

# サーバーレスでプログラムを動かす

Lambda を使うと、EC2 インスタンスを使うことなくプログラムを稼働できます。

常時実行している必要がないサービスは、Lambda を使って構築すると、コストを削減できるばかりか、開発工数も抑えられます。

## ○ファイルが保存されたときに、サムネイルを作りたい

### 【導入】

N 社では、Web ブラウザーで参照できる商品カタログを作成しています。商品カタログの運用には、S3 の静的ウェブホスティング機能を使っており、商品の写真を S3 にアップロードしています。

オリジナルの商品写真はサイズが大きいので、サムネイルも一緒にアップロードしています。サムネイルは担当者がフォトレタッチソフトを用いて、手作業で作っているので、とても手間がかかっています。

この手間を軽減するため、画像をアップロードしたら、自動的にサムネイルが作れるといいと考えています。

## ○利用サービス

S3、Lambda

## ○解説

AWS 以外のソリューションで、こうした機能を実現する場合、すぐ思いつく方法は、サムネイルを作成する Web アプリケーションを作り、Web フォームから画像をアップロードする手法です（図 3-3-1）。

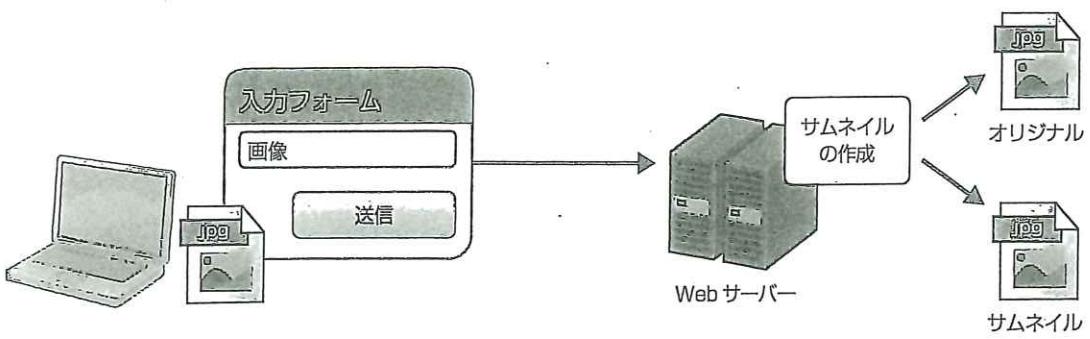


図 3-3-1 Web アプリケーションでサムネイルを作る

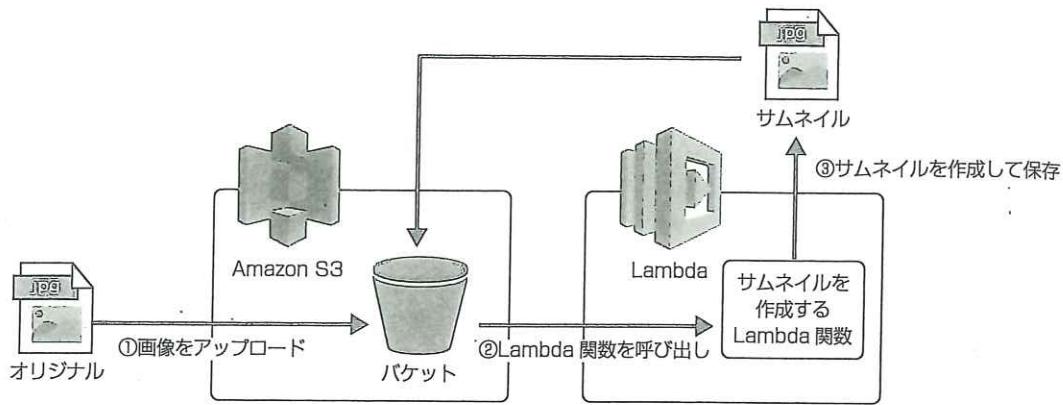


図 3-3-2 Lambda 関数を使ってサムネイルを作成する

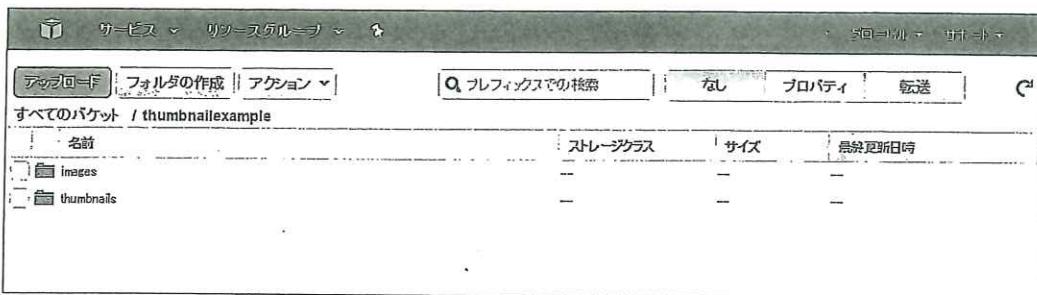


図 3-3-3 S3 バケットを作成し、images ディレクトリーと thumbnails ディレクトリーを作成しておく

しかし AWS なら、もっとシンプルな方法で作れます。それは、Lambda を使う方法です。

S3 にファイルをアップロードすると、それをイベントのトリガにして Lambda 関数を実行できます。つまり、サムネイルを作成して保存するプログラムを Lambda 関数として作っておけば、画像がアップロードされたタイミングでサムネイルを作れます（図 3-3-2）。

この方法だと、S3 に画像を保存する方法は問いません。AWS マネジメントコンソールでも、Cyberduck のようなツールでも、もしくは、CLI や何か開発したアプリケーションなどからでも、S3 に画像が保存されれば、サムネイルが作成されます。

## ○手順

### 【S3 バケットを準備する】

まずは、何かしらの S3 バケットを用意しておきます。

そして、この images ディレクトリーに JPEG 形式のファイルを保存したときに、thumbnails ディレクトリーに同名でサムネイル画像が作られるようにします。images ディレクトリーと thumbnails ディレクトリーをあらかじめ作成しておいてください（図 3-3-3）。

### 【Lambda 関数を実行するロールを作成する】

Lambda 関数は、何らかの IAM ロールの権限で実行されます。これから作る Lambda 関数を実行するには、次の権限が必要です。

① Lambda 関数自体を実行するための基本的な権限

→ AWSLambdaBasicExecutionRole ポリシー

② 図 3-3-3 で作成した S3 バケットに対するフルアクセス権

→ AmazonS3FullAccess ポリシー

**裏技** AmazonS3FullAccess ポリシーは、すべての S3 バケットに対するアクセス権を与えるため、この権限を与えることは、あまり望ましい選択肢ではありません。本書では、話を簡単にするために、AmazonS3FullAccess ポリシーを使いますが、より安全に使うには、図 3-3-3 の S3 バケットにだけアクセスできるカスタムポリシーを作成して、そのポリシーを適用するのが望ましい運用です。

そこで、これらの権限を持つ IAM ロールを作成します。まずは、AWS マネジメントコンソールの IAM 画面から [ロール] メニューをクリックして開き、[新しいロールの作成] をクリックしてください（図 3-3-4）。

ロール名を入力します。ここでは「myLambdaS3Role」としておきます（図 3-3-5）。

ロールタイプを選びます。ここでは、Lambda 関数が AWS のさまざまなサービスを呼び出せるようにするロールを作りたいので [AWS Lambda] を選択します（図 3-3-6）。

次にポリシーをアタッチします。ここでは、二つのポリシーをアタッチします。

一つは、「AWSLambdaBasicExecutionRole」です。これは Lambda のログ出力先となる CloudWatch Logs へのアクセス権を付与します。フィルターのところに「lambda」と入力すると、lambda という文字列を含むポリシーが表示されるので、その中から選びます（図 3-3-7）。

そしてそのまま今度はフィルターのところに「S3」と入力して S3 関連のポリシーを表示し、「AmazonS3FullAccess」にチェックを付けます。これが S3 に対するフルアクセス権となります。

AWSLambdaBasicExecutionRole と AmazonS3FullAccess の二つにチェックを付けたなら、[次のステップ] をクリックして、次の画面に進んでください（図 3-3-8）。

すると確認画面が表示されます。[ロールの作成] ボタンをクリックして、ロールの作成を完了してください（図 3-3-9）。

### 【Lambda 関数を作る】

ロールができたら、Lambda 関数を作成していきます。AWS マネジメントコンソールで [コンピューティング] のなかの [Lambda] を開いてください（図 3-3-10）。

まだ Lambda 関数を一つも作っていないときは、図 3-3-11 の画面が表示されるので [Get Started Now] をクリックしてください。

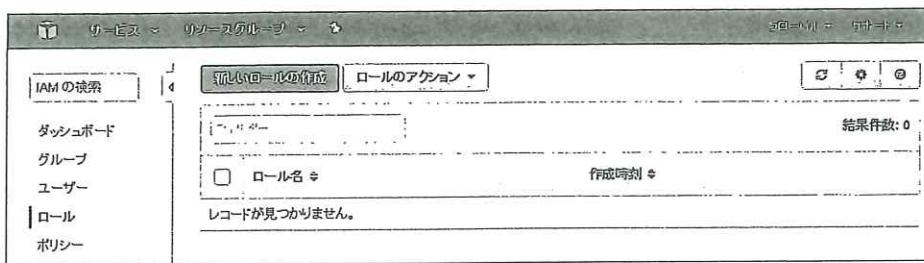


図 3-3-4 新しいロールを作成する

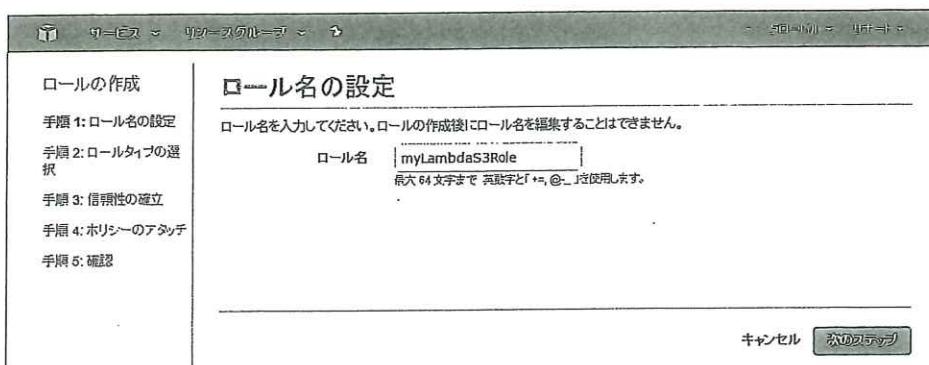


図 3-3-5 ロール名を付ける

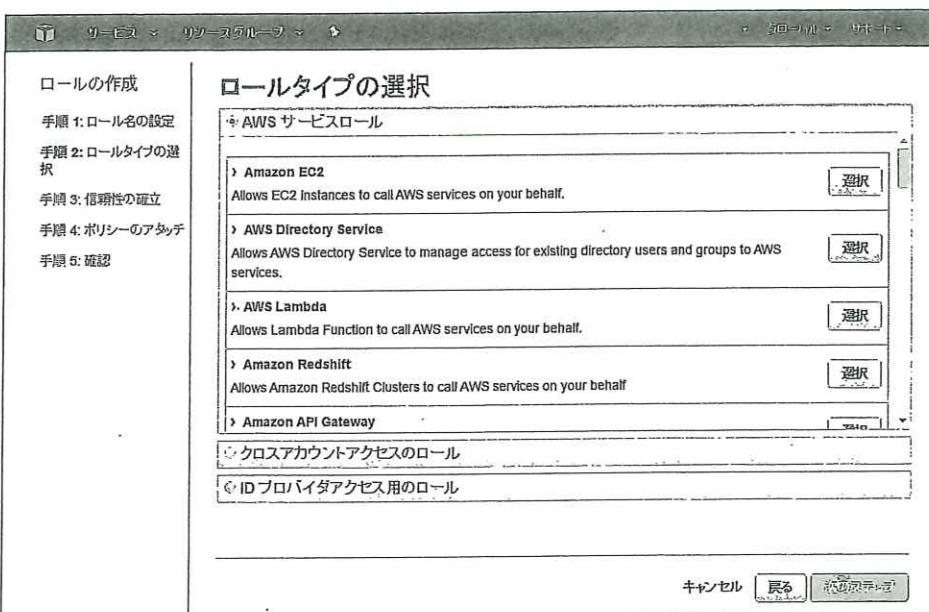


図 3-3-6 AWS Lambda ロールタイプを選択する

サービス リソースグループ

手順 1: ロール名の設定  
手順 2: ロールタイプの選択  
手順 3: 信頼性の確認  
手順 4: ポリシーのアタッチ  
手順 5: 確認

### ポリシーのアタッチ

アッタチするポリシーを1個以上選択してください。ロールは、それぞれ10個までのポリシーをアタッチできます。

ポリシー名	アッタチされたエンティティ	作成時刻	権限時刻
AWSLambdaBasic...	0	2015-04-10 00:0...	2015-04-10 0...
AWSLambdaDyna...	0	2015-04-10 00:0...	2015-04-10 0...
AWSLambdaExec...	0	2015-02-07 03:4...	2015-02-07 0...
AWSLambdaFullA...	0	2015-02-07 03:4...	2016-05-10 0...
AWSLambdaInvoc...	0	2015-02-07 03:4...	2015-02-07 0...
AWSLambdaKines...	0	2015-04-10 00:1...	2015-04-10 0...
AWSLambdaRead...	0	2015-02-07 03:4...	2016-05-10 0...
AWSLambdaRole	0	2015-02-07 03:4...	2015-02-07 0...
AWSLambdaVPC...	0	2016-02-12 08:1...	2016-02-12 0...

キヤンセル 戻る 次のタブ

図 3-3-7 AWSLambdaBasicExecutionRole を選ぶ

サービス リソースグループ

手順 1: ロール名の設定  
手順 2: ロールタイプの選択  
手順 3: 信頼性の確認  
手順 4: ポリシーのアタッチ  
手順 5: 確認

### ポリシーのアタッチ

アッタチするポリシーを1個以上選択してください。ロールは、それぞれ10個までのポリシーをアタッチできます。

ポリシー名	アッタチされたエンティティ	作成時刻	権限時刻
AmazonDMSReds...	0	2016-04-21 02:0...	2016-04-21 0...
AmazonS3FullAcc...	0	2015-02-07 03:4...	2015-02-07 0...
AmazonS3ReadO...	0	2015-02-07 03:4...	2015-02-07 0...

キヤンセル 戻る 次のタブ

図 3-3-8 AmazonS3FullAccess を選ぶ

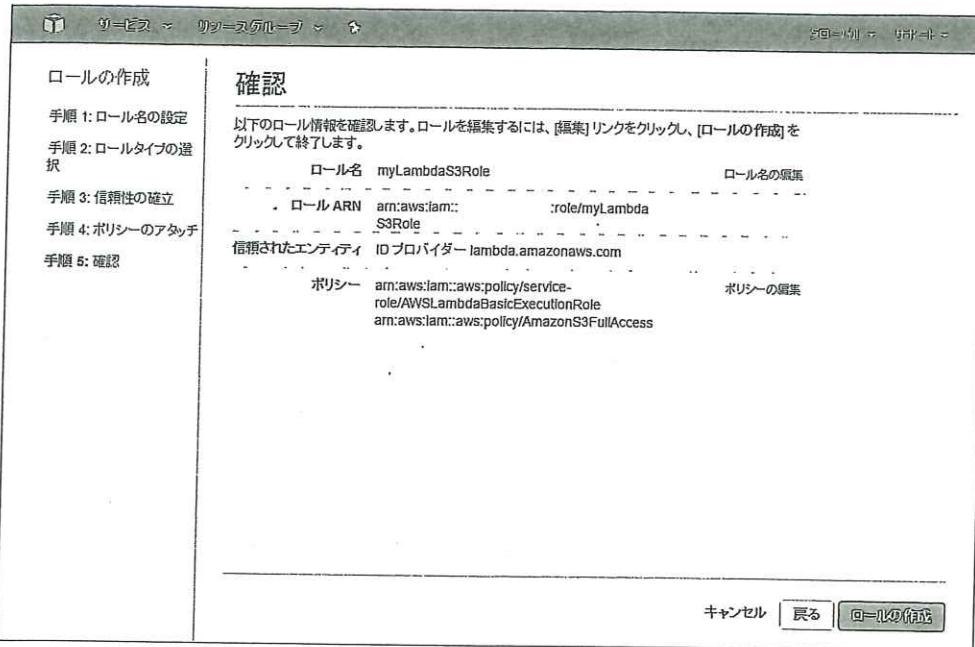


図 3-3-9 作成するロールの確認

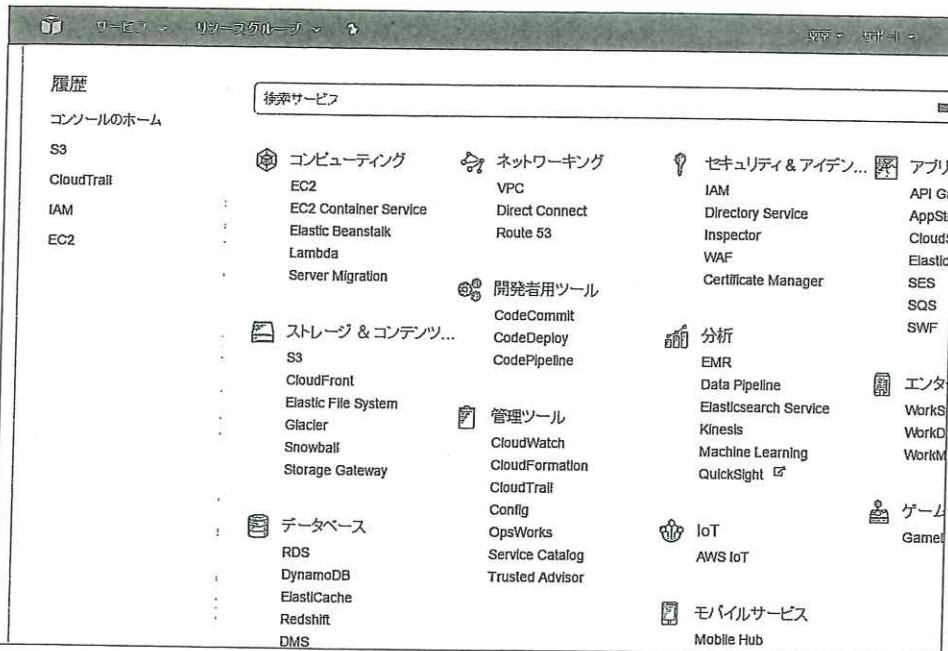


図 3-3-10 Lambda を開く

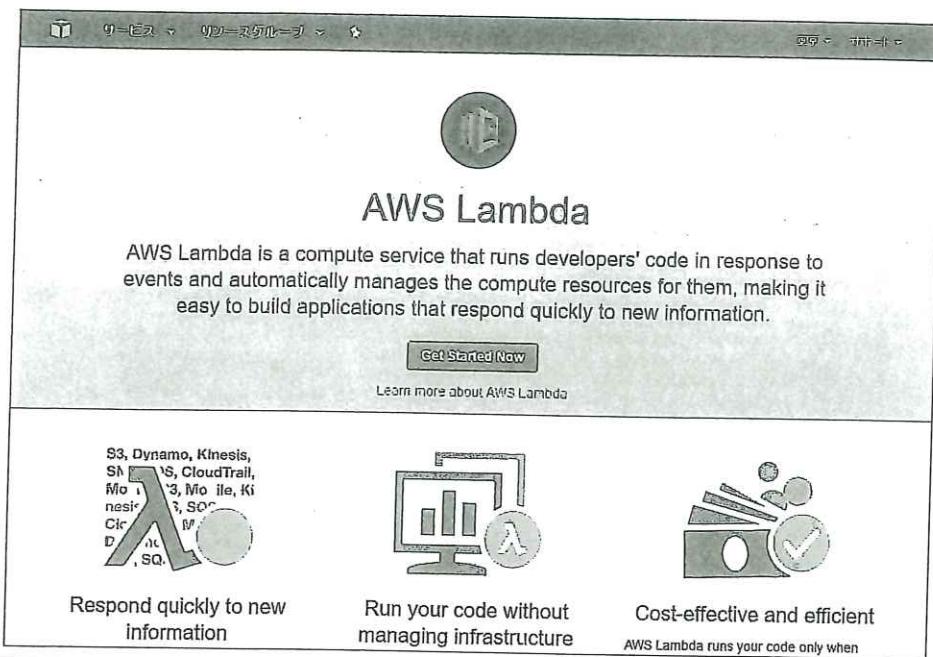


図 3-3-11 Get Started Now をクリックする

【参考】すでに作っているときは [Functions] メニューから [Create a Lambda Function] をクリックすることで作成してください（後掲の図 3-3-3 を参照）。

すると、Lambda 関数の作成画面になります。Lambda 関数は「blueprint」と呼ばれるひな形（青写真）を基に作ることができます。Lambda 関数を作るときは、最も機能が近い blueprint を選ぶのが簡単です。

S3 に関する blueprint は、2016 年 11 月時点で二つあり、フィルターのところに「s3」と入力すると、S3-get-object と s3-get-object-python の二つに絞り込みます。ここでは、Node.js (JavaScript) を利用した「s3-get-object」を選択します（図 3-3-12）。

すると、トリガの選択画面（図 3-3-13）になります。次のように設定してください。

#### ・ Bucket

対象とするバケットを選択します。図 3-3-3 で事前に用意しておいたバケットを選択してください。

#### ・ Event type

トリガの種類を選択します。ここでは、ファイルが配置されたときにトリガを実行したいので「Object Created (ALL)」を選択してください。

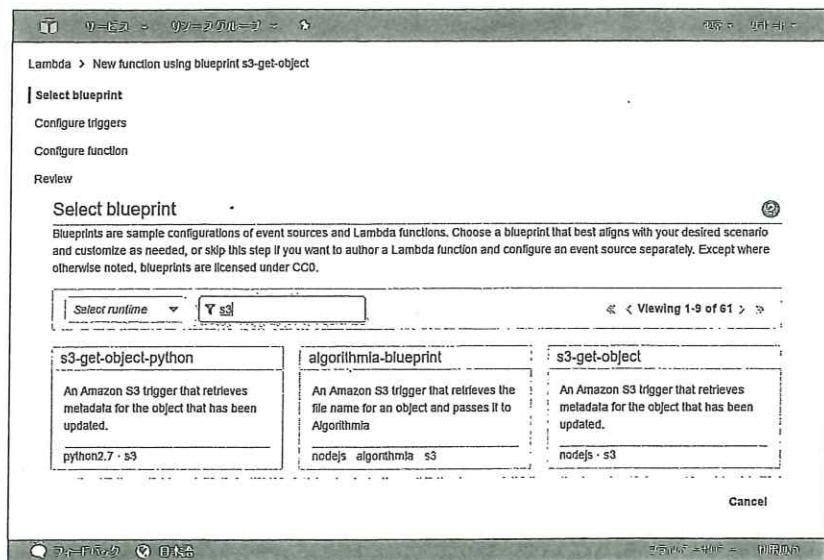


図 3-3-12 s3-get-object を選択する

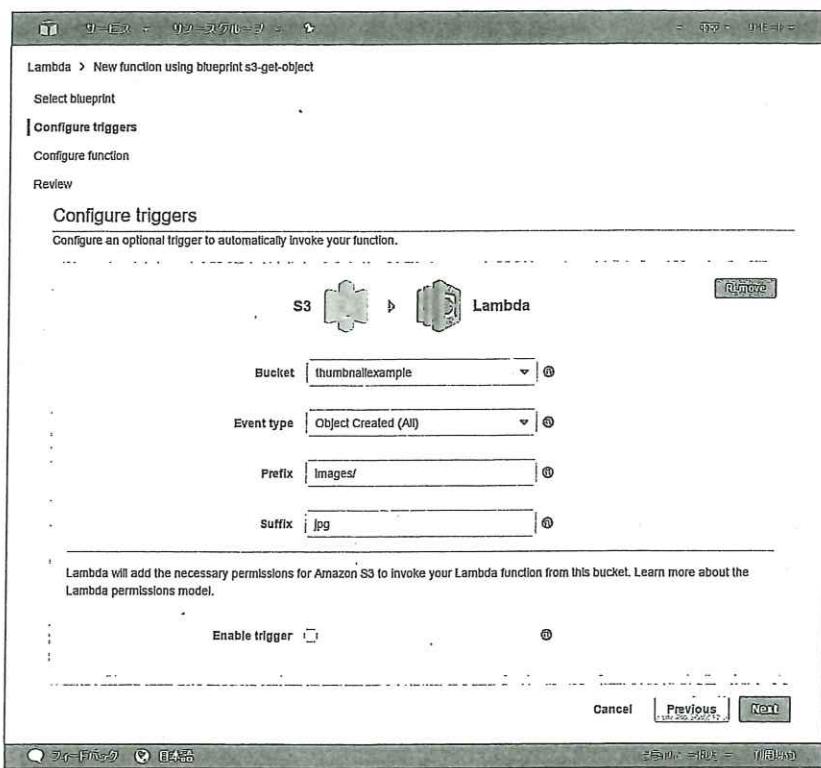


図 3-3-13 トリガを設定する

**Configure function**

A Lambda function consists of the custom code you want to execute. Learn more about Lambda functions.

Name*	myThumbnailFunc
Description	make thumbnail
Runtime*	Node.js 4.3

図 3-3-14 Configure function

・Prefix

トリガをかけるファイル名の接頭辞を設定します。ここでは「images/ ファイル名.jpg」のように「images から始まるファイルが置かれたとき」に、トリガを発生したいので、「images/」と入力してください。

・Suffix

トリガをかけるファイル名の接尾辞を設定します。ここでは「ファイル名.jpg」のように末尾が「jpg」で終わるファイルが置かれたとき」にトリガを発生したいので「jpg」と入力してください。

・Enable trigger

作成したトリガをすぐに有効にするかどうかの設定です。チェックを付けると、すぐにこのトリガが有効になりますが、多くの場合、コードを記述して少しデバッグしてから有効にしたほうがよいので、ここではチェックを付けないでおきます。

次の画面では、Lambda 関数を作り始めます。次の設定があります。

### ① Configure function

Lambda 関数名や実行環境を設定します。

Name には、任意の名前を入力します（図 3-3-14）。ここでは「myThumbnailFunc」という名前にしました。Description は解説文です。何でもかまいません。

Runtime は実行環境です。Lambda 関数は「Java」「Python」「Node.js (JavaScript)」のいずれかの言語を使って記述できます。ここでは「Node.js 4.3」を選択しました。

### ② Lambda function code

ここに Lambda 関数のコードを書きます。「Edit code inline」を選択すると、この画面のテキストボックスでコードを書けます（図 3-3-15）。

それ以外に「Upload a ZIP file」を選択して ZIP ファイルをアップロードしたり、「Upload a file from Amazon S3」を選択して S3 にアップロードしたファイルを取り込んだりできます。

blueprint から選んだときは、ここにひな形のプログラムが書かれているので、これをあとで編集することにして、このままの状態にしておきます。

Lambda function code

Provide the code for your function. Use the editor if your code does not require custom libraries (other than the aws-sdk). If you need custom libraries, you can upload your code and libraries as a ZIP file. Learn more about deploying Lambda functions.

Code entry type	Edit code inline	▼
-----------------	------------------	---

```

5  const aws = require('aws-sdk');
6
7  const s3 = new aws.S3({ apiVersion: '2006-03-01' });
8
9
10 exports.handler = (event, context, callback) => {
11   //console.log('Received event:', JSON.stringify(event, null, 2));
12
13   // Get the object from the event and show its content type
14   const bucket = event.Records[0].s3.bucket.name;
15   const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));
16
17   const params = {
18     Bucket: bucket,
19     Key: key,
20   };
21   s3.getObject(params, (err, data) => {
22     if (err) {
23       console.log(err);
24       const message = `Error getting object ${key} from bucket ${bucket}. Make sure they exist and you
25       console.log(message);
26       callback(message);
27     } else {
28       console.log('CONTENT TYPE:', data.ContentType);
29       callback(null, data.ContentType);
30     }
31   });
32 };

```

You can define Environment Variables as key/value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more:

Environment variables	Key	Value
-----------------------	-----	-------

図 3-3-15 Lambda function code

**メモ》** インラインで書けるのは、標準ライブラリしか利用しない場合です。何かライブラリを使いたいときは、そのライブラリを含んだ ZIP ファイルを作成し、ライブラリも一緒にアップロードする必要があります。

**メモ》** その下にある「Environment variables」は、Lambda 関数に対して環境変数を設定できる機能です。外部から Lambda 関数に対して設定情報を与えたいときなどに利用できます。

### ③ Lambda function handler and role

Lambda 関数の実行ハンドラとロールを設定します。

#### (a) Handler

プログラムを実行するときに呼び出すメソッド名を設定します。デフォルトは「index.handler」となっています。これは「handler」というメソッドを実行するという意味です。

blueprint のコードは、メインルーチンが次のように定義されています。

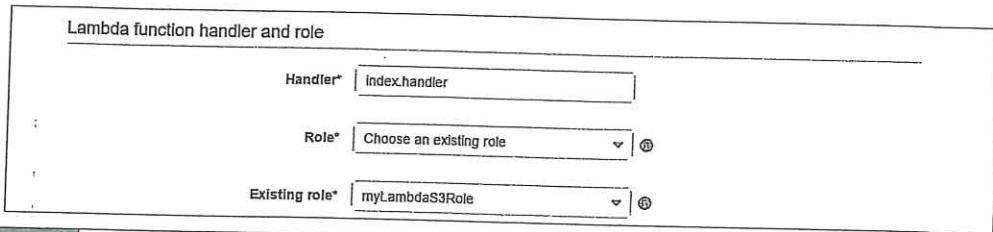


図 3-3-16 Lambda function handler and role

```
exports.handler = (event, context, callback) {
...プログラム...
}
```

「index.handler」は、この「handler」を指しています。もし、プログラムのメソッド名を変更したときは、この設定をそれに合わせる必要があります。

#### (b) Role, Role name, Policy templates

この Lambda 関数を実行するロールを指定します。三つの選択肢があります。

- Choose an existing role

既存のロールから選択します。

- Create new role from template(s)

Policy tempaltes で選択したポリシーテンプレートをひも付けた新しいロールを作ります。作成するロール名は、「Role name」のところで決めます。

- Create a custom role

カスタムロールを指定して、新しいロールを作成します。このとき作成するロール名は「Role name」のところで決めます。

ここではすでにロールを作っているので、[Choose an existing role] を選び、先に作成しておいた myLambdaS3Role を選択してください（図 3-3-16）。

#### ④ Advanced settings

Lambda の実行環境を設定します。

Memory の部分で、割り当てるメモリーを設定します。最小値は 128MB で、デフォルトはそれに設定されていますが、ここで作成する Lambda 関数は画像を扱うため、それではメモリーが足りないかもしれません。そこでここでは、256MB を設定することにします。メモリーを増やすと、利用可能な CPU パワーも容量に比例して増加します。

**Advanced settings**

These settings allow you to control the code execution performance and costs for your Lambda function. Changing your resource settings (by selecting memory) or changing the timeout may impact your function cost. Learn more about how Lambda pricing works.

Memory (MB)*	256	④		
Timeout*	0	min	15	sec
All AWS Lambda functions run securely inside a default system-managed VPC. However, you can optionally configure Lambda to access resources, such as databases, within your custom VPC. Learn more about accessing VPCs within Lambda. Please ensure your role has appropriate permissions to configure VPC.				
VPC	No VPC	④		
Environment variables are encrypted at rest using a default Lambda service key. You can change the key below to one of your account's keys or paste in a full KMS key ARN.				
KMS key	(default) aws/lambda	④		

\* These fields are required.

Cancel Previous Next

図 3-3-17 Advanced settings

Lambda 関数は、割り当てたメモリーと実行時間によって金額が異なります。大きなメモリーを割り当てるごとに費用が高くなります。

「VPC」の項目は、この Lambda 関数を VPC の内部で実行するときに指定します。例えば、EC2 インスタンスや RDS などと通信したいときは、VPC を指定する必要があります。ここで作る Lambda 関数は、VPC 上のリソースと通信する必要がないので、ここでは [No VPC] とします。

なお、Lambda から VPC 上のリソースにアクセスする場合は、特有の制約があります。設計上やむを得ない場合は [https://docs.aws.amazon.com/ja\\_jp/lambda/latest/dg/vpc.html](https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/vpc.html) を参照して、制約事項を確認しておくことをお勧めします。

[KMS key] の項目は、環境変数を、どのキーで暗号化するかという設定です。デフォルトの [aws/lambda] にしておきます（図 3-3-17）。

以上を設定すると、確認画面が表示されます。[Create function] をクリックして Lambda 関数を作成してください（図 3-3-18）。

すると、Lambda 関数が作られます。この時点では、まだトリガが [Disable] になっており、無効になっていることが分かります（図 3-3-19）。

### 【サムネイルを作成するコードを記述する】

作成した Lambda 関数は、[Code] タブで編集できます。

サムネイルを作成するコードを List 3-3-1 に示します（図 3-3-20）。これを入力して、[Save] ボタンをクリックしてください。

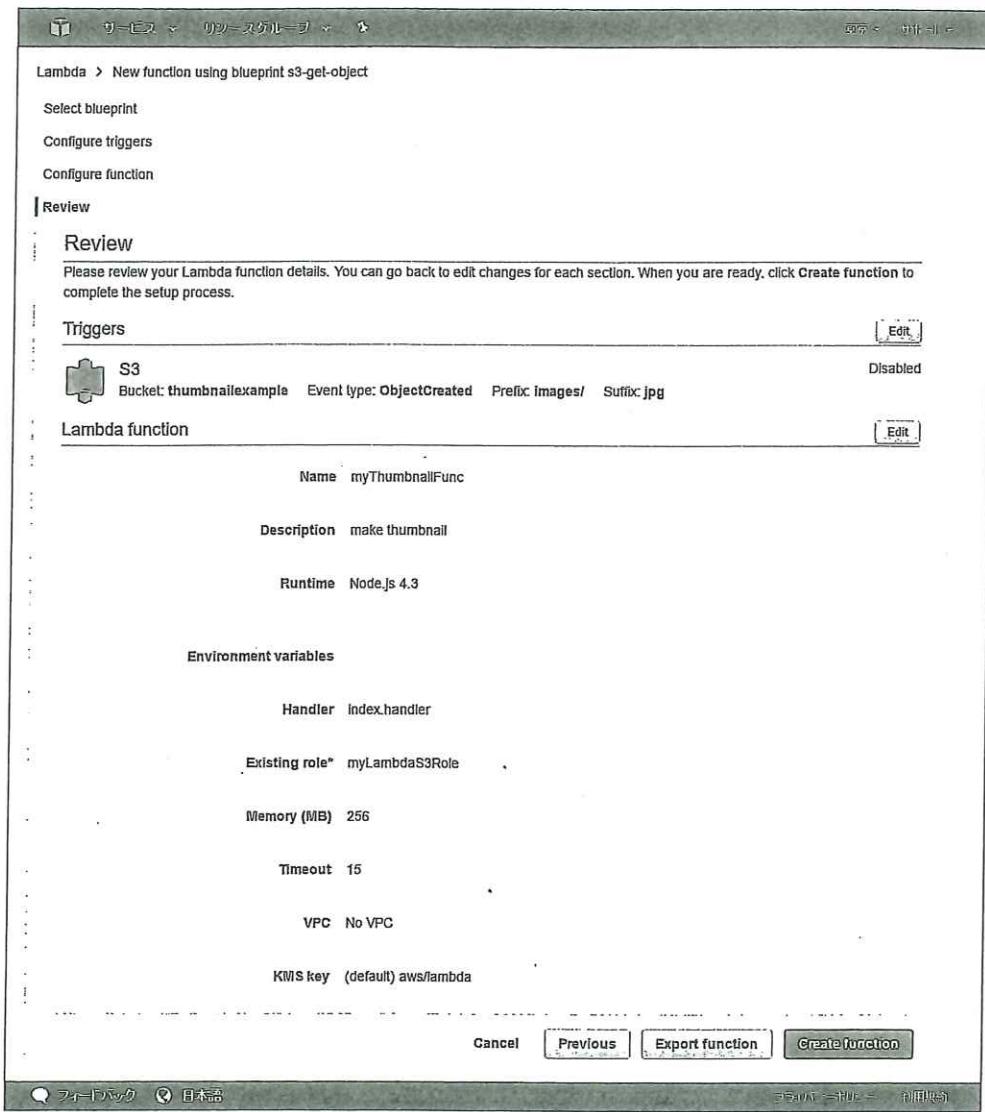


図 3-3-18 Lambda 関数を作成する

**メモ》** 何かコードを変更すると [Save] ボタンが表示されます。変更していないときは [Test] ボタンしかありません。

### List 3-3-1 サムネイルを作成するための Lambda 関数

```
'use strict';
console.log('Loading function');
const aws = require('aws-sdk');
```

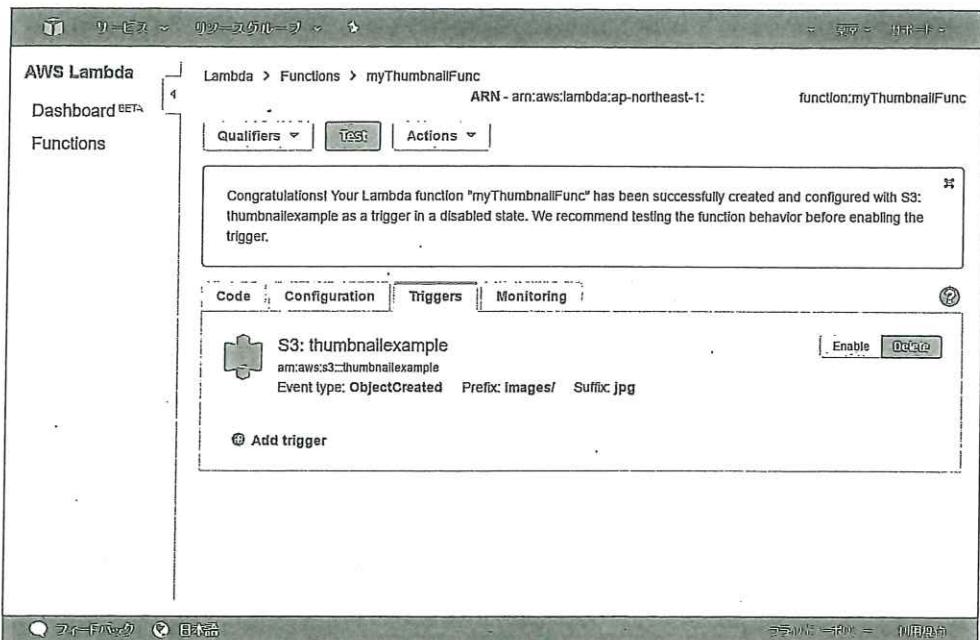


図 3-3-19 作成された Lambda 関数

```
1 'use strict';
2
3 console.log('Loading function');
4 const aws = require('aws-sdk');
5 const s3 = new aws.S3({ apiVersion: '2006-03-01' });
6 const im = require('imagemagick');
7
8 exports.handler = (event, context, callback) => {
9   // デバッグ用にeventとコンテキストを出力
10   console.log(JSON.stringify(event, null, 2));
11   console.log(JSON.stringify(context, null, 2));
12
13   // バケット名とキー名(ファイル名)
14   const bucket = event.Records[0].s3.bucket.name;
15   const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));
16   // 設置されたデータを取得
17   s3.getObject({
18     Bucket: bucket,
19     Key: key
20   }, (err, data) => {
21     if (err) {
```

図 3-3-20 サムネイルを作るためのコードを記述する

```

const s3 = new aws.S3({ apiVersion: '2006-03-01' });
const im = require('imagemagick');

exports.handler = (event, context, callback) => {
    // デバッグ用にeventとコンテキストを出力
    console.log(JSON.stringify(event, null, 2));
    console.log(JSON.stringify(context, null, 2));

    // バケット名とキー名(ファイル名)
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));
    // 配置されたデータを取得
    s3.getObject({
        Bucket: bucket,
        Key: key
    }, (err, data) => {
        if (err) {
            console.log(err);
            callback(`Cannot read ${key} from bucket ${bucket}`);
        } else {
            // 幅320pxのサムネイルを作成
            im.resize({
                srcData: data.Body,
                format: "jpeg",
                width: 320
            }, function(err, stdout, stderr) {
                if (err) {
                    console.log(err);
                    callback('Cannot create thumbnail.');
                } else {
                    // 作成したサムネイルを保存
                    var thumbnailkey = key.replace("images/", "thumbnails/");
                    s3.putObject({
                        Bucket: bucket,
                        Key: thumbnailkey,
                        Body: new Buffer(stdout, "binary"),
                        ContentType: data.ContentType
                    }, function(err, res) {
                        if (err) {
                            callback(`Cannot write ${thumbnailkey}`);
                        } else {
                            // 成功
                            callback(null);
                        }
                    });
                }
            });
        }
    });
};


```

プログラムの冒頭では、利用するライブラリを読み込んでいます。ここではawsのライブラリ、S3のライブラリ、そして、画像を縮小するためのImageMagickライブラリを読み込みました。

**※注** ImageMagick は、Lambda 環境に標準で組み込まれているため、このようにして読み込みます。標準外のライブラリを使いたいときは、この画面でソースコードを記述する方法ではなく、ソースコードとライブラリを ZIP 形式でまとめて、それをアップロードするという方法で Lambda 関数を作る必要があります。特定のバージョンの ImageMagick を利用したい場合は、あえて利用したいバージョンの ImageMagick を自分でアップロードしても構いません。

```
const aws = require('aws-sdk');
const s3 = new aws.S3({ apiVersion: '2006-03-01' });
const im = require('imagemagick');
```

なお、ソースコードの先頭にある、

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

というのは、コンソールに出力したいメッセージです。console.log で出力した内容は、CloudWatch Logs（あとで説明します）で参照できるので、デバッグ時に役立ちます。

Lambda 関数の本体は、次のように指定した handler メソッドです。イベントが発生したときには、このメソッドが呼び出されます。

```
exports.handler = (event, context, callback) => { }
```

event 引数はイベントに関するデータが格納されています。データの内容は、トリガとなったイベントの種類によって異なります。例えば S3 ならバケット名などが含まれていますし、SES ならメールアドレスなどが含まれているという具合です。

context 引数はイベントのコンテキスト情報です。そして callback 引数は結果を返すときに使うメソッドです。

S3 バケットに対する書き込みイベントから Lambda 関数が呼び出された場合、書き込まれたデータのバケット名とキー名（ファイル名）は、event 引数から、次のようにして取得できます。

```
const bucket = event.Records[0].s3.bucket.name;
const key =
  decodeURIComponent(
    event.Records[0].s3.object.key.replace(/\+/g, ' '));
```

このバケットからデータを取得するには、getObject メソッドを呼び出します。

```
s3.getObject({
  Bucket: bucket,
  Key: key
}, (err, data) => {
...処理...
})
```

失敗した場合は、errにエラーメッセージが設定されます。エラーが発生したら、callbackメソッドに「失敗した旨のメッセージを返す」という処理をします。これでLambda関数は失敗したという意味になります。

```
if (err) {
  console.log(err);
  callback(`Cannot read ${key} from bucket ${bucket}`);
}
```

成功したら、サムネイルを作ります。ここではImageMagickライブラリを使って生成しました。幅は320pxとしました。

```
im.resize({
  srcData: data.Body,
  format: "jpeg",
  width: 320
}, function(err, stdout, stderr) {
...処理...
})
```

生成したサムネイルは、putObjectメソッドを呼び出すとS3バケットに書き込みます。

```
var thumbnailkey = key.replace("images/", "thumbnails/");
s3.putObject({
  Bucket: bucket,
  Key: thumbnailkey,
  Body: new Buffer(stdout, "binary"),
  ContentType: data.ContentType
}, function(err, res) {
...処理...
})
```

処理に成功した場合は、callbackメソッドにnullを渡してください。

```
callback(null);
```

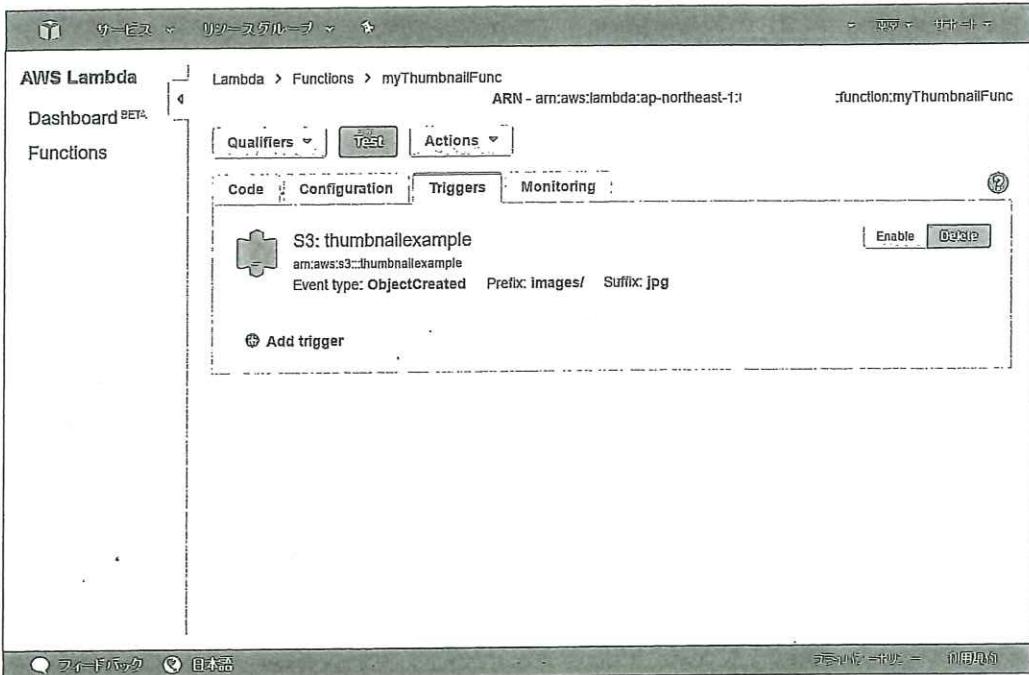


図 3-3-21 トリガを有効にする

### 【トリガを有効にしてテストする】

では、実際にテストしてみましょう。まずは、S3 のトリガを有効にします。[Triggers] タブを開き、[Enable] をクリックして、トリガを有効にしてください（図 3-3-21）。確認メッセージが表示されたら [Enable] をクリックしてください（図 3-3-22）。

**△△△** ここでは Lambda 関数のテストをせずに、いきなりトリガを有効にしていますが、本来なら、Lambda 関数のテスト機能を使って、一通りの動作テストをしてから、トリガを有効にすべきです。しかし画像のサムネイルを作る処理を Lambda 関数のテスト機能で実行するのは難しいので、ここではテストを省略しました。Lambda 関数のテストの方法については「○コンテンツが役立ったかどうかをアンケートを取りたい」で説明します。

この状態で、先の S3 バケットの images ディレクトリーに、適当な JPEG 画像をアップロードしてください（図 3-3-23）。すると、thumbnails ディレクトリーに、同名のサムネイル画像が保存されるはずです。

実際に Lambda 関数が実行されたかどうかは、[Monitoring] タブで確認できます（図 3-3-24）。

[View logs in CloudWatch] をクリックすると、CloudWatch Logs でログを確認できます（図 3-3-25）。ログストリームの各行をクリックすると、その詳細が表示されます。この詳細には、Console.log メソッドで書き出したメッセージも含まれています（図 3-3-26）。

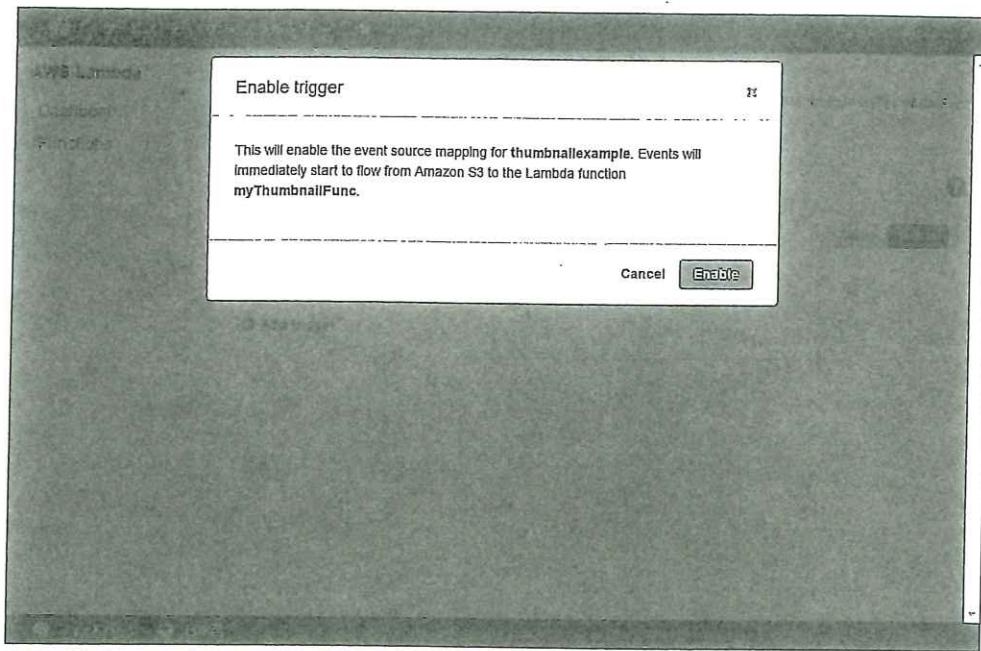


図 3-3-22 確認メッセージ

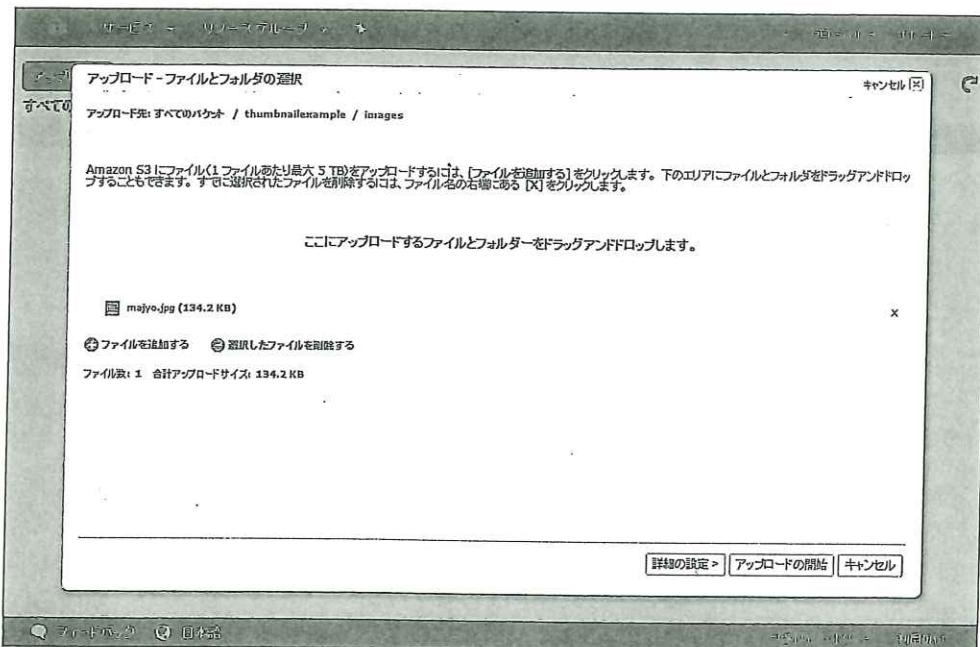


図 3-3-23 JPEG 画像をアップロードする

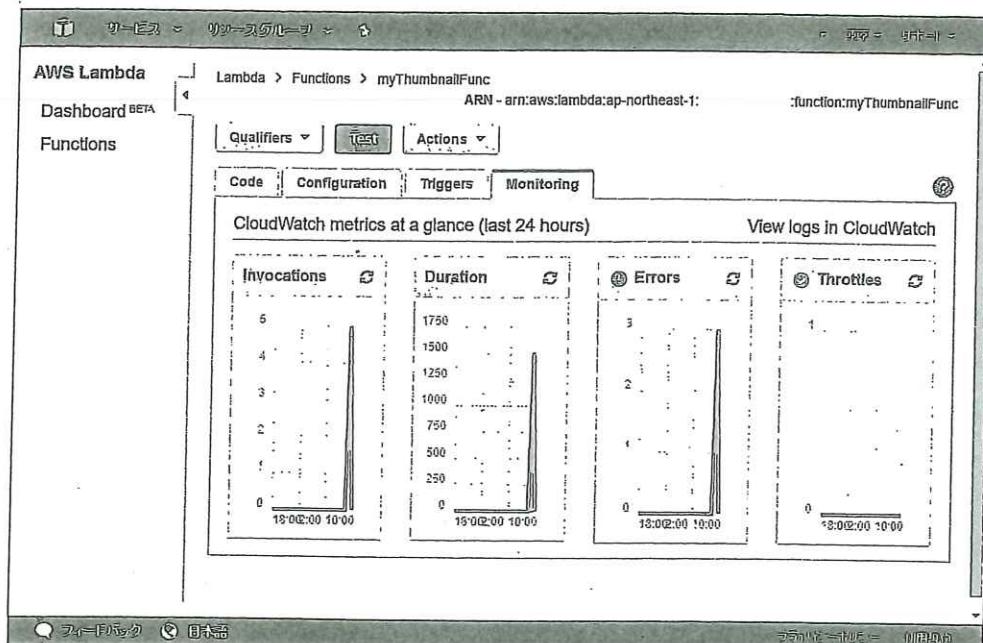


図 3-3-24 [Monitoring] タブで確認する

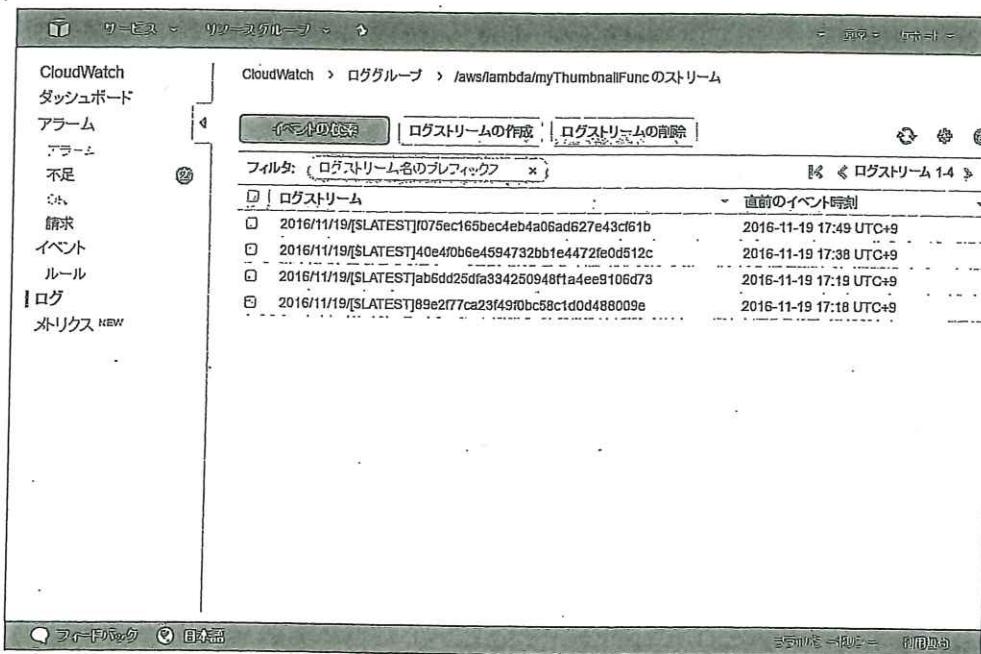


図 3-3-25 CloudWatch Logs で確認する