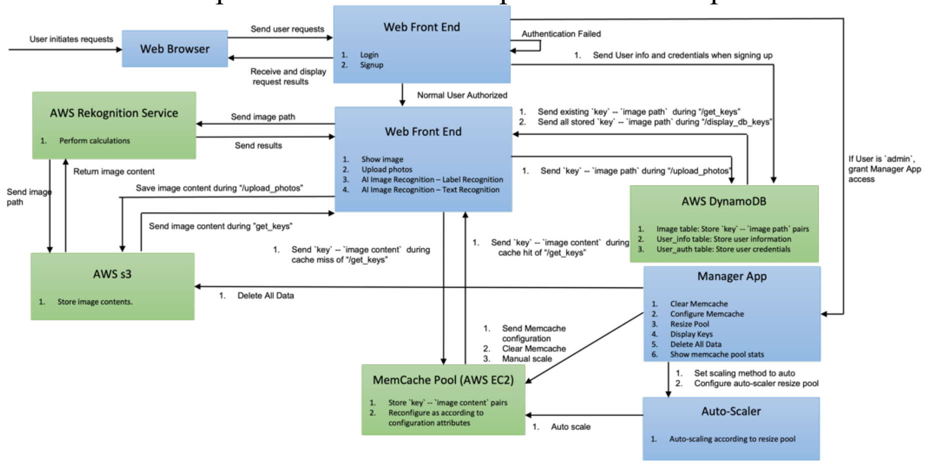
ECE 1779 Project 3 Documentation

Group 3

Part 1. Project Abstract

In this project, we continued to build on assignment 2 by adding new features and modifying the backend to run serverless. We implemented a new login system to separate different user sessions. Persistent states such as user information and credentials are saved via DynamoDB tables. User uploaded images will continued be saved in AWS s3. We also utilized AWS Route53 and ACMs to provide our website a proper and decent URL (https://ece1779-a3-test.com). Label and text recognition services for images are added with AWS Rekognition services. All functionalities of assignment 2 are migrated to Lambda (except the MemCache pool, which scales accordingly) so that everything runs serverless. In addition, user info and user credential tables in DynamoDB can be backed up in case of failure. Admin user are able to perform manual backups. Figure 1 shows the overview of our system. Detailed architecture and implementations will be presented in the part 2 and 3.



User initiates requests
Web Browser
Send user requests
Receive and display request results
Web Front End
1. Login
2. Signup
Authentication Failed
1. Send User info and credentials when signing up
Normal User Authorized
AWS Rekognition Service
1. Perform calculations
Send image path
Send results
Web Front End
1. Show image
2. Upload photos
3. AI Image Recognition – Label Recognition
4. AI Image Recognition – Text Recognition
1. Send existing `key` -- `image path` during "/get_keys"
2. Send all stored `key` -- `image path` during "/display_db_keys"
1. Send `key` -- `image path` during "/upload_photos"
If User is `admin`, grant Manager App access
Send image path
Return image content
Save image content during "/upload_photos"
Send image content during "get_keys"
1. Send `key` -- `image content` during cache miss of "/get_keys"
1. Send `key` -- `image content` during cache hit of "/get_keys"
AWS DynamoDB
1. Image table: Store `key` -- `image path` pairs
2. User_info table: Store user information
3. User_auth table: Store user credentials
AWS s3
1. Store image contents.
1. Delete All Data
Manager App
1. Clear Memcache
2. Configure Memcache
3. Resize Pool
4. Display Keys
5. Delete All Data
6. Show memcache pool stats
1. Send Memcache configuration
2. Clear Memcache
3. Manual scale
MemCache Pool (AWS EC2)
1. Store `key` -- `image content` pairs
2. Reconfigure as according to configuration attributes
1. Set scaling method to auto
2. Configure auto-scaler resize pool
Auto-Scaler
1. Auto-scaling according to resize pool
1. Auto scale

Figure 1. Overall System Architecture

Part 2. Application Components and Functions

1. Front End: Manages user requests
    a. Upload photos: Allows users to upload a photo along with a key. The key-filename pair is stored in the AWS DynamoDB while the file itself, i.e., the image content is stored in AWS S3.
    b. Get a photo with key: The full workflow when the front end receives a get photo request is displayed in figure 2. There are three possible cases when a key is an input by the user. Their respective workflows are shown below. Numbers indicate their steps labelled in figure 3.

Cases	Workflow
Key in MemCache	Step 1 → Step 2: Yes → Show image
Key in RDS	Step 1 → Step 2: No → Step 3: Yes → Step 4 → Step 5 → Step 6 → Show image
Key not found	Step 1 → Step 2: No → Step 3: No → "Unknown Key!"

Table 1. /get_photos request workflow

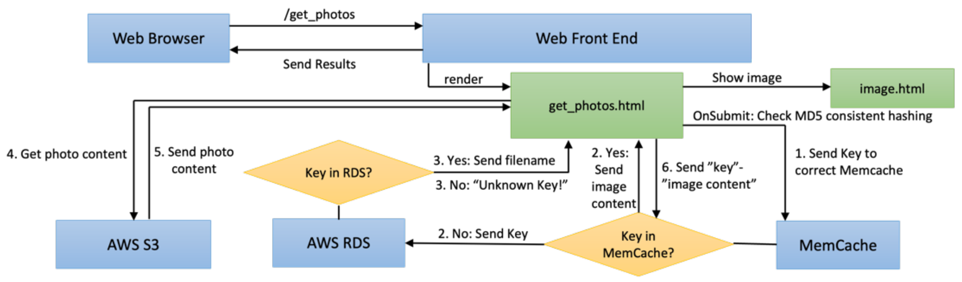


Figure 2. /get_photos request workflow

c. Login and Signup system: Before users can use our application, we ask the user to signup for an account first. Once signed-up, the user info along with credentials will be stored in DynamoDB, which can live through stateless serverless cycles. In addition, for website management, we allow an admin user to be set, which will grant him or her access to the manager app portal, allowing robust control over scaling policy, checking stats, configuring memcache, and so on.

d. The whole system is packaged into a custom domain, ece1779-a3-test.com, which can be accessed by link https://ece1779-a3-test.com.

2. DynamoDB Database:

User_info and User_auth table are backed up to s3 on daily basis (default). Admin user can change the default backup time or perform manual backup in the manager app portal.

a. Image table:

User_id	**Image Key**	**Image Path**
(Primary Key)	**(Sort Key, Unique for same User_id)**	
The User_id, i.e. the current user's session. This allows the backend to correctly identify individual users.	Image Key the user setup. No duplicated image keys are allowed under same User_id.	The respective path in s3 to find the image content.

b. User_info table:

User_id	**Username**	**First Name**	**Last name**	**email**	**birthday**	**phone**
(Primary Key)	**(Unique)**					
The User_id, i.e. the current user's session.	Set up by user.	Set up by user.	Set up by user.	Set up by user.	Set up by user.	Set up by user.

c. User_auth table:

User_id	**Username**	**Password**
(Primary Key)	**(Unique)**	
The User_id, i.e. the current user's session. This allows the backend to correctly identify individual users.	Set up by user. This is used to check credentials of users	Set up by user. This is used to check credentials of users.

3. Manager App:

a. Configure Memcache attributes: Two attributes can be set, being the capacity of the cache (in MB) and the replacement policy (Random Replacement (*RR*) / Least Recently Used (*LRU*)). The same attributes will be configured for all running memcache nodes. For new memcache nodes due to scaling up, they will all be configured to the most recently set configurations.

b. Resize Pool: There are two methods to resize the pool, being manual mode and auto scaling.

   i. Manual mode: Resizing of the memcache pool depends on user set values. Whenever a new node is running, its configuration data is automatically set according to the most recent memcache attributes.

   ii. Auto-scaling mode: Four attributes can be configured as shown in table 2. Whenever the user sets scaling to this mode, the following attributes' values will be posted to auto-scaler for handling all auto-scaling jobs.

c. Display current database keys: Front End requests the database for all key-filename pairs stored. There shall be a one-to-one relationship between each of these pairs to the image contents stored in the local file system.

d. Display current Memcache keys: Front End request MemCache for all keys stored in MemCache. The order starting from the top represents the least recently used keys down to the most recently used keys.
e. Clear memcache data: Clear all currently running memcache node.
f. Delete all data: Delete all image content in AWS S3, all key-filename pair in AWS RDS database, and clear all currently running memcache node.
g. Display Memcache stats chart: The following data are shown for the last 30 minutes at 1-minute granularity. 1. Number of nodes 2. Miss rate 3. Hit rate 4. Number of items in cache 5. Total size of items in cache. Since the period of each cloud watch data put by memcache is 5 seconds, the statistics are averaged over 1-minute granularity.
h. Configure DB backup: DynamoDB backups immediately on the current timestamp. A maximum of 3 backups will be stored in AWS. Only tables user_info and user_auth will be backed up

4. Auto-Scaler: Auto scale MemCache pool according to the attributes in Table 2 set by Manager-App. Each MemCache is a new AWS EC2 node. The miss rate is checked every one minute, and averaged through all nodes.

5. MemCache: Memcache stores part of the data in a "key"-"image content" form according to locality. It can improve the performance of the code by reducing the number of times to retrieve data from the database. The two attributes are the same as assignment 1, being Capacity and Replacement Policy. Each running MemCache will send its stats to AWS CloudWatch every five seconds. It will contain the current No_items, the current Total_size, the No_requests received during this five seconds, the Miss_rate, Hit_rate over the last five seconds, which is the same as in Assignment 1 and 2.

6. AWS S3: Stores the content of the images to be accessed by Front End.

7. AI Recognition: AI recognition has two functions to recognize the image: text detection and label detection. In text detection, it would detect all English text in the image. And the label detection would recognize all objects appeared in the graph with a convenience rate (probability).

Rekognition	
Label	It can identify the items in the images and will return the 10 highest properties with the highest confidence and the value of confidence.
Text	It can detect all text in the images and return the text.

Part 3. Application Architecture

With the whole application going serverless, all background processes are moved to AWS Lambda.

1. Interaction with DynamoDB: All CRUD operations to the database are implemented through AWS Lambda. All lambda functions are triggered by click events, which can all be seen in Figure 1. We will list two examples to demonstrate the workflow. The other operations such as uploading images and signing up, follow similar workflows.
    a. Login (Front End): User enters username and password → Application hashes the password → Application calls lambda function with parameters {username, hashed password} → Lambda function checks database for user credentials → if found: check with hashed password; if not found: Create response → Lambda function sends result back to Application → Application acts according to result.
    b. Database backup (Manager App): Admin logins (step a.) → Admin performs database backup → Application calls lambda function → Lambda function backs up database → Lambda function deletes old backup if # backup > 3 → Lambda function sends result back to Application.

2. Custom Domain Name for application: We deployed our application on custom domain ece1779-a3-test.com through routing the traffic to our API gateway API towards the custom domain. This is done with AWS Route53. With the purchased domain, we certified it with a DNS record in Route53 and created the domain in AWS API Gateway. This allowed us to successfully route all traffic to https://ece1779-a3-test.com.

3. Rekognition: In our program, we utilized AWS Rekognition service, which allows the user to get the information returned from Rekognition. We created a lambda function and used the AWS Rekognition API to implement this feature. Once the program sending requests to AWS API Gateway. API gateway can trigger the lambda functions to get the image in S3 through the image's name and detect the labels or text in the image. When the front end has received a request for AWS Rekognition from the user, two workflows will be shown below using Figure 2.

Cases	**Workflow**
Key in DynamoDB	Step 1 → Step 3: Yes → Step 4 → Step 5 → Show image and return value
Key not found	Step 1 → Step 2: No → Step 3: No → "Unknown Key!"

Part 4. How to use our application

- **Normal User**
  As a normal user, creating an account is required before using our application. Our application allows authorized users to upload and store images with a key, access these images with the key, and performing simple classifications and labeling towards the image. The signup process is only needed once since the data is safely stored in a DynamoDB. The original password is not stored in the database, rather the hashed password. This is to prevent even authorized personals such as the admin to gain access to the password or to avoid potential data leaks were it for our database to be hacked.
- **Admin User**
  Admins are granted configuration and stat checking operations. They can modify database backup options, configure parameters of the memcache, setup auto-scaler rules, and so on. In addition, admins are also allowed to check memcache stats, memcache keys, database keys, and other operations.

Part 5. Cost Model

The cost model is calculated with AWS pricing calculator under US-East-1 region. We calculate base on 100 total requests per user per day. Total cost for 10 Users/month for 6 months: 591.6 USD, 1000 Users/month for 6 months: 2695.92 USD. 1 million Users/month for 6 months: 375638.28 USD.

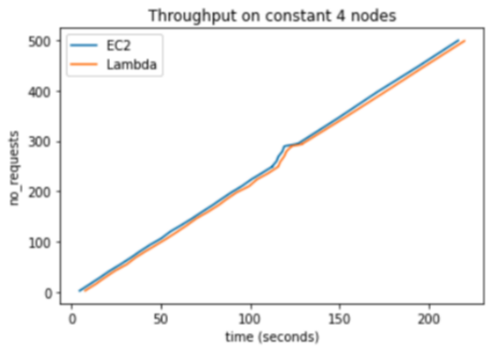
User	**Service**	**Upfront**	**Monthly**	**6 months**	**Configuration Summary**
10 Users / month	Lambda	0	1.28	7.68	**# of invocations: 100 / day per user**
	DynamoDB provisioning	180	26.39	248.34	Reserved capacity term: 1 Year Data storage size: 1 GB
	EC2	0	54.56	315.36	# of instances: 10, t2.micro
	Rekognition	0	0.5	3	**# of invocations: 500 / month**
	Route53	0	0.51	3.06	Hosted Zones (1)
	S3 standard	0	2.36	14.16	100 GB / month (2MB / image)
1000 Users / month	Lambda	0	13.28	79.68	**# of invocations: 100 / day per user**
	DynamoDB provisioning	180	28.64	261.84	Reserved capacity term: 1 Year Data storage size: 10 GB
	EC2	0	105.12	630.72	# of instances: 10, t2.small
	Rekognition	0	50	300	**# of invocations: 50000 / month**
	Route53	0	1.7	10.2	Hosted Zones (1)
	S3 standard	0	235.58	1413.48	10 TB / month (2MB /image)
	Lambda	0	13281.97	79691.82	**# of invocations: 100 / day per user**

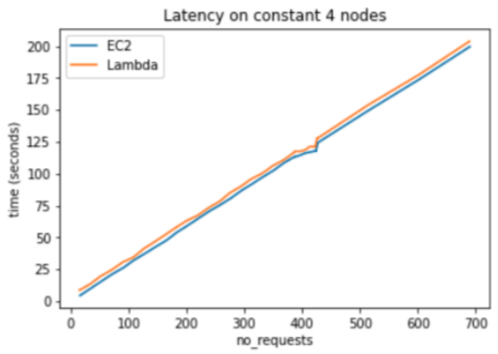
1M Users / month	DynamoDB provisioning	180	282.14	1782.84	Reserved capacity term: 1 Year Data storage size: 1 TB
	EC2	0	209.51	1257.06	# of instances: 10, t2.medium
	Rekognition	0	25950	155700	**# of invocations: 50000000 / month**
	Route53	0	800.5	4803	Hosted Zones (1)
	S3 standard	0	22067.26	132403.56	1000 TB / month (2MB /image)

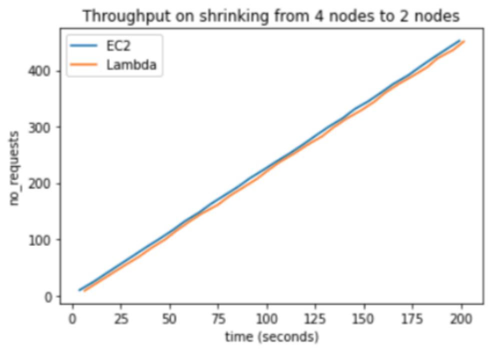
Part 6. Results

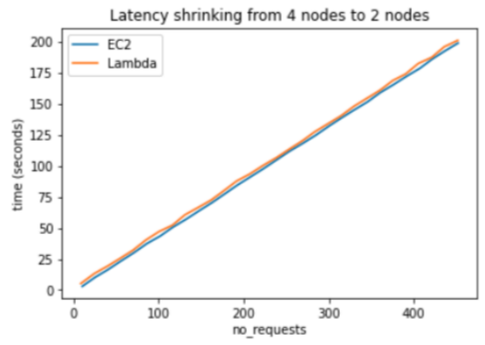
In project3, we tested the throughputs and latencies of previous tests on serverless computing (on lambda functions). We compared the ec2 program and lambda problems performance in the following plots.

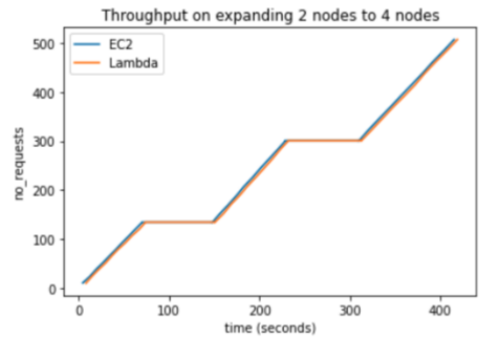
1. Average image size: 100kb per image
2. Memcache Capacity: 1000Mb; Memcache Policy: Random Replacement
3. Read:Write ratio: 50:50

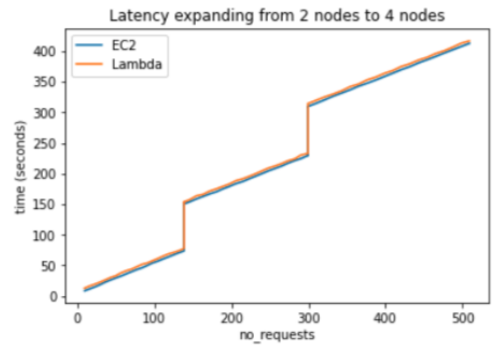












Part 7. Result Discussions

Detailed results regarding EC2 have been discussed in assignment 2. For server-less computing, the performance is similar to the EC2 server one except that server-less computing needs more time to solve the request. The result is expected because there will be a delay between sending a request and application execution and also cold start issues in AWS Lambda functions. Besides, the lambda function is stored in the AWS Lambda, which should not contain any states to prevent collision. Therefore, we can conclude that the server-less computing is good for short-term tasks and EC2 is good for long-term and steady state operation as of today.

Part 8. Contribution Table

Yih CHENG	Finished Login System and Backend Lambda migration
JiaWei MA	Finished Label Rekognition implementation and application result analysis
Ho Wan LUO	Finished Text Rekognition implementation and application result analysis