Project 3 (Part 2): PaaS

[CSE 546] [Fall_2024] Cloud Computing *Due*
*by **11/24/2024 at 11:59:59pm***

Summary

In the second project, we will build an elastic application that can automatically scale out and in on-demand and cost-effectively by using the PaaS cloud. 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. Our application will offer a meaningful cloud service to users, and the technologies and techniques that we learn will be useful for us to build many others in the future.

Description

Our complete cloud app will be a video analysis application that uses four Lambda functions to implement a multi-stage pipeline to process videos sent by users.

1. The pipeline starts with a user uploading a video to the input bucket.
2. Stage 1: The *video-splitting* function splits the video into frames and chunks them into the group-of-pictures (GoP) using FFmpeg. It stores this group of pictures in an intermediate stage-1 bucket.
3. Stage 2: The *face-recognition* function extracts the faces in the pictures using a Single Shot MultiBox Detector (SSD) algorithm and uses only the frames that have faces in them for face recognition. It uses a pre-trained CNN model (ResNet-34) for face recognition and outputs the name of the extracted face. The final output is stored in the output bucket.

The architecture of the cloud application is shown in Figure 1. We will use AWS Lambda to implement the functions and AWS S3 to store the data required for the functions.

Copyright © Ming Zhao, PhD

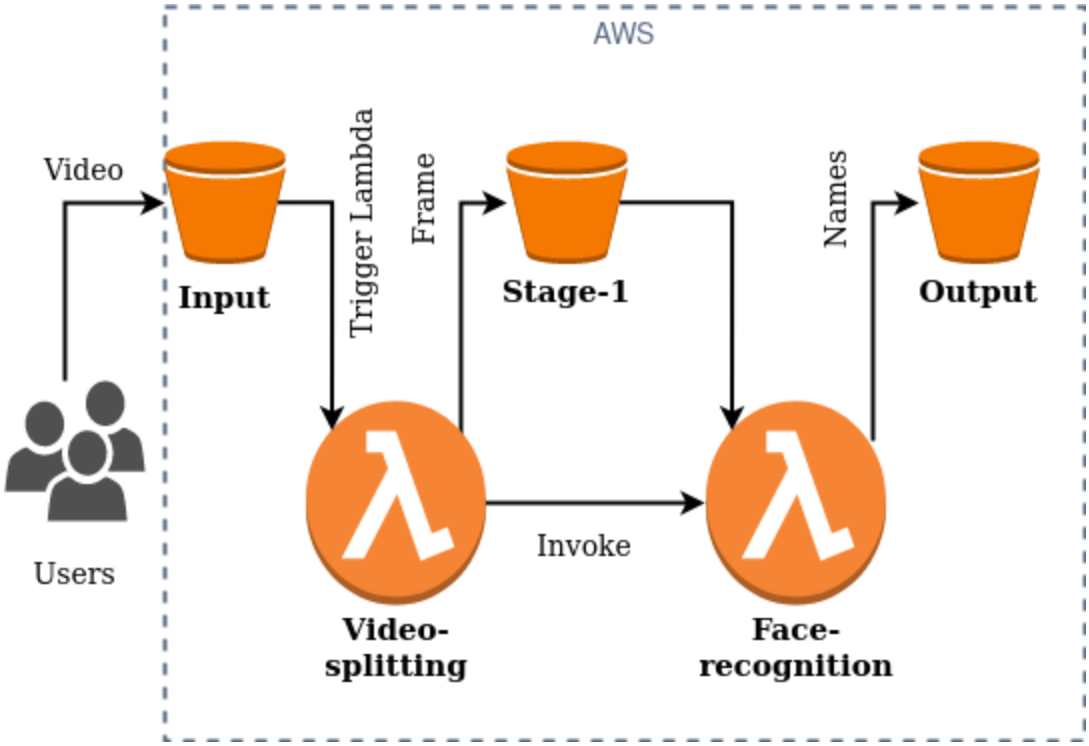

Fig 1: Architecture Diagram of Project 2

Input Bucket

- This bucket stores the videos uploaded by the workload generator.
- The name of the input bucket MUST be **<ASU ID>-input**. Each student submission MUST have only one input bucket with the exact naming. For example, if your ASU ID is "12345678910", your bucket name will be 12345678910-input.
- The input bucket must contain only .mp4 video files sent by the workload generator. (100-video dataset link)
- Each new video upload should trigger an invocation of the **video-splitting** Lambda function.

Stage-1 Bucket

- This bucket stores the result of the video-splitting function.
- The name of the input bucket MUST be **<ASU ID>-stage-1**. Each student submission MUST have only one "**<ASU ID>-stage-1**" bucket with the exact naming.
- The <ASU ID>-stage-1 bucket should have JPG images with the exact name as the input video name with .jpg extension. E.g. If the input bucket has 5 videos with the names test_00.mp4, test_01.mp4, test_02.mp4, test_03.mp4, test_04.mp4, then the stage-1

Copyright © Ming Zhao, PhD

bucket **MUST** have 5 images with the names exactly as the input video files: test_00.jpg, test_01.jpg, test_02.jpg, test_03.jpg, test_04.jpg.

Output Bucket

- This bucket stores the result of the face-recognition function.
- The name of the output bucket MUST be **<ASU ID>-output**. Each student submission MUST have only one "**<ASU ID>-output**" bucket with the exact naming.
- The output bucket should contain text files (.txt) with the same name as the input video with .txt extension. E.g. If the stage-1 bucket has 5 images with the names test_00.jpg, test_01.jpg, test_02.jpg, test_03.jpg, test_04.jpg, then the output bucket should have 5 text files with the names exactly as the original input video files: test_00.txt, test_01.txt, test_02.txt, test_03.txt, test_04.txt.
- The content of each text file is the identified person's name in the image. E.g. test_00.txt should contain only the name of the person identified, i.e. Trump.

Video-splitting Function

- The name of this Lambda function MUST be "**video-splitting**".
- The function is triggered whenever a new video is uploaded to the "**<ASU ID>-input**" bucket.
- The function gets the video file and splits the video into frames using the ffmpeg library. The function splits the given input in frames using the following command:

```
ffmpeg -i ' +video_filename+ ' -vframes 1 ' + '/tmp/' +outfile
```

  Note: To reduce the number of PUT requests, we will extract only one frame from each video.
- The function stores one frame with the same name as the input video in the "**<ASU ID>-stage-1**" bucket. E.g. If the <input_video> is test_00.mp4, then its corresponding frame is stored in a file named test_00.jpg in the "**<ASU ID>-stage-1**" bucket.
- This function invokes the face-recognition function asynchronously as soon as it finishes processing the function.
    - The face-recognition function is invoked with the invocation parameters: "bucket_name" and "image_file_name".
    - The output of this function stored in "stage-1" is passed as the "bucket_name" and the file in which the frame is stored is passed as the "image_file_name" in the invocation parameters.

Copyright © Ming Zhao, PhD

    - E.g. For a given input test_00.mp4, the face-recognition function invocation parameters would be {“bucket_name”:”<ASU ID>-stage-1”, “image_file_name”:”test_00.jpg”}.
- You can use the provided video-splitting code to implement your Lambda function.
  Note: To conserve our usage of ECR, do not use a container image to deploy this function. You can use Lambda UI to write your function code and add the FFmpeg library externally.

Face-recognition Function

- The name of this Lambda function MUST be “**face-recognition**”.
- The face-recognition function is triggered when the preceding video-splitting function finishes processing.
    - This function requires two parameters- “bucket_name” and “image_file_name”.
    - E.g. if the previous video-splitting function processed a video “test_00.mp4”, then the invocation parameters for the face-recognition function would be “bucket_name” set as “<ASU ID>-stage-1” and “image_file_name” set as “test_00.jpg”.
- The face-recognition function processes every image in the following manner.
    - Each image is the frame from the original video which goes through a sequence of OpenCV APIs to detect and extract faces from the image.
    - It computes the face embedding using a ResNet model and compares it with embeddings from the data.pt file.
    - It identifies the closest match. If the match is found, the recognized name is stored as a text file with the same name as the input image, and the file's content is the name of the recognized face.
    - The result of this function is stored in the ”<ASU ID>-output” bucket.
      E.g. if the image_file_name in the invocation parameter is test_00.jpg, then the result of the function is stored as test_00.txt in <ASU ID>-output bucket.
- You can refer to the face-recognition-code to implement your Lambda function.
- You can use the provided templates for the Dockerfile and handler code as the starting point for building your container image for deploying this function.
  Note: You **MUST** add the required code/packages in the template code to run and compile your function. You can use the following techniques to reduce the image size in AWS ECR:
    - Use a smaller base image such as python:${VERSION}-slim
    - Use only the required packages and libraries in the Dockerfile and requirements.txt

Copyright © Ming Zhao, PhD

- Install Torch and Torchvision for CPU (without CUDA) from the official torch webpage- https://download.pytorch.org/whl/torch_stable.html
- Save the data.pt in a separate S3 bucket instead of storing it in the container image and download from S3 when needed in the code.

Testing & Grading

- Create an IAM user with the following permissions to facilitate automated grading using our grading script. You can create policies from AWS UI -> IAM -> Policies.
  - **s3:Get***
  - **s3:PutObject**
  - **s3:List***
  - **lambda:GetFunction**
  - **cloudwatch:GetMetricData**
  - **s3:Delete***
- Use the provided workload generator to test your app thoroughly.
- The grading will be done using an automated grading script and following the rubrics provided below.
- You must use AWS region 'us-east-1' for all the S3 buckets and Lambda.

	Test Case	Test Criteria		Total Points
1	Validate the Lambda function	Check if there exist 2 Lambda functions with the names video-splitting and face-recognition.	There are 2 Lambda functions with the names- "video-splitting and face-recognition"; otherwise deduct 10 points	
2	Validate the names and initial states of S3 buckets	1) Check if there exist 3 S3 buckets with names - <ASU ID>-input, <ASU ID>-stage-1, and <ASU ID>-output. 2) Check if all the S3 buckets are empty initially.	<10-digit-number>-input, <10-digit-number>-stage-1, <10-digit-number>-output buckets exist and are empty before the 100-video test; otherwise deduct 10 points	
5	Check the total end-to-end latency.	Run workload generator on the provided URL by the students with 100 requests. Run grading script with Option 3. Grading script stops when either all 100 outputs are produced or 400 seconds have passed. Check the total end-to-end	The total end-to-end latency is <= **300** sec (30) The total end-to-end latency is <= **400** sec (20) The total end-to-end latency is > **400** sec: Deduct 1 point from **20** for each incomplete request	30

Copyright © Ming Zhao, PhD

		latency of the 100 video workload.		
4	Validate 100 images in the bucket stage-1, for the 100-video test with names the same as input videos	Run workload generator script on the provided URL by the students with 100 requests. Check there are 100 jpg images with names- test_00.jpg, …, test_99.jpg in stage-1 bucket.	Deduct 1 point for each wrong/missing image in the stage-1 bucket	20
5	Validate 100 text files in the bucket output, for the 100-video test with names the same as input videos	Run workload generator script on the provided URL by the students with 100 requests. Check there are 100 text files with names- test_00.txt, …, test_99.txt in the output bucket.	Deduct 1 point for each wrong/missing file in the output bucket	20
6	Check the correctness of the text file in the output bucket with the input video	Check if every identified name in the text file matches the input video.	Deduct 1 point for each wrong name	30

- Test your code using the provided workload generator and grading script. If they fail to execute, you will receive **0** points.

Submission

The submission requires three components; all must be done by ***11/24/2024 at11:59:59pm***.

1. Upload your source code to Canvas as a single zip file named by your full name: <lastname><firstname>.zip. Submit only the source code that you have created/modified: **handler.py, Dockerfile,** and **requirements.txt** for each function. Do not include any code not developed by you. Do not include any binary files.
2. Provide both the bucket names and your grading user credentials.

IMPORTANT:

- Failure to follow the submission instructions will cause a penalty to your grade.
- Do not change your code after the submission deadline. The grader 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

Copyright © Ming Zhao, PhD

Policies

1) 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 than to submit late for no credit.

2) You need to work independently on this project. We encourage high-level group discussions to help others understand the concepts and principles. However, code-level discussion is prohibited, and plagiarism will directly lead to failure in this course. We will use anti-plagiarism tools to detect violations of this policy.

Copyright © Ming Zhao, PhD