設定方針

詳細な手順の説明の前に、AWS Lambda設定方針を整理します。

１．S3によるバケットの作成

写真をS3に保持するため、まずバケット(Bucket)を作成します。バケット名は任意ですが、バケット名を指定してLambdaプログラムが動作しますので、バケット名はメモしておいてください。本実装では、バケツ名としてwificambucketを前提に進めます。
S3の写真に対してアクセスするのはアクセス権限付きのURLを生成してSNS経由で通知しますので、S3のアクセス権限はPrivate（一般へのアクセスは禁止）とします。

２．Lambdaの設定

AWSの設定を説明します。まずAWS　Lambdaの設定を行います。Lambdaにより、
（１）写真アップロード機能、（２）RAW->JPEG形式の変換機能、（３）SNSの呼び出しによる関係者への通知
を実現しています。

WiFiCameraからLambdaの呼び出しWebAPIで行います。このため、API Gatewayを設定して、WiFiCameraからLambdaの写真アップロード機能を呼び出せるようにします。

RAW->JPEG変換は、＃＃＃ですが、画像が確認できる程度の最低限の機能のみを実現しています。ビットマップ形式のデータをJPEGに変換するのは画像変換ライブラリ(PIL/Pillow)を用いています。PILを用いることで、ビットマップ形式の画像を簡単にJPEGフォーマットに変換することができます。PILライブラリは、Lambdaの環境では利用可能になっていなため、PILライブラリを別途用意してLambda環境にアップロードする必要があります。この手順は＃＃＃に示します。

３．API Gatewayの設定

APIでは機器からのHTTP接続を処理します。
WiFIカメラからの通信を受信し、Lambdaを呼び出します。Typeが、Body部がバイナリであるため、通常の＃＃＃とは異なり＃＃＃＃を指定します。

作成するAPIの仕様：
リージョンはいずれもap-northeast-1（東京)を選択
ap-northeastは3種類あり拠点は以下となっています
ap-northeast-1 アジアパシフィック（東京）
ap-northeast-2 アジアパシフィック（ソウル）

ap-northeast-3 アジアパシフィック (大阪)

作成詳細手順１（画像アップロード用関数の登録）

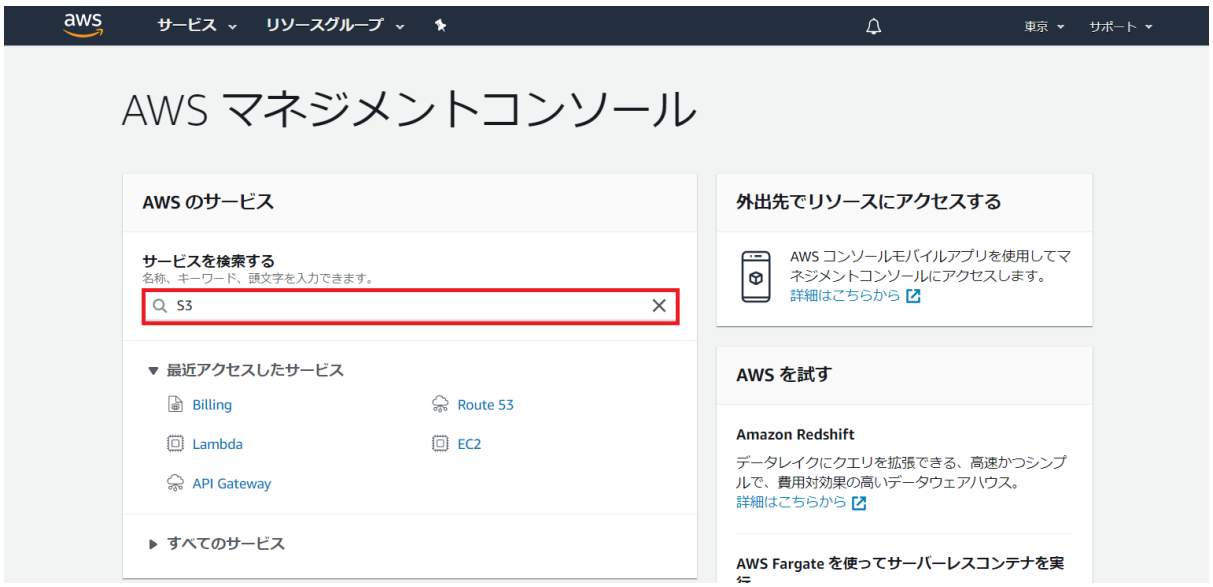
本資料では、２種類あるLambda関数のうちの前半、

1. uploader.py
　　アップロード用APIの提供、ならびに、RAW画像を受け取りS3に保存する機能

について説明します。

(1) S3バケットを作る

バケットの仕様は以下です。
バケット名：wificambucket
リージョン：アジアパシフィック（東京）
アクセス：バケットとオブジェクトは非公開



AWSマネージメントコンソールから、サービス検索で S3 と入力

画面：「Amazon S3 管理画面」

[+バケットを作成する]ボタン押下、「バケットの作成」画面に移ります。

画面：「バケットの作成」

・バケット名：wificambucket01を入力します。
　バケット名は任意ですが、Lambdaのアップローダ、画像変換でバケット名を指定しますので、バケット名を記録しておいてください。
・リージョンは、アジアパシフィック（東京）を選択

[作成]ボタン押下

画面：「Amazon S3 管理画面」

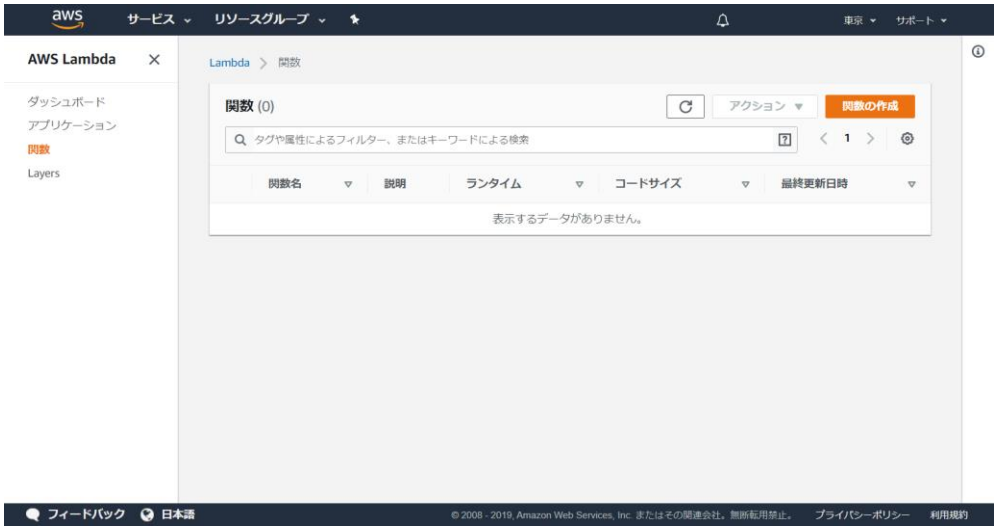
仕様通りに作成できたことを確認します。
バケット名：wificambucket
リージョン：アジアパシフィック（東京）
アクセス：バケットとオブジェクトは非公開


aws
サービス
リソースグループ
グローバル
サポート
Amazon S3
バケット
バッチオペレーション
ブロックパブリックアクセス (アカウント設定)
注目機能 2
S3 バケット
コンソールのご紹介
バケット検索
すべてのアクセスタイプ
+ バケットを作成する
パブリックアクセス設定を編集する
空にする
削除
1 バケット
1 リージョン
バケット名
アクセス
リージョン
作成日
wificambucket01
バケットとオブジェクトは非公開
アジアパシフィック (東京)
11月 23, 2019 4:29:31 午後 GMT+0900

(2)AWS Lambdaでアップロード用関数を作成する

AWSマネージメントコンソールから、サービス検索で lambda と入力

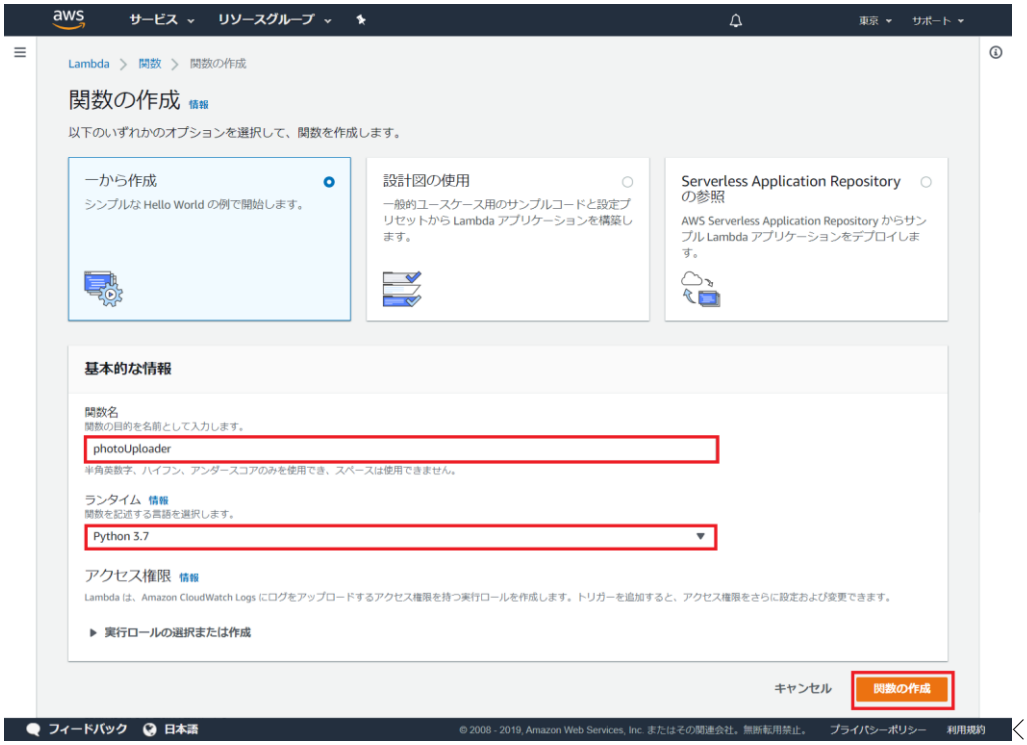
画面：「AWS Lambda 管理画面」



[関数の作成]ボタン 押下

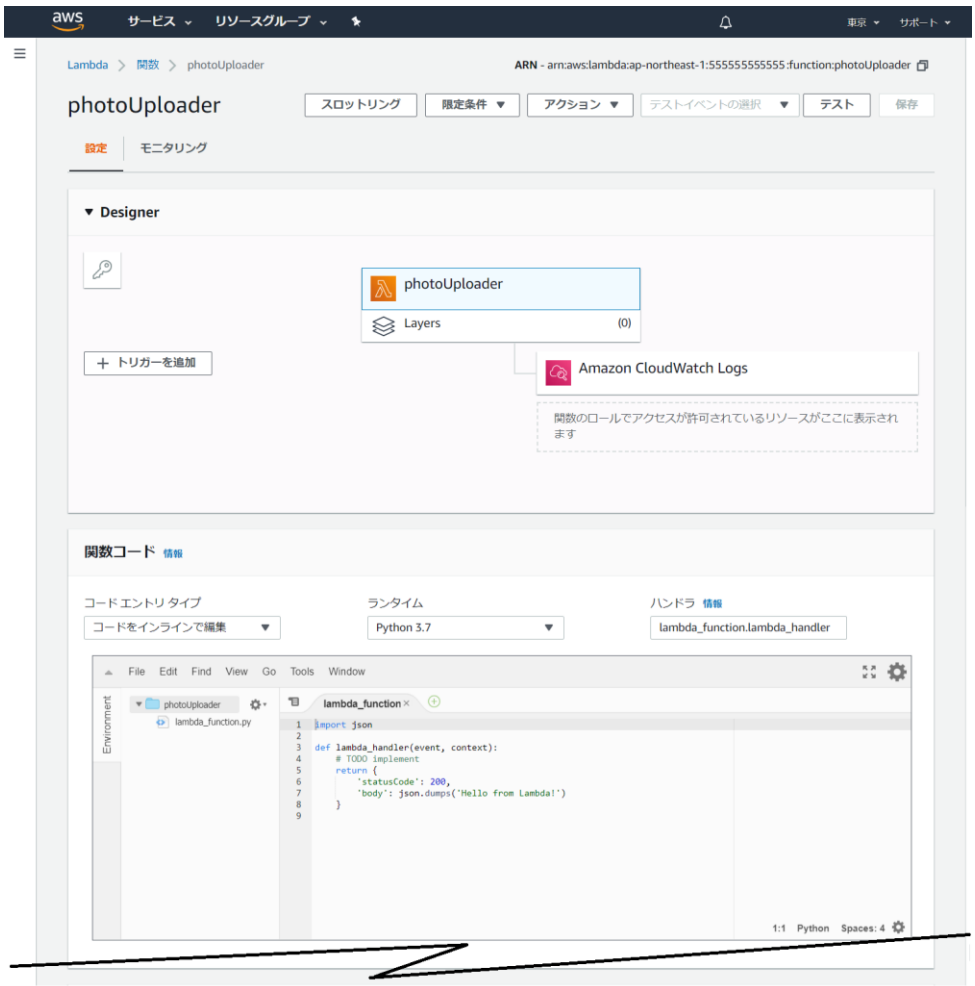
画面：「関数の作成」

・関数の作成：一から作成
・関数名：photoUploader（任意の関数名）
・ランタイム：Python3.7（PILライブラリのバージョンがPython3.7のため）
・アクセス権限：変更しない（後から編集）



<lam_mk_func.png>

[関数の作成]ボタン押下

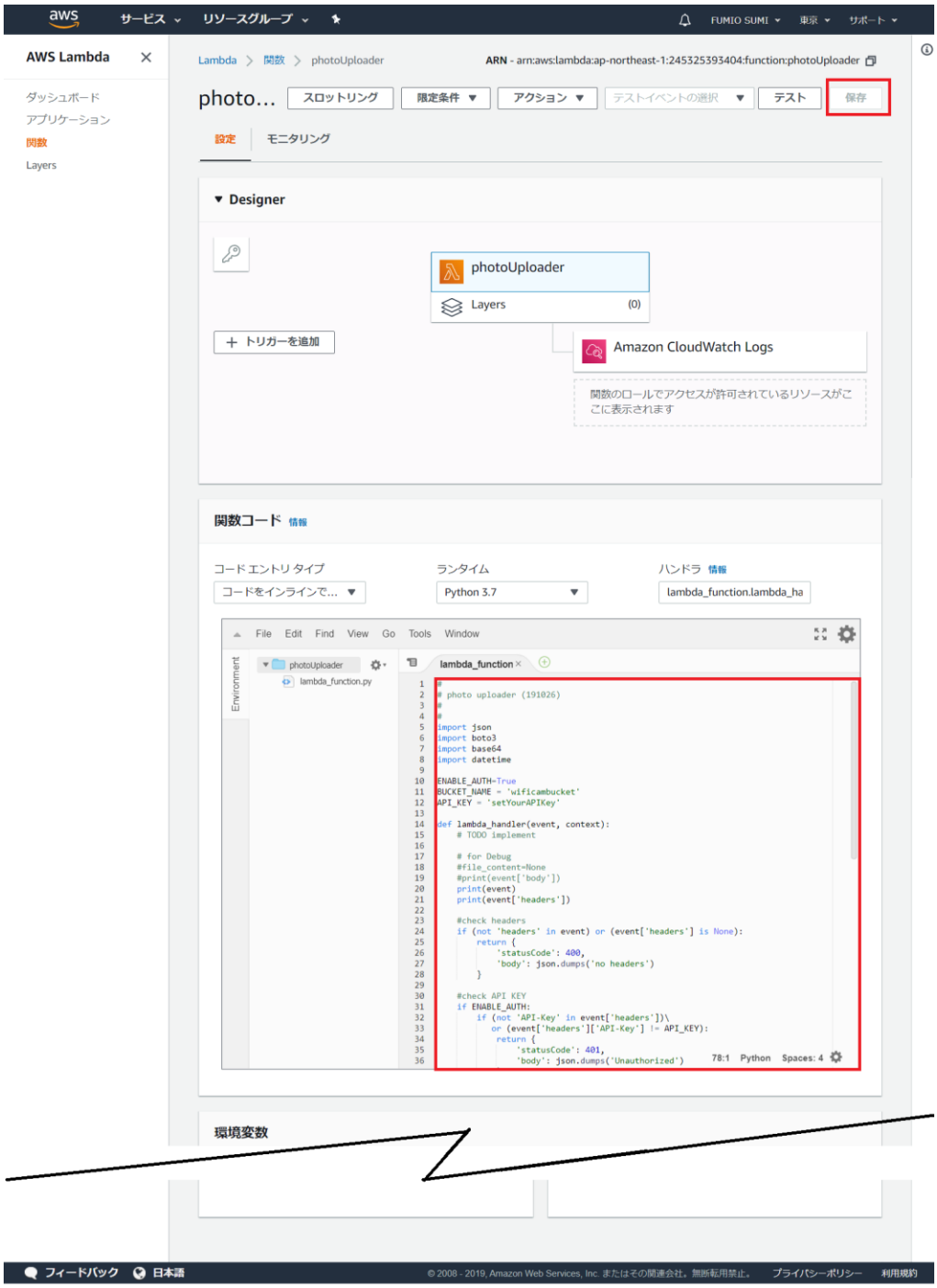


< lam_photo_up_console.png>

画面：「関数:photoUploader」

「photoUploaderの関数を設定」

・lambda_functionにはプログラム例が提示されているので、アップローダプログラムをCopy&Pasteで貼り付ける。
・画面右上の「保存」ボタン押下して保存



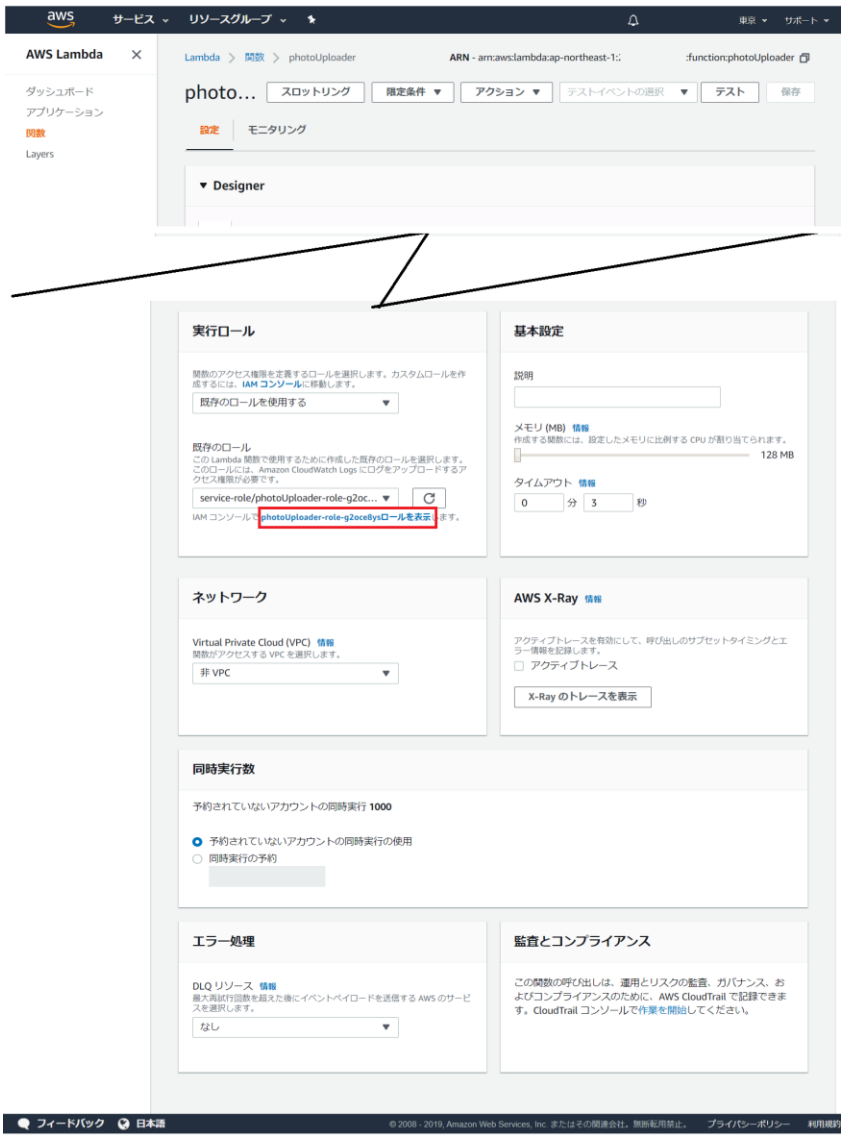
<lam_photo_up_console_paste.png>

「S3へのアクセス権限を付与」

この状態ではAmazon CloudWatch Logsへの書き込み権限しかないので
先ほど作成したS3への書き込み権限を付与します。

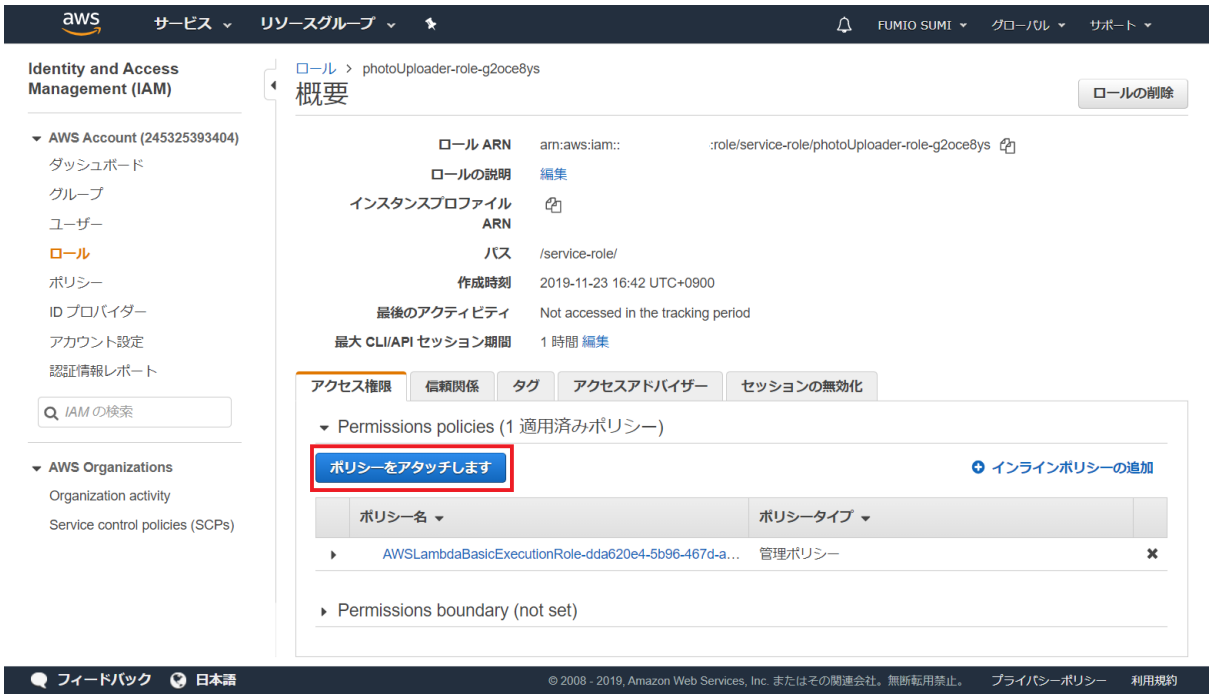
画面下に、実行ロールの表示枠があり、既存のロールが表示されています。
既存のロールが自動で作成されたphotoUploader用のロールであり
このロールにS3への書き込み権限を追加します。

追加するには、IAMコンソールに移る必要があるが、画面下に
「IAM コンソールで photoUploader-role-XXXXXX ロールを表示します。]」
とあるのでこれをクリックします。



画面：Identity and Access Management (IAM)
　　ロール：photoUploader-role-g2oce8ys

Identity and Access Management (IAM)のロール画面に遷移します。
photoUploader用のロールが表示されています。「ポリシーをアタッチします」を押下します。



画面：「アクセス権限追加画面」

追加可能なポリシー一覧が表示されるので、S3と入力して検索
AmazonS3FullAccessを選択して「ポリシーのアタッチ」を押下

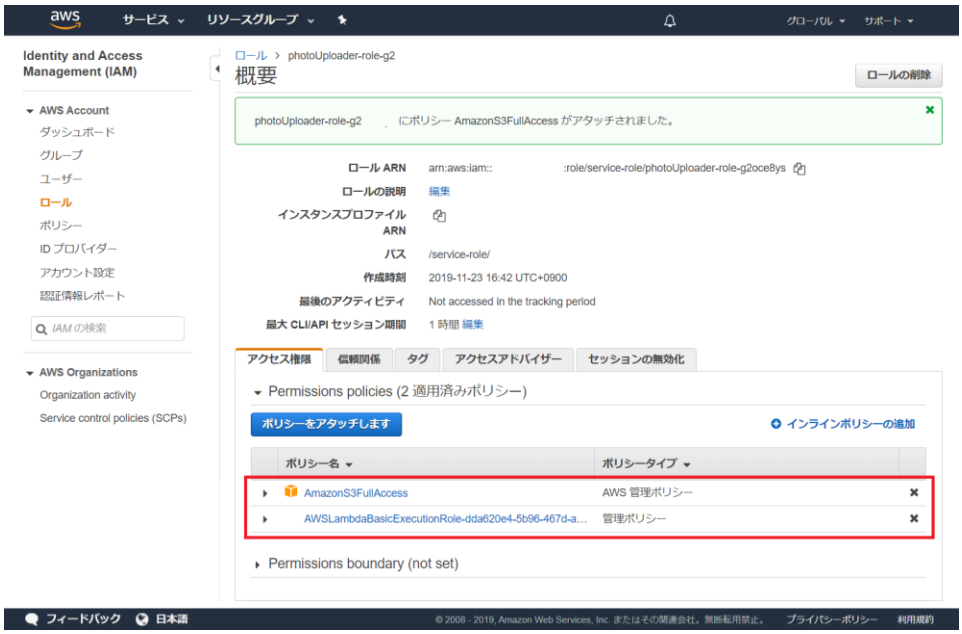
<IAM_cons.png>

<IAM_attach.png>

ロール画面に切り替わり

画面：Identity and Access Management (IAM)
　　ロール：photoUploader-role-g2oce8ys

今回追加した新しいポリシー「AmazonS3FullAccess」が追加されていることを確認
<IAM_role_fin.png>



IAMでの作業は終わり、Lambdaコンソールに戻る

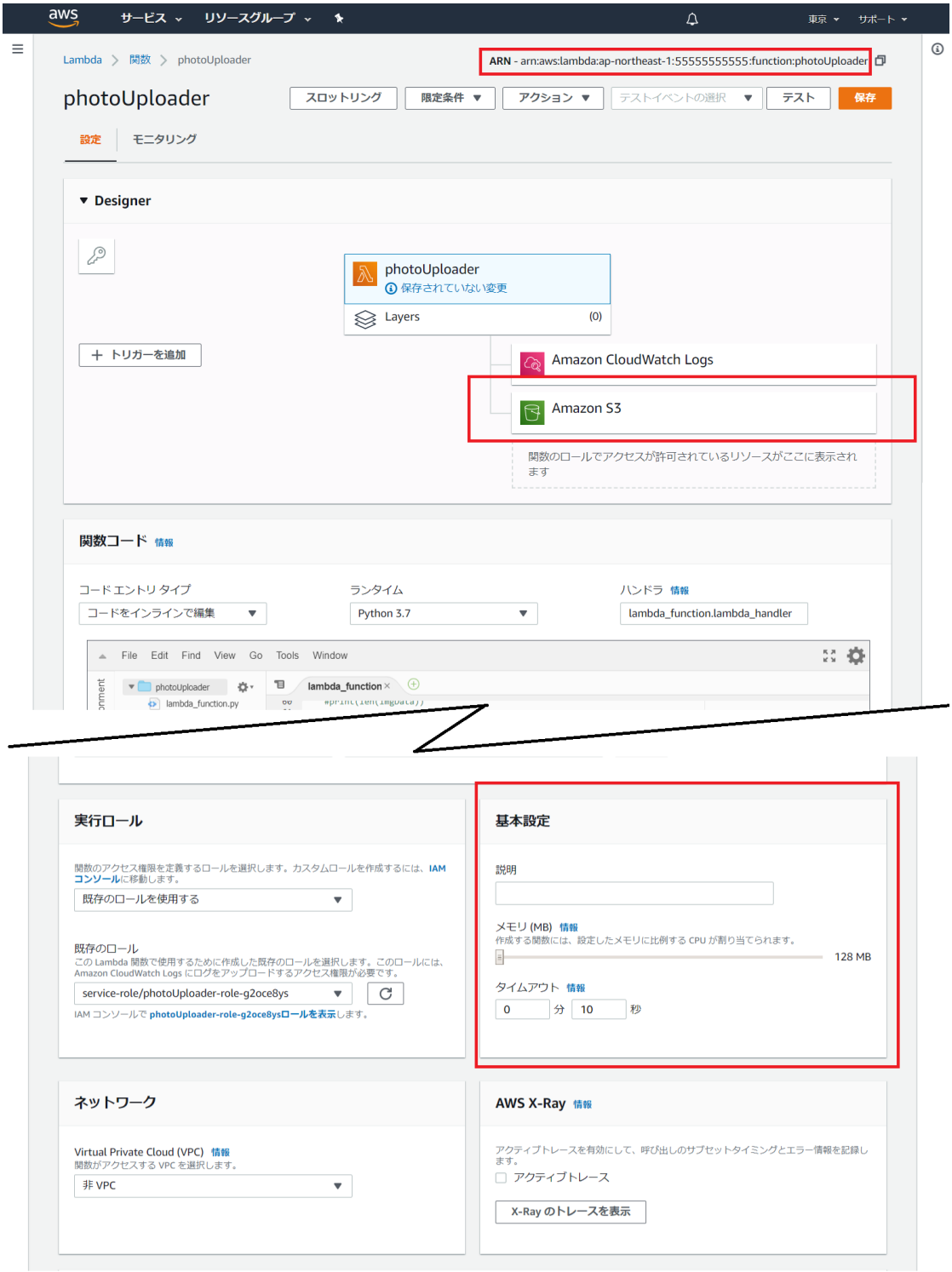
画面：「関数:photoUploader」

<IAM_permit_S3.png>
photoUploaderの画面でAmazon S3が追加されたことが確認できます。

ソース内の以下を設定したバケット名、ESP32のアップローダプログラムの
APIキーに変更する（サンプル通りの場合は以下）

```
BUCKET_NAME = 'wificambucket'    # set your S3 Bucket Name
API_KEY = 'setYourAPIKey'        # set your API key
```

実行時間3秒となっている。アップローダは余裕をみて10秒とする。
<lam_timeout_10.png>



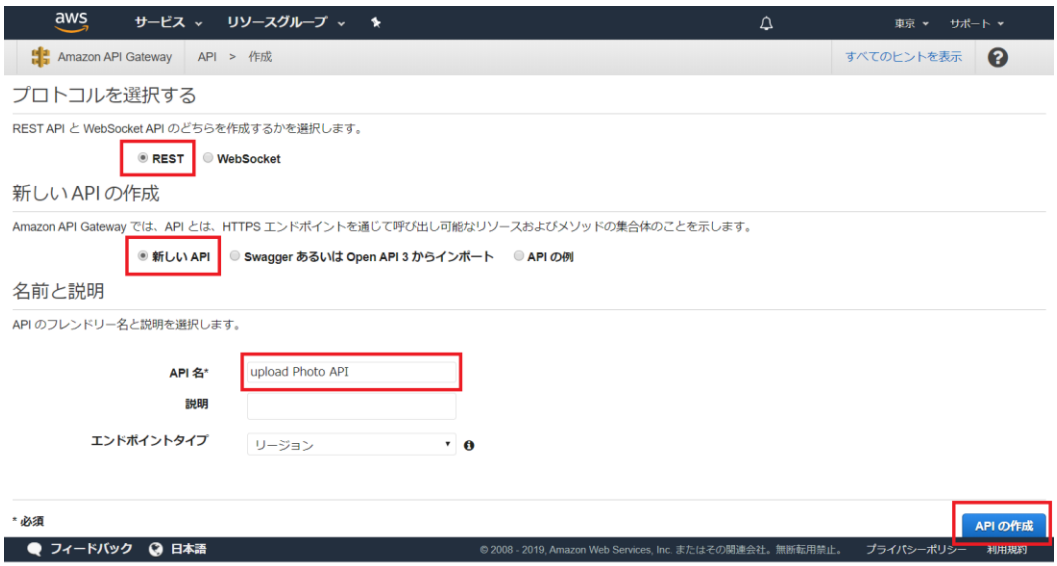
uploader用の登録作業は終わり、APIからの呼び出しの指定を行うため
関数名をメモする（Lambda photoUploaderの画面右上にARNが表示されている）
(ARM Amazon Resource Name)

arn:aws:lambda:ap-northeast-1:55555555555:function:photoUploader(仮)

APIの作成

<API_GW_start.png>
AWSマネージメントコンソールから、サービス検索で API と入力
画面デザインは結構変わるようですが、、[今すぐ始める]押下

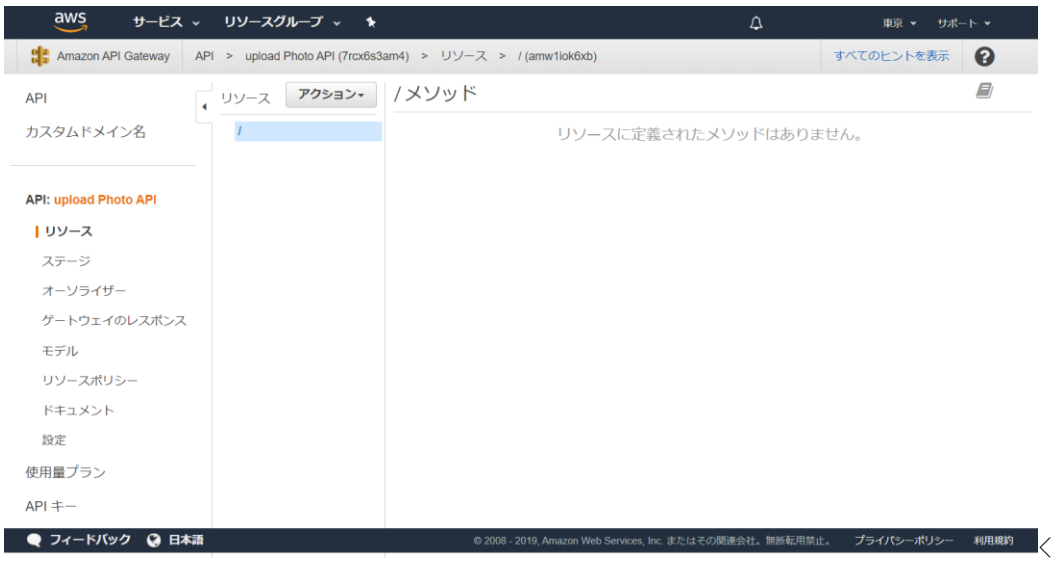
プロトコルを選択： REST
新しいAPIの作成： 新しいAPI
API名：upload Photo API （内部管理用、任意）
エンドポイントタイプ：リージョン（初期設定）
<API_gen.png>



「APIの作成」ボタン押下

画面： 「API:upload Photo API 管理画面」

API名「upload Photo API」の管理画面が表示される



<api_res.png>

＜api_act_pulldown.png＞

まずリソースを定義します

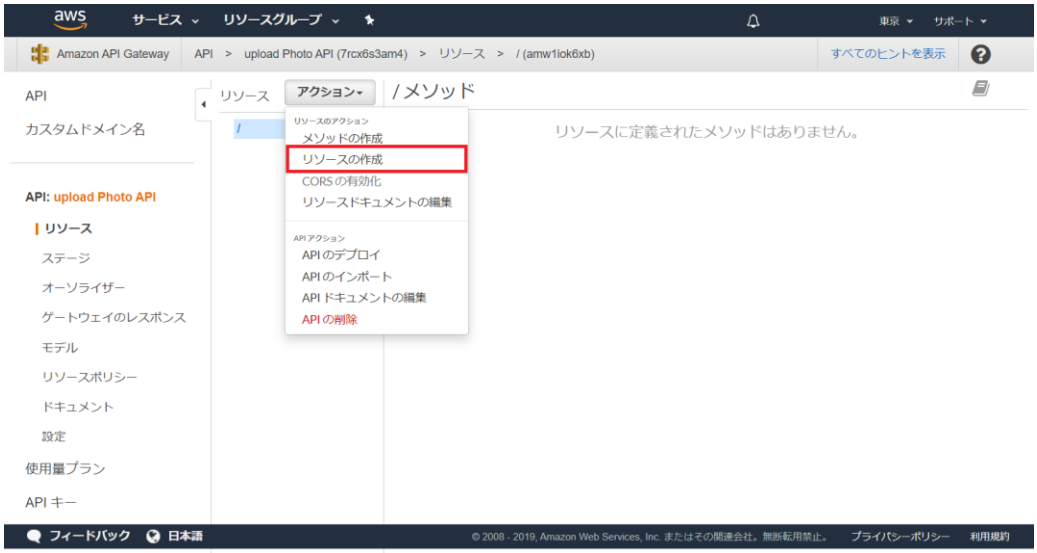
リソースはURL名に含まれる文字列となります（ESP32のアップロード時に指定される）

任意ですが分かりやすく uploadphotoとします。

https://xxx.amazonaws.com/<stage名>/<resource名>

https://xxx.amazonaws.com/<stage>/uploadphoto

「アクション」のプルダウンメニューより、「リソースの作成」をクリックします



画面：「新しい子リソース」

リソース設定画面が表示されます。

プロキシリソースとして設定する：チェックを入れない（初期設定のまま）
（リソースとりまとめは不要のため）

リソース名：uploadphoto（リソース名は任意です。API用URL内のパスとなります）

API Gateway CORS を有効にする：チェックを入れない（初期設定のまま）

「リソースの作成」ボタン押下

<api_sel_rel.png>

＜api_new_res.png＞

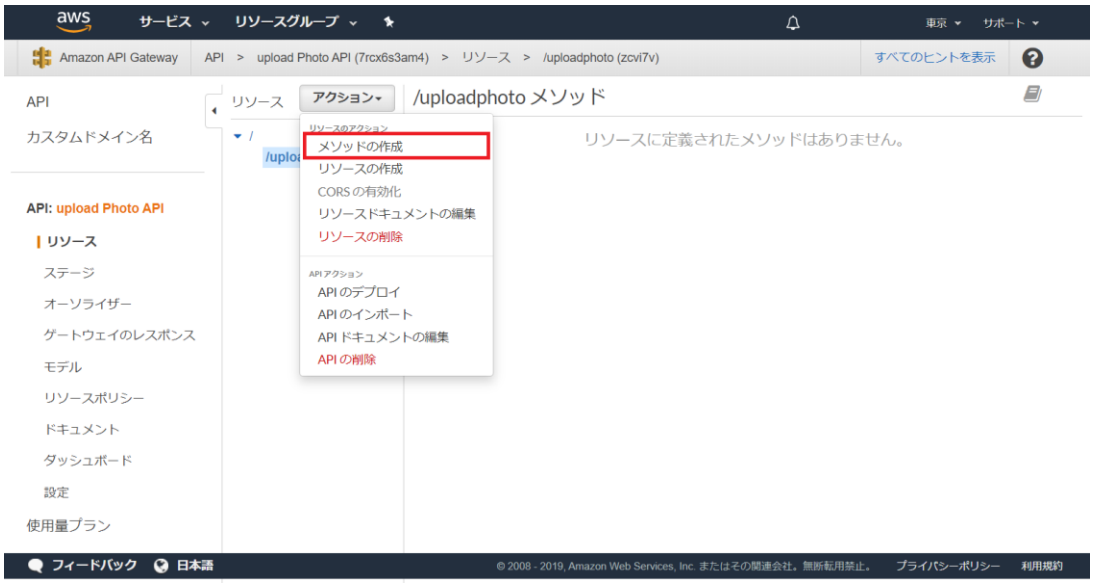
リソース：/uploadphoto　が作成されました。

画面：「/uploadphotoメソッド」

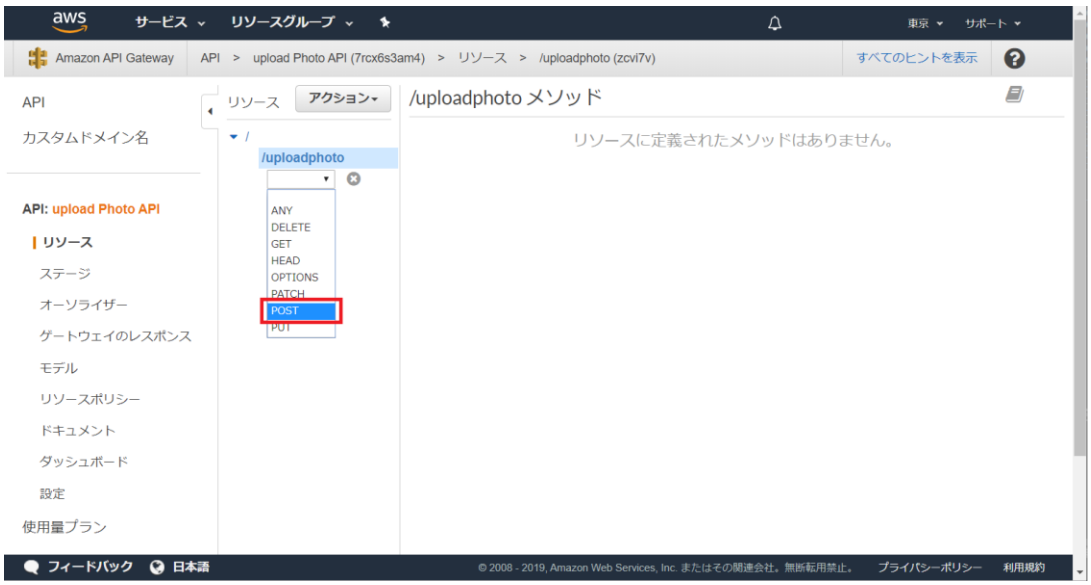
次にPOSTメソッドを設定します。

「アクション」プルダウンメニューより、「メソッドの作成」を選択しますす。

＜ api_new_method.png＞



プルダウンメニューよりPOSTを選択、チェックボタンをクリックします



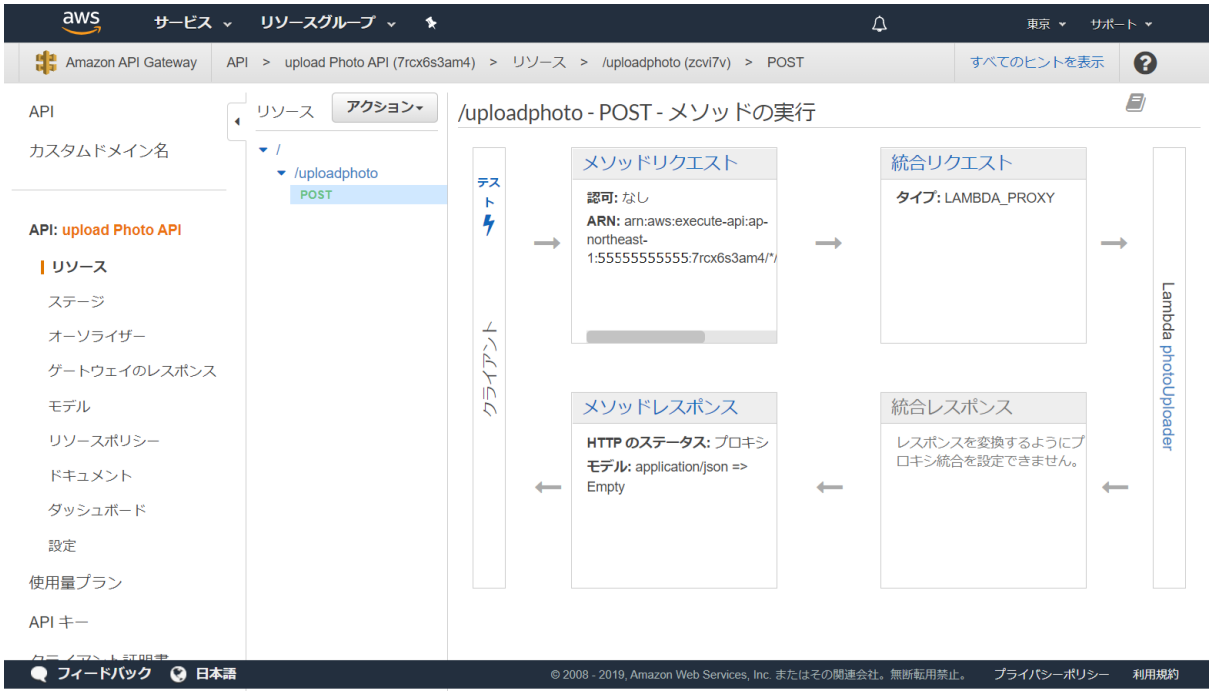
統合タイプ：Lambda関数
Lambdaプロキシ統合の使用[X]
Lambdaリージョン:ap-northeast-1（東京）
Lambda関数：arn:aws:lambda:ap-northeast-1:245325393404:function:photoUploader
(Lambda関数名について補完機能もあるようですが、ARNで指定するのが確実です)
<API_post_setup.png>
「保存」ボタンを押下します

Lambda 関数に権限を追加する
API Gateway に、Lambda 関数を呼び出す権限を与えようとしています:
arn:aws:lambda:ap-northeast-1:365701690774:function:photoUploader2

実行権限が自動で付与されますが、確認してきますので、「OK」を選択
APIの登録が完了

画面：「/uploadphoto - POST メソッドの実行」

＜api_create_fin.png＞



メソッドリクエスト

アップロードするデータのBODY部がTextではなくバイナリであり、バイナリメディアタイプとして、＃＃＃を指定しますので、これを受理してもらう必要があります。

「設定」メニューをクリック

画面：「設定」

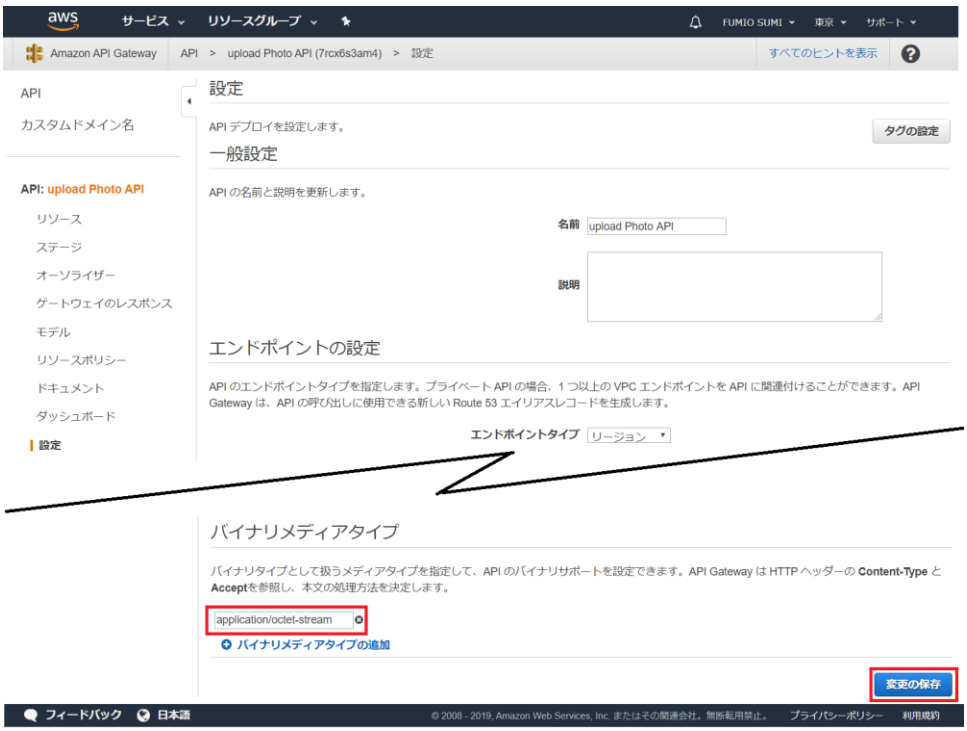
バイナリメディアタイプの欄に、(+)バイナリメディアタイプの追加の
メニューがありこれをクリック

バイナリメディアタイプに以下を追加

設定画面を選択
バイナリメディアタイプの段落で、「バイナリメディアタイプの追加」を押して

application/octet-stream

を入力、　「変更の保存」ボタン押下



APIの作成が終わったので、サービス公開します。

アクション＞APIのデプロイ　を選択

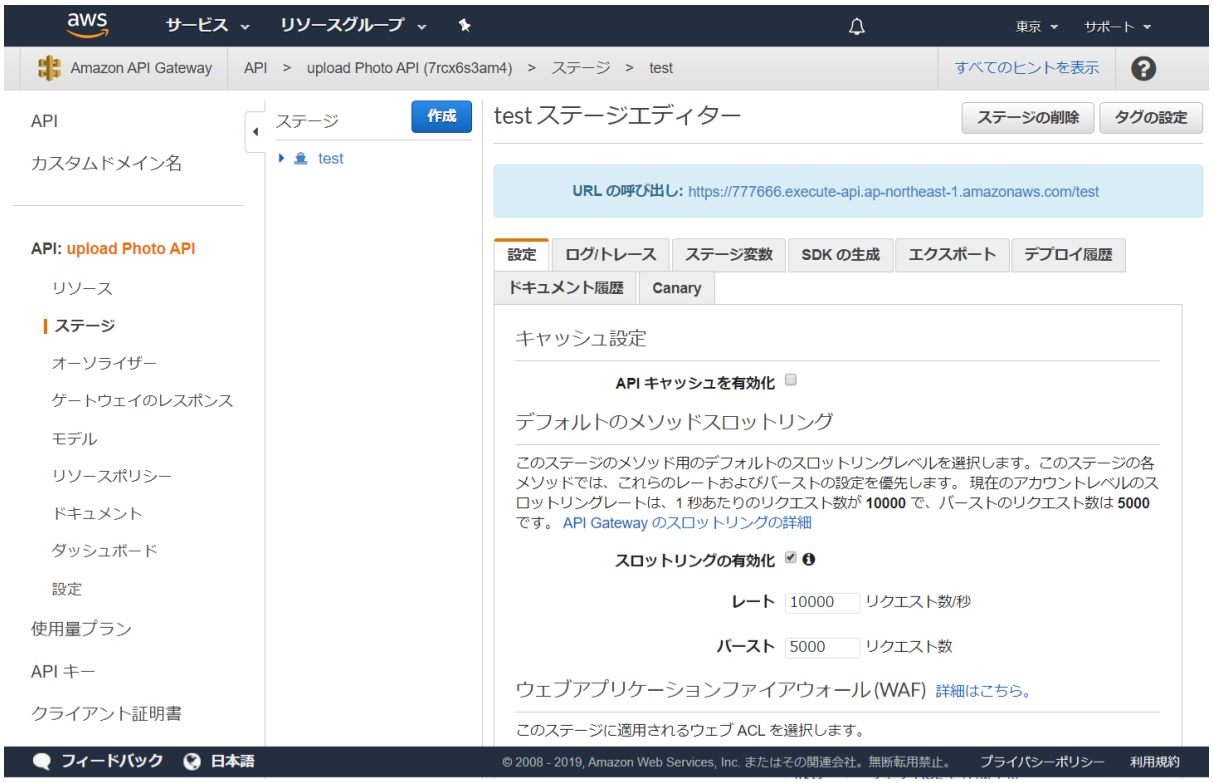
初回のデプロイになるため、ステージの作成が必要です。デプロイされるステージで「新しいステージ」を選択、ステージ名：test と入力します。（任意ですが、APIのURL名に含まれます）

api_deploy.png

「デプロイ」ボタン押下

デプロイが完了するとステージ画面に切り替わります。

test_stg_edit.png



画面：「testステージ」

以上の操作により作成されたuploadphoto用URLは以下から確認できます。

ステージ画面より

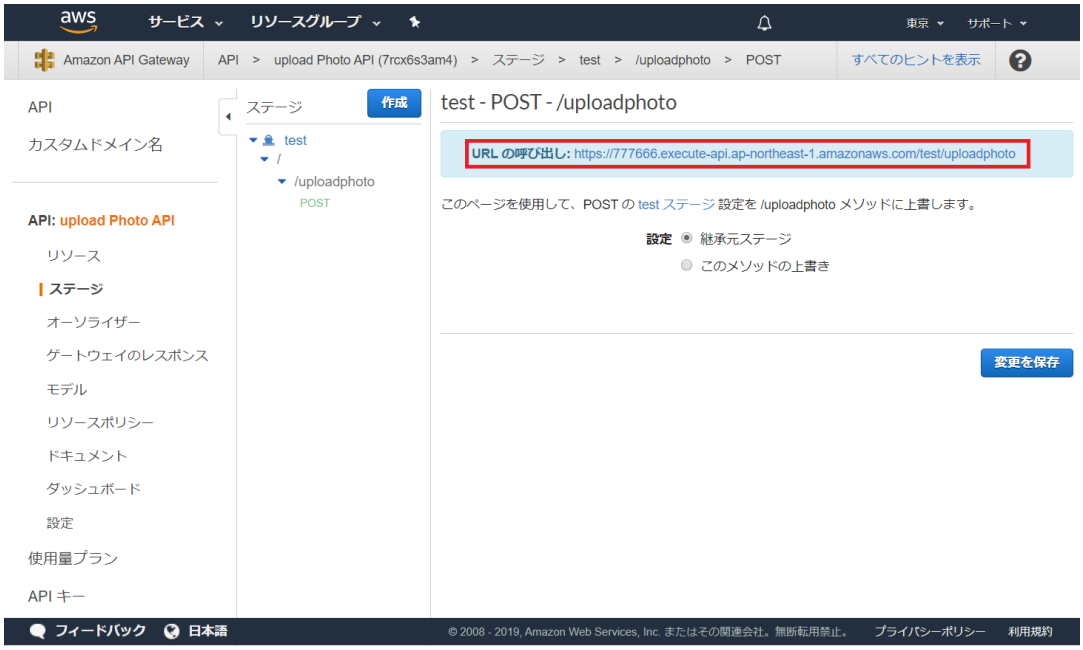
＞test

/

/uploadphoto

POST

＜api_url.png＞



URL の呼び出し: https://777666.execute-api.ap-northeast-1.amazonaws.com/test/uploadphoto

ESP32から接続する前に、正しく設定されたかテストします。

リソース画面より、
  /uploadphoto
    POST をクリック　/uploadphoto - POST - メソッドの実行　が表示されます

画面内のテストをクリック

FireShot Capture 031 - API Gateway - ap-northeast-1.console.aws.amazon.com.png

ヘッダのフォームに以下を指定

Content-Type:application/octet-stream
API-Key:setYourAPIKey

「テスト」ボタンを押す

画面右側に実行結果が表示され、レスポンス本文が以下であることを確認

"internal Error, not Base64Encoded"

上記以外の場合は実装上の問題があり調査します

"Unauthorized"と表示される場合、ヘッダー名がAPI-Keyになっていないか、

APIキーの指定文字列が間違っていると考えられます。

"not specified Content-Type"と表示される場合、コンテントタイプの
ヘッダー名がContent-Typeとなっていないと考えれます

"unsupported Content-Type"と表示される場合、コンテントタイプの
指定文字列(application/octet-stream)が間違っています。

Lambda側の画面でも以下となっており、photoUploader関数が、API Gatewayから呼び出され、Amazon CloudWatch Logs、Amazon S3へのロールが付与されていることが確認できます。

＜lambda_up_final.png＞

curlコマンド等でサーバ外からテストします。
このテストではどのような画像データでも問題ありませんが、Bayer形式のRAW画像を指定しています。

```
curl -X POST -H 'Content-Type: application/octet-stream' -H 'API-Key: setYourAPIKey' --data-binary "@test_191026.raw" https://bva6z6igki.execute-api.ap-northeast-1.amazonaws.com/test/uploadphoto
```

```
curl -X POST -H 'Content-Type: application/octet-stream' -H 'API-Key: setYourAPIKey' --data-binary "@191117_200040.raw" https://7rcx6s3am4.execute-api.ap-northeast-1.amazonaws.com/test/uploadphoto
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  300k  100    16  100  300k      8   150k  0:00:02  0:00:02 --:--:--  139k
```

"upload is done"

"upload is done"となっていればアップロードが正常に行われてます。

もしエラーとなった場合、同様に確認します。

以上でアップロード用APIの作成は終了です