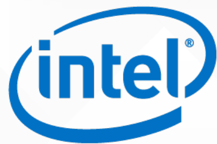
AWS Summit

Tokyo

モバイル開発における AWS Lambda 活用法

Amazon Data Service Japan K.K.
Solutions Architect
Keisuke Nishitani(@Keisuke69)

■Gold Sponsors



Empowered by Innovation

■Global Sponsors

■Silver Sponsors

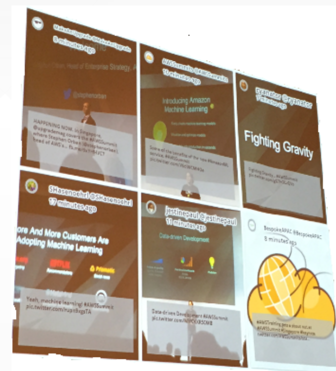
野村総合研究所

■Bronze Sponsors

■Global Tech Sponsors

■Logo Sponsors

ハッシュタグ **#AWSSummit**
と **#DevCon**で、**皆さんのツイート**
が展示エリアの**大画面に表示されます**



公式アカウント**@awscloud_jp**
を**フォローすると**、ロゴ入り
コースターを**プレゼント**

【コースター配布場所】
メイン展示会場、メイン会場1F受付、デベロッパーカンファレンス会場

自己紹介

- 西谷圭介
  - @Keisuke69
  - www.facebook.com/keisuke69
- ロール
  - ソリューションアーキテクト
  - Webサービス / EC / スタートアップを担当
  - モバイルなどアプリ寄りなプロダクトを担当

amazon web services

AWS Lambda知ってますか？

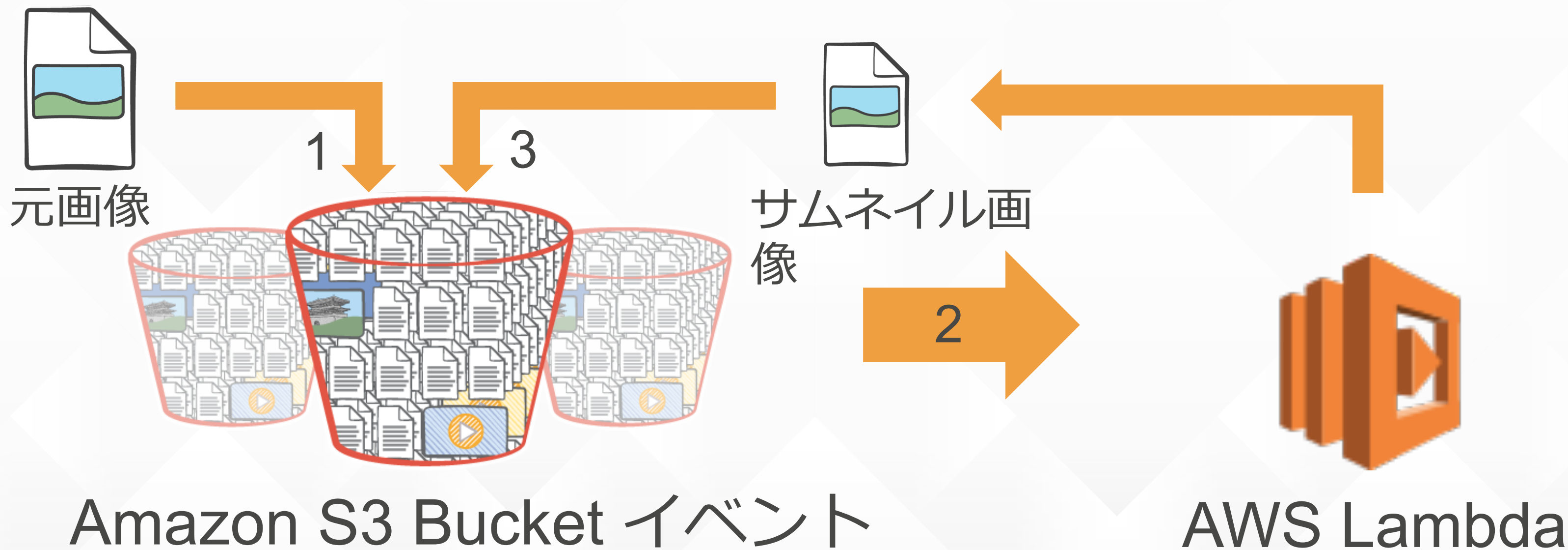
amazon web services

AWS Lambdaとは
イベントをトリガーに
独自のコードを実行させる
Computeサービス

amazon web services

サムネイルの生成やリサイズ

- S3に画像がアップロードされたときにサムネイルの生成やリサイズを実行



amazon web services

もう少し詳しく、、、

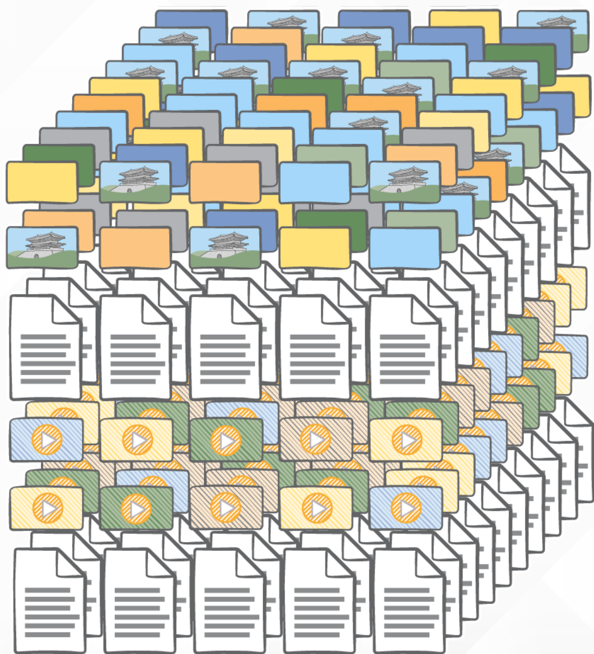
amazon web services

インフラの管理が不要

- ビジネスロジックにフォーカスできる
- コードをアップロードするだけで、あとはAWS Lambdaが以下をハンドリング
  - キャパシティ
  - スケール
  - デプロイ
  - 耐障害性
  - モニタリング
  - ロギング
  - セキュリティパッチの適用

amazon web services

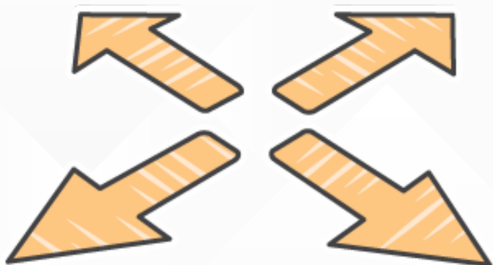
オートスケール



- イベントのレートに合うようにLambdaが自動でスケール
- プロビジョニング中や完了を気にする必要なし
- コードが稼働した分だけのお支払い

amazon web services

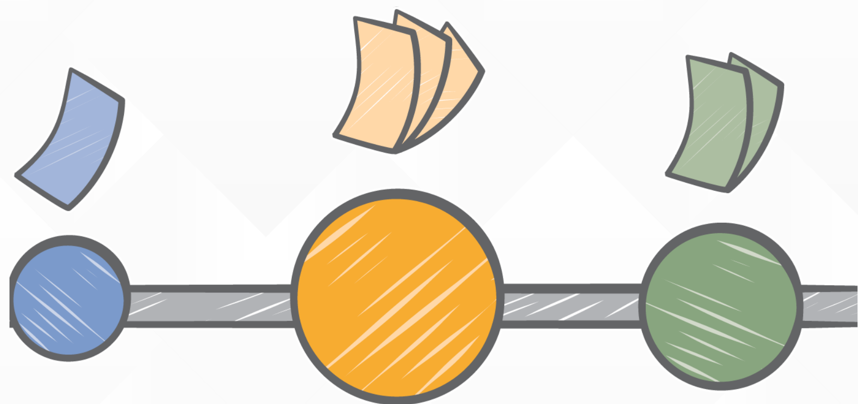
Bring your own code



- Node.jsで書かれたコードを実行
- コード内では以下も可能
  - スレッド/プロセスの生成
  - バッチスクリプトや何らかの実行ファイルの実行
  - /tmpのread/write
- 各種ライブラリも利用可能
  - ネイティブライブラリも可能
  - 利用するライブラリを一緒にアップロード

amazon web services

細やかな料金体系



- 100ミリ秒単位でのコンピュート時間に対する価格設定
- リクエストに対する低額の課金
- 十分な無料枠
- アイドル状態は一切課金されない

amazon web services

Lambdaファンクションの呼び出し

- イベントの発生元となるAWSサービス
  - Amazon S3
  - Amazon Kinesis
  - Amazon DynamoDB Stream(Preview)
  - Amazon Cognito
  - Amazon SNS
- ユーザアプリケーションからの呼び出し
  - カスタムイベント
  - 各種SDKを利用

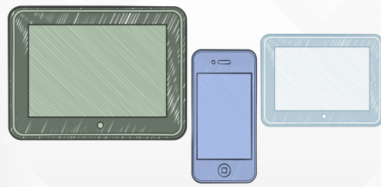
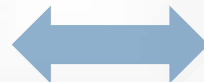
amazon web services

モバイルバックエンドとしての AWS Lambda

amazon web services

モバイルバックエンドとしてのAWS Lambda

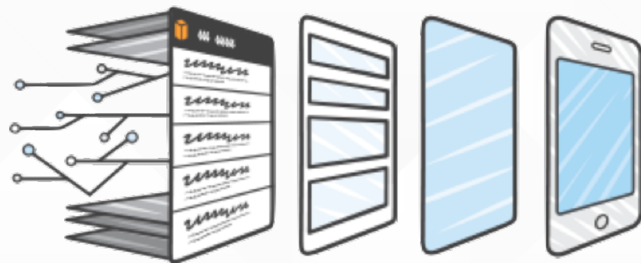
- AWS Mobile SDKによるサポート
  - AWS Mobile SDK for iOS
  - AWS Mobile SDK for Android
- Lambdaファンクションをモバイルアプリから呼び出すことが可能
  - 簡単・即座にスケーラブルなバックエンドとして利用可能
  - バックエンドのためのインフラ管理不要

amazon web services

AWS Mobile SDK

- 全てのサービスに共通の認証機構
- オンライン・オフラインを自動でハンドリング
- クロスプラットフォームのサポート：
  Android, iOS, Fire OS, Unity, Xamarin
- Mobile OSへの最適化
  - 例： ローカルオフラインキャッシュを利用するアーキテクチャ
- モバイルに最適化されたクライアントライブラリ
  - Amazon DynamoDB Object Mapper
  - Amazon S3 Transfer Manager
  - Amazon Kinesis Recorder
- メモリフットプリントの削減
  - 導入するパッケージをサービス単位で選択することが可能



2種類の呼び出し方式

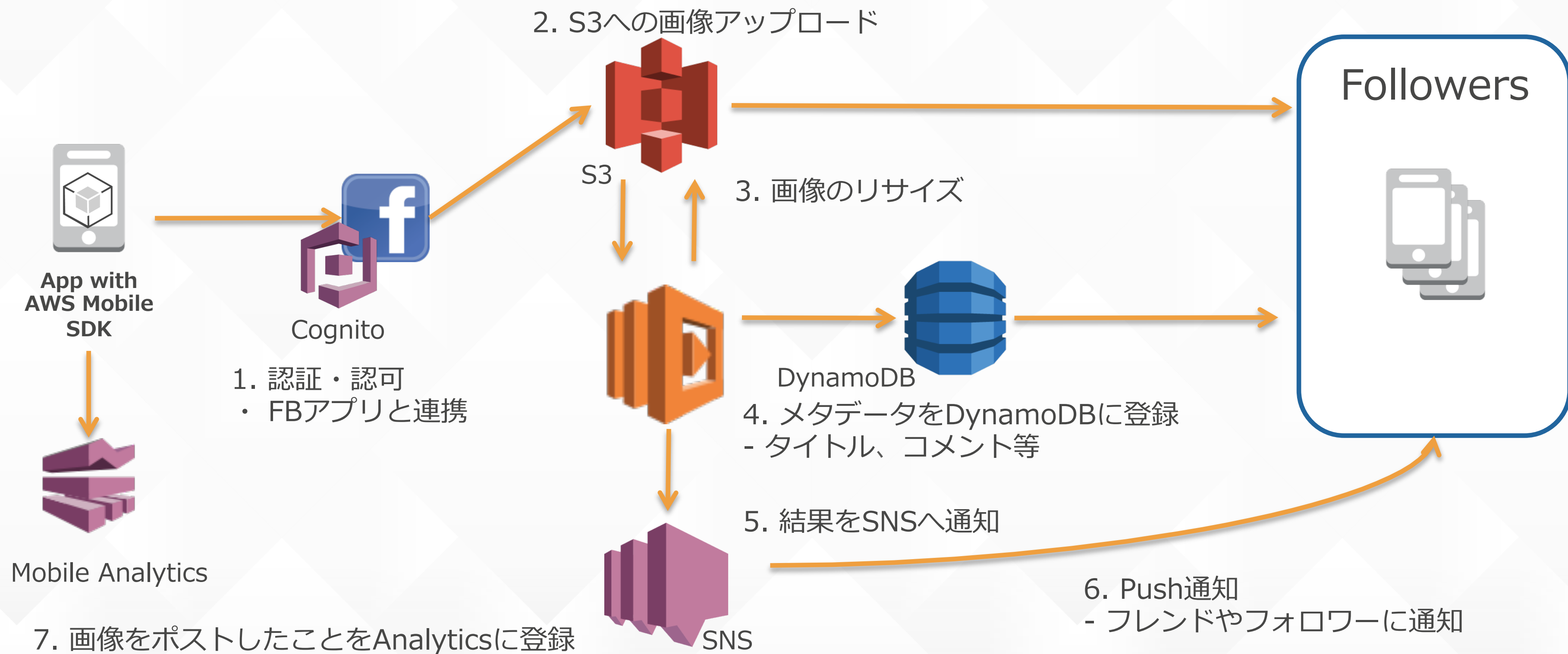
- 呼び出し（Invoke）時にInvocation Typeを指定することで実行形式を選択可能
  - 非同期実行
    - レスポンスはリクエストが正常に受け付けられたかどうかのみ
  - 同期実行
    - 実行完了時にレスポンスが返ってくる。レスポンス内容はLambdaファンクション内でセット可能

amazon web services

ユースケース

amazon web services

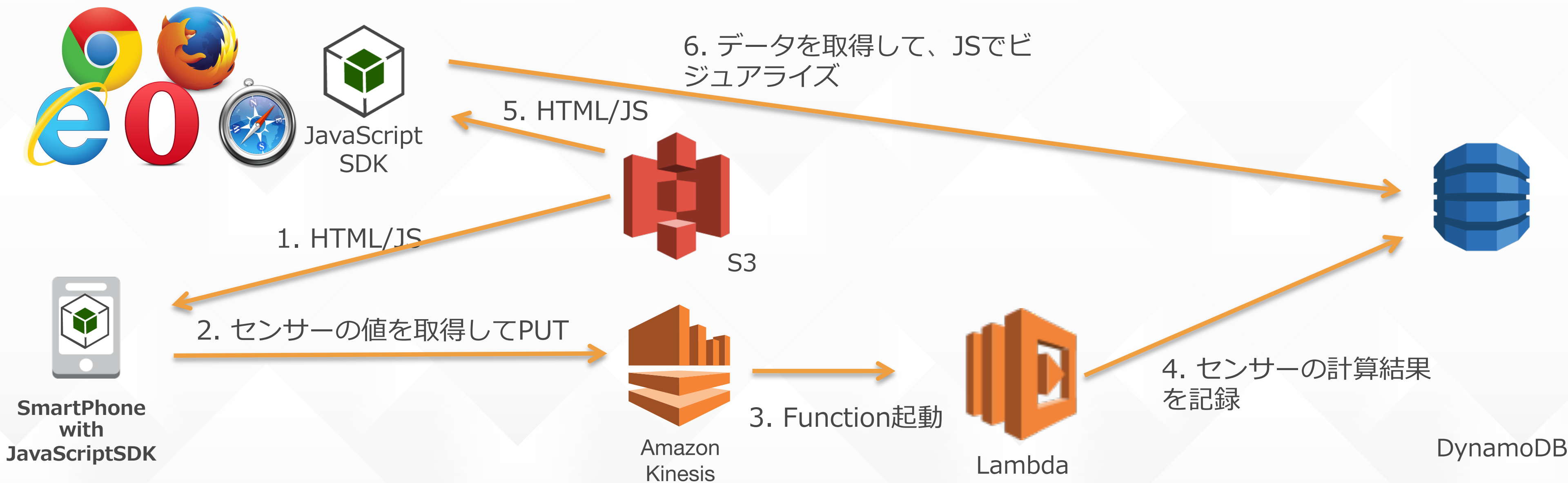
写真共有モバイルアプリ



amazon web services

モーションセンサーを利用した観客参加型ゲーム

- 加速度センサーの値をKinesisに流し込み、Lambdaで計算し、結果をDynamoDBへ
- PCブラウザから結果を取得してリアルタイムにビジュアライズ



APIサーバの代わりとして

- 例えば、ユーザによってコンテンツの出し分けをしたい場合

amazon web services

モバイルバックエンドとしての使い方

とてもシンプル

1. Lambdaファンクションを用意
2. Mobile SDKを使ってモバイルアプリからInvokeする

amazon web services

各種データへのアクセス

- モバイルSDK経由で実行すると、デバイスやアプリ、アイデンティティといった情報にアクセス可能
  - SDKが自動的に収集して設定
  - アプリ内の行動に対するレスポンスをパーソナライズすることも可能

プロパティ名	内容
awsRequestId	Lambdaファンクション呼び出しリクエストのID
logStreamName	CloudWatch LogsのLogストリーム名
clientContext	クライアントアプリおよびデバイスに関する情報
identity	Amazon CognitoのIdentity providorに関する情報

amazon web services

Client Context

プロパティ名	Android	iOS
client.installation_id	初回起動時に生成されたUUID	初回起動時に生成されたUUID
client.app_version_code	Android ManifestのversionCode	CFBundleShortVersionStringの値
client.app_version_name	Android ManifestのversionName	CFBundleVersionの値
client.app_package_name	パッケージ名	CFBundleIdentifierの値
client.app_title	アプリケーションのタイトル	CFBundleDisplayNameの値
env.platform_version	Build.VERSION.RELEASE	systemVersionの値（OSバージョン）
env.platform	Android（固定）	systemNameの値（OS名）
env.make	Build.MANUFACTURER	apple(固定)
env.model	Build.MODEL	モデル名
env.locale	Locale.getDefault()で返ってくる値	autoupdatingCurrentLocaleの localeIdentifier

amazon web services

Lambdaファンクションを モバイルアプリから呼び出す

amazon web services

Lambdaファンクションを用意する

```
exports.handler = function(payload, context) {
    console.log("Received event");
    context.succeed("Hello "+ payload.firstName + ". Your user ID is " +
context.identity.cognitoIdentityId  + " and your platform is " +
context.clientContext.client.platform);
};
```

- Client ContextはSDKによってデバイス上で収集され、Lambdaファンクションに渡される
  - 特定バージョンだけ処理を分岐するなども可能

- クレデンシャルプロバイダとしてAmazon Cognitoを利用している場合、そのアイデンティティも渡される

※上記サンプルコードはLambdaファンクション作成時に提供されるテンプレート

amazon web services

Androidの場合

amazon web services

Lambdaファンクションに対応するメソッドを定義する

```
public interface MyInterface {
@LambdaFunction(functionName = “hello”, invocationType =
"RequestResponse")
    String hello(NameInfo nameInfo);
}
```

- Lambdaで実行したい処理をinterfaceとして定義
- 各メソッドに@LambdaFunctionというアノテーションを付与する
  - ファンクション名やInvocation Type、LogTypeはパラメータで指定
- 例では引数としてNameInfoというオブジェクトが渡されている（後述）

amazon web services

カスタムデータタイプ

```
public class NameInfo {

    String firstName;
    String lastName;

    public NameInfo(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

- Lambdaファンクションに渡すデータをカスタムデータとして定義
  - 単なるJavaオブジェクトとして実装すればよい

- 標準ではシリアライズ/デシリアライズにLambdaJSONBinderが使われる

amazon web services

LambdaInvokerFactoryの初期化

```
AWSCredentialsProvider provider = new
CognitoCachingCredentialsProvider(myActivity, xxxxxxxxxxxxxxxxxx,
Regions.US_WEST_1);

LambdaInvokerFactory factory = new LambdaInvokerFactory(myActivity,
Regions.US_WEST_1, provider);
```

- Amazon Cognitoを使ってLambdaInvokerFactoryを初期化
  - Cognitoを使うことでテンポラリで権限の制限されたクレデンシャルが提供される
  - 利用するIdentityPoolIdを指定し、AWSCredentialProviderを初期化する

amazon web services

Proxyオブジェクトの作成とLambdaファンクションの呼び出し

```
//Proxyオブジェクトの作成
MyInterface invoker = factory.build(MyInterface.class);

//Lambdaファンクションの呼び出し
String result = invoker.hello(nameInfo);
```

- 作成したInterfaceのProxyオブジェクトを作成し、それを利用してファンクションを呼び出す
- ネットワークコールが発生するのでメインスレッドでは実行しないこと

amazon web services

エラーハンドリング

```
try {
    result = invoker.hello(nameInfo);
} catch (LambdaFunctionException lfe) {
    // Lambdaファンクション失敗時
    Log.e(TAG, "Failed to execute echo", lfe);
} catch (AmazonServiceException ase) {
     // クレデンシャルの間違い、など
} catch (AmazonClientException ace) {
    // ネットワーク起因のエラー
}
```

- Lambda側でエラーが発生した場合はLambdaFunctionExceptionがthrowされる
  - エラーメッセージおよび呼び出し結果を取得可能
- 不正なクレデンシャルやネットワークなどその他の理由で失敗した場合はLambdaFunctionExceptionはthrowされない

amazon web services

カスタムデータバインダ

- シリアライズ/デシリアライズを独自に行いたい場合
  - 標準のLambdaJSONBinderを使いたくない場合
- LambdaDataBinderを実装し、Proxyオブジェクト作成時に指定する

amazon web services

（例）カスタムデータバインダ

```
public class JacksonDataBinder implements LambdaDataBinder {
   private final ObjectMapper mapper;

   public JacksonDataBinder() {
      mapper = new ObjectMapper();
      mapper.setPropertyNamingStrategy(PropertyNamingStrategy.CAMEL_CASE_TO_LOWER_CASE_WITH_UNDERSCORES);
   }

   public  T deserialize(byte[] content, Class clazz) {
     try {
         return mapper.readValue(content, clazz);
     } catch (IOException e) {
         throw new AmazonClientException("Failed to deserialize content", e);
     }
   }

   public byte[] serialize(Object object) {
      try {
          return mapper.writeValueAsBytes(object);
      } catch (IOException e) {
          throw new AmazonClientException("Failed to serialize object", e);
      }
  }
}

MyInterface myInterface = lambdaInvokerFactory.build(MyInterface.class, new JacksonDataBinder());
```

amazon web services

iOSの場合
(Objective-C)

amazon web services

AWSCognitoCredentialsProviderのセットアップ

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    AWSCognitoCredentialsProvider *credentialsProvider =
[[AWSCognitoCredentialsProvider alloc] initWithRegionType:AWSRegionUSEast1

identityPoolId:@"YourCognitoIdentityPoolId"];
    AWSServiceConfiguration *configuration = [[AWSServiceConfiguration alloc]
initWithRegion:AWSRegionUSWest2

credentialsProvider:credentialsProvider];
    AWSServiceManager.defaultServiceManager.defaultServiceConfiguration =
configuration;

    return YES;
}
```

amazon web services

Lambdaファンクションの呼び出し

```
AWSLambdaInvoker *lambdaInvoker = [AWSLambdaInvoker defaultLambdaInvoker];
[[lambdaInvoker invokeFunction:@”hello"
                    JSONObject:@{@”firstname" : @"YourFirstName",
                                 @”lastname" : @"YourLastName”
                                }] continueWithBlock:^id(BFTask *task)
{
    if (task.result) {
        NSLog(@"Result: %@", task.result);
        NSString *result = task.result;
    }
    return nil;
}];
```

- invokeFunctionで同期呼び出しを行う
  - パラメータでファンクション名を指定
- パラメータとしてJSONObjectが指定された場合、シリアライズされて送信される
  - LambdaはJSON形式のレスポンスを返し、JSONObjectにデシリアライズされる

amazon web services

エラーハンドリング

```
if (task.error) {
  NSLog(@"Error: %@", task.error);
  NSLog(@"Function error: %@", task.error.userInfo[AWSLambdaInvokerFunctionErrorKey]);
}
```

- Lambdaの実行に失敗した場合、NSErrorが返る
  - ドメインはAWSLambdaErrorDomain
  - エラーコードは以下の4種類から失敗内容に応じてセットされる
    - AWSLambdaErrorUnknown / AWSLambdaErrorService
    - AWSLambdaErrorResourceNotFound / AWSLambdaErrorInvalidParameterValue
- Lambdaファンクションの実行に失敗した場合
  - ドメインはAWSLambdaInvokerErrorDomain
  - エラーコードは AWSLambdaInvokerErrorTypeFunctionError
  - userInfoにはAWSLambdaInvokerFunctionErrorKeyというキーでLambdaファンクションが返すエラーが含まれる

amazon web services

Lambdaファンクションの書き方

Lambdaファンクション

- コードはJavaScript（Node.js）で記述
- メモリ容量はデフォルトで128MB
  - 64MBごとに設定可能
  - 容量に応じてCPU能力なども変動
- 実行時間のタイムアウトはデフォルトで3秒、最大60秒まで
- それぞれが隔離されたコンテナ内で実行される

amazon web services

基本

- Node.jsのベストプラクティスに従う
  - AWS SDKやImageMagickは組み込み済みで標準で使えるようになっている
- ステートレス
  - データを永続化するためにはS3、DynamoDBもしくはその他のインターネット経由で利用可能なストレージを利用すること
  - 実際に実行されるサーバは毎回異なり、ログインもできない
  - /tmpへのread/writeは可能だがあくまでも一時的な用途として使用すること
- その他
  - プロセス、スレッド、/tmp、ソケットを利用可能
  - インバウンドのソケット接続は不可能
  - 各種ライブラリを利用可能

amazon web services

Lambdaファンクションへの登録

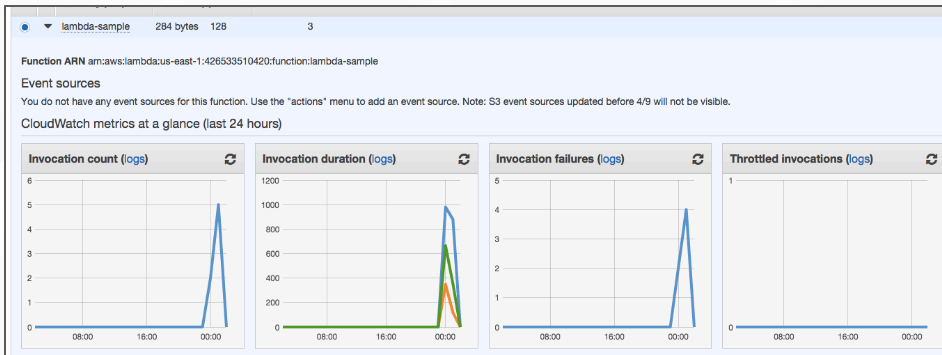
- コンソールに組み込まれたコードエディタ
  - サンプルテンプレートあり
- Zipファイルによるコードのアップロード
  - 外部ライブラリを含める場合はこの方式となる
  - Lambdaファンクションへの直接アップロード
  - S3にアップロードしてそれを指定することも可能

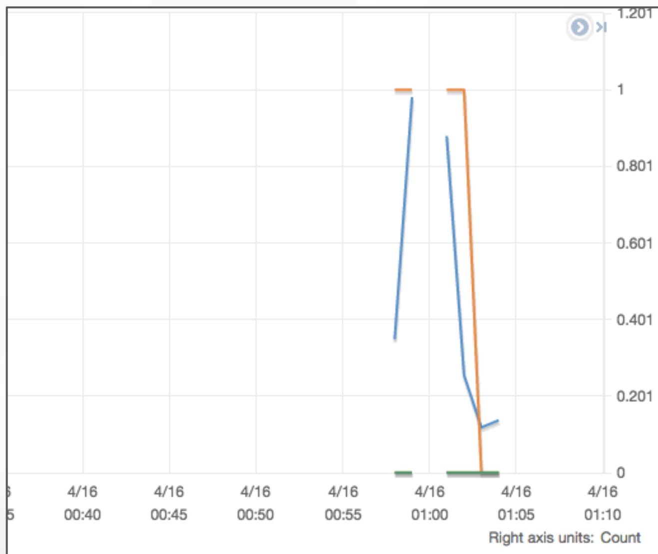
CLI実行例

```
aws lambda update-function-code --function-name sample  --zip-file fileb:///path/to/zipfile/function.zip
```

amazon web services

モニタリング





- ダッシュボード
  - 全てのLambdaファンクションのリスト
  - 可視化されたメトリクス
- CloudWatchを用いたMetricsの監視
  - Invocations
  - Errors
  - Duration
  - Throttle

amazon web services

デバッグ

```
exports.handler =
  function(event, context)  {
    console.log('Received event:');
    var bucket =
      event.Records[0].s3.bucket.name;
    var key =
      event.Records[0].s3.object.key;
    console.log('Bucket: '+bucket);
    console.log('Key:    '+key);
    …
```

- 実行ログがCloudWatchに出力される
  - 各Lambdaファンクションごとのロググループ
- 実行開始/終了と消費したリソースに関するデフォルトのログエントリ
  - メモリ使用量（Max Memory Used）
  - 実行時間（Duration）
  - 課金対象時間（Billed Duration）
- カスタムログエントリの追加も可能
  - ファンクション内でconsole.logを用いて出力

amazon web services

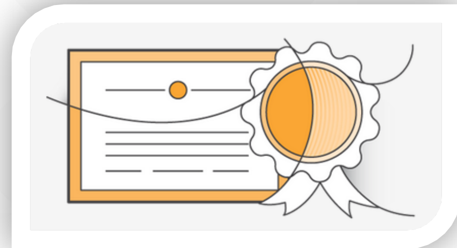
Run Code in the cloud!

amazon web services

AWSトレーニング @ AWS Summit Tokyo

セルフペースラボ：@パミール1F 瑞光

AWS クラウドに実際に触れてみませんか？
ご自分の AWS アカウントをおつくりいただけなくても、
AWS クラウドを体験いただけます。



AWS認定試験（有償）：@ パミール1F 黄玉

特設認定試験会場を AWS Summit Tokyo 2015 会場に開設
Devopsエンジニア-プロフェッショナル認定試験を先行受験いただけます。

AWS認定資格者取得専用ラウンジ：@ パミール1F 青玉

他の AWS 認定資格をお持ちの方とのネットワーキングにぜひラウンジをご活用ください。
お席や充電器、お飲物などを用意し、皆様をお待ちしております。

amazon web services

Thank You

amazon web services