Project 2 (Part I): PaaS (ECR approach)

Due by **April 14 10:00 AM (firm deadline)**

Summary

In the second project, we will develop another cloud application using PaaS resources. Specifically, we will build this application using AWS Lambda and other supporting services from AWS. AWS Lambda is the first and currently the most widely used function-based serverless computing service. We will develop a more sophisticated application than Project 1, as we are now more experienced with cloud programming and PaaS also makes it easier for us to develop in the cloud. This project will also give us the opportunity to learn a few more important cloud services, including AWS Lambda and Elastic Container Registry (ECR).

Our PaaS application will provide face recognition as a service on video frames streamed from the clients (e.g., security cameras). This is an important cloud service to many users, and the technologies and techniques that we learn will be useful for us to build many others in the future.

Description

Our cloud app will use AWS Lambda to implement a multi-stage pipeline to recognize faces in video frames collected from Internet of Things (IoTs) such as smart cameras.

1. The pipeline starts with a client, which represents an IoT, sending video frames to the cloud app.
2. The *face-detection* function accepts video frames from the client, performs face detection using a machine learning model, and produces the detected faces.
3. The *face-recognition* function performs face recognition on the detected faces using a machine learning model, and produces the names of the recognized faces.
4. The names of the recognized faces are sent back to the client.

In the first part of the project, we will focus on implementing the face detection and face recognition functions on Lambda.

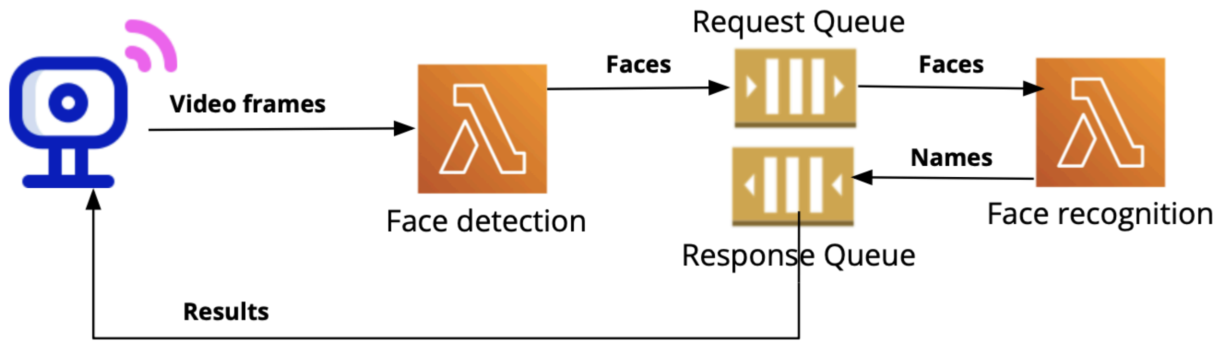

Fig 1: Architecture diagram of Project 2 (Part I)

Step 1: Set up ECR

Amazon Elastic Container Registry (Amazon ECR) is a fully managed container registry that allows you to store, manage, and deploy container images securely and reliably. We will use Docker images to store our Lambda function code along with dependencies in the Dockerfile and additional scripts, and store the Docker image in ECR. Amazon's free tier allows only 500 MB per month of storage for your private repositories in ECR. The machine learning platform (PyTorch) that we use for this project exceeds this limit and needs no more than 10 GB of ECR storage, which would incur no more than $2.00 (10 GB per month x 0.10 USD x 2 months = 2.00 USD) by the end of the project.

1) You can use the provided Dockerfile template as the starting point for building your container image for your Lambda functions. The provided Dockerfile serves as a basic template that sets up a Docker image for Lambda function, includes a *python-alpine* base image, essential packages such as awslambdaric for Lambda execution, a command to install Python packages using pip requirements.txt, and sets up an entry point for Lambda execution.
   Note: You **MUST** add the required code/packages in the template code to run and compile your function as follows:
   a) boto3
   b) facenet_pytorch
   c) awslambdaric
   d) requests
   e) Pillow
   f) opencv-python

2) You can use the following techniques to reduce the image size in AWS ECR (but focus on the core requirements of the project first):

a) Use the same image for both functions that you will implement. In order to do you can override "CMD" configuration as per your handler code when you deploy Lambda functions
b) Use a smaller base image such as python:${VERSION}-slim
c) Use only the required packages and libraries in the Dockerfile and requirements.txt
d) Install Torch and Torchvision for CPU (without CUDA) from the official torch webpage: https://download.pytorch.org/whl/torch_stable.html

Step 2: Set Up SQS Queues

1) Create the SQS Queues
The Lambda functions shown in Fig 1 relies on SQS queues for message passing and loose-coupling. For this we will create two SQS queues:
    a) SQS Request Queue
        i) The face-detection Lambda sends the detected faces to the request queue to trigger the face recognition function.
        ii) You MUST name your request queue: <ASU ID>-req-queue.
    b) SQS Response Queue
        i) The face-recognition Lambda sends the classification results to the response queue.
        ii) You MUST name your response queue: <ASU ID>-resp-queue.
        iii) The SQS response queue is polled by the client for classification results.

Step 3: Create the Face Detection Function

1) Configuration**:**
    a) The name of this Lambda function MUST be "face-detection".
    b) Create a "Function URL". This URL will be used by the client to send requests to the Lambda function.
    c) Roles: Ensure at least the following permission policies are attached to the role associated with the face-detection Lambda:
        i) AWSLambdaSQSQueueExecutionRole
        ii) AWSLambdaVPCAccessExecutionRole

2) Design**:**
    a) The face detection function is invoked by the client via a POST request on the function URL. The function should extract "content" (Base64-encoded input file), "request_id" (a unique ID to identify the requests), and "filename" (na me of

the input test image) from the JSON-formatted "body", obtained from the "event" of the Lambda handler.

b) It performs face detection on the received video frame using MTCNN (Multi-task Cascaded Convolutional Networks). It is a cascaded network that consists of three stages: face detection, facial landmark localization, and face alignment. These stages work together to detect faces and facial landmarks in an image. You can refer to face-detection code to implement the Lambda function.

c) It sends the detected faces to a SQS request queue (<ASU ID>-req-queue) to trigger the face recognition function.

Step 4: Create the Face Recognition Function

1) Configuration:
   a) The name of this Lambda function MUST be "face-recognition".
   b) Roles: Ensure at least the following permission policies are attached to the role associated with the face-detection Lambda:
      i) AWSLambdaSQSQueueExecutionRole
      ii) AWSLambdaVPCAccessExecutionRole
2) Design:
   a) The face-recognition function is triggered when a detected face is sent by the face-detection function to the SQS request queue.
   b) It performs face recognition by:
      i) Directly initialize the model "resnet = InceptionResnetV1(pretrained='vggface2').eval()"
      ii) Compute the face embedding using a ResNet model and compare it with embeddings from the weights file.
      iii) Identify the closest match and return the label.
      iv) You can refer to the face-recognition-code to implement your Lambda function.
   c) It sends the result to the SQS response queue (<ASU ID>-resp-queue).
      The message pushed to the response queue is a dict with following key-values {"request_id": *<ID of the request received by the face-detection>* , "result": *<the classification results from face-recognition>* }

Note: During development when your setup is idle make sure no messages remain "In-flight" in the SQS response queue which is used to trigger the face-recognition Lambda. If the message is not properly deleted from the queue it will trigger the Lambda recursively and might incur cost after the AWS free tier limit is over. To avoid this make sure to purge any message in the queue or disable the trigger to stop Lambda invocations.

Testing

Make sure that you use the provided autograder and follow the instructions below to test your project submission. Failure to do so may cause you to lose all the project points and there will be absolutely no second chance. Use a Linux-based system to test your project.

1. Download the zip file you submitted from Canvas.
2. Download the autograder from GitHub:
   https://github.com/CSE546-Cloud-Computing/CSE546-SPRING-2025.git
   In order to clone the Github repository follow the below steps:
   a. git clone https://github.com/CSE546-Cloud-Computing/CSE546-SPRING-2025.git
   b. cd CSE546-SPRING-2025/
   c. git checkout project-2-part-1
   d. Create a directory "submissions" in the CSE546-SPRING-2025 directory and move your zip file to the submissions directory.
3. To facilitate the testing, a standard video frame dataset and the expected recognition output of each image are provided to you at:
   a. Input: dataset/video-frames
   b. Output: dataset/Results_video_frames_100.csv
4. Create a Grading IAM User. The TA will use this only for grading.
   a. You MUST name the grading IAM user as "cse546-AutoGrader"
   b. For Project-2**,** the Grading IAM requires only these permission
      i. IAMReadOnlyAccess
      ii. AmazonSQSFullAccess
      iii. AWSLambdaSQSQueueExecutionRole
      iv. AWSLambda_ReadOnlyAccess
5. Prepare to run the autograder
   a. Install Python: sudo apt install python3
   b. Populate the class_roster.csv
      i. If you are a student; replace the given template only with your details.
      ii. If you are a grader; use the class roster for the entire class.
   c. Clean up your SQS queues.
6. Run the autograder

- To run the autograder: python3 autograder.py --num_requests 100 --img_folder="<dataset folder path>" --pred_file="<output classification csv file path>"
- The autograder will look for submissions for each entry present in the class_roster.csv
- For each submission the autograder will
  - The autograder extracts the credentials.txt from the submission and parses the entries.
  - Use the Grader IAM credentials to test the project as per the grading rubrics and allocate grade points.
- The autograder has a workload generator component to generate requests to your face-detection Lambda function.

7. To facilitate testing, you MUST use only the resources from the US-East-1 region.

8. Sample Output

```
+++++++++++++++++++++++++++++++ CSE546 Autograder  +++++++++++++++++++++++++++++++
- 1) Extract the credentials from the credentials.txt
- 2) Execute the test cases as per the Grading Rubrics
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++ Autograder Configurations ++++++++++++++++++++++++++++
Project Path: /home/local/ASUAD/kjha9/git/GTA-CSE546-SPRING-2025/Project-2/part-1/grader
Grade Project: Project-1
Class Roster: class_roster.csv
Zip folder path:
/home/local/ASUAD/kjha9/git/GTA-CSE546-SPRING-2025/Project-2/part-1/grader/submissions
Grading script:
/home/local/ASUAD/kjha9/git/GTA-CSE546-SPRING-2025/Project-2/part-1/grader/grade_project2_p1.py
Test Image folder path: ../../datasets/frames
Classification results file: ../../datasets/FaceRecognitionResults.csv
Autograder Results: Project-1-grades.csv
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++ Grading for Doe John ASUID: 1225754101 +++++++++++++++++++++
Extracted
/home/local/ASUAD/kjha9/git/GTA-CSE546-SPRING-2025/Project-2/part-1/grader/submissions/Project2-12257
54101.zip to extracted
File: extracted/credentials/credentials.txt has values ('XXXXXXXXXXX, 'XXXXXXXXXXX, ''XXXXXXXXXXX',
''XXXXXXXXXXX')
Credentials parsing complete.
-----------------------------------------------------------------
IAM ACCESS KEY ID: XXXXXXXXXXX
IAM SECRET ACCESS KEY: 'XXXXXXXXXXX
```

```
-----------------------------------------------------------------
Following policies are attached with IAM user:cse546-AutoGrader: ['IAMReadOnlyAccess',
'AmazonSQSFullAccess', 'AWSLambda_ReadOnlyAccess', ‘AWSLambdaSQSQueueExecutionRole’]
[IAM-log] AmazonSQSFullAccess policy attached with grading IAM
[IAM-log] AWSLambda_ReadOnlyAccess policy attached with grading IAM
[Cloudwatch-log] CAUTION !! You do not have a Cloudwatch alarm set. Kindly refer to the Project-0
document and learn how to set a billing alarm
-------------- CSE546 Cloud Computing Grading Console -----------
IAM ACCESS KEY ID: XXXXXXXXXXX
IAM SECRET ACCESS KEY: XXXXXXXXXXX
Face detection Lambda Function URL: XXXXXXXXXXX
SQS Response Queue URL:XXXXXXXXXXX
-----------------------------------------------------------------
----------------- Executing Test-Case:1 ----------------
[Lambda-log] The function: face_detection exists
[Lambda-log] The function: face-recognition-part-1 exists.
[Lambda-log] Points deducted:0
[Lambda-log] ---------------------------------------------------------
[Lambda-Trigger-log] SQS Trigger Found with Lambda func:face-recognition-part-1:
arn:aws:sqs:us-east-1:XXXXXXXXXXX:1225754101-req-queue-fifo.fifo
[Lambda-Trigger-log] Points deducted:0
[Lambda-Trigger-log] ---------------------------------------------------------
[SQS-log] The expectation is that both the Request and Response SQS should exist and be EMPTY
[SQS-log] - WARN: This will purge any messages available in the SQS
[SQS-log] ---------------------------------------------------------
[SQS-log] SQS Request Queue:1225754101-req-queue has 0 pending messages.
[SQS-log] SQS Response Queue:1225754101-resp-queue has 0 pending messages.
[SQS-log] Points deducted:0
----------------- Executing Test-Case:2 ----------------
Starting workload generator...
Workload complete! Dumping statistics...
[Workload-gen] ----- Workload Generator Statistics -----
[Workload-gen] Total number of requests: 100
[Workload-gen] Total number of requests completed successfully: 100
[Workload-gen] Total number of failed requests: 0
[Workload-gen] Total number of correct predictions : 100
[Workload-gen] Total number of wrong predictions: 0
[Workload-gen] Total test duration: 178.29808926582336 (seconds)
[Workload-gen] ------------------------------------------

[Test-Case-3-log] 100/100 completed successfully.Points:[20.0/20]
[Test-Case-3-log] 100/100 correct predictions.Points:[20.0/20]
[Test-Case-3-log] Test Average Latency: 1.7829808926582337 sec. `avg latency<3s`.Points:[60/60]
[Test-Case-3-log] ---------------------------------------------------------
Total Grade Points: 100.0
Removed extracted folder: extracted
Total time taken to grade for Doe John ASUID: 1225754101: 183.52135705947876 seconds
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Grading complete for Project-2. Check the Project-2-grades.csv file.
```

Submission

1. Submit the following in a zip file to Canvas:
   a. Credentials: Make a directory named “credentials”, and include your txt file named ‘credentials.txt’
      i. In the ‘credentials’ folder, create a txt file “credentials.txt” with following parameter values separated by commas in the order as mentioned below
         1. ACCESS KEY ID of the grading IAM user
         2. SECRET ACCESS KEY of the grading IAM user
         3. Function URL of the Face detection Lambda
         4. SQS response queue URL

```
kjha9@en4113732l:~/git/GTA-CSE546-SPRING-2025/Project-2/part-1/grader/cre
dentials$ cat credentials.txt
XXXXXXXXXXXXXXX,XXXXXXXXXXXXXXX,https://abcdefghijklmnopqr.lambda-url.us-
east-1.on.aws/
,https://sqs.us-east-1.amazonaws.com/123456789123/1225754101-resp-queue.f
ifo
```

   b. Face detection code: Make a directory named ‘face-detection’ and include your code file named ‘fd_lambda.py’
   c. Face recognition code: Make a directory named ‘face-recognition’ and include your code file named ‘fr_lambda.py’
   d. Note: DO NOT submit any additional Python files

2. You MUST name the zip file following the below naming convention: “Project2-<Your ASU ID>.zip”, e.g., Project2-1225754101.zip
   You can use the following command to create your zip file:
   zip -r Project2-1225754101.zip credentials/ face-detection/ face-recognition/
   Note: Extensions automatically generated by Canvas due to multiple submission attempts will be managed by the grading script.

3. Do not submit any other file.

4. Do not change your code after the submission deadline. We will compare the code you have submitted on Canvas and the code you run on AWS. If any discrepancy is detected, it will be considered as an academic integrity violation.

Rubrics

#	Test Objective	Test Criteria	Pass	Points
1	Validate the initial state of the resources (Lambda and SQS)	To check if 1) There exists two lambda functions with name “face-detection” and “face-recognition” 2) The SQS request and response queues must exist and be empty 3) The face-recognition Lambda has a SQS trigger enabled	1) Deduct 100 points if either of the Lambda functions does not exist. 2) Deduct 100 points if either of the SQS queues does not exist. 2) Deduct 10 points each if any queue not empty 3) Deduct 100 points if face-recognition Lambda does not have a SQS trigger.	
2	Validate the completeness, correctness and the latency of the requests	Run workload generator script on the provided function URL of the face-detection Lambda by the students with 100 requests 1) The HTTP response code must be 200 for all the requests 2) All the classification results must be correct 3) Total runtime must be within the limits suggested in the project document	For each of the 100 requests 1) +0.2 correct classification result (20) 2) +0.2 for completion of every request. (20) 3) Average latency for 100 requests (60) < 3 secs (60) >= 3 secs and < 4 secs (40) between >= 4 secs < 5 secs (20) >= 5 secs (0)	100

Policies

- Late submissions will **absolutely not** be graded (unless you have verifiable proof of emergency). It is much better to submit partial work on time and get partial credit for your work than to submit late for no credit.
- Every student needs to **work independently** on this exercise. We encourage high-level discussions among students to help each other understand the concepts and principles.

However, a code-level discussion is prohibited and plagiarism will directly lead to failure of this course. We will use anti-plagiarism tools to detect violations of this policy.

- **The use of generative AI tools is not allowed** to complete any portion of the assignment.