

AWS IAMユーザーアカウントキー 期限確認&通知&削除 設定方法

2020.1.10
作成：平井秀明

※下記は2020.1.10時点での操作方法である（操作方法・画面がよく変わるので注意）
※CloudFormation化したが、そのデプロイ手順については別マニュアル参照（ローカルPycharmでのAWS Lambda Functionデプロイ方法）

1.概要

定期的にLambdaを起動し、アカウントキー管理スプレッドシート情報を収集する。
期限一週間前にユーザーにSLACKで通知、期日の次の日にIAMアカウントキー、ユーザー、ポリシーを削除する。

2.AWS構成

Cloudwatch EventsにてLambdaを定期実行（今回はJST15時の1日1回）。
Lambda関数にてアカウントキー管理スプレッドシートにアクセスし、
一行ずつ処理を行い、処理対象はSLACK通知、IAM操作を行う。



3.動作仕様

1. 日時はJST（日本時間）で扱う
2. 一週間後に終了日を迎えるアカウントキー申請ユーザーに延長可否を尋ねる
3. 延長可否はSLACKにて対象者にメンションする
4. 対象者に加えて管理者にも通知の写しを送る
5. メンションにはbotアプリを用いる
6. 延長申請が行われた場合は、延長済みの印をスプレッドシートに付ける（別Lambdaで行う）
7. 延長済み印が付いた対象アカウントキーは処理対象外とする

8. 延長申請が行われず、期日を過ぎた場合（次の日）、アカウントキーを削除する
9. 削除はAPIにて行う
10. 削除したアカウントキー情報には削除済みの印をスプレッドシートに付ける
11. Lambdaの動作は1回／日行う
12. SLACKメッセージには、アカウントキー情報を載せて通知する

4. 基本設計

- a. 利用終了日列を1行ずつチェックして一週間後の日付がないか検索する
 - i. あった場合はアカウントキー利用ユーザーに通知を行う
 1. 通知は対象者を氏名からSLACKのユーザーIDを割り出し、botで通知する
 2. 割り出しにはユーザーIDと氏名対比のテーブル（users_json）を使う
 3. 管理者には通知を行ったことを固定名称にてbotで通知する
 - ii. ない場合は何もしない
- b. 一週間後の日付があっても状態列に「削除済み」「延長申請済み」が設定されている場合は何もしない
 - i. 延長申請済みの入力とは別のLambdaで設定する
- c. アクセスキー利用終了期日の次の日に、「削除済み」「延長申請済み」になっていないアカウントは以下の処理を行う
 - i. APIにて対象のアカウントを削除する
 - ii. 状態列に「削除済み」を設定する
- d. 氏名対比テーブル（users_json）に記載がなかった場合
 - i. ユーザー情報が見つからなかった旨を管理者にSlackで通知する

5. Slack botの設定

下記にアクセス。

<https://api.slack.com/apps>

「Create New App」を押下。

- ・ App Name : アプリの名前
- ・ Development Slack Workspace : FutureStandardを選択
- ・ 「Create App」を押下。

「Add features and functionality」から「Bots」を押下。

- ・ Display name、Default usernameはそのまま
- ・ 「Add Bot User」を押下。

「Permissions」を押下。

- ・ 「Install App to Workspace」を押下。

- ・「許可する」を押下。
- ・「Bot User OAuth Access Token」をコピーしておく。

Add features and functionality



Choose and configure the tools you'll need to create your app (or review all our [documentation](#)).

Incoming Webhooks

Post messages from external sources into Slack.

Interactive Components

Add components like buttons and select menus to your app's interface, and create an interactive experience for users.

Slash Commands

Allow users to perform app actions by typing commands in Slack.

Event Subscriptions

Make it easy for your app to respond to activity in Slack.

✔ Bots

Add a bot to allow users to exchange messages with your app.

✔ Permissions

Configure permissions to allow your app to interact with the Slack API.

Bot User OAuth Access Token

xoxb-10711501747-752648249329-WGXFUDahv2Dyb6Hg3yGDByRm

Copy

6. スプレッドシートの設定

<https://console.developers.google.com/project>にアクセス。

「プロジェクトを作成」を押下。

≡ Google APIs

リソースの管理

+ プロジェクトを作成

適当なプロジェクト名を設定して作成を押下。（他変更なし）

新しいプロジェクト

プロジェクト名 *

My Project 46537



プロジェクト ID: absolute-brook-250609。後で変更することはできません。 [編集](#)

組織 *

futurestandard.co.jp



プロジェクトに関連付ける組織を選択します。この選択を後で変更することはできません。

場所 *

futurestandard.co.jp

[参照](#)

親組織またはフォルダ

作成

キャンセル

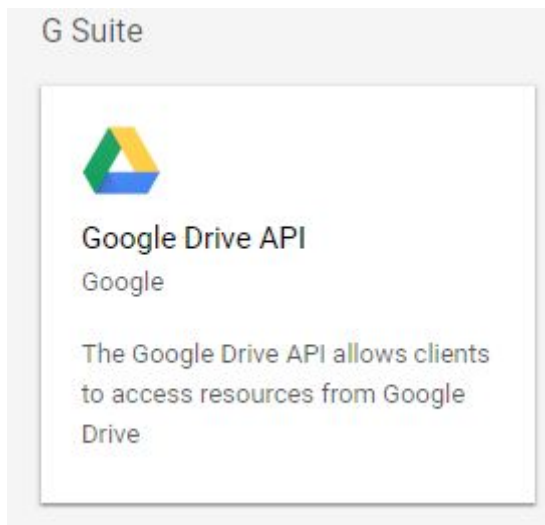
左上のメニューから「APIとサービス」→「ライブラリ」を選択。



上部のメニューバーで作成したプロジェクトを選択。



「Google Drive API」を選択。



「有効にする」を押下。



左上のメニューから「APIとサービス」→「認証情報」を選択。



「認証情報を作成」を押下。

API 認証情報

API にアクセスするには認証情報が必要です。使用する API を有効にしてから、必要な認証情報を作成します。API に応じて、API キー、サービス アカウント、または OAuth 2.0 クライアント ID が必要です。詳細については、[認証のドキュメント](#)をご覧ください。

認証情報を作成

サービスアカウントキーを選択。

API キー

シンプル API キーを使用してプロジェクトを識別し、割り当てとアクセスを確認します

OAuth クライアント ID

ユーザーのデータにアクセスできるようにユーザーの同意をリクエストします

サービス アカウント キー

ロボット アカウントによるサーバー間でのアプリレベルの認証を有効にします

ウィザードで選択

使用する認証情報の種類を決定するため、いくつかの質問をします

認証情報を作成

以下を設定し、作成を選択。

- ・サービスアカウント：新しいサービスアカウント
- ・サービスアカウント名：適当
- ・役割：Project→編集者
- ・キーのタイプ：JSON

← サービスアカウントキーの作成

サービスアカウント

新しいサービスアカウント

サービスアカウント名 ?

csv_output

役割 ?

編集者

サービスアカウントID

csv-output

@absolute-brook-250609.iam.gserviceaccount.com

キーのタイプ

秘密鍵を含むファイルをダウンロードします。この鍵を紛失すると復元できないため、大切に保管してください。

☒ JSON

推奨

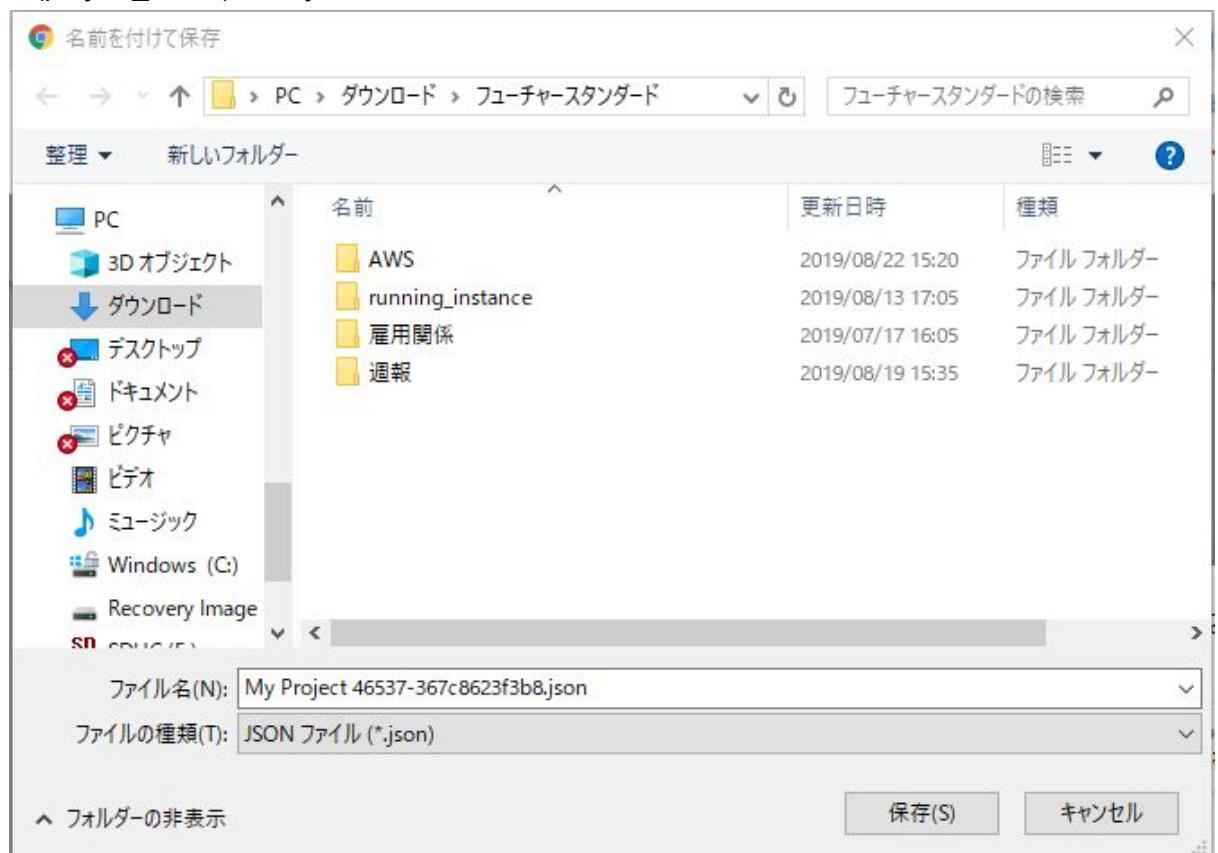
☐ P12

P12 形式を使用したコードとの下位互換性を目的としています

作成

キャンセル

「(project_name)-*****.json」のファイルが生成されるので適当な場所に保存する。



「(project_name)-****.json」のファイルを開いて、「client_email」:部分のメールアドレスをコピーする


```
↓  
"type": "service_account", ↓  
"project_id": "steady-cascade-248409", ↓  
"private_key_id": "27b1d4b595f3a2939fc9429ffe89b3df16733d0b", ↓  
"private_key": "-----BEGIN PRIVATE KEY-----¥nMLIEvQIBADANBgkqhkiG9w0BAQEEAAASC  
"client_email": "csv-output@s[REDACTED]", ↓
```

アクセスしたいスプレッドシートを開き、右上の「共有」を押下する。



ユーザー一部にコピーしたメールアドレスをユーザー欄に張り付け、「送信」を押下する。

他のユーザーと共有

共有可能なリンクを取得 

リンクの共有がオンです [詳細](#)

リンクを知っている全員が閲覧可 ▼	リンクをコピー
https://docs.google.com/spreadsheets/d/1duJLKZt_x9SUrYlXk_COCWEnCSDg8_yt	

ユーザー

名前かメールアドレスを入力...	
------------------	---

平井秀明 さんと他 1 人と共有しています

完了

[詳細設定](#)

7.CloudWatch Eventsの作成

Designer画面で「+トリガーを追加」を押下。
Cloudwatch Eventsを押下。



ルールで「新規ルールの作成」を選択。

ルール名を入力

ルールタイプで「スケジュール式」を選択。

スケジュール式に以下を入力し、追加を押下。

cron(0 6 ? * * *)

読み

(分 時 日 月 曜日 コマンド)

0 /6 ? * * * 6時間毎に実施

0 6 ? * * * 実施時間を指定

※日または週日の値は疑問符である必要がある (?)

※設定はUTC+0なのでJST (UTC+9) に合わせる

8.Lambda関数の作成

a. Lambda関数の作成

AWSサービスにて「Lambda」を選択。



関数の作成を押下。



一から作成を選択。



基本的な情報欄で以下を入力

- ・関数名：任意（例でここでは、cloudwatch_alarm_to_slack）。
- ・ランタイム：「Python 3.7」を選択。

関数名

関数の目的について説明する名前を入力します。

cloudwatch_alarm_to_slack

半角英数字、ハイフン、アンダースコアのみを使用でき、スペースは使用できません。

ランタイム [情報](#)

関数を記述するために使用する言語を選択します。

Python 3.7 ▼

アクセス権限で、「基本的なLambdaアクセス権限で新しいロールを作成」を選択。

アクセス権限 [情報](#)

Lambda は、Amazon CloudWatch Logs にログをアップロードするアクセス権限を持つ実行ロールを作成します。トリガーを追加すると、アクセス権限をさらに設定および変更できます。

▼ 実行ロールの選択または作成

実行ロール

関数のアクセス権限を定義するロールを選択します。カスタムロールを作成するには、[IAM コンソール](#)に移動します。

基本的な Lambda アクセス権限で新しいロールを作成 ▼

「関数の作成」を押下。

キャンセル

関数の作成

実行ロール部の既存のロール下の「cloudwatch_alarm_to_slack-role-***ロールを表示」を押下。

実行ロール

関数のアクセス権限を定義するロールを選択します。カスタムロールを作成するには、[IAM コンソール](#)に移動します。

既存のロールを使用する ▼

既存のロール

この Lambda 関数で使用するために作成した既存のロールを選択します。このロールには、Amazon CloudWatch Logs にログをアップロードするアクセス権限が必要です。

service-role/cloudwatch_alarm_to_slack-role... ▼



IAM コンソールで [cloudwatch_alarm_to_slack-role-dfx464ou](#) ロールを表示します。

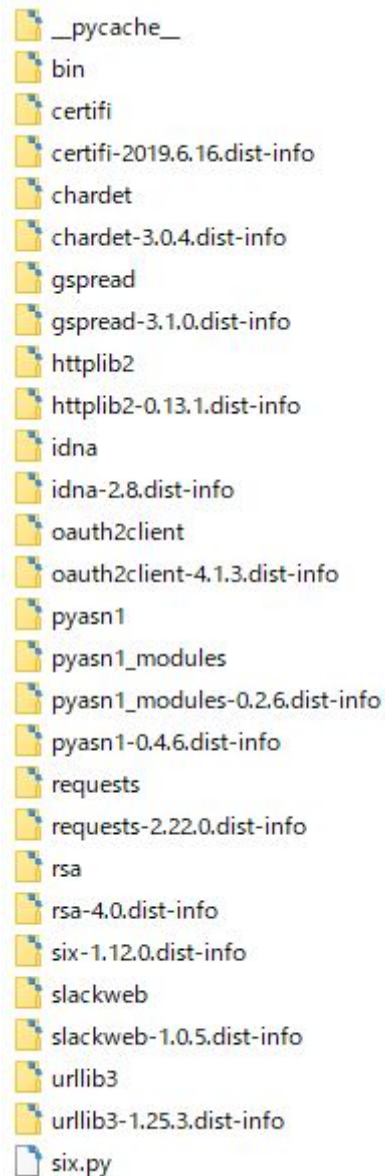
「AWSLambdaBasicExecutionRole...」を押下し、「ポリシーの編集」を押下。
今回使用しているAPIに該当するサービス、アクションを選び、「ポリシーの確認」
→「変更の保存」を押下。

サービス	アクション	概要
iam	DeleteUser	IAMユーザーの削除
iam	DeletPolicy	IAMポリシーの削除
iam	DetachUserPolicy	IAMユーザーとポリシーのデタッチ
iam	DeleteAccessKey	IAMユーザーアクセスキーの削除

b. 関数コード・インポートモジュールの作成

Slackに投稿するためのモジュールを以下にてダウンロードする。

- ・ コマンドプロンプトを起動。
- ・ 保存するフォルダを作成（例でここでは、aws-key-delete-announce）。
- ・ 保存する場所に移動 `cd 、aws-key-delete-announce`
- ・ 以下コマンド入力
 - `pip install slackweb -t .`
 - `pip install gspread -t .`
 - `pip install oauth2client -t .`
（最後にドットがある）
- ・ 以下のフォルダ、ファイルが生成される



- __pycache__
- bin
- certifi
- certifi-2019.6.16.dist-info
- chardet
- chardet-3.0.4.dist-info
- gspread
- gspread-3.1.0.dist-info
- httplib2
- httplib2-0.13.1.dist-info
- idna
- idna-2.8.dist-info
- oauth2client
- oauth2client-4.1.3.dist-info
- pyasn1
- pyasn1_modules
- pyasn1_modules-0.2.6.dist-info
- pyasn1-0.4.6.dist-info
- requests
- requests-2.22.0.dist-info
- rsa
- rsa-4.0.dist-info
- six-1.12.0.dist-info
- slackweb
- slackweb-1.0.5.dist-info
- urllib3
- urllib3-1.25.3.dist-info
- six.py

同じ場所に「lambda_function.py」のPythonファイルを作成し、以下のコードを記載。

```
# ----- #
# 定期的にアクセスキー発行スプレッドシートの利用終了日を検索
# 終了日を過ぎた場合はSlackに通知
# author:hirai
# date:2020/1/6
# ----- #

from __future__ import print_function

### sentry
import sentry_sdk
from sentry_sdk.integrations.aws_lambda import AwsLambdaIntegration
```

```

sentry_sdk.init(
    "https://e90c4b5fb19e4fdc8b708a1b1af75263@sentry.io/1875771",
    integrations=[AwsLambdaIntegration()]
)

import json
import boto3
import urllib.parse
import os
import re
import datetime
import time

import gspread
from oauth2client.service_account import ServiceAccountCredentials

import slackweb
import requests

def get_google_api():
    # Google Driv API認証設定
    SP_CREDNTIAL_FILE = 'My Project 48058-0025e18f407a.json'
    SP_SCOPE = 'https://spreadsheets.google.com/feeds'
    SP_SHEET_KEY = os.environ['SP_SHEET_KEY'] # 環境変数から読み込み
    # sp_sheet_key = "1ZLJf6GQE5ywcATKZxYWWl_EdLjsNHjlCYazrS_DeGcs"

    # スプレッドシートへのアクセス（認証）
    scope = [SP_SCOPE]
    credentials =
ServiceAccountCredentials.from_json_keyfile_name(SP_CREDNTIAL_FILE, scope)
    gs_client = gspread.authorize(credentials)
    sp = gs_client.open_by_key(SP_SHEET_KEY)

    return (sp)

def slack_notify(lst):
    # user_id = "ULF3WH04C" # 【暫定】現状通知対象のIDを固定
    user_id = lst[0]
    msg = "\nアクセスキー発行申請管理者からのお知らせです。" \
        "\nアクセスキー利用終了日まで一週間となりました。" \
        "\n延長が必要な場合は、延長申請をお願いします。" \
        "\n延長されない場合はアクセスキーは自動的に削除されます。" + "\n" + \
        "アカウント名：" + lst[1] + "\n" + \
        "利用目的：" + lst[2] + "\n" + \
        "AWSリソース名：" + lst[3] + "\n" + \
        "アクセスキー利用開始日：" + lst[4] + "\n" + \

```

```

        "アクセスキー利用終了日：" + lst[5]

url = os.environ['SLACK_API_URL'] # 環境変数から読み込み
# url = "https://slack.com/api/chat.postMessage"

token = os.environ['BOT_ACCESS_TOKEN'] # 環境変数から読み込み
# token = "xoxb-10711501747-752648249329-WGXFUDahv2Dyb6Hg3yGDByRm"

headers = {"Authorization": "Bearer " + token}
# channel = "<channel or user to alert>" # ユーザーを指定するとDMが送られ
る
params = {
    'channel': user_id,
    'text': msg,
    'as_user': True
}

requests.post(url, headers=headers, data=params)
# requests.post(url, json = {"text":f"<@{id}> {msg}"})

def slack_notify_for_admin(lst, user_id, fullname):
    msg = "\nアクセスキー利用終了日まで一週間の" + fullname + "さんへ通知を行いました。" \
        "\n延長申請がなされない場合は自動的にアカウントを削除します。" + "\n" + \
        "\n"
    "アカウント名：" + lst[1] + "\n" + \
    "利用目的：" + lst[2] + "\n" + \
    "AWSリソース名：" + lst[3] + "\n" + \
    "アクセスキー利用開始日：" + lst[4] + "\n" + \
    "アクセスキー利用終了日：" + lst[5]

url = os.environ['SLACK_API_URL'] # 環境変数から読み込み
# url = "https://slack.com/api/chat.postMessage"

token = os.environ['BOT_ACCESS_TOKEN'] # 環境変数から読み込み
# token = "xoxb-10711501747-752648249329-WGXFUDahv2Dyb6Hg3yGDByRm"

headers = {"Authorization": "Bearer " + token}
# channel = "<channel or user to alert>" # ユーザーを指定するとDMが送られ
る
params = {
    'channel': user_id,
    'text': msg,
    'as_user': True
}

requests.post(url, headers=headers, data=params)

```

```

# requests.post(url, json = {"text":f"<@{id}> {msg}"})

def slack_notify_for_nouser(user_id, fullname):
    msg = "\nアクセスキー発行申請管理者からのお知らせです。" \
          "\nユーザーへSlack通知実施時に、" \
          + fullname + "さんのユーザー情報が見つかりませんでした。" \
          "\nスプレッドシートのusers_json情報を更新してください。"

    url = os.environ['SLACK_API_URL'] # 環境変数から読み込み
    # url = "https://slack.com/api/chat.postMessage"

    token = os.environ['BOT_ACCESS_TOKEN'] # 環境変数から読み込み
    # token = "xoxb-10711501747-752648249329-WGXFUDahv2Dyb6Hg3yGDByRm"

    headers = {"Authorization": "Bearer " + token}
    # channel = "<channel or user to alert>" # ユーザーを指定するとDMが送られ
る
    params = {
        'channel': user_id,
        'text': msg,
        'as_user': True
    }

    requests.post(url, headers=headers, data=params)
    # requests.post(url, json = {"text":f"<@{id}> {msg}"})

def detach_iam_user_policy(iam_name, policy_arn):
    return (
        boto3.client("iam").detach_user_policy(
            UserName=iam_name,
            PolicyArn=policy_arn
        ))

def delete_iam_user(iam_name):
    return (
        boto3.client("iam").delete_user(
            UserName=iam_name,
        ))

def delete_iam_policy(policy_arn):
    return (
        boto3.client("iam").delete_policy(
            PolicyArn=policy_arn
        ))

def delete_iam_access_key(iam_name, access_key_id):

```



```

return (
    boto3.client("iam").delete_access_key(
        UserName=iam_name,
        AccessKeyId=access_key_id
    ))

# 以下lambda_function()関数内部
def lambda_handler(event, context):
    # initial
    user_id = ""

    # set date
    nowdatetime = datetime.datetime.today() # 今日の日付
    nowdate = nowdatetime.date()
    one_week_ago = nowdate + datetime.timedelta(days=7) # 一週間前
    next_day = nowdate - datetime.timedelta(days=1) # 次の日

    # google_spreadsheet_api
    sp = get_google_api()
    SP_SHEET = os.environ['SP_SHEET'] # 環境変数から読み込み
    wks = sp.worksheet(SP_SHEET)

    values = wks.get_all_values() # 全情報をvaluesに配列として入れ込む

    for row, name in enumerate(values, start=0): # セル内容の取得
        if row == 0:
            continue

        # get cell value for spreadsheet(date)
        end_day = values[row][8] # アクセスキー利用終了日セルデータを取得する
        full_name = values[row][1] # 氏名セルデータを取得
        status = values[row][10] # アカウントの状態
        access_key_id = values[row][11] # アクセスキーID

        # exchange datetime (from str to datetime)
        strdatetime = end_day.replace("/", "") # /を""置き換えで無くす
        imptime = datetime.datetime.strptime(strdatetime, "%Y%m%d") #
datetime形式に変換
        imptime = imptime.date() # date形式に変換

        # compare one_week_ago vs import_date_value
        if not (re.search("削除", status) or re.search("延長", status)) \
            and one_week_ago == imptime: # 状態が削除もしくは延長でなく、残り一週間の場合
            # get cell value for user_json
            sp_sheet_tbl = os.environ['SP_SHEET_TBL'] # 環境変数から読み込み
            # sp_sheet_tbl = "users_json" # シート名設

```

定

```
wks_tbl = sp.worksheet(sp_sheet_tbl) # 名前テーブルシートへアクセス
```

ス

```
values_tbl = wks_tbl.get_all_values() # 全情報をvalues_tblに配列として入れ込む
```

```
for row2, name2 in enumerate(values_tbl, start=0): # 名前シートからセル内容の取得
```

```
    last_name = values_tbl[row2][3]
```

```
    first_name = values_tbl[row2][4]
```

```
    if last_name == "" or first_name == "":
```

```
        user_id = "none"
```

```
    elif last_name in full_name and first_name in full_name:
```

```
        lst = []
```

```
        user_id = values_tbl[row2][0] # ユーザーID
```

```
        obj = values_tbl[row2][2] # 申請目的・理由
```

```
        resource = values_tbl[row2][4] # AWSサービス名
```

```
        start_day = values_tbl[row2][7] # アクセスキー利用開始日
```

```
        account = values_tbl[row2][9] # アカウント名
```

```
        lst = [user_id, account, obj, resource, start_day,
```

```
end_day]
```

```
        # to slack
```

```
        slack_notify(lst)
```

```
        admin_id_1 = os.environ['ADMIN_ID_1'] # 【暫定】現状通知
```

対象のIDを固定

```
        slack_notify_for_admin(lst, admin_id_1, full_name)
```

```
        admin_id_2 = os.environ['ADMIN_ID_2'] # 【暫定】現状通知
```

対象のIDを固定

```
        slack_notify_for_admin(lst, admin_id_2, full_name)
```

```
        break
```

```
    else:
```

```
        user_id = "none"
```

```
    # ユーザーが見つからなかった場合の通知
```

```
    if user_id == "none":
```

```
        admin_id_1 = os.environ['ADMIN_ID_1'] # 【暫定】現状通知対象の
```

IDを固定

```
        slack_notify_for_nouser(admin_id_1, full_name)
```

```
        admin_id_2 = os.environ['ADMIN_ID_2'] # 【暫定】現状通知対象の
```

IDを固定

```
        slack_notify_for_nouser(admin_id_2, full_name)
```

```
    elif not (re.search("削除", status) or re.search("延長", status)) \
```

```
        and next_day == imdate: # 状態が削除もしくは延長でなく、終了日
```

を過ぎている場合

```

# アカウントを削除する
account = values[row][9] # アカウント名
arn = "arn:aws:iam::729700637718:policy/"
policy_arn = arn + account
# detach
detach_iam_user_policy(account, policy_arn)

# detachした後スリープいるかも
time.sleep(5)

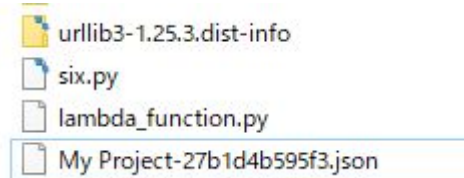
# delete
delete_iam_access_key(account, access_key_id)
delete_iam_user(account)
delete_iam_policy(policy_arn)

# 状態列に「削除済み」を記載する
wks.update_cell(row + 1, 11, "削除済み")
else:
    pass

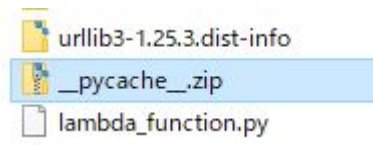
print("end_function")

```

同じ場所に、「3. スプレッドシートの設定」で作成した「(project_name)-****.json」を保存する。



全てのフォルダ、ファイルを一緒にZIPファイルに圧縮する。（ファイル名は任意）



AWSのLambdaサービスを表示し、関数コードの「コードエントリタイプ」で「.zip ファイルをアップロード」を選択する。

関数コード 情報

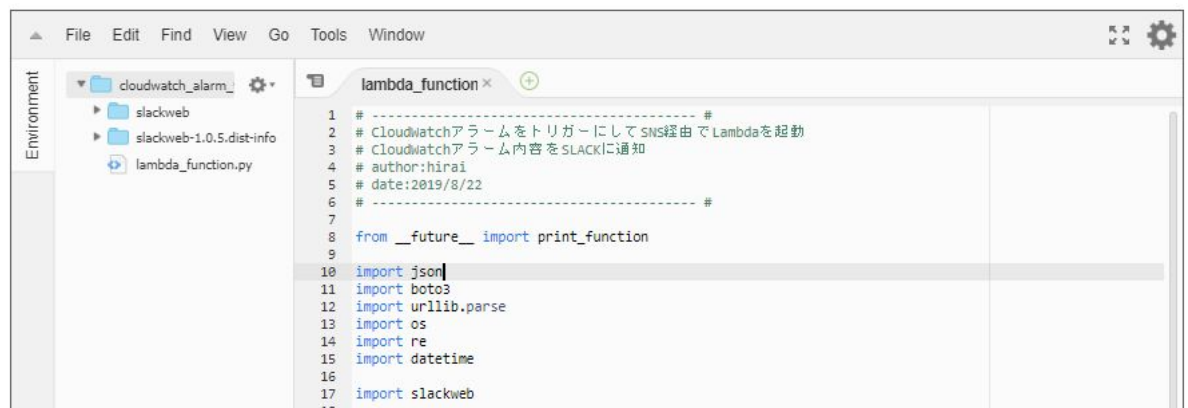
コードエントリタイプ

.zip ファイルをアップ...

関数パッケージ

アップロード

「アップロード」ボタンを押下し作成したZIPファイルを選択して、「開く」を押下。



c. 環境変数の設定

「環境変数」の設定を以下に列挙する。実際の設定は、template.yamlで行う。
※別マニュアル参照（ローカルPycharmでのAWS Lambda Functionデプロイ方法）

キー	値	内容
SLACK_API_URL	https://slack.com/api/chat.postMessage	BotでのSlack通知アドレス
BOT_ACCESS_TOKEN	xoxb-10711501747-752648249329-WGXFUDahv2Dyb6Hg3yGDBByRm	Botアプリアクセストークン（5章でコピーした値）
SP_SHEET_KEY	1ZLJf6GQE5ywcATKZxYW WI_EdLjsNHjICYazrS_DeGcs	書き込むスプレッドシートのID（ブラウザのアドレスバーに表示されているURLの一部）
SP_SHEET	key_request_form	スプレッドシートのシート

		名
SP_SHEET_TBL	users_json	氏名とユーザーID対比シートのシート名
ADMIN_ID_1	U75R8M3J7	管理者用ユーザーID 1
ADMIN_ID_2	ULF3WH04C	管理者ユーザーID 2

「保存」を押下。

テスト

保存

以上