AWS Summit

Tokyo

amazon web services

Run Code in The Cloud

AWS Lambda 概要

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

amazon web services

■Gold Sponsors

intel
Empowered by Innovation
NEC
NTT DATA
NetApp
Serverworks
HITACHI
Inspire the Next
TREND MICRO
WORKS APPLICATIONS

■Global Sponsors

infor

■Silver Sponsors

CTC
Challenging Tomorrow's Changes
FPT
JBCC
SKYARCH
あなたの側で、あなた以上に考える。
iSiD
IT Solution Innovator
ナレコムクラウド
NRI 未来創発
Dream up the future.
野村総合研究所
FUJISOFT
amazon web services

■Bronze Sponsors

APPRESSO
ISAO
Logstorage
infoteria
IMPERVA
WingArc 1st
intra-mart
GitHub
KDDI
colt STRONGER TOGETHER
JMAS
STS Innovation Working For You® システムサポート
siteROCK
CAC
Sky365
TerraSky
SMART InSight.
SOPHOS
splunk>
+ tableau
DATADOG
MKI
redhat.

■Global Tech Sponsors

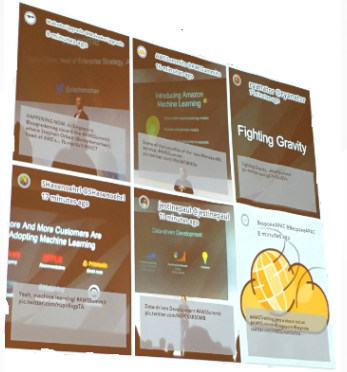
AVERE
ZADARA STORAGE
talend
druva

■Logo Sponsors

AUDIT MASTER
Asgent アスジェント
駅すぱあと
EQUINIX
Canon キヤノンITソリューションズ株式会社
class method
Voxer
Global Knowledge
TAG BANGERS
TECHORUS
FUJITSU 富士通ラーニングメディア
HANDS LAB

amazon web services

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

amazon
web services

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

課題はシンプル

amazon web services

例えば、

S3のバケットに画像が保存されたらサムネイルイメージを用意したい

例えば、

DynamoDBに保存されるアドレス
が全て正しい形式かチェックしたい

amazon web services

AWS Lambda以前

amazon web services

解決方法は複雑

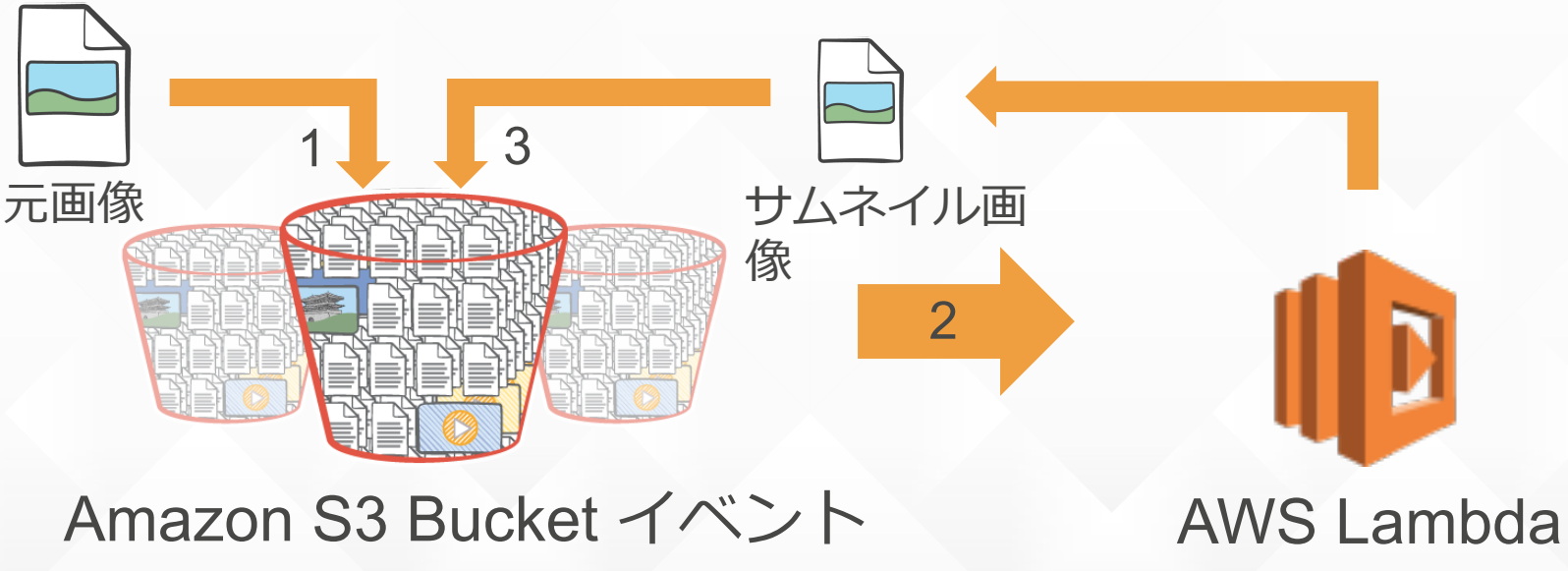
- アップロードを検知するためのサーバと仕組みを用意
- イベントに応じて処理を実行するサーバ群を用意
- OSの設定や言語環境の構築
- パッチ適用や更新をし続ける必要もある
- 予測困難なリクエスト数に対し、スケールや耐障害性を高める仕組みを自身で構成
- キャパシティや状態、セキュリティなどを24時間365日モニタリング

AWS Lambda以降

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



値チェックや別テーブルへのコピー

- DynamoDBへの書き込みに応じて値チェックをしつつ別テーブルの更新やプッシュ通知を実行

つまり、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

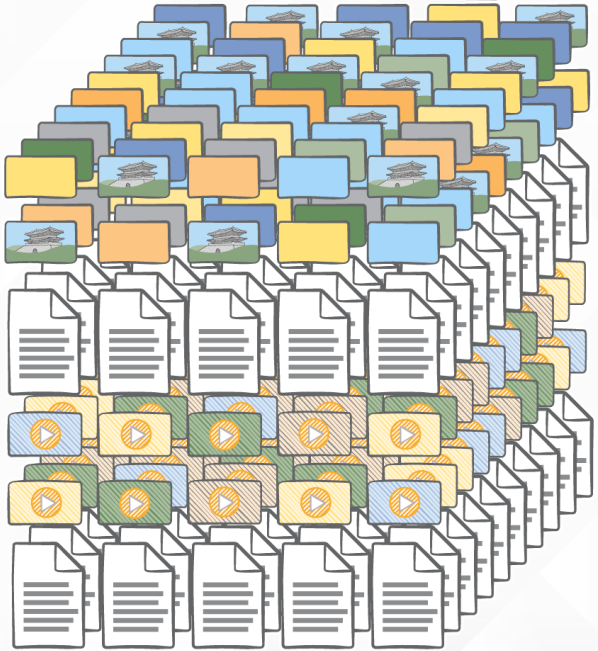
AWS Lambdaの特徴

amazon web services

インフラの管理が不要

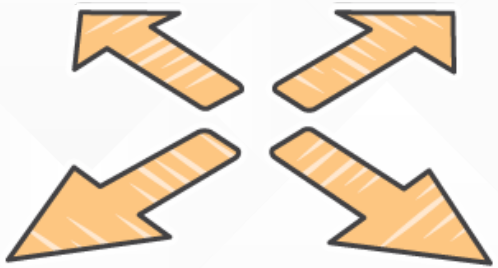
- ビジネスロジックにフォーカスできる
- コードをアップロードするだけで、あとはAWS Lambdaが以下をハンドリング
  - キャパシティ
  - スケール
  - デプロイ
  - 耐障害性
  - モニタリング
  - ロギング
  - セキュリティパッチの適用

オートスケール



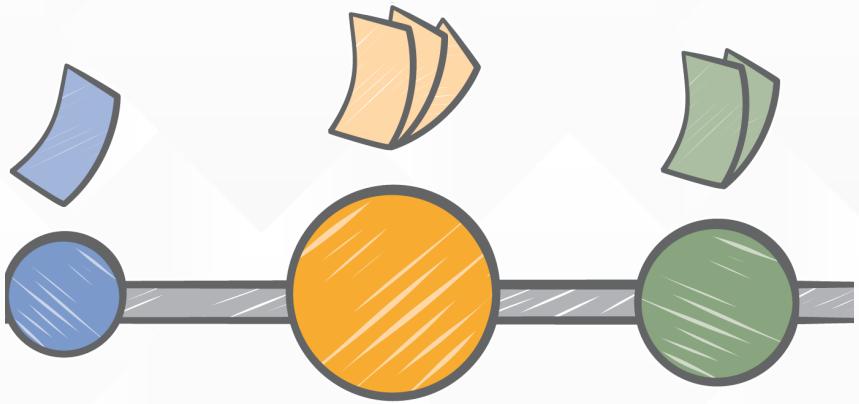
- イベントのレートに合うようにLambdaが自動でスケール
- プロビジョニング中や完了を気にする必要なし
- コードが稼働した分だけのお支払い

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

細やかな料金体系



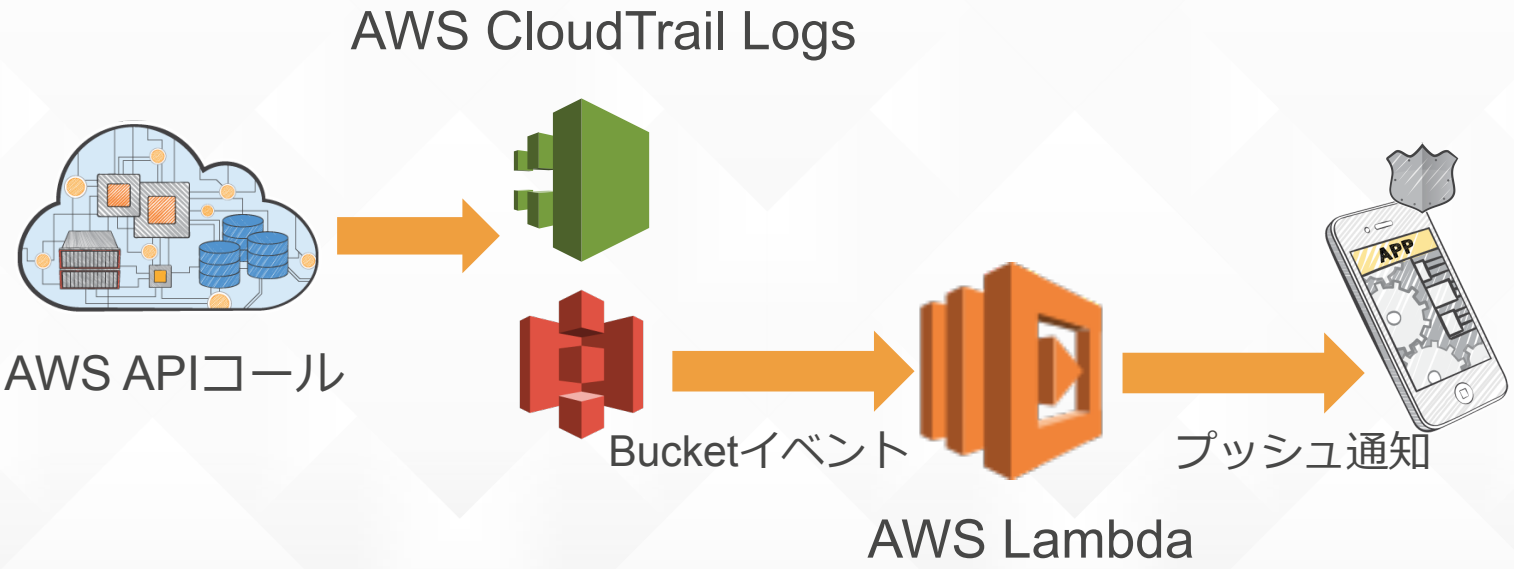
- 100ミリ秒単位でのコンピュート時間に対する価格設定
- リクエストに対する低額の課金
- 十分な無料枠
- アイドル状態は一切課金されない

ユースケース

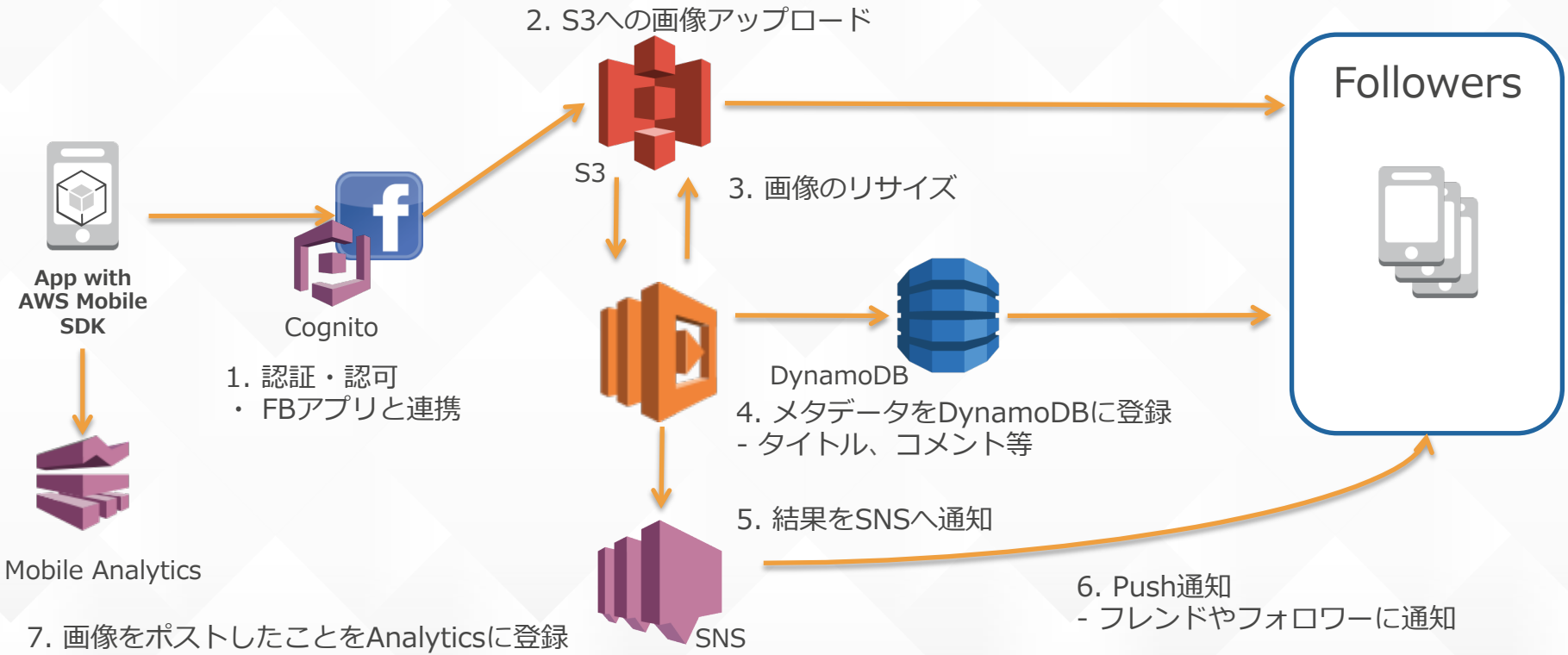
amazon web services

監査と通知

- S3に保管されるCloudTrailのログを分析し、怪しい行動や異常を検知したら通知する



写真共有モバイルアプリ



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

JavaScript SDK

6. データを取得して、JSでビジュアライズ

5. HTML/JS

S3

1. HTML/JS

2. センサーの値を取得してPUT

SmartPhone with JavaScriptSDK

Amazon Kinesis

3. Function起動

Lambda

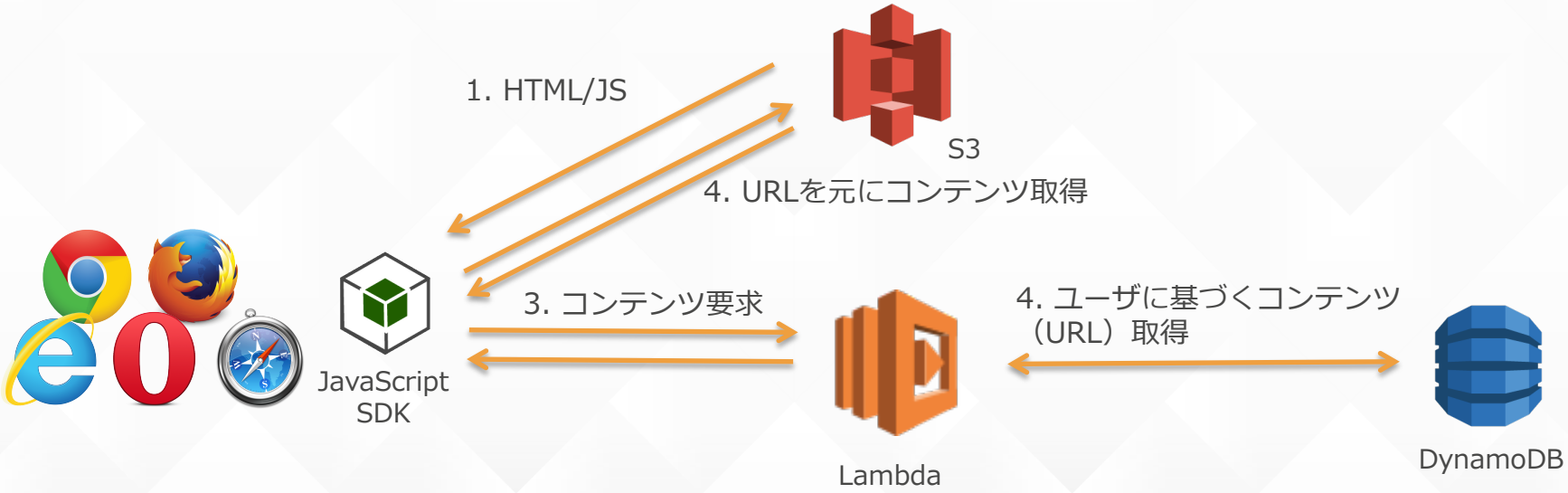
4. センサーの計算結果を記録

DynamoDB

amazon web services

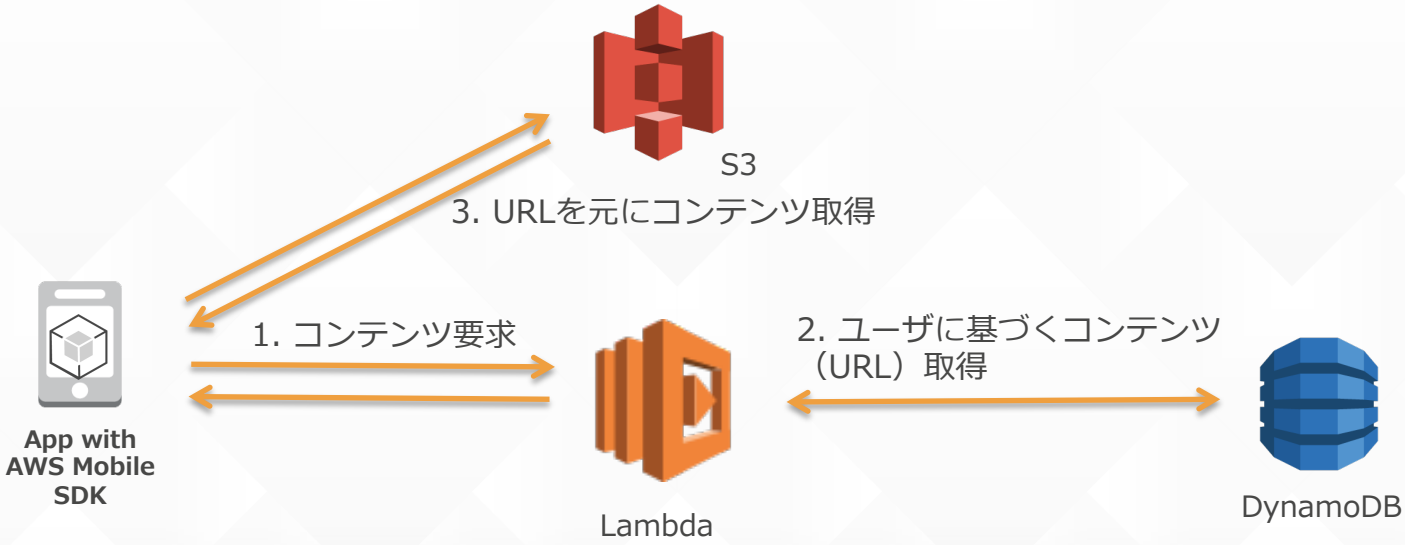
APIサーバの代わりとして

- 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
- 同期呼び出しで実現



APIサーバの代わりとして（モバイルの場合）

- 前述と同様のことをモバイルアプリからも可能



AWS Lambdaの使い方

amazon web services

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

イベントソース

- イベントの発生元となるAWSリソース

- サポートする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

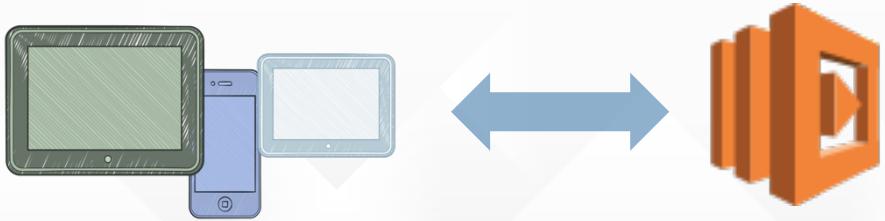
- モバイルもしくはWebアプリからの呼び出しが可能
  - AWS SDKもしくはAWS Mobile SDKを利用

- 2種類の実行タイプ
  - 非同期実行
    - レスポンス内容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

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ファンクションの同期呼び出し
  - 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



同期呼び出しコード例（JavaScript）

```
var params = {
    IdentityPoolId: "Cognito Identity Pool ID”,
};

AWS.config.region = 'us-east-1’;
AWS.config.credentials = new AWS.CognitoIdentityCredentials(params);
AWS.config.credentials.get(function(err) {
    if (!err) {
        var event = {
            'key': 'value’
        };
        var params = {
            FunctionName: 'Sample’,
            InvocationType: 'RequestResponse’,
            LogType: 'Tail’,
            Payload: JSON.stringify(event)
        };

        var lambda = new AWS.Lambda();
        lambda.invoke(params, function(err, data) {
            if (err){
                console.log(err, err.stack);
            }else{
                console.log(data.Payload);
            }
        });
    } else {
        console.log("Error:" + err);
    }
});
```

イベント

- イベントはJSON形式でLambdaに渡される
- Lambdaファンクションはイベントごとに実行される
  - PUSHモデル: Amazon S3、Amazon Cognito、Amazon SNSとカスタムイベント
    - 順不同
    - サービスもしくはアプリケーションが直接実行
    - 3回までリトライ
  - PULLモデル: Amazon DynamoDB と Amazon Kinesis
    - 順序性あり。ストリームに入ってきた順に処理される
    - イベントソースとして登録したストリームに対してLambdaが自動的に取得しにいく
    - イベントごとに複数のレコードを取得可能
    - データが期限切れになるまでリトライ

イベント例（S3）

```
{
  "Records": [

     -- 省略 -

     "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "sourcebucket",
          "ownerIdentity": {
            "principalId": "A3NL1KOZZKExample"
          },
          "arn": "arn:aws:s3:::mybucket"
        },
        "object": {
          "key": "sourcebucket/HappyFace.jpg",
          "size": 1024,
          "eTag": "d41d8cd98f00b204e9800998ecf8427e"
        }
      }
    }
  ]
}
```

イベント例（Kinesis）


```
{
  "Records": [
    {
      "awsRegion": "us-east-1",
      "sequenceNumber": "19680000000000000000374",
      "partitionKey": "2efdb0ea22685b46993e42a67302a001",
      "eventSource": "aws:kinesis",
      "data": "SOME CUSTOM DATA 1"
    },
    {
      "awsRegion": "us-east-1",
      "sequenceNumber": "19680000000000000000571",
      "partitionKey": "2efdb0ea22685b46993e42a67302a003",
      "eventSource": "aws:kinesis",
      "data": "{ \"key\": \"value\" }"
    }
  ]
}
```

パーミッションモデル

- Executionパーミッションと Invocationパーミッションの2種類
  - IAMロールとして用意する
- Lambdaファンクション作成時にAWS Lambda側で自動で作成することも可能
- クロスアカウントアクセスも可能

ExecutionRole

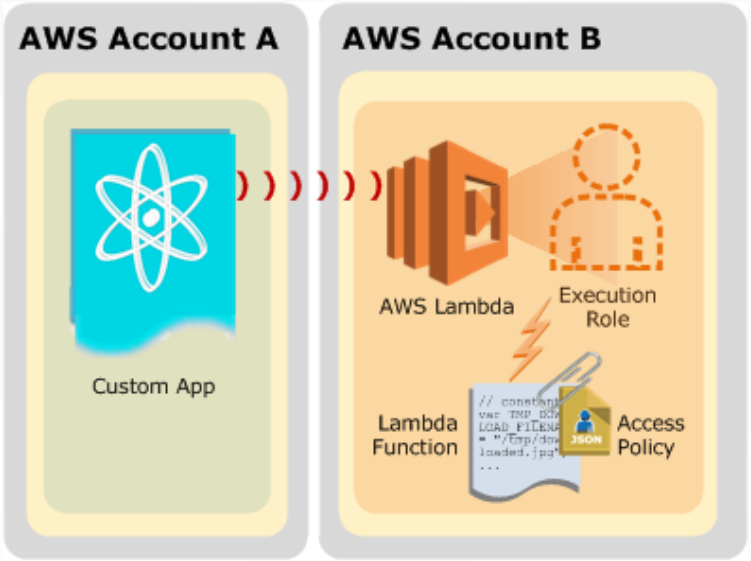
- 必要なAWSリソースへのアクセスを許可するIAMロール
- 指定されたIAMロールにそってAWSのリソースへのアクセスが許可される

Execution Role例（イベントソースがKinesisの場合）

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:GetRecords",
        "kinesis:GetShardIterator",
        "kinesis:DescribeStream",
        "kinesis:ListStreams",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

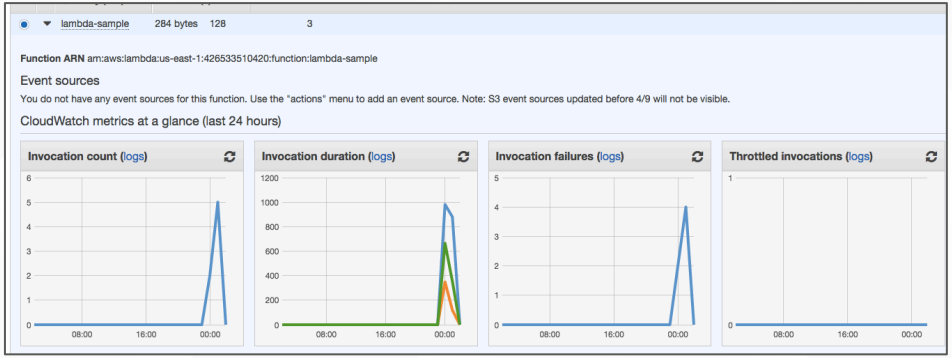
- Kinesisの場合、Pullモデルとなるため、LambdaがKinesisにポーリングできるよう権限を与える
- LambdaのInvokeFunctionというアクションの許可も必要

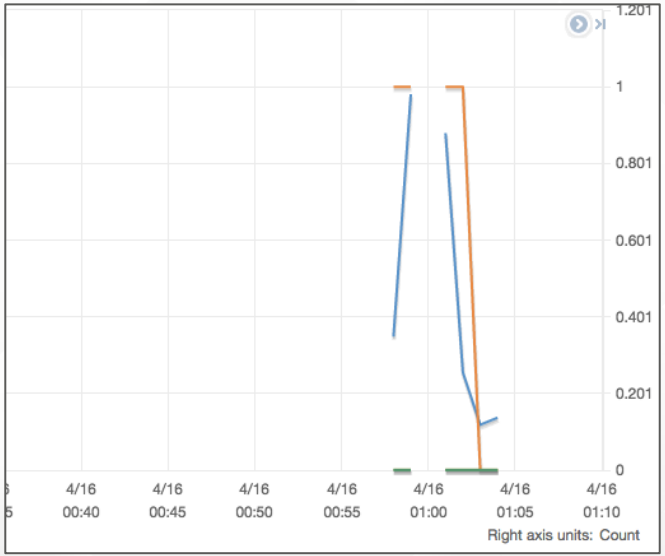
クロスアカウントアクセス



- 異なるAWSアカウントのアプリケーションからのアクセスが可能
- Lambdaファンクションのオーナーが Access Policyでクロスアカウントアクセスを許可する設定を追加する

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

Lambdaファンクションの書き方

amazon web services

基本

- Node.jsのベストプラクティスに従う
  - AWS SDK、ImageMagickは組み込み済み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、ソケットを利用可能
  - 各種ライブラリを利用可能
  - インバウンドのソケット接続は不可能

コード例

```
console.log('Loading function');
var aws = require('aws-sdk');
var s3 = new aws.S3({apiVersion: '2006-03-01'});

exports.handler = function(event, context) {
    console.log('Received event:', JSON.stringify(event, null, 2));
    //渡されたイベントの情報からオブジェクトを取得しContentTypeを出力
    var bucket = event.Records[0].s3.bucket.name;
    var key = event.Records[0].s3.object.key;
    s3.getObject({Bucket: bucket, Key: key}, function(err, data) {
        if (err) {
            console.log("Error getting object " + key + " from bucket " + bucket +
                ". Make sure they exist and your bucket is in the same region as this function.");
            context.fail('Error', "Error getting file: " + err);
        } else {
            console.log('CONTENT TYPE:', data.ContentType);
            context.succeed();
        }
    });
};
```

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

DEMO

Amazon Web Services

Compute

- **EC2** — Virtual Servers in the Cloud
- **Lambda** — Run Code in Response to Events
- **EC2 Container Service** — Run and Manage Docker Containers

Storage & Content Delivery

- **S3** — Scalable Storage in the Cloud
- **Storage Gateway** — Integrates On-Premises IT Environments with Cloud Storage
- **Glacier** — Archive Storage in the Cloud
- **CloudFront** — Global Content Delivery Network

Database

- **RDS** — MySQL, Postgres, Oracle, SQL Server, and Amazon Aurora
- **DynamoDB** — Predictable and Scalable NoSQL Data Store
- **ElastiCache** — In-Memory Cache
- **Redshift** — Managed Petabyte-Scale Data Warehouse Service

Networking

- **VPC** — Isolated Cloud Resources
- **Direct Connect** — Dedicated Network Connection to AWS
- **Route 53** — Scalable DNS and Domain Name Registration

Administration & Security

- **Directory Service** — Managed Directories in the Cloud
- **Identity & Access Management** — Access Control and Key Management
- **Trusted Advisor** — AWS Cloud Optimization Expert
- **CloudTrail** — User Activity and Change Tracking
- **Config** — Resource Configurations and Inventory
- **CloudWatch** — Resource and Application Monitoring

Deployment & Management

- **Elastic Beanstalk** — AWS Application Container
- **OpsWorks** — DevOps Application Management Service
- **CloudFormation** — Templated AWS Resource Creation
- **CodeDeploy** — Automated Deployments

Analytics

- **EMR** — Managed Hadoop Framework
- **Kinesis** — Real-time Processing of Streaming Big Data
- **Data Pipeline** — Orchestration for Data-Driven Workflows
- **Machine Learning** — Build Smart Applications Quickly and Easily

Application Services

- **SQS** — Message Queue Service
- **SWF** — Workflow Service for Coordinating Application Components
- **AppStream** — Low Latency Application Streaming
- **Elastic Transcoder** — Easy-to-use Scalable Media Transcoding
- **SES** — Email Sending Service
- **CloudSearch** — Managed Search Service

Mobile Services

- **Cognito** — User Identity and App Data Synchronization
- **Mobile Analytics** — Understand App Usage Data at Scale
- **SNS** — Push Notification Service

Enterprise Applications

- **WorkSpaces** — Desktops in the Cloud
- **WorkDocs** — Secure Enterprise Storage and Sharing Service
- **WorkMail** PREVIEW — Secure Email and Calendaring Service

Resource Groups

A resource group is a collection of resources that share one or more tags. Create a group for each project, application, or environment in your account.

Create a Group | Tag Editor

Additional Resources

Getting Started
See our documentation to get started and learn more about how to use our services.

AWS Console Mobile App
View your resources on the go with our AWS Console mobile app, available from Amazon Appstore, Google Play, or iTunes.

AWS Marketplace
Find and buy software, launch with 1-Click and pay by the hour.

AWS Summit - San Francisco
Learn about the exciting new services and features announced at the AWS Summit in San Francisco

Service Health

All services operating normally.

Updated: May 23 2015 01:54:00 GMT+0900

Service Health Dashboard

Lambdaファンクション作成時の注意事項

Lambdaファンクション作成時の注意事項

- ロールを正しくセットアップする
  - Execution rolesはLambdaファンクションがアクセスするAWSリソースに対する権限を与える
  - Kinesis、DynamoDBをイベントソースとする場合はLambdaがポーリングできるよう権限を与える

- 再帰処理は慎重に
  - イベントハンドラ内部でS3やDynamoDBへオブジェクトの書き込みを行うと、別のイベントがトリガーされ無限ループに陥る可能性がある

- Node.jsでは処理が非同期で実行されることを意識する

Kinesisのイベントを処理する場合の例

- KinesisやDynamoDBのイベントはBatchサイズを指定することで複数レコードを処理可能

- 1イベントに含まれる複数レコードに対してループ処理を行う場合、ループ内で別の非同期な処理を呼ぶとコールバックを受け取れない
  - async等のライブラリの利用が必須

KinesisのレコードをDynamoDBへ1件ずつ登録する例

```
var AWS = require('aws-sdk');
var async = require('async');
var dynamodb = new AWS.DynamoDB();

exports.handler = function(event, context) {
    async.eachSeries(event.Records, function(v, callback){
        var encodedPayload = v.kinesis.data;
        var payload = new Buffer(encodedPayload, 'base64').toString('ascii');
        var data = JSON.parse(payload);
        var params = {
            Item: {
                someKey: {  N: 'STRING_VALUE'},
            },
            TableName: 'STRING_VALUE'
        };
        dynamodb.putItem(params, function(err) {
            if (err) console.log(err, err.stack); callback();
        });
    }, function(err,result){
        if (err) {
            context.fail(err);
        } else {
            context.succeed('success');
        }
    });
}
```

- DynamoDBのputItem()は非同期のため、Array.forEach()ではコールバックを実行できず意図した結果にならない
- asyncを使うことで非同期処理の順序制御が可能


Deep Dive into AWS Lambda
amazon
web services

コンテナのライフサイクルについて

amazon web services

コンテナのライフサイクル

- Lambdaファンクションはコンテナとして実行される

- コンテナのライフサイクルとして開始・終了・再利用の3つのサイクルがある

開始

- ファンクション作成後、もしくはコードや設定更新後の初回実行時は新たにコンテナが作成され、ファンクション用のコードがコンテナ内にロードされる

- Node.js側では、ハンドラが最初に呼び出される前に初期化コードがコンテナの生成ごとに一度だけ実行される

終了

- 複数の終了パターン
  - Timeout
    - ユーザが指定した実行時間を超えた場合。その時どういう処理が行われているかは関係なく、即時停止される
  - Controlled termination
    - コールバックの1つがcontext.done()を呼び出して終了した場合。この時点で他のコールバックが実行されてるかに関わらず終了する
  - Default termination
    - 全てのコールバックが終了してファンクション自体が終了した場合。この時、context.done()を実行していなくてもログに“Process exited before completing request”と出力される
- クラッシュもしくはprocess.exit()を呼び出すことでも終了
  - 例えば不具合のあるライブラリが含まれていて、セグメンテーション違反を起こした場合、そのコンテナは実行終了となる

再利用

- 実行からある程度時間が経過した後に再度実行される場合は新たにコンテナが作成される
  - 初回実行時と同様

- コードの変更がなく前回の実行から時間が立っていない場合は以前のコンテナを再利用することがある
  - Node.jsの初期化処理をスキップできるなどパフォーマンス上のアドバンテージ
  - 再利用された場合、最後に/tmpに書き込んだ内容も残っているがあてにはしないこと


プロセスの凍結と再開
amazon
web services

プロセスの凍結と再開

- ファンクションの終了時に実行中のバックグラウンドプロセスがある場合、Lambdaはプロセスをfreezeさせ、次回ファンクションを呼び出した際に再開する
  - ただし、コンテナが再利用される場合だけであり保証はされていない
- この場合、バックグラウンドプロセスは残っていても処理は行われていない
  - プロセス再作成のオーバーヘッドを減らせる

制限事項

amazon web services

制限事項（リソース）

リソース	制限
/tmpスペース	512MB
ファイルディスクリプタ数	1024
プロセス数およびスレッド数（合計）	1024
同時リクエスト数	100
アカウントにおける1秒あたりのリクエスト数	1000
1リクエストあたりの実行時間	60秒

制限事項（インプット）

インプット	制限
zipファイル（圧縮）	30MB
zipファイル（無圧縮）	250MB
Invokeのリクエストボディ（JSON）	6MB
アクティブでないコードストレージの保持期間	90日
コードストレージの容量	1.5GB

料金

料金体系

- リクエスト (全リージョン)
  - 月間100万リクエストまでは無料
  - 超過分は$0.20/100万リクエスト (1リクエストあたり$0.0000002)
- 実行時間 (全リージョン)
  - 100ms単位で課金
  - 100ms以下は繰り上げで計算
  - メモリー容量により単価および無料時間が異なる

- かなり複雑なのでこちらも参考に
  - http://qiita.com/Keisuke69/items/e3f79b50b6039175401b

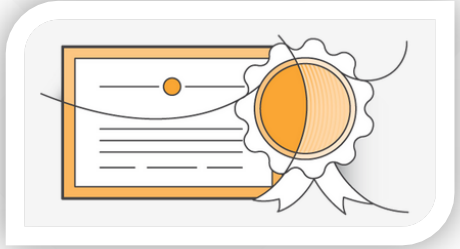
Memory (MB)	Price per 100ms ($)	Free tier seconds per month
128	0.000000208	3,200,000
192	0.000000313	2,133,333
256	0.000000417	1,600,000
320	0.000000521	1,280,000
384	0.000000625	1,066,667
448	0.000000729	914,286
512	0.000000834	800,000
576	0.000000938	711,111
640	0.000001042	640,000
704	0.000001146	581,818
768	0.00000125	533,333
832	0.000001354	492,308
896	0.000001459	457,143
960	0.000001563	426,667
1024	0.000001667	400,000

Run Code in the cloud!

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