

AUTOMATING CELL COUNTING WITH LAMBDA

1. INTRODUCTION

The aim of this practical session is to learn about the AWS services known as **Lambda** and **S3** through a practical exercise that counts the number of cells from an image. Nevertheless, since we will be using Docker images, we will also learn about the Elastic Container Registry (**ECR**) service.

What is Lambda?

- Lambda is a serverless computing service that allows us to run code without provisioning or managing servers.
- We can upload code written in language to Lambda and it will automatically scale and execute it code in response to triggers.
- Triggers can include events from other AWS services, such as S3, DynamoDB, API Gateway, etc., or they can be custom events.
- Lambda functions can be used for various purposes like data processing, automation, real-time file processing, etc.

What are S3 buckets?

- S3 is an object storage service provided by AWS, designed to store and retrieve large amounts of data.
- It allows us to store any type of data (files, images, videos, etc.) in the form of objects within buckets.
- Objects stored in S3 are organized into buckets, which act as containers for the objects. Each bucket must have a unique name globally within AWS.
- S3 provides high durability, availability, and scalability for storing data.

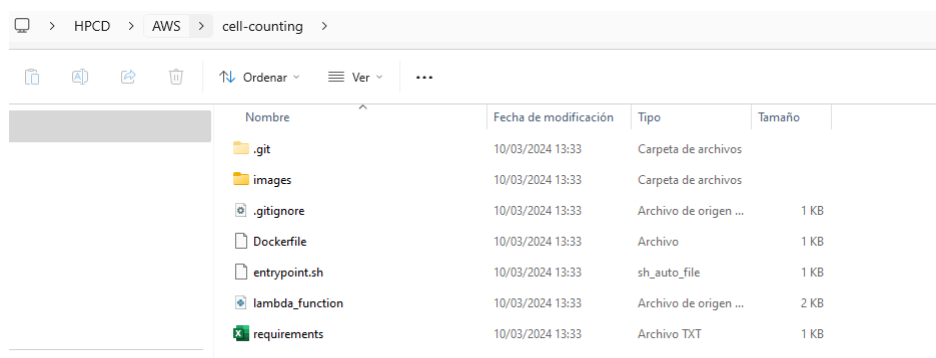
We are going to work with Lambda and AWS to automate the cell count in images. The overall idea is that cell images will be uploaded into an s3 bucket and this will trigger Lambda, which executes a script that counts the cells on that image and returns a new image to a bucket with the cell contours.

Note: The objective of this exercise is to be able to solve the cell counting task using AWS resources, not to create the code and the Docker file from scratch.

2. CREATING THE DOCKER IMAGE

We start from a repository [1] that already contains the following files:

- [lambda_function.py](#): This is the script that Lambda needs to run and that performs the cell counting. The code downloads the image file from the source S3 bucket specified in the Lambda event and uses OpenCV to process the image, specifically to detect circles using the Hough Circle Transform. Then it draws circles and rectangles around the detected circles on the image and generates a plot using matplotlib, showing the original image with detected circles. It finally saves the plot as a JPEG file in the /tmp directory. It also uploads the processed image to the destination S3 bucket specified by the *DST_BUCKET* environment variable and returns a response indicating the number of detected circles.
- [Dockerfile](#): This is a text file that contains instructions for building a Docker image. The Dockerfile specifies the environment and steps required to assemble an image that can be run as a container.
- [requirements.txt](#): Specifies the necessary libraries to run the script.
- [entrypoint.sh](#): It is a shell script that defines the initial command to be executed when the container starts. This script often sets up the environment, performs any necessary configurations, and then launches the main application process.
- [images](#): A folder that contains cell images to test Lambda.



Nombre	Fecha de modificación	Tipo	Tamaño
.git	10/03/2024 13:33	Carpeta de archivos	
images	10/03/2024 13:33	Carpeta de archivos	
.gitignore	10/03/2024 13:33	Archivo de origen ...	1 KB
Dockerfile	10/03/2024 13:33	Archivo	1 KB
entrypoint.sh	10/03/2024 13:33	sh_auto_file	1 KB
lambda_function	10/03/2024 13:33	Archivo de origen ...	2 KB
requirements	10/03/2024 13:33	Archivo TXT	1 KB

After cloning the repository, we open Docker Desktop to ensure the engine is running and create the Docker image.

The Docker image is created by running this command on the terminal:

```
docker build -t hdbc-cell-counting .
```

3. PUSHING DOCKER IMAGE TO ECR REPOSITORY

ECR is a fully managed Docker container registry service provided by AWS. It allows us to store, manage, and deploy Docker container images.

We log into ECR and create a new repository with the name “hdbc-cell-counting” and keeping the default settings.

Next:

1. Go to the local terminal and log in with the AWS CLI through this command. (Remember to replace your ECR URI from the ECR repository created before in all the steps):

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS \  
--password-stdin 654654389363.dkr.ecr.us-east-1.amazonaws.com
```

The command above retrieves a login password for ECR and logs your Docker client to your AWS ECR repository.

2. Tag the image:

```
docker tag hdbc-cell-counting:latest \  
654654389363.dkr.ecr.us-east-1.amazonaws.com hdbc-cell-counting:latest
```

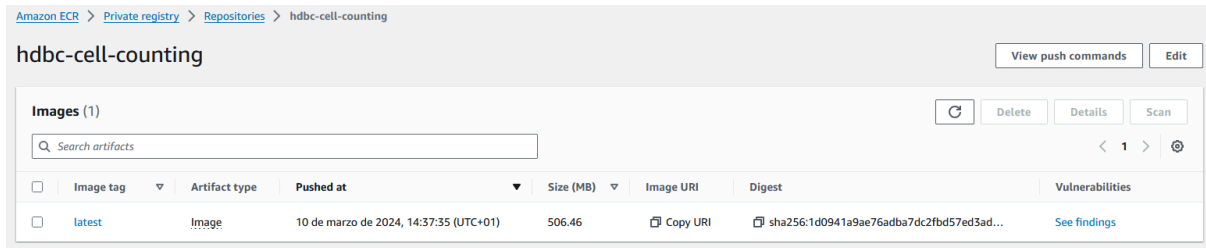
This command tags your local Docker image (hdbc-cell-counting:latest) with the full ECR repository URI, preparing it for the push to AWS ECR. This tag tells Docker where the image should reside in the cloud.

3. Push the image to the ECR:

```
docker push  
654654389363.dkr.ecr.us-east-1.amazonaws.com/hdbc-cell-counting:latest
```

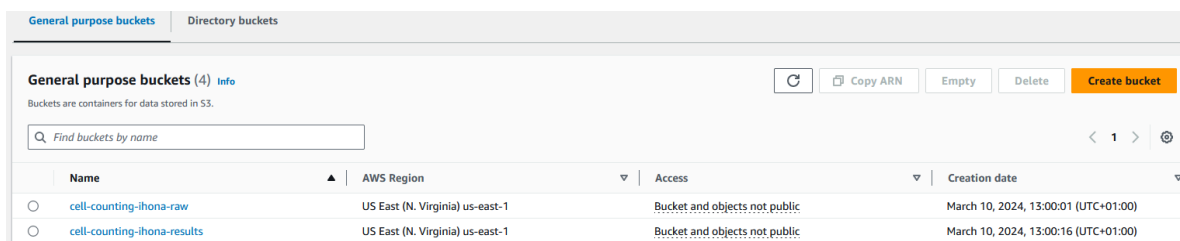
This uploads the Docker image to the specified AWS ECR repository, making it accessible for AWS services like Lambda.

After this last step, we are able to see the image pushed in ECR:

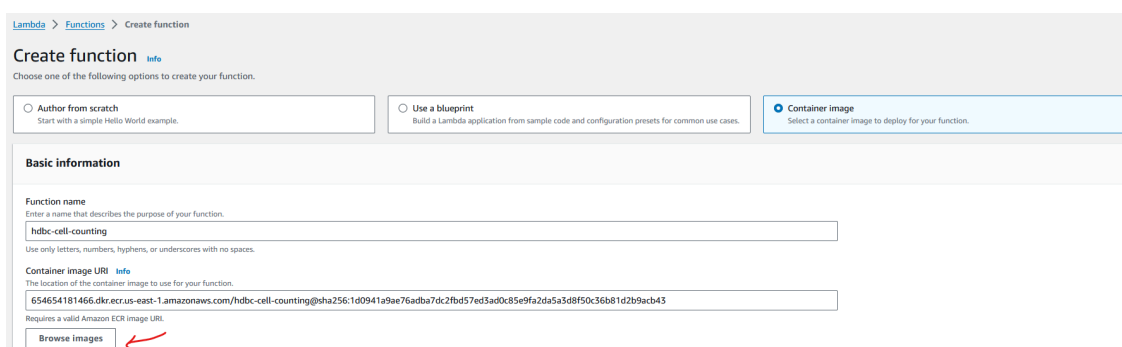


4. LAMBDA AND S3 CREATION

First of all, we will create two s3 buckets, one to store the images of the cells and the other one to store the results of the cell counting task:



Next, we create a lambda function from a container image and browse the Docker image that was previously created:




Then, we add an s3 trigger to the Lambda, specifying that, whenever a file is put in the specified s3 bucket, Lambda should be executed:

Lambda > Add trigger

Add trigger

Trigger configuration [Info](#)


S3
 aws asynchronous storage

Bucket
 Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.

Bucket region: us-east-1

Event types
 Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

Moreover, in the *lambda_function.py* there are configured two environmental variables:

Code Blame 50 lines (39 loc) · 1.75 KB

```

1  import boto3
2  import cv2
3  import os
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  def handler(event, context):
8
9      # Set the matplotlib config directory to /tmp
10     # This is required to avoid permission issues when saving the plot
11
12     os.environ['MPLCONFIGDIR'] = '/tmp/matplotlib'
13     output_bucket = os.environ['DST_BUCKET']
14
15     s3 = boto3.client('s3')
16     input_bucket = event['Records'][0]['s3']['bucket']['name']
17     input_key = event['Records'][0]['s3']['object']['key']
18     filename = os.path.basename(input_key)
19     local_path = '/tmp/' + filename
20

```

Therefore, we add these two environmental variables to Lambda:

Edit environment variables

Environment variables

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](#)

Key	Value	
DST_BUCKET	cell-counting-ihona-results	<button>Remove</button>
MPLCONFIGDIR	/tmp/matplotlib	<button>Remove</button>
<button>Add environment variable</button>		

► Encryption configuration

Environmental variables are very convenient since they can be edited without changing Lambda. For example, if we decide to store the results in a different bucket, we would just go and edit the DST_BUCKET variable and it would be much easier than having to modify the lambda itself.

This is the overview of the created Lambda:

Lambda > Functions > hdbc-cell-counting

hdbc-cell-counting

Throttle Copy ARN Actions

▼ Function overview Info

Export to Application Composer Download

Diagram Template

hdbc-cell-counting

S3

+ Add trigger

+ Add destination

Description

-

Last modified

6 days ago

Function ARN

arn:aws:lambda:us-east-1:654654181466:function:hdbc-cell-counting

Function URL Info

-

Image Test Monitor Configuration Aliases Versions

Image

Deploy new image

No code preview available

Your function code is deployed as a container image. The AWS Cloud9 IDE cannot display your code.

Image URI

654654181466.dkr.ecr.us-east-1.amazonaws.com/hdbc-cell-counting@sha256:1d0941a9ae76adba7dc2fbd57ed3ad0c85e9fa2da5a3d8f50c36b81d2b9acb43

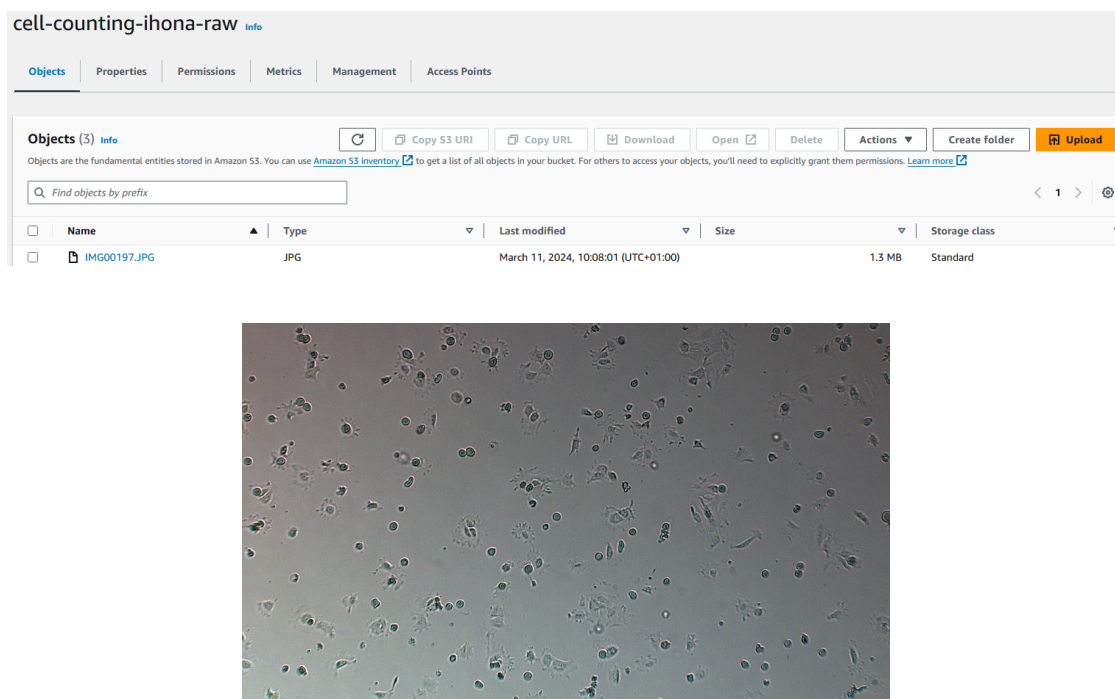
Architecture Info

x86_64

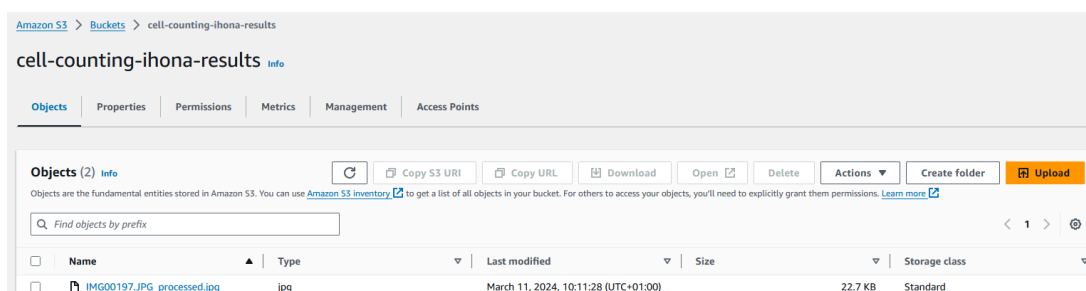
5. RESULTS

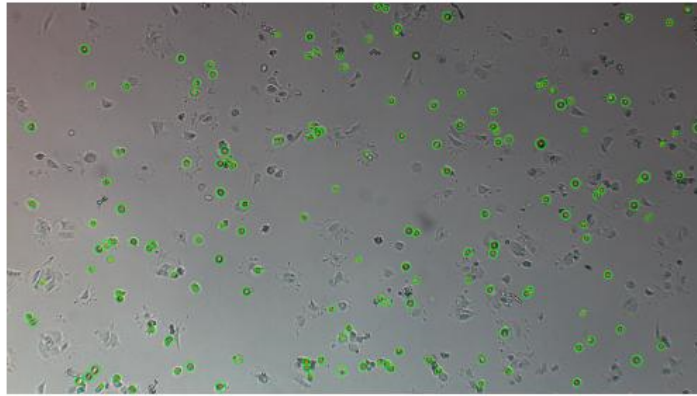
After configuring the Lambda and s3 buckets, we can test that everything works by doing the following:

1. Go to the raw s3 bucket and upload a new cell image from the images folder in the repository:



2. Go to the results bucket and check the processed image:





6. REFERENCES

This exercise was proposed and provided by Jordi Mateo, professor of the subject High Performance and Distributed Computing for Big Data (URV).

In the following link there is the repository used to solve the exercise, which contains the Python file, the Docker file, and all the other files mentioned before which are needed for the successful implementation of the exercise in AWS.

[1] <https://github.com/HDBC-17705110-MDBS/cell-counting>