.com/awslabs/aws-serverless-java-container/wiki/Quick-start---Spring

Spring Framework – 既存プロジェクトへの適用

- LambdaHandler の作成

```
public class LambdaHandler implements
                            RequestHandler<AwsProxyRequest, AwsProxyResponse> {
    private SpringLambdaContainerHandler<AwsProxyRequest, AwsProxyResponse> handler;

    public AwsProxyResponse handleRequest(AwsProxyRequest awsProxyRequest,
                                                        Context context) {
        if (handler == null) {
            try {
                handler = SpringLambdaContainerHandler.
                                getAwsProxyHandler(PetStoreSpringAppConfig.class);
            } catch (ContainerInitializationException e) {
                e.printStackTrace();
                return null;
            }
        }
        return handler.proxy(awsProxyRequest, context);
    }
}
```

@see https://github.com/awslabs/aws-serverless-java-container/wiki/Quick-start---Spring

Spring Framework – 既存プロジェクトへの適用

- Lambda ファンクションをパッケージングしてデプロイ
  - AWS Documentation "デプロイパッケージの作成 (Java)" を参照のこと
    http://docs.aws.amazon.com/ja_jp/lambda/latest/dg/lambda-java-how-to-create-deployment-package.html
- API Gateway、CloudFormation（AWS SAM）の作成や設定は pet-store を参考にすることをオススメ

@see https://github.com/awslabs/aws-serverless-java-container/wiki/Quick-start---Spring

Spring Framework – How to Get Started

- ~~awslabs の sample(pet-store) を走らせてみる~~
- ~~既存の Spring プロジェクトに aws-serverless-java-container を適用する~~

amazon web services

Spring Framework

✓ How to Get Started

始め方を知る

How It Works

動作原理を知る

amazon web services

Spring Framework – How It Works

- バックエンドの構成、原理
  → Express.js の example とあまり変わらないので割愛
  - API Gateway は /{proxy+} ANY で、 Lambda Proxy Integration で…
- awslabs/aws-serverless-java-container
  - spring, sample/pet-store の
    （ソフトウェアとしての）アーキテクチャ俯瞰
    - クラスレベル
    - パッケージレベル

amazon web services

Spring on Serverless Classes (1/2)

com.amazonaws

serverless.sample.spring

LambdaHandler
+ handleRequest(…)

Create

@Configuration,
@ComponentScan("…sample.spring")
PetStoreSpringAppConfig

Scan

@RestController, @EnableWebMvc
PetsController
@RequestMapping(POST "/pets")
+ createPet(…):Pet
@RequestMapping(GET "/pets")
+ listPets(…):Pet[]
@RequestMapping(GET "/pets/{userId}")
+ listPets():Pet

Create

services.lambda.runtime

<<Interface>>RequestHandler
+ handleRequest(…)

serverless.proxy.spring

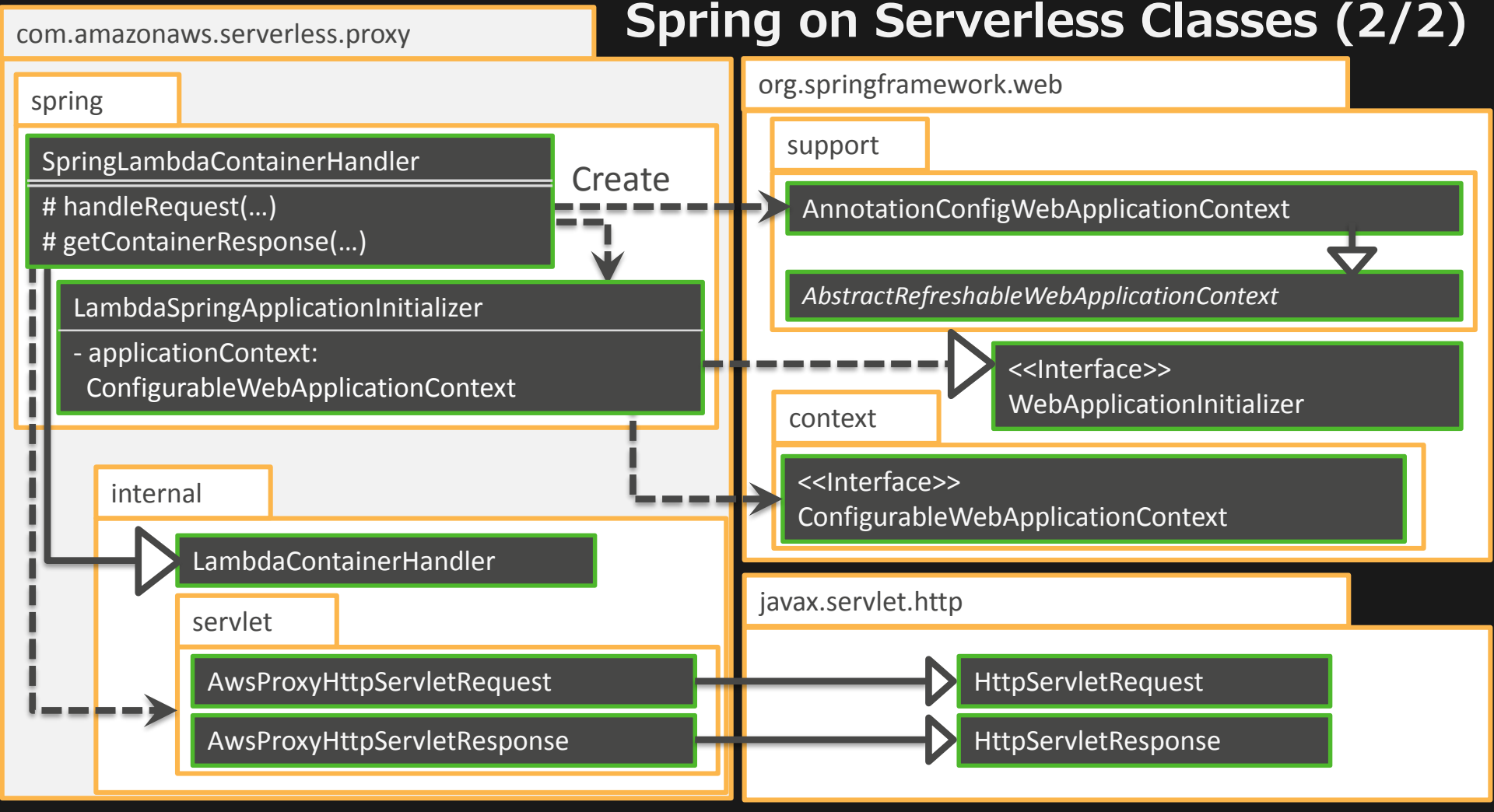
SpringLambdaContainerHandler
handleRequest(…)
getContainerResponse(…)

LambdaSpringApplicationInitializer
- applicationContext

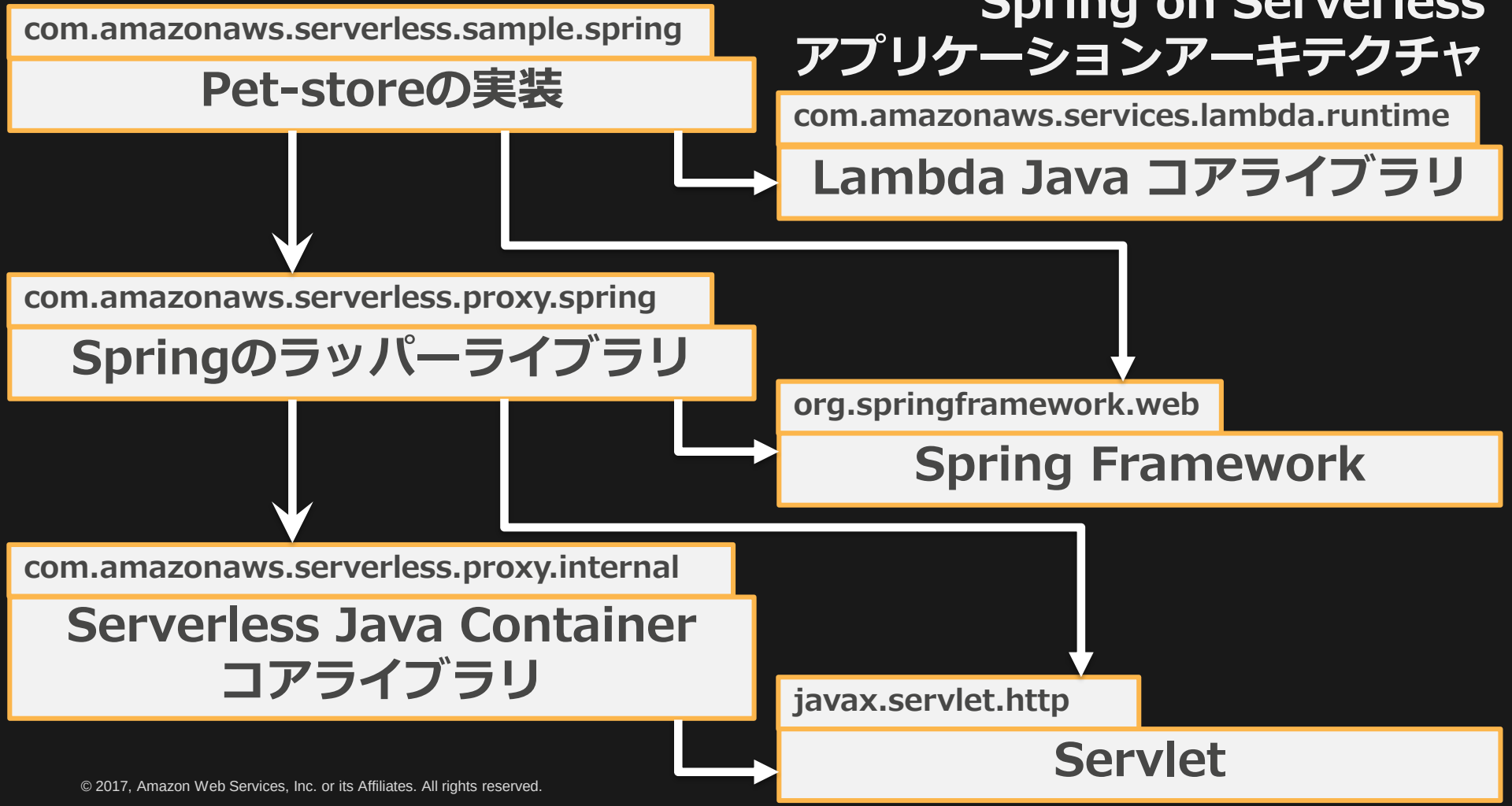
serverless.proxy.internal

LambdaContainerHandler
+ proxy(…)
 # handleRequest(…)
 # getContainerResponse(…)

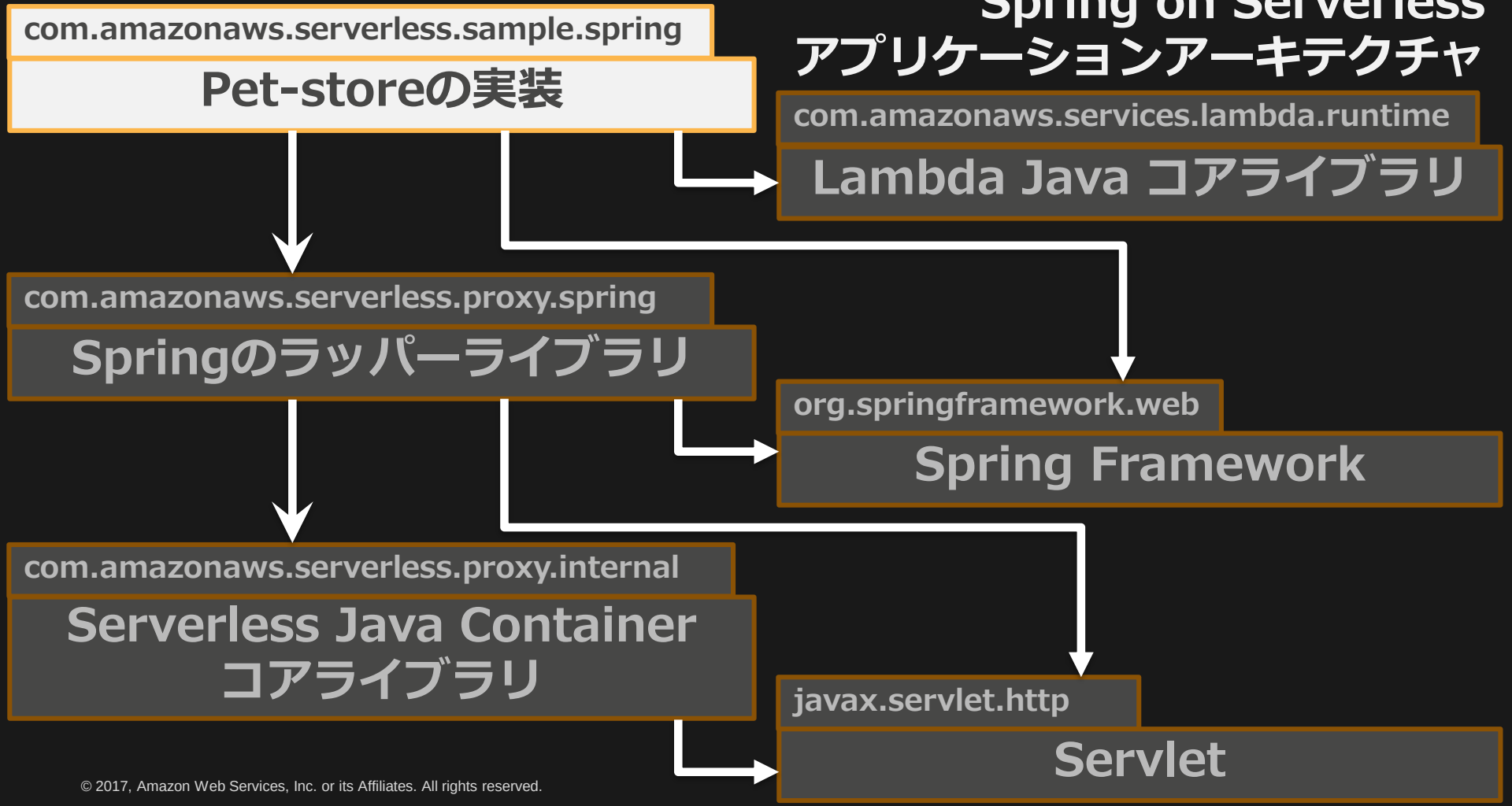
Spring on Serverless Classes (2/2)

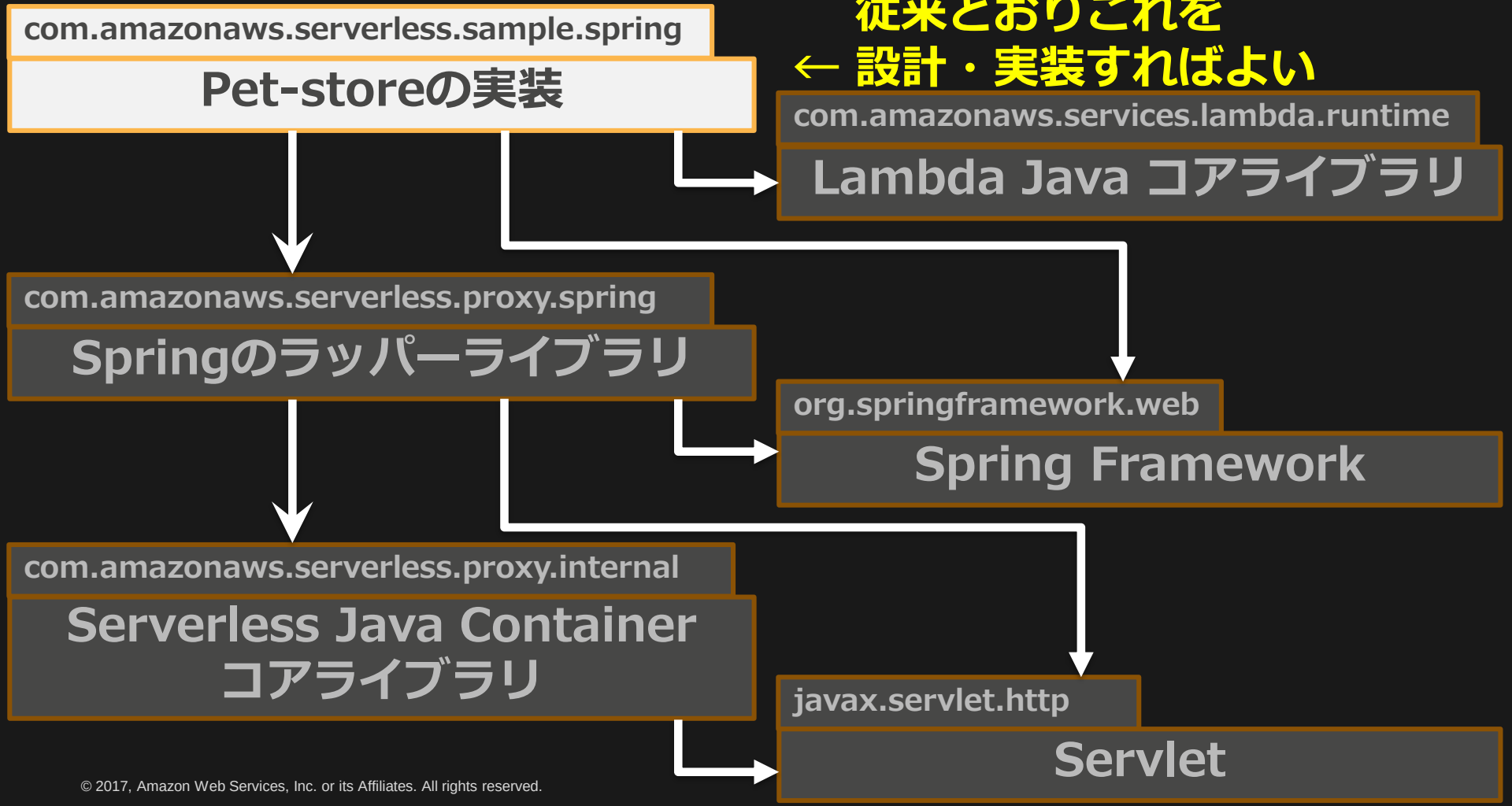


Spring on Serverless アプリケーションアーキテクチャ



Spring on Serverless
アプリケーションアーキテクチャ





アジェンダ

- ~~このセッションについて~~
- ~~github/awslabs~~
- ~~Express.js~~
- ~~Spring Framework~~
- AWS CodeStar
- まとめ

amazon web services

AWS CodeStar

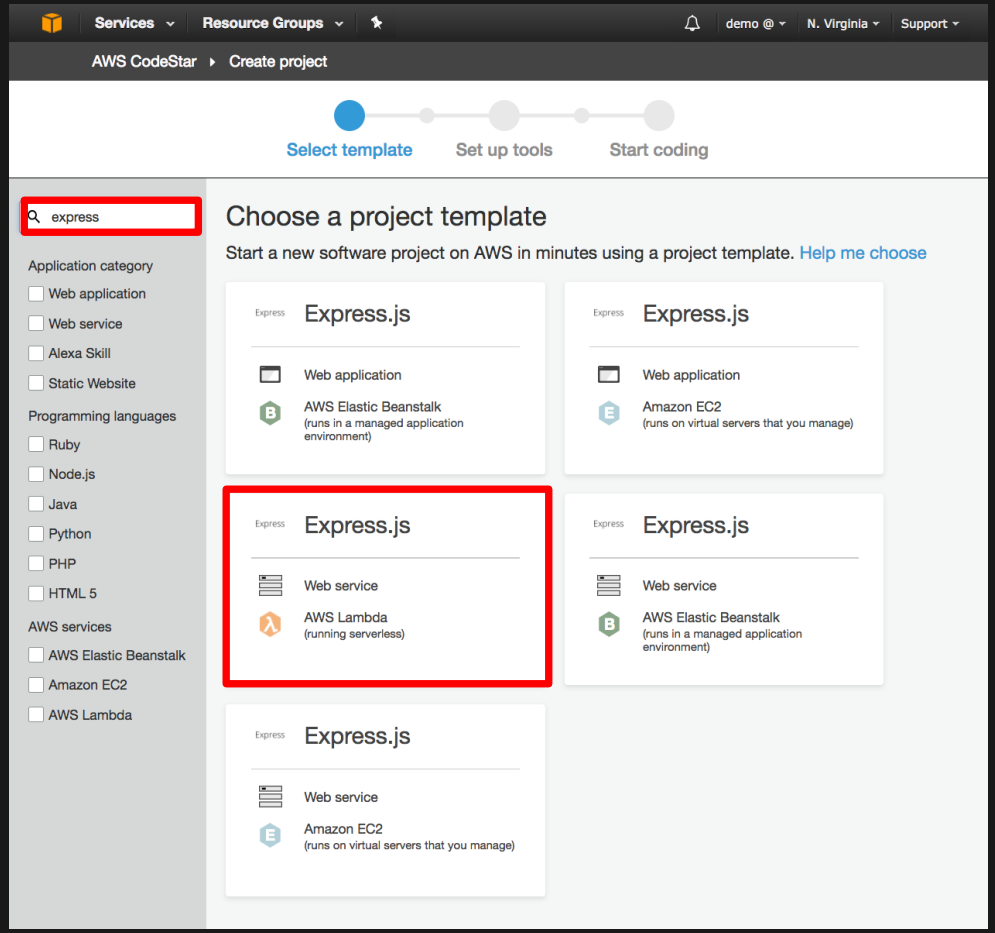
AWS CodeStar

Quickly develop, build, and deploy applications on AWS

https://aws.amazon.com/jp/codestar/

amazon web services

AWS CodeStar で Web フレームワーク on Serverless

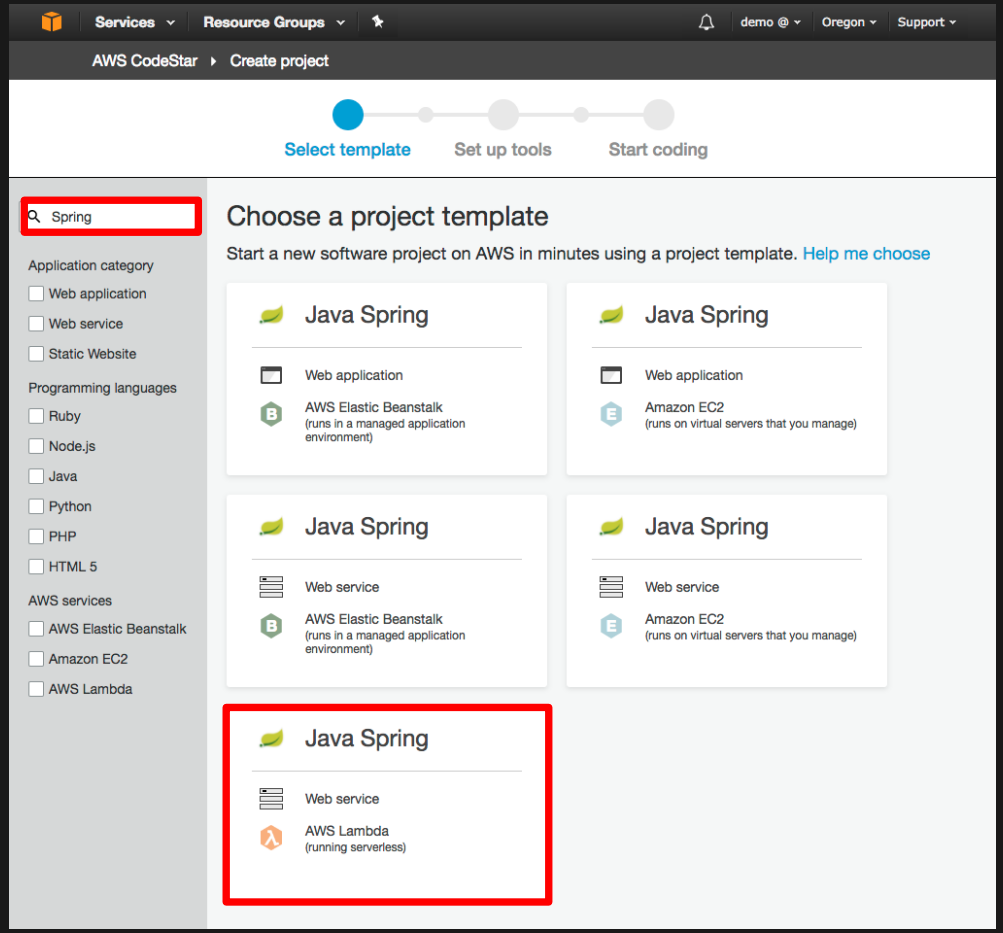


express と入力

↓

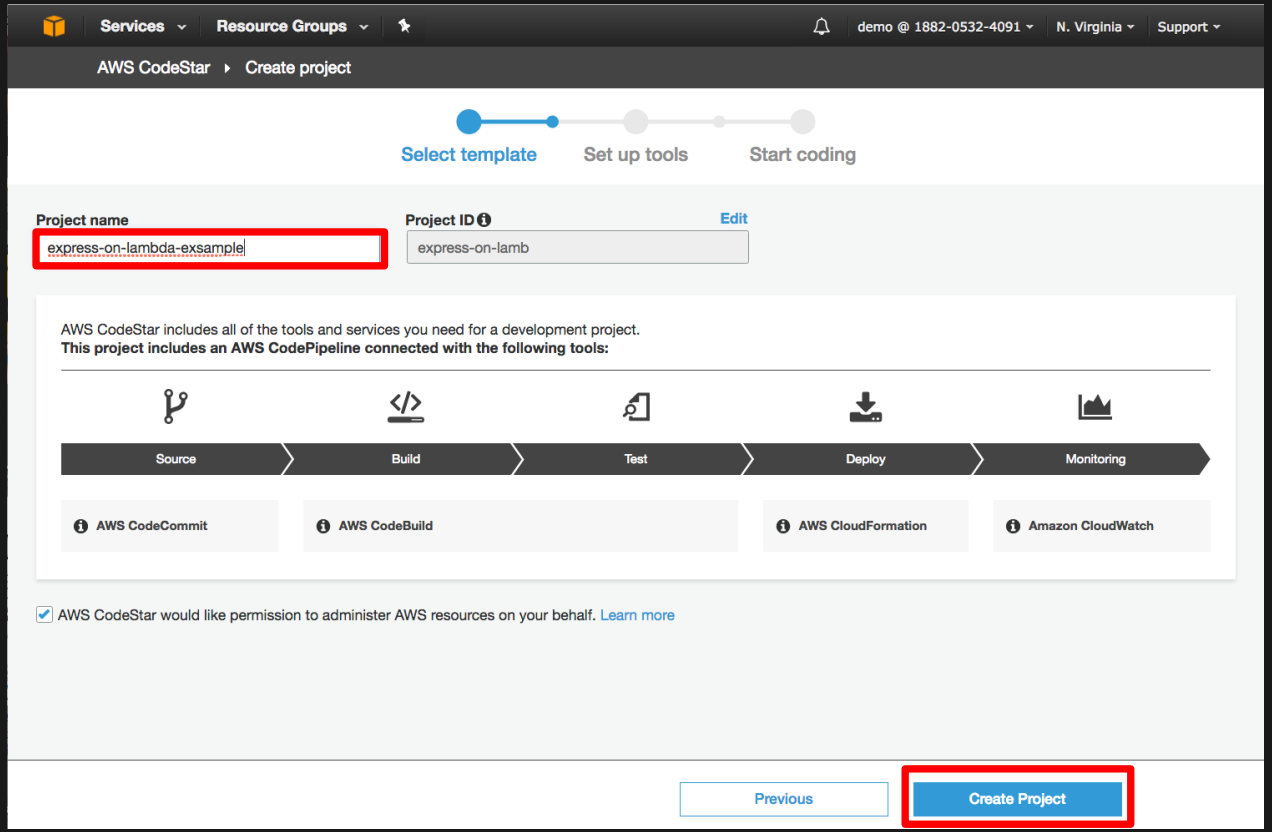
Express.js, Web service, AWS Lambda （running serverless）を選択

AWS CodeStar で Web フレームワーク on Serverless



Spring も対応

AWS CodeStar で Web フレームワーク on Serverless



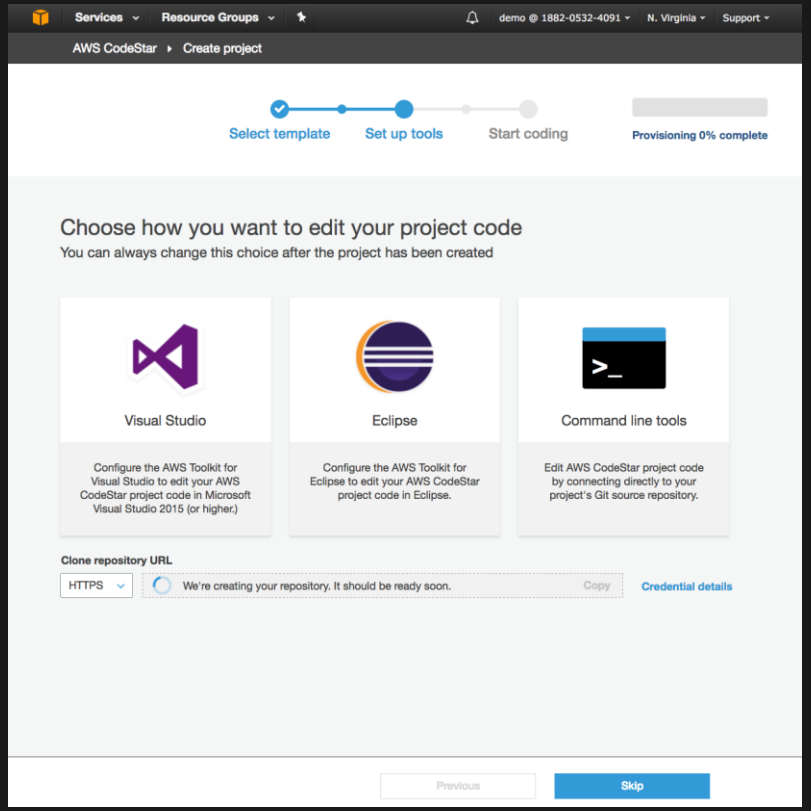
プロジェクト名を入力

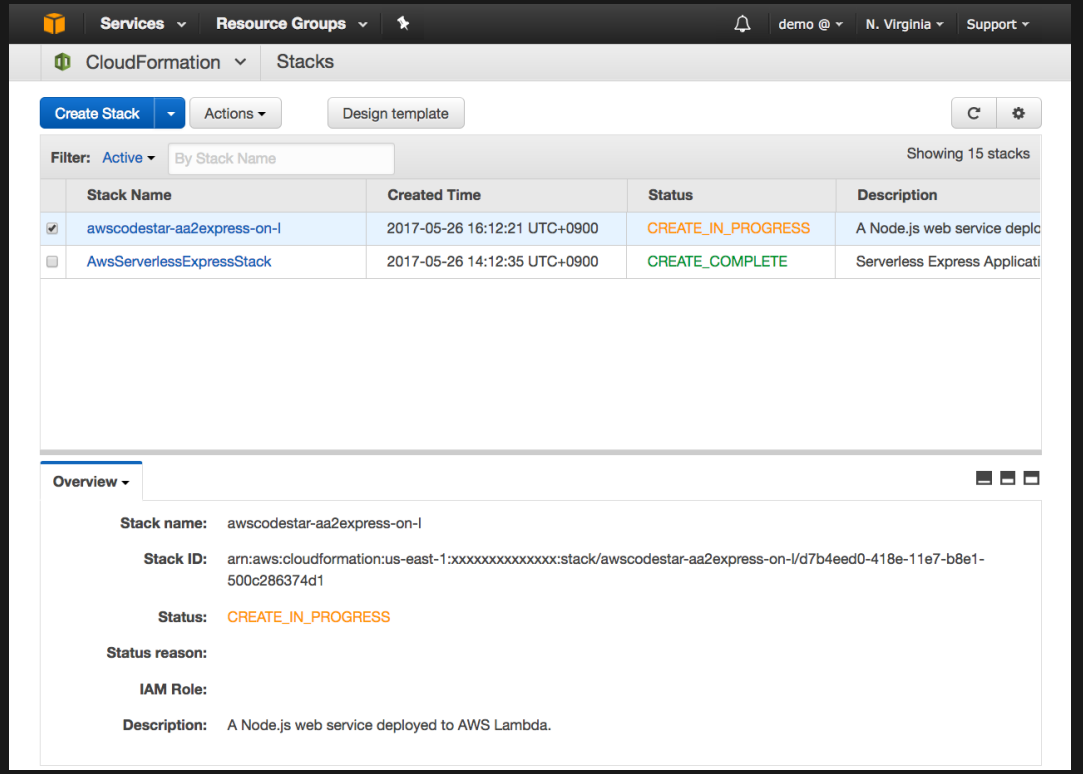
↓

Create Project ボタンをポチっと

amazon web services

AWS CodeStar で Web フレームワーク on Serverless

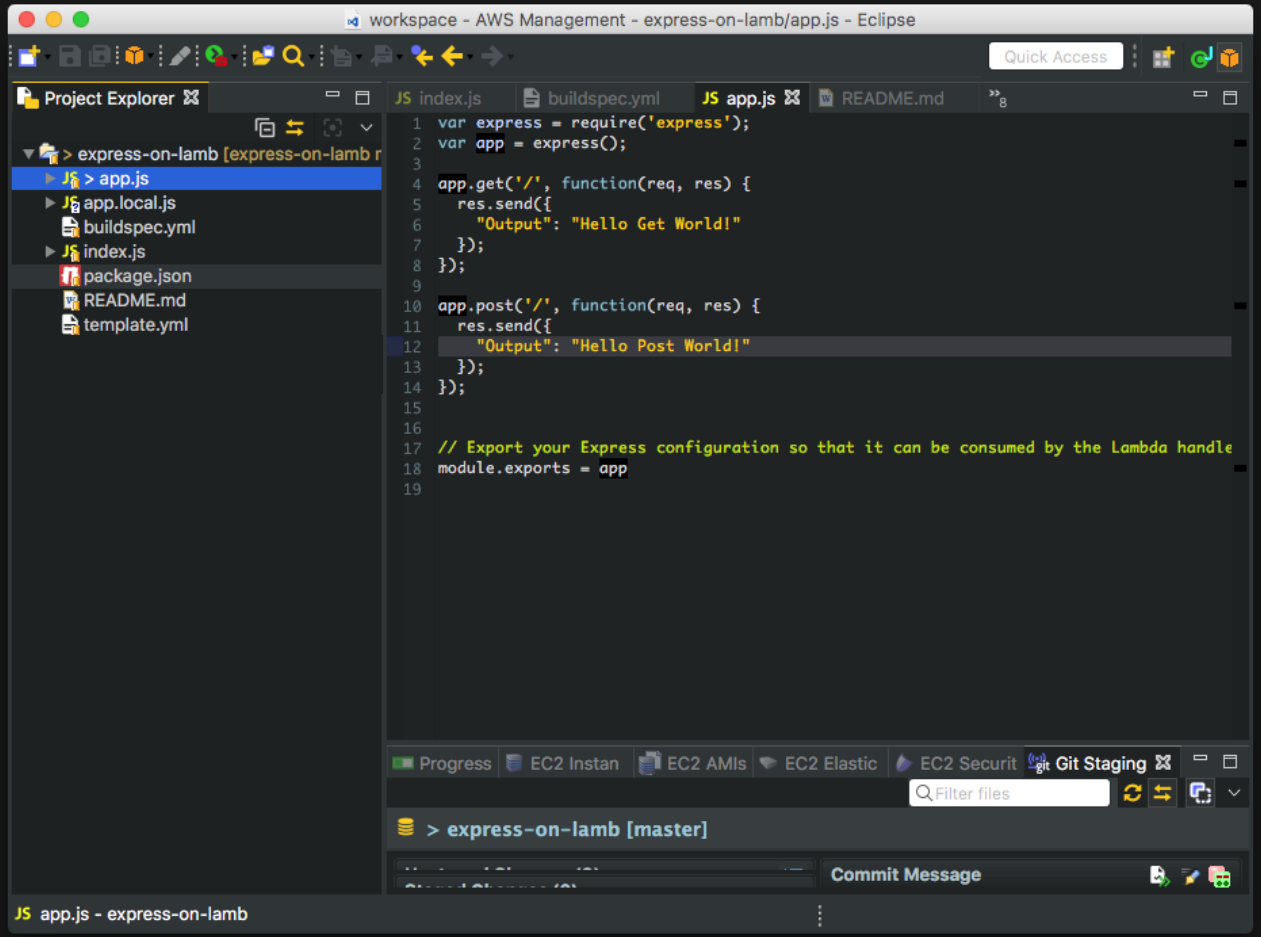




※ 生成されるのは awslabs のライブラリに含まれるサンプルとは別の実装

amazon web services

AWS CodeStar で Web フレームワーク on Serverless



AWS Toolkit を
IDE に追加し
CodeStar プロジェクト
をインポート

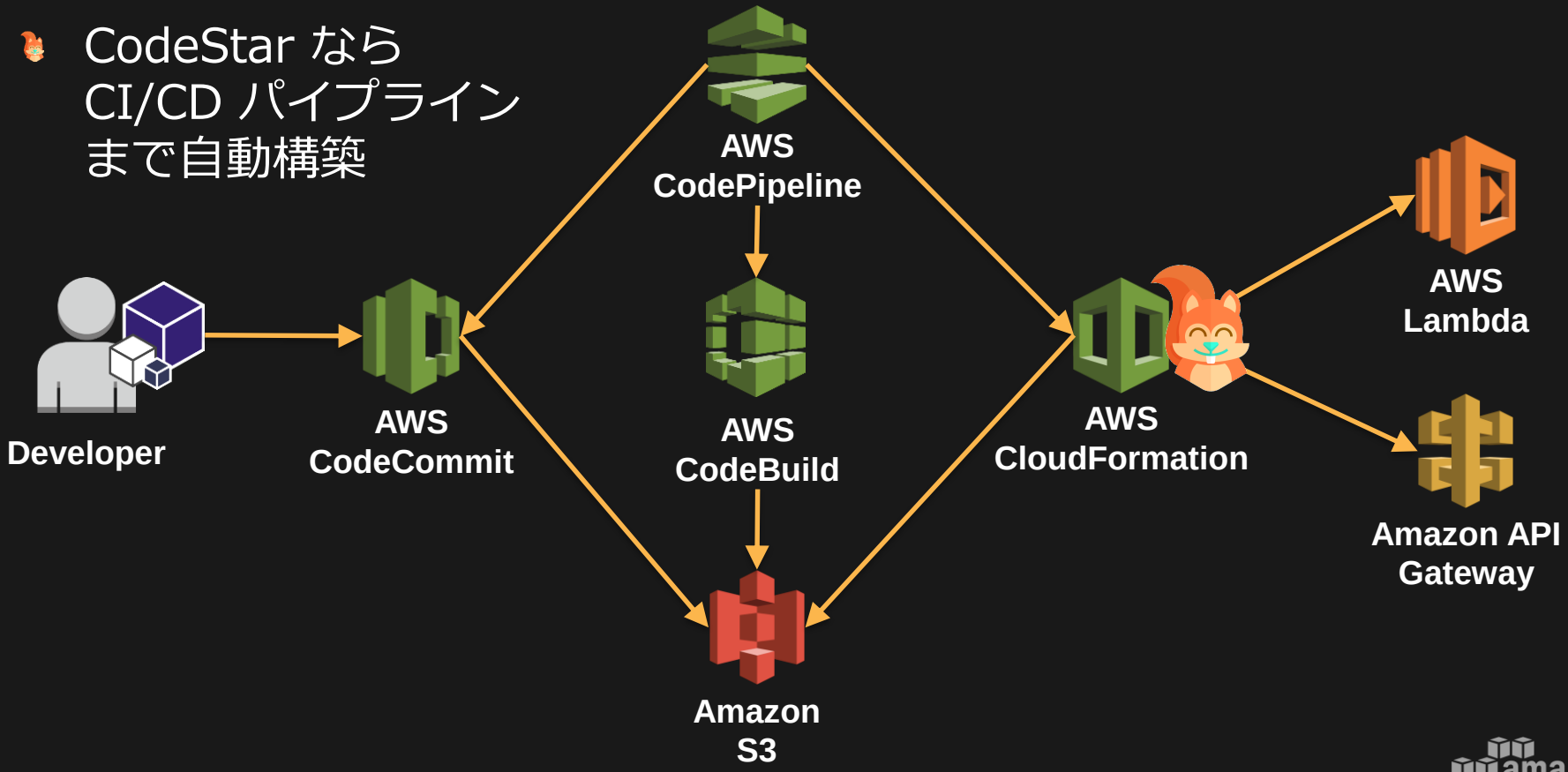
↓

編集し Commit & Push
すればデプロイされる

amazon web services

AWS CodeStar で Web フレームワーク on Serverless

- CodeStar なら CI/CD パイプラインまで自動構築



まとめ

amazon web services

まとめ

- github/awslabs を見ると楽しい✌('ω'✌)三 ✌('ω')✌三(✌'ω')✌
- Express.js と Spring Framework は
  AWS 提供のライブラリや AWS CodeStar を使って
  API Gateway と Lambda で従来どおりの開発が可能
- awslabs の sample を走らせるのも既存プロジェクトをマイグレーションするのも非常に簡単
- AWS CodeStar は CI/CD パイプラインまで自動構築

Thank You!

Don’t Forget Evaluations!


aws
DEVDAY
GLOBAL SERIES