CSCI 4145 Term Project Report

Dhairy Raval B00845519

→ Menu Item Requirements

- **What did I build?**

For 4145's term project I wanted to build a project that could solve some real-world problem that may/may not already have existing solutions, however, what I wanted was something lightweight, effective, and easy to modify, retrofit into existing applications. I decided to cover the following use case:

A Dalhousie student wants to enter the CS building after it's closed. The student has been authorized to do so but he forgot to bring his Dal card to verify his identity and gain access to the front door. Now the student is locked out of the building even though he has the appropriate authorization. If the building can implement a cost-effective tool to verify a person's identity and can authorization in real-time we can tackle this challenge.

I decided to build a simple application that can cover the previous use case of verifying users without an external id card. For this, I am storing each authorized user's face data and verifying it through AWS Rekognition. Additionally, I wanted to make sure that someone was always notified about such events where people are attempting to enter a building after closing. This was covered using AWS SNS.

Here's the list of all the AWS services I have used, their justifications, and a brief mention of alternatives (if applicable):

→ **Compute Category:**

- **Lambdas** for creating faceprints (user's face's data), testing/verifying against other images, processing input, and executing the storage and output processes. I decided on Lambda for its cost-effectiveness and I only need my application to execute on an *as-need* or *on-call* basis. I can get away with a serverless implementation and avoid the work needed to host stuff (for ex: ec2)

- **Step Function** used in the verification process to take the user input from the API gateway, pass it to the lambda implementation responsible for the verification process and then instruct the SNS about the publishing. Using Step Functions for this process at such an early stage makes sense and I can easily attach new services or features to this process with ease. For ex: If I want to add the ability to process multiple users' faces parallelly (if they are in the same image), using step functions makes this possible and relatively easy to implement.

→ **Storage Category:**

- **S3** to store raw images containing the user's faces. Since S3 is highly scalable and you only pay for what you use, I don't have to worry about transitioning databases if the project requirements increase. Additionally, it's highly durable, has low latency and high throughput which is very important for this project as it's dealing with real-time data/requests.

- **DynamoDB(additional)** is used to store the user's faceprints in collections used by recognition. Similar to S3, DynamoDB provides high durability and availability while maintaining good performance. Also as this is just used alongside Rekognition I benefit from the fact that it's fully managed by AWS itself and I don't have to spend time configuring it.

→ **Network Category:**

- **API Gateway** to handle client requests and input data. API Gateways are flexible (you can communicate with them through a script, an API call, or even a third-party application) this is great for my project as someone looking to add it to their organization probably has the architecture set up already. It's easy to create additional endpoints and modify existing ones, and it provides built-in security features, such as authentication and authorization. We could consider a few alternatives like EventBridge or AppSync however, both of those services are not as lightweight or cost-effective as Gateways while also costing more.

→ **General Category:**

- **Rekognition** is the most important aws service in this project. Rekognition is highly scalable and provides many built-in security features like data encryption and access control which will be required if the scale of the project were to increase. Additionally, we know that the data it produces is highly accurate so the rate of false-positive and false negatives is expected to be low.

- **SNS** is used to send email messages to a predefined email address about the events and interactions with the application. I decided to use SNS as it provides real-time messaging capabilities. It is also very flexible and can be used to support different messaging protocols such as SQS, SMS, email, HTTPS, etc.

→ Deployment Model

The deployment model for this project could be roughly summarized as a serverless architecture that utilizes AWS Lambda and other serverless services, combined with cloud-based storage, messaging, and database services. This deployment model leverages the serverless architecture offered by AWS Lambda, combined with other managed services such as S3, DynamoDB, Rekognition, API Gateway, and SNS, to build a scalable, cost-effective, and highly available application. The serverless approach allows me to modify and edit the lambda functions as I please without having to manage the underlying infrastructure, while the other services provide scalable storage, messaging, and database capabilities.

→ Delivery Model

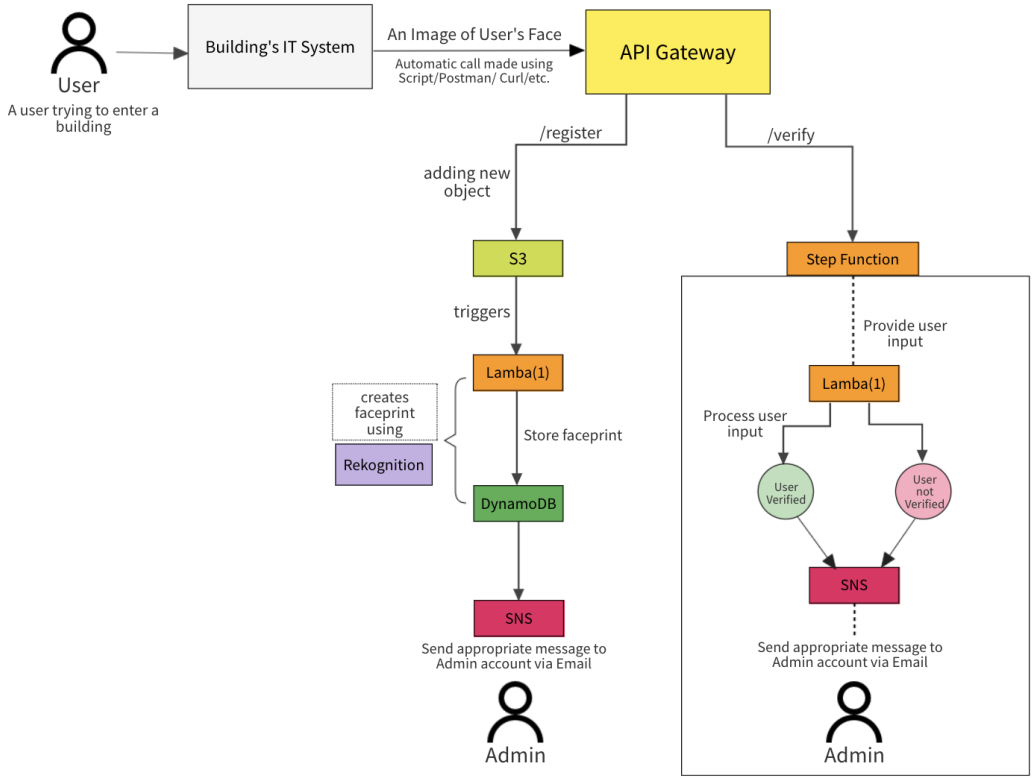
The most appropriate delivery model for this project would be Function-as-a-Service (FaaS) as we expect the user to interact with the API end of our application and expect an output at the SNS (or email) end of our application. What happens inside the application has little to no use for our users or even the organizations adopting our application (as long as it's cost-effective and accurate.

This is the image of a FaaS used in the lecture slides, we can see how if I were to add on DBs for store and Rekognition, and SNS all of it would be inside the application and the services platform. The user only interacts with the API Gateway and expects the output at the email address.

Img ref: https://thenewstack.io/why-netflix-rolled-its-own-node-js-functions-as-a-service-runtime/

→ Final Architecture

A high-level overview of the project and how the different aws services interact with each other:



Tool used: Mockflow.com

- **Where is the data stored?**

The application data is stored in a combination of S3 objects and DynamoDB collections. S3 stores the raw, unprocessed images of users (these images may contain more than 1 face). The collections are responsible for storing the processed faceprints from these raw images. Aws Rekognition utilizes these faceprints to find the best possible match (if it exists) among other items in the collection.

- **Programming language used and where?**

The language that I decided to code in for this project is Python, mainly because it is not as verbose, and development in Python is faster for me as I already have some experience building lambdas and scripts using Python. Additionally, it is worth noting that all the code files themselves only do a very basic task (as most of the heavy lifting is handled by aws) so the language chosen does not play a very big role in this project.
There are 3 main parts of the application that use Python:

- app.py script – this will be the script used by an organization to register or verify a user
- lambda func (register) – lambda function used to register new users
- lambda func (verify) – lambda function used to verify a given user

- **How is your system deployed to the cloud?**

The application starts with a call to the app's API (handled by aws API Gateway). All the processing is done via serverless components like lambdas and step functions which means that I don't spend resources and credits trying to host services. There are called and executed as needed.
→ If the **"\register**" endpoint is called, then a new object is created in S3 which in turn triggers the lambda to process this new object and create its faceprint using Rekognition. If the faceprint is successfully created we store it in a dynamo DB collection. Finally, we send an appropriate message to the admin acc stating whether the user was registered successfully or not.
→ If the **"\verify"** endpoint is called, then the input is passed to a step function which triggers another lambda to process this input – create faceprint, and verify if faceprint matches any previously stored ones in the collection. Depending on whether we find a match, don't find a match or if the lambda crashes during execution we send the appropriate message to the admin acc.

- **About the system architecture**

The application follows a similar architecture to the A4 where we mostly rely on serverless components to do the processing and talk with the outside world using API gateways. I decided to use this architecture because it fit well with the use case of the project. I needed something lightweight that could be easily modified, or retrofitted on top of existing applications.
In the current state, the step function is not being effectively utilized as I haven't yet implemented the functionality of parallel processing of multiple images into the application. Additionally, I could store the messages into multiple SQS queues and then publish them through SNS when required instead of directly sending all replies to SNS.
There are 2 main things I want to fix in the coming weeks after the term ends.

→ Data Security & Vulnerabilities

Currently, there are many vulnerabilities in my application. In this section, I try to highlight some of the major security vulnerabilities that I could figure out, the threats they could cause, and how I can address them in future iterations.

- **Lack of Authentication and Authorization**

As the project does not use proper access-control policies and the IAM role (labRole) used to provision the services is overly permissive and grants permissions to other services that are not required, an attacker can take advantage of such vulnerabilities. This can affect the confidentiality, integrity, authenticity, and availability of data. A potential fix for this is to implement cloud security mechanisms like Digital Signatures, Transport Layer Security, and Hashing of data.

- **Denial of Service (DoS) attacks**

As the system is not properly configured to handle high traffic loads, it is vulnerable to DoS attacks. For example, API gateways or Lambda functions are not properly rate-limited, so an attacker could flood the system with requests, causing it to become unresponsive or unavailable. To prevent such attacks I need to configure the API gateway and add user authentication.

- **Injection attacks & Cross-Site Scripting (XSS) attacks:**

Currently, the lambdas responsible for processing the user input through the '\verify' or '\register' endpoints are not sanitizing the data. This allows threats of injection attacks where an attacker could manipulate the data stored in my Dynamo DB collections and affect the availability, integrity, and authenticity of my application. Moreover, if an attacker were to use a cross-site scripting attack and inject malicious scripts into the API, this could lead to the theft of sensitive user data or other malicious activities. This would affect all the parts of the CIA triad: Confidentiality, Integrity, Availability, Authenticity, and Non-repudiation.
In order to prevent such attacks, I can modify the lambda functions to sanitize the data, and implement services like CloudWatch or CloudTrail to monitor the services in my app at all times.

→ Transitioning to Private Cloud

When transitioning to a private cloud, there will be high upfront costs of setting up the servers and environment where the server should be stored. There will be additionally costs that we don't have to worry about when provisioning with aws such as software licencing costs, management and monitoring costs and wages for the people managing the software and hardware resources. Additionally, it will won't able to maintain the durability and availability of ~99.999% with less than a minute of recovery time needed that AWS provides. There are able to achieve this as they have multiple data centers and server rooms spread out across the world. In order to achieve something even remotely similar to this we would have to spend millions of dollars just setting up back-ups across the world for our application. For almost all companies and organizations this is simply not feasible.

In this section I have tried to come up with a rough estimate of costs that what we should be expecting to pay to set up and run the application on a private server rather than an AWS (in 1 year).

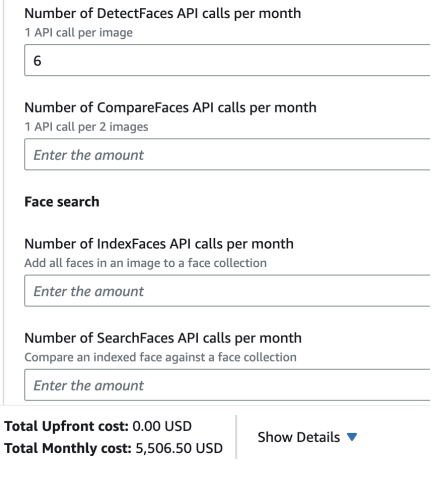
All the references are mentioned at the end of the document.

Sr No.	Service	Setup Cost ($)	Maintenance Cost ($)	Total Cost ($)	Note
1	API Gateway	~5000	~2500	7500	Setup costs will decrease for the next year ref [1]
2	Lambda * 2	1500 * 2 = 3000	150-200$	3150	Assuming that there is little to no maintenance required here. Depends on whether the clients change their requirements. ref [2]
3	S3 storage + Dynamo DB	37500	~ 600 * 12 = 7200	44,700	Assuming a medium-level storage solution. Setup – $37500 (ref [3-1]) Maintenance – 600 (per month) (ref[3-2])
4	Rekognition	6000 - 300,000	50,000	150,000	Assuming set-up costs ~100,000 Maintenance (not very accurate) ~50000 ref [4]
5	Step Function	1000	200	1200	Assuming we will use an alternative like Astronomer airflow ref [5]
6	SNS service	–	~10497	10500	Assuming we will use an alternative like RabbitMQ ref [6]

Final total cost: 7500 + 3150 + 44,700 + 150,000 + 1200 + 10500 = $ **217050**
The final total cost represents the rough estimate to host and maintain the application for the initial year on a private cloud. This estimate assumes that our application is scaled to a mid-sized client with ~200k connections per day.

→ Cloud Mechanisms and their Costs

Out of all the cloud mechanisms and services that I have deployed for this application. AWS Rekognition. If we assume we want to process ~200k images a day, which is ~6 million images per month. We are looking at ~ 5,500 USD per month just to support Rekognition (ref [7]).



→ This is a quote from the Amazon Pricining Calculator for our projected demands. Additionally, we still have to consider custom labels and video recognition as future implementations which will increase the cost even further.

→ Future implementations

As mentioned previously there are certain fixes that I want to implement in the future to better secure the data. These include

- Implementing cloud security mechanisms like Digital Signatures, Transport Layer Security, and Hashing of data.
- Configuring API Gateways and implementing proper user authentication
- Sanitizing user data to prevent Injection and Cross-site scripting attacks

Once these basic security features are added to the application I plan on modifying the step function to support parallel processing to images or video. That way the application will be able to verify multiple users at the same time. This ultimately results in a better UX for the people trying to verify their identity to enter the premises along with other employees.

Another feature to add is logging and monitoring events and data that interact with and passes through the application. Currently, AWS CloudWatch or CloudTrail look like the ideal services to cover these use cases.

References

[1] How much does it cost to build an API
[2] How Much Does it Cost to Build a Python Application?
[3-1] How much does it cost to build a data storage app?
[3-2] Database Server Price
[4] AI Pricing: How Much Does Artificial Intelligence Cost?
[5] Managed Apache Airflow
[6] CloudAMQP - manage rabbitmq
[7] AWS Rekognition Pricing Calculator