Contents

1 Idea

The idea is to leverage AWS machine learning services to provide an image description and read the text when the device is connected to the internet:

1. The user selects an image and requests to process it
2. The image is loaded into the cloud
3. Text in image is detected and read
4. Contextual description tags are generated
5. All the information is aggregated and vocalized
6. An MP3 file is created and stored
7. The MP3 file to be played is made available to the user

The frontend parts 1 and 7 as described above are only for demonstration purpose: they can be adapted to match the end-application needs.

2 Key Components

The solution comprises the following components:

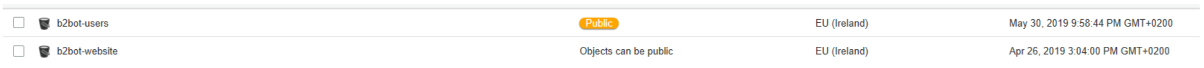
- S3 bucket
- Dynamo (NoSQL) database
- SQS message queue
- API Gateway
- Lambda functions
- API functions to connect the frontend and backend parts

3 Ingredients

3.1 S3 buckets

There are two S3 buckets:

- The first one (b2bot-website) hosts the frontend files for the static hosting of the website
- The second one (b2bot-users) hosts the user data (image and audio files)



b2bot-users	Public	EU (Ireland)	May 30, 2019 9:58:44 PM GMT+0200
b2bot-website	Objects can be public	EU (Ireland)	Apr 26, 2019 3:04:00 PM GMT+0200

Bucket name	Bucket permission	Bucket policy	CORS	Object permission
B2bot-website	private	no	yes	public
B2bot-users	public	yes	yes	Made public-read upon upload or creation

<u>Bucket_cors.txt</u>
Cross origin resource sharing (CORS) defines a way for a client web application that is loaded in one domain, to interact with resources in a different domain. In our case, the website endpoint must interact with the S3 bucket endpoint.
The bucket_cores.txt file provides the CORS definition for the S3 bucket.
The file content must be copied to the CORS bucket policy.

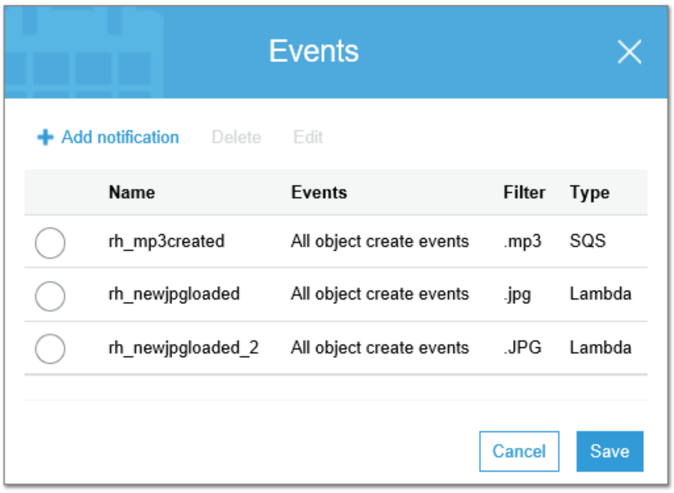
<u>Bucket_policy.txt</u>
The bucket_policy.txt file provides the permission policy for the S3 bucket.
The file content must be copied to the bucket policy.
Currently, the defined policy allows only the PUTObjects (into the S3 bucket).

3.1.1 B2bot-website

B2bot-website is set up for static website hosting.
So far, 4 files are used for that purpose: index.html, error.html, rh_scripts.js, styles.css

3.1.2 B2bot-users

B2bot-users is set up to trigger 3 events:



Name	Events	Filter	Type
rh_mp3created	All object create events	.mp3	SQS
rh_newjpgloaded	All object create events	.jpg	Lambda
rh_newjpgloaded_2	All object create events	.JPG	Lambda

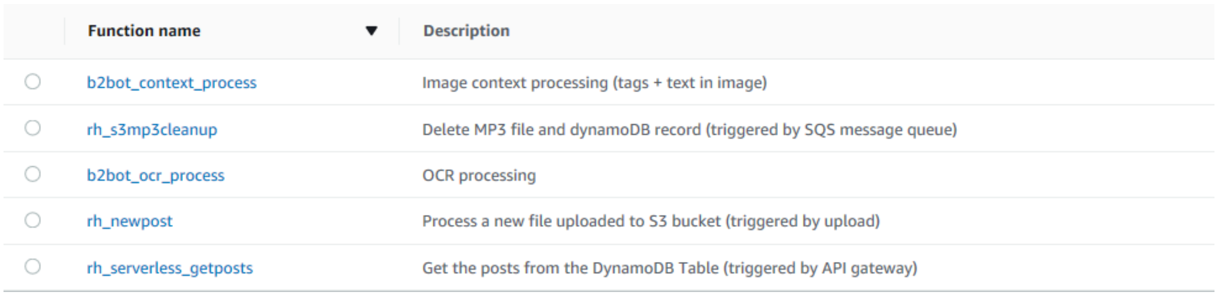
Event	**Suffix filter**	**Trig**	**Goal**
Mp3created	.mp3	SQS message	When a .MP3 file is created in the bucket, a message is pushed to the SQS queue for a pre-defined delivery time. When the delivery time has elapsed, a lambda function is triggered to delete the MP3 file and the related DB record automatically.
Newjpgloaded	.jpg	Lambda function	When a .jpg or .JPG file is loaded into the bucket, a lambda function is triggered to process the image file.
Newjpgloaded_2	.JPG	Lambda function	

The folders are organized as follows:
B2bot-users -> user -> function -> filename (jpg, mp3)
The function depends on the frontend selection (context or ocr). The uploaded file is stored in the related folder and the lambda processing forks accordingly.

3.2 Lambda

3.2.1 Overview

The architecture comprises 5 lambda functions:
x

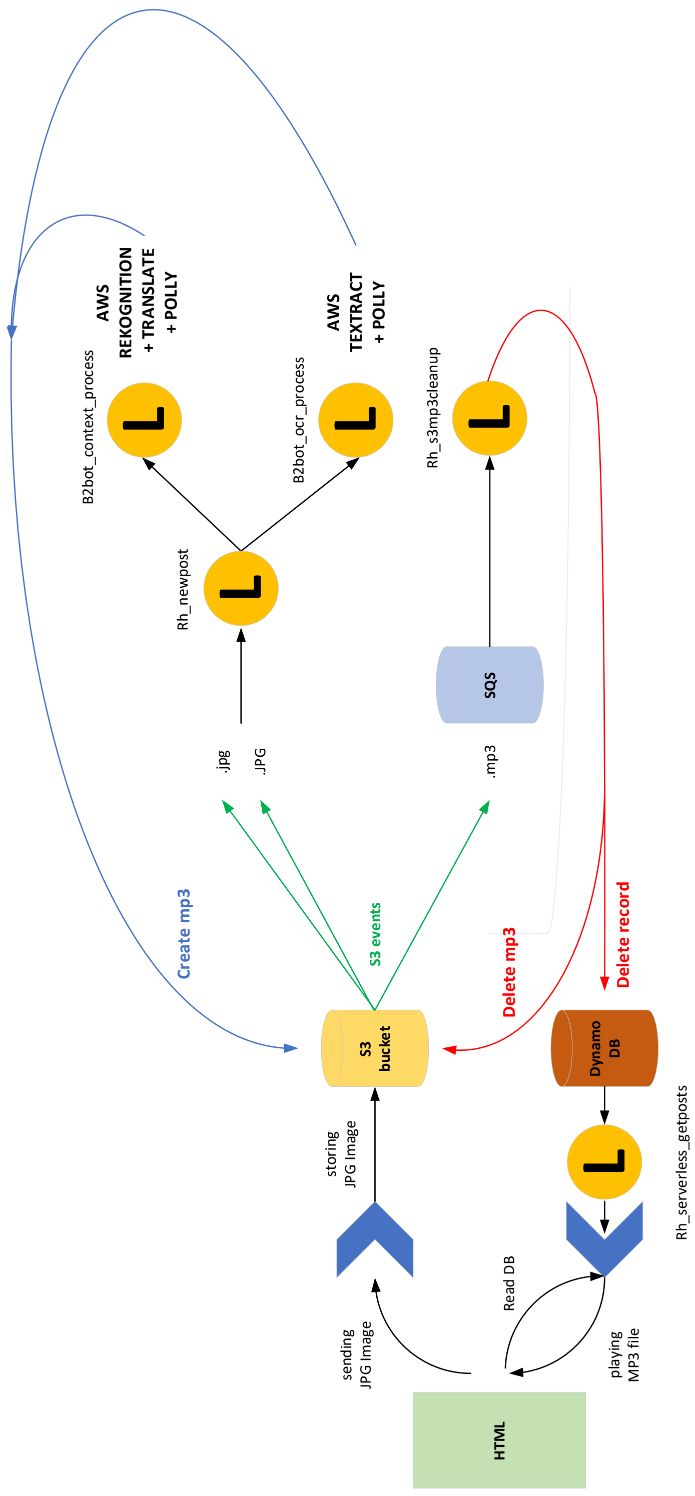


Function name	Description
b2bot_context_process	Image context processing (tags + text in image)
rh_s3mp3cleanup	Delete MP3 file and dynamoDB record (triggered by SQS message queue)
b2bot_ocr_process	OCR processing
rh_newpost	Process a new file uploaded to S3 bucket (triggered by upload)
rh_serverless_getposts	Get the posts from the DynamoDB Table (triggered by API gateway)

Lambda_policy.txt
An IAM role *rh_lambdapolly* is set to be applied to the lambda functions with a policy attached. The policy must be copied from the lambda_policy.txt file. It lists the services that the lambda function can access.

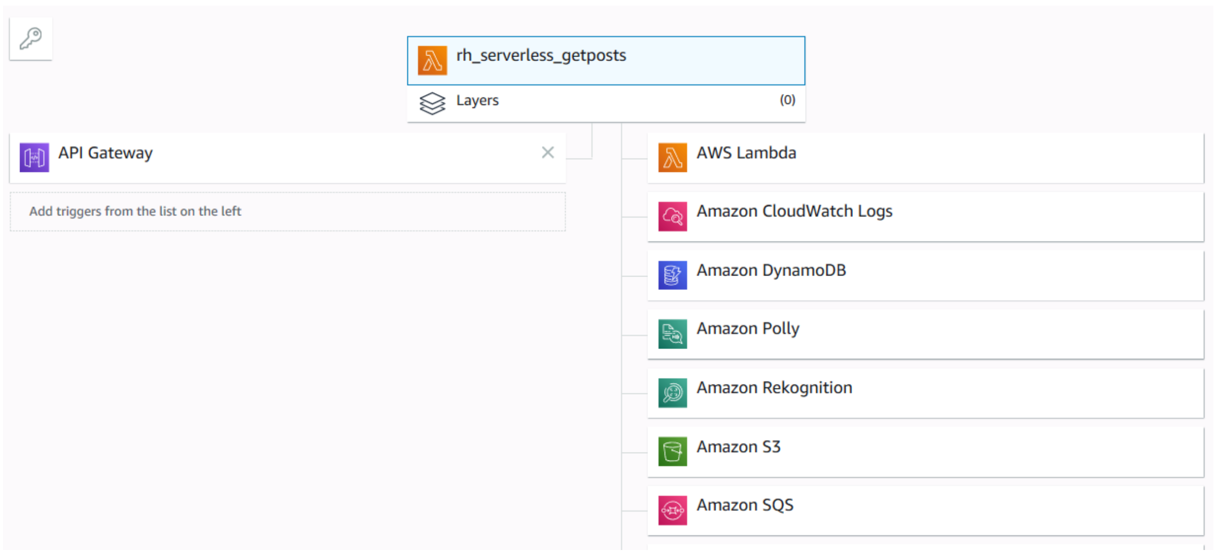
Notice:
For simplicity, the same policy is applied to all the lambda functions. Ideally a specific role and a specific policy should be defined for each lambda function individually to comply with the least privilege permission best practice (give every user or process the minimal amount of permissions that are required to get job done).

3.2.2 Process flow



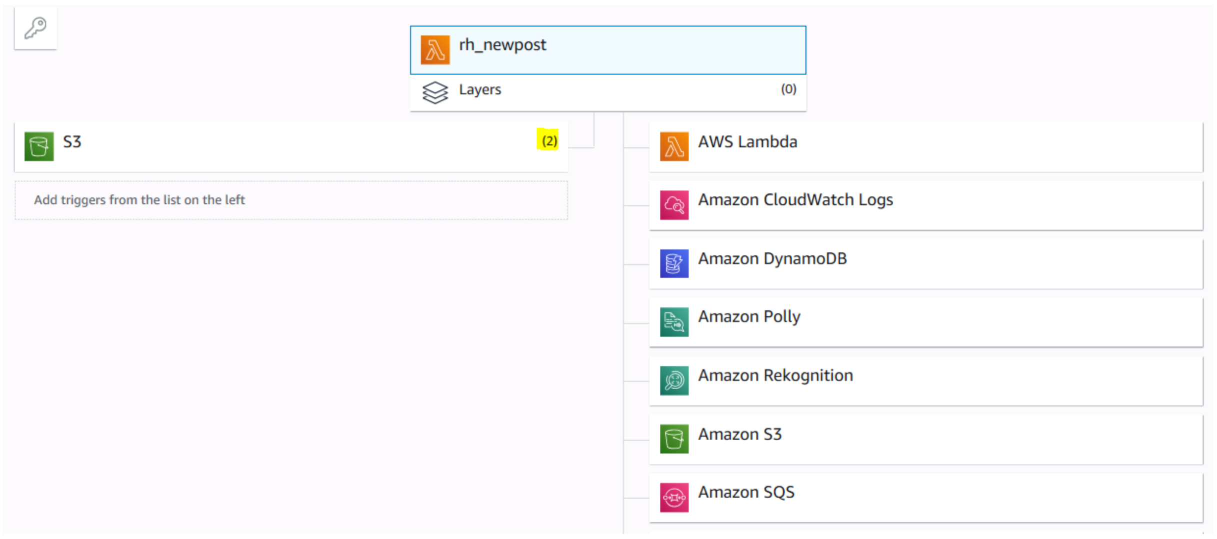
3.2.3 Functions

<u>Rh_getposts.py</u>
Upon request from the frontend interface, the lambda function retrieves the selected post or all posts (*) from the database.
This function is triggered by the API gateway.



<u>Rh_newpost.py</u>
This function is triggered by the PUT of a new object into the S3 bucket.
The PUT operation generates an event that triggers the execution of the lambda function.

Depending on the selected function (text or context) that corresponds also to the location where the image file is stored in the S3 bucket, the lambda function calls another nested lambda function, either b2bot_context_process or b2bot_ocr_process.



B2bot_context_process.py

The function executes the following tasks:

- Clean the polly tmp directory (for old submissions)
- Retrieve the post ID (key) from the events log
- Use Rekognition service to extract the text from the image (detect_text)
- Use Rekognition service to extract the context tags from the image (detect_labels)
- Build the character string, aggregating the text and the tags
- Use Polly service to synthesize the text into a MP3 file
- Store the MP3 file into the S3 bucket
- Update the DynamoDB NoSQL database

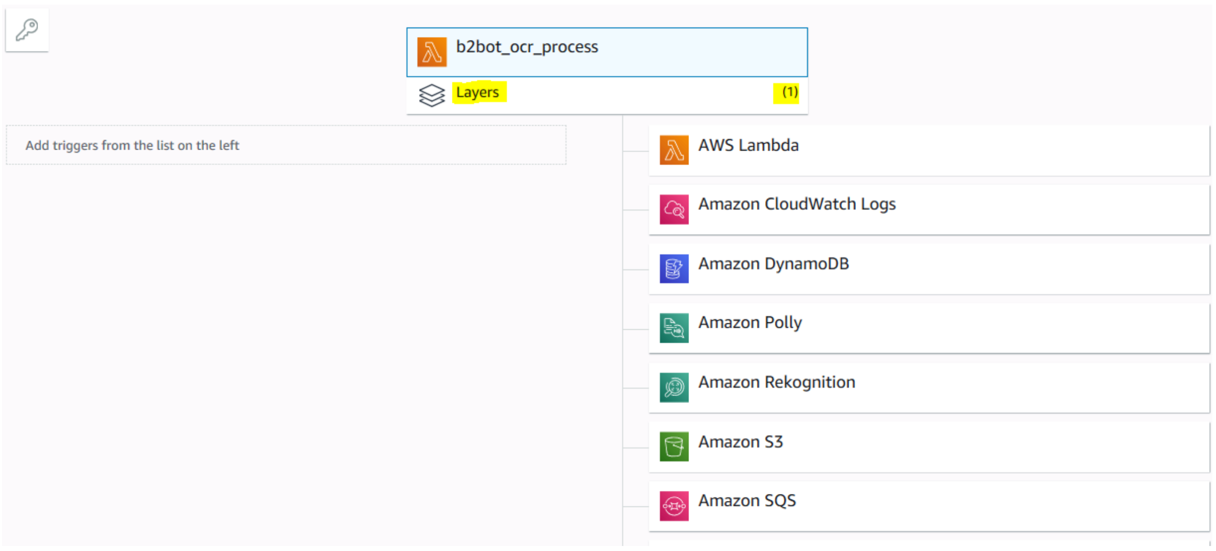
B2bot_ocr_process.py

The function executes the following tasks:

- Clean the polly tmp directory (for old submissions),
- Retrieve the post ID (key) from the events log
- Use Textract service to extract the text from the image
- Build the text block chunks to be managed by Polly
- Use Polly service to synthesize the text into a MP3 file
- Store the MP3 file into the S3 bucket
- Update the DynamoDB NoSQL database

As the boto3 library used by Lambda is not up to date and currently does not support Textract, additional steps are required to make it work:

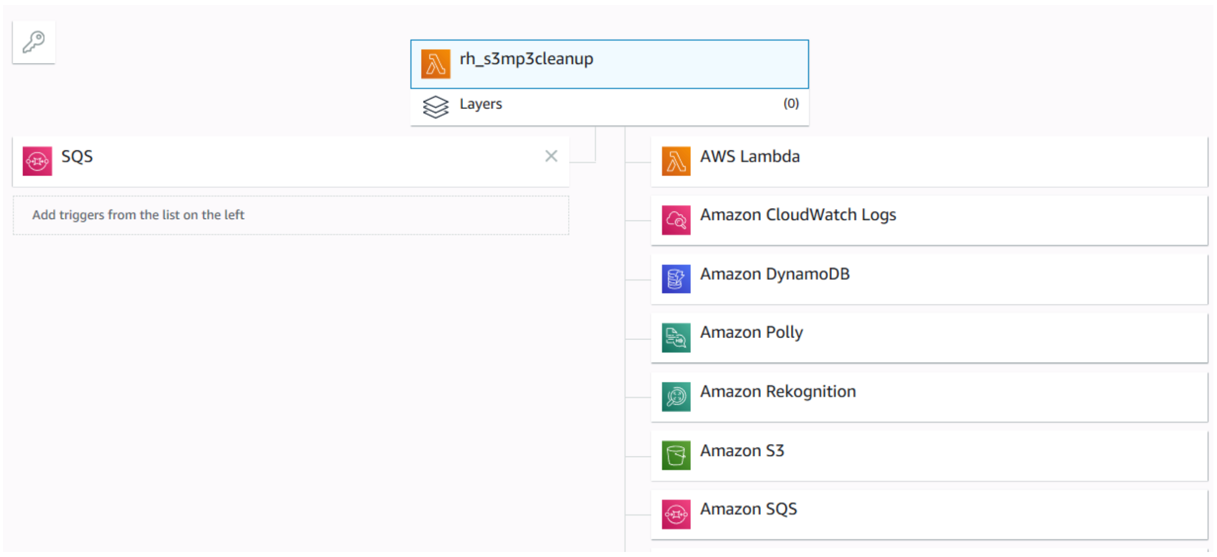
- Install the latest boto3 lib on the local drive
- Compress it to zip
- Create a layer in the lambda service and upload the zip file
- Assign the layer to the b2bot_ocr_process function



Rh_s3mp3cleanup

The function executes the following tasks:

- Delete the mp3 file from the S3 bucket
- Delete the record (associated to the MP3 file) from the DynamoDB table



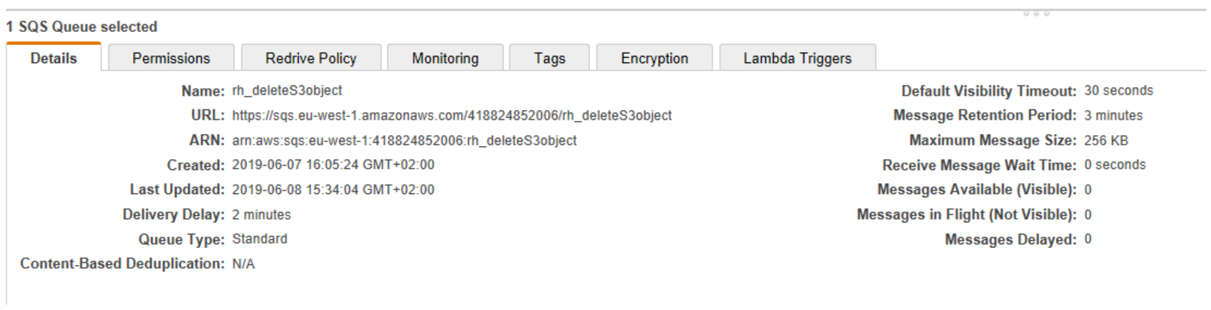
3.3 SQS

Whenever a mp3 file is created in the S3 bucket, an event is generated to push a message into the SQS queue. The delivery delay is predefined (2 minutes) and corresponds to the time that the message remains in the queue before it is delivered and triggers the subsequent lambda function (rh_s3mp3cleanup). The lambda function is responsible to delete the previously created mp3 file and the related DynamoDB record.

The delivery delay is therefore the lifetime of the mp3 file in the S3 bucket.

The message retention period is set to 3 minutes so that the message is automatically deleted in the SQS queue.

These times can be changed for longer values if needed.



Name:	rh_deleteS3object	Default Visibility Timeout:	30 seconds
URL:	https://sqs.eu-west-1.amazonaws.com/418824852006/rh_deleteS3object	Message Retention Period:	3 minutes
ARN:	arn:aws:sqs:eu-west-1:418824852006:rh_deleteS3object	Maximum Message Size:	256 KB
Created:	2019-06-07 16:05:24 GMT+02:00	Receive Message Wait Time:	0 seconds
Last Updated:	2019-06-08 15:34:04 GMT+02:00	Messages Available (Visible):	0
Delivery Delay:	2 minutes	Messages in Flight (Not Visible):	0
Queue Type:	Standard	Messages Delayed:	0
Content-Based Deduplication:	N/A		

The SQS requires a permission for the S3 bucket to push messages into it. The permission is described in the sqs_permission.txt file. In the AWS console, it can be updated in the permission tab of the SQS queue.

3.4 Frontend APIs

Rh_scripts.js
The file comprises the 2 functions to:

1. Upload the JPG image file to the S3 bucket
2. Retrieve the fields of the records from the dynamo DB NoSQL database

Index.html
Error.html
Styles.css