클라우드 컴퓨팅 과제 #1

: AWS Lambda 실습

2016130927 박준영

1. 시나리오 구성

AWS Lambda 실습 영상과 동일한 시나리오를 자바스크립트가 아닌 파이썬 언어를 통해 구현하였다. 자세한 시나리오 구성은 다음과 같다. 서버리스 환경에서 트리거로 연결된 S3 버킷에 이미지 파일을 업로드할 때, Lambda 함수는 이 파일을 다운로드 한 뒤 환경변수에 설정된 세 가지 함수(흑백, 투명, 반전) 중 한 가지를 실행한다. 이후 결과물이설정된 버킷에 자동으로 업로드된다.

기존 실습에서는 업로드되는 파일 확장자명과 입력되는 환경변수에 해당 사항이 없으면 call back하는 방식으로 구현되었으나, 본 시나리오는 이중 확장자명만 검사하여 jpeg 또는 png만 받아들이고, 나머지는 return None으로 그대로 종료한다. 반면 환경변수 값은 설정한 변수 중에 존재하지 않더라도 가공 없이 업로드되도록 구현하였다.

2. Lambda 함수 코드

Lambda 함수 CL-ASSIGNMENT1의 구성은 다음과 같다.

- (1). Lambda_handler: AWS Lambda 함수의 메인 함수로 이벤트 트리거로 입력된 event를 통해 버킷과 키, 파일 정보 등을 인식하고, S3 버킷에서 파일을 다운로드한다. Splitext 함수를 통해 다운로드한 파일의 확장자명이 jpeg 혹은 png인지 확인한다. 이후 환경변수 입력값이 설정된 세 가지 함수(흑백, 투명, 반전) 중에 존재하면 각 함수를 실행, 다운로드한 이미지 파일을 변형시키고, 존재하지 않으면 아무 가공도 하지 않는다. 마지막으로 트리거로 연결된 버킷 이름에 '-output'이 추가된 이름을 가진 버킷에 위의 과정을 거친 이미지 파일을 업로드한다.
- (2). gray_image: 가공할 이미지 파일을 받아 convert 함수를 이용해 grayscale, 즉 흑백으로 만든다. 가공한 파일은 이후 업로드할 곳에 저장한다.
- (3). transparent_image: 가공할 이미지 파일을 받아 RGBA가 아니라 RGB로 convert한다. RGB 값을 가지고 있는 이 이미지 파일은 다시 흑백 반전되어 흰색 값이 검은색 값이된다. 이 상태를 흑백 모드로 convert하면, 이 이미지를 기존의 RGB에 alpha(투명도를 결

정하는 값)로 putalpha시켜줄 수 있다. 즉, Convert("L")을 통해 검정색이 된 배경을 투명하게 설정한 이미지를 업로드한다.

- (4). negative_image: transparent_image가 알파 값을 절반으로 하여 투명화하는 과정이라면, negative 즉 흑백 반전은 각 픽셀이 가지고 있는 RGB 값을 완전히 반전(즉 255 각 RGB 색상값)한 것이다.
- (5). Others: PIL로부터 Image를 Import해야 하므로, Lambda에 추가할 계층에 업로드할 필요가 있다. 또한 어떤 종류의 이미지 가공을 할지 사용자의 입장에서 결정해야 하므로, 환경변수 transform을 고려해야 한다(세 가지 설정값 이외는 Default로 가공 없음)

다음은 AWS Lambda 함수 코드를 캡쳐한 스크린샷이다.

```
В
          lambda_function × +
         import boto3
2 import os
3 import sys
         import uuid
        from PIL import Image, ImageOps, ImageFilter
         transform = os.environ.get('TRANSFORM_FUNC')
         s3_client = boto3.client('s3')
  10
  11 def gray_image(image_path, gray_path):
               with Image.open(image_path) as image:
gray = image.convert("L")
  13
                      gray.save(gray_path)
  14
  15
                      # grayscale image
  16
         def transparent_image(image_path, transparent_path):
               with Image.open(image_path) as image:
    transparent = image.convert("RGB")
    image_invert = ImageOps.invert(transparent)
  18
  19
                     image_invert_L = image_invert.convert("L")
transparent.putalpha(image_invert_L)
  21
  22
                      transparent.save(transparent_path,
                      #transparent image
  25
        def negative_image(image_path, negative_path):
              with Image.open(image_path) as image:
negative = image.point(lambda i: 255 -i)
               negative.save(negative_path)
# negativize image
  30
  31
       def raw_image(image_path, raw_path):
    with Image.open(image_path) as image:
  32
  33
                   raw = image
raw.save(raw_path)
  35
                # raw(original) image
  36
       def lambda_handler(event, context):
  38
               famular (event['Records']:
    bucket = record['s3']['bucket']['name']
    key = record['s3']['object']['key']
    download_path = '/tmp/{}{}'.format(uuid.uuid4(), key)
    upload_path = '/tmp/resized-{}'.format(key)
  40
  41
  43
 44
45
46
                      s3_client.download_file(bucket, key, download_path)
                      if (os.path.splitext(download_path)[1] == '.jpeg'): print("Image type jpeg")
elif (os.path.splitext(download_path)[1] == '.png'): print("Image type png")
  48
                      else:
  49
                            print("Unsupported type detected. jpeg and png only allowed.")
  50
                      #check whether its format jpeg/png or not. otherwise return.
  51
  52
                      if (transform == 'gray'): gray_image(download_path, upload_path), print("grayscale transform")
elif (transform == 'transparent'): transparent_image(download_path, upload_path), print("transparent transform")
elif (transform == 'negative'): negative_image(download_path, upload_path), print("negative transform")
  53
  54
                      else: raw_image(download_path, upload_path), print("No transform")
s3_client.upload_file(upload_path, '{}-output'.format(bucket), key)
# Upload_processed_file_to_output_bucket
  56
  57
```

3. 레이어 설정

		계층 추가					
		함수 런타임 설정					
		런타임 Python 3.8		아키탈 x86_6			
		계층 선택					
	계층 소스 Info 호환 런타임 및 명령 세트 아키텍처가 있는 계층에서 선택하거나 계층 버전의 Amazon 리소스 이름(ARN)을 지정합니다. 새로운 계층을 선수도 있습니다.					생성할 	
		○ AWS 계층 AWS에서 제공하는 계 계층을 선택합니다.	층 목록에서 A'	ト용자 지정 계층 WS 계정 또는 조직에서 충 목록에서 계층을 선	생성한	ARN 지정 ARN을 제공하여 계층을 지정 다.	합니
	ARN 지정 Amazon 리소스 이름(ARN)을 제공하여 계층을 지정합니다.						
		am:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-Pillow:14 확인 설명 Pillow==8.4.0 0fd903094d8acec7efb6130dc6da17f442df50dfd3e9ea0d9eaf8e2dcdcedca0 호환 런타임 Python 3.6, Python 3.7, Python 3.8					
		호환 아키텍처 -					
						취소	추가
계층 Info							편집 [Add a layer
령합 주문	이름		Layer 버전	호환 런타임	호환 아키텍처	버전 ARN	
	Klayers-pyt	hon38-Pillow	14	python3.6, python3.7, python3.8	-	arn:aws:lambda:us-east-1:77 Pillow:14	0693421928:layer:Klayers-python38-

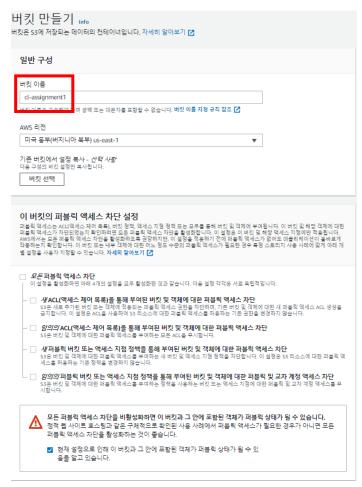
실습 영상에서 imageMagic, node mudule 계층을 설정하여 사용하였으나, 본 시나리오에서는 파이썬 언어를 사용하며, 이미지 가공을 위한 도구로 PIL을 이용하고자 하였다. 위 두경우는 직접 계층에 zip 파일을 업로드해 Lambda 함수에 추가하여 사용했는데, PIL 라이브러리는 ARN을 통해 다른 사용자가 업로드한 계층를 사용 가능했다. 따라서 AWS 계층이나 사용자 지정 계층이 아닌 ARN만 지정해서 곧바로 레이어를 추가할 수 있었다.

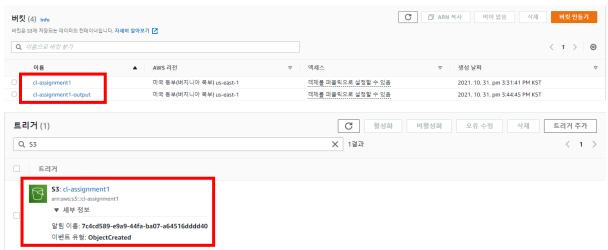
추가한 레이어 Klayers-python38-Pillow를 통해 Lambda 함수에서 PIL.Image 등을 import할 수 있었다.

4. S3 구성 및 이벤트 트리거 설정

Lambda 함수에 이벤트 트리거로 연결된 S3 버킷 cl-assignment1은 ObjectCreated 유형에 반응하도록 설정되었다. 즉 특정 오브젝트가 만들어졌을 때(따라서 파일 업로드와 본 시나리오에서는 그 의미가 상동), lambda handler를 통해 그 event를 관리할 수 있다.

모든 퍼블릭 액세스 차단을 비활성화하여 퍼블릭 상태가 가능하도록 만든 S3 버킷을 총 두 개(트리거용, 업로드용) 만들어 각각 cl-assignment1, cl-assignment1-output으로 이름 붙였다. 트리거 버킷에 '-output' 문자가 추가된 것으로 Lambda 함수에서 업로드용 버킷을 판별하기 때문이다. 이후 cl-aissignmnet1을 Lambda 함수에 트리거로 추가하였다.



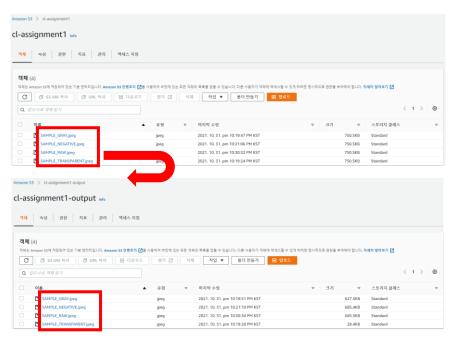


5. 이벤트 발생 이후 결과

(1). JPEG 파일 업로드/환경변수 'raw', 'gray', 'transparent', 'negative':



위는 환경 변수 값에 각각 'raw(사실 이후 세 값을 제외한 그 어떤 값을 넣어도 작동한다)', 'gray', 'transparent', 'negative'로 준 모습이다.



Lambda 함수의 트리거로 설정한 S3 버킷 cl-assignmnet1에 SAMPLE 파일(jpeg 확장자)을 각각 네 번 업로드한 모습이다. 모두 동일한 SAMPLE이며 업로드 시 퍼블릭 환경을 허용하도록 설정하였다. Lambda 함수 코드 내에서 자동적으로 트리거 이벤트 버킷 'cl-assignmnet1'에 '-output'을 추가한 이름을 가진 버킷에 가공한 이미지 파일을 업로드하였으므로, 자동적으로 본 버킷에 파일 업로드가 이루어진다. 다음은 업로드할 때 사용한 SAMPLE 이미지 파일과 각결과물이다.



SAMPLE.jpeg



(1). SAMPLE_RAW.jpeg



(2). SAMPLE_GRAY.jpeg



(3). SAMPLE_TRANSPARENT.jpeg

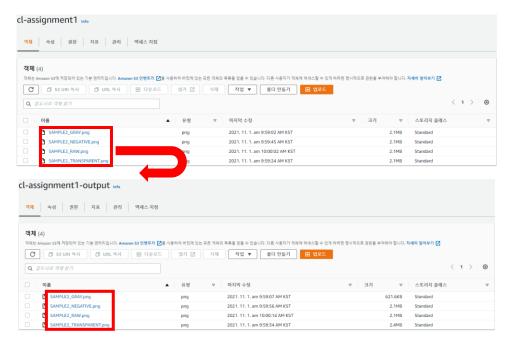


(4). SAMPLE_NEGATIVE.jpeg

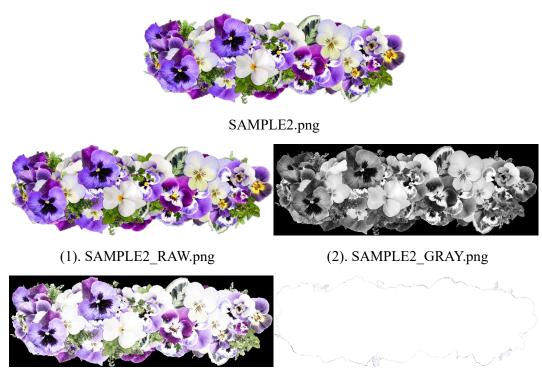
SAMPLE 이미지 파일은 각각 아무런 가공도 거치지 않은 (1) raw, 흑백으로 표현한 준 (2). Gray, 배경에 투명도를 더한 (3). Transparent, 각 RGB 값을 반전한 (4). Negative 과정이 적용된 이미지 파일로 S3 버킷(cl-assignment1-output)으로 업로드되었다.

(2). PNG 파일 업로드/환경변수 'raw', 'gray', 'transparent', 'negative':

(1)과 마찬가지로 PNG 이미지 파일을 각각 환경 변수 네 가지 조건을 준 상태로 이벤트 트리거 S3 버킷 cl-assignment1에 업로드하였다.



JPEG 이미지 파일을 업로드한 경우와 마찬가지로, PNG 파일을 업로드하였을 때에도 자동으로 트리거 함수의 이름에 매핑된 다른 S3 버킷 cl-assignment1-output에 자동으로 Lambda 함수가 적용된 SAMPLE2 이미지 파일이 업로드된다.



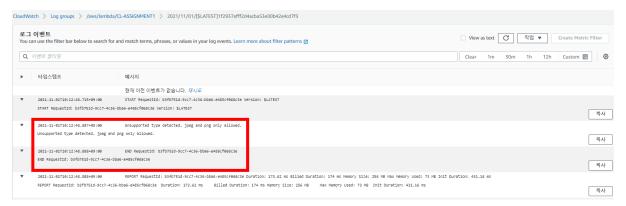
31). SAMPLE2 TRANSPARENT.png

(4). SAMPLE2 NEGATIVE.png

(3). JPG 파일 업로드/환경변수 'gray' (설정된 확장자명이 아님):



SAMPLE3.jpg



Lambda 함수 코드에서 jpeg와 png 확장자명만 허용했으므로 본 jpg 파일은 가공하지 않는다고 설정하였다. 따라서 곧바로 return None 처리되어, 비록 트리거인 버킷에는 파일이 업로드되었으나 업로드 과정은 이루어지지 않았다.

